

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO
RIO GRANDE DO NORTE

Iéber Santos de Moura

SISTEMA DE FOTOALBUM UTILIZANDO TECNOLOGIAS MÓVEIS

NATAL/RN
2007

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO
RIO GRANDE DO NORTE

Iéber Santos de Moura

SISTEMA DE FOTOALBUM UTILIZANDO TECNOLOGIAS MÓVEIS

NATAL/RN
2007

IÉBER SANTOS DE MOURA

SISTEMA DE FOTOÁLBUM UTILIZANDO TECNOLOGIAS MÓVEIS

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do grau de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador:

Ulisses Telêmaco, Msc.

**NATAL/RN
2007**

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Moura, Ieber Santos de

Sistema de fotoalbum utilizando tecnologias móveis / Ieber Santos de
Moura. – Natal: FARN, 2007.
54f.

Orientador: Ulisses Telemaco (M.Sc.)

Monografia (Especialização em Desenvolvimento de Sistemas
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio
Grande do Norte.

1. Desenvolvimento de Softwares – Monografia. 2. Fotoalbum –
Monografia. 3. Tecnologias Móveis – Monografia. Plataforma JAVA –
Monografia. I. Título.

RN/FARN/BC

CDU 004.4

RB- 30.712

FARN

BIBLIOTECA

Nº Reg. 10000903

16/04/08

1045
M884S
2007
P201

IÉBER SANTOS DE MOURA

SISTEMA DE FOTOÁLBUM UTILIZANDO TECNOLOGIAS MÓVEIS

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em _____ de _____ de _____, pela banca examinadora composta pelos seguintes membros:



Ulisses Telemaco, Msc.

Orientador



Membro

Examinador

Agradeço em primeiro lugar a Deus, aos meus pais, irmã, minha namorada, parentes e amigos, ao professor Gleydson Lima, ao professor e orientador Ulisses Telemaco pelo apoio e encorajamento contínuos na pesquisa, aos demais Mestres da casa, pelos conhecimentos transmitidos e à Diretoria do Curso de Graduação da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte pelo apoio institucional e pelas facilidades oferecidas.

*"A imaginação é mais importante que o
conhecimento."*

A. Einstein.

RESUMO

Dois grandes avanços do homem moderno foram: a invenção do computador e a criação da telefonia móvel. Essas descobertas criaram uma nova era na comunicação. Com surgimento da *Internet*, as possibilidades de criação de aplicações utilizando essas duas ferramentas combinadas se solidificou. O objetivo deste trabalho acadêmico foi desenvolver uma aplicação que permita ao usuário atualizar um site *web* com fotos através de um dispositivo móvel. O Sistema proposto é dividido em dois módulos: módulo cliente ou aplicação móvel e módulo servidor. Enquanto o primeiro é responsável por capturar as imagens e enviá-las digitalizadas para um serviço *web*, o segundo é responsável por receber as imagens e publicá-las em um site *web*. Para modelar o Sistema utilizamos a linguagem de *UML* (*Unified Modeling Language*). Durante a fase de Análise e Projeto, alguns Padrões de Projeto foram utilizados no modelo de software. A aplicação foi desenvolvida na plataforma Java. O módulo cliente foi implementado utilizando a *API JME* (*Java Micro Edition*) enquanto o módulo servidor utilizou a *API JEE* (*Java Enterprise Edition*). O módulo servidor utiliza ainda a *API JAlbum* que permite a criação de álbuns de foto no formato *HTML* (*Hipertext Markup Language*).

Palavras-chave: Java, J2ME, J2EE, Internet, HTTP, UML, Padrões de projeto, Dispositivos móveis, celulares, JAlbum, JDom, álbum virtual, comunicação.

ABSTRACT

Two grate progresses of the men were: the invention of the computer and the creation of the movable telephony. Those discovered created a new age in the communication. With appearance of Internet, the possibilities of creation of applications using those two combined tools is very strong. The objective of this academic work was to develop an application that allows to the user to update a web site with pictures through from a movable device. The proposed system is divided in two modules: module client or movable application and module server. While the first is responsible to capture the images and to send them for a service web, the second is responsible for receive the images and to publish them in a web site. To model the System we used the language Unified Modeling Language (UML). During the phase of Analysis and Project, some Patterns of Project were used in the software model. The application was developed in the Java platform. The module client was implemented using Java Micro Edition (JME) API, while the module server was implemented with Java Enterprise Edition (JEE) API. The module server still uses the JAlbum API that allows the creation of picture albums in the format of Hypertext Markup Language (HTML).

Keywords: Java, J2ME, J2EE, Internet, HTTP, UML, Design Patterns, Movable devices, cellular, JAlbum, JDom, virtual album, communication.

SUMÁRIO

INTRODUÇÃO	10
2 REFERENCIAL TEÓRICO	12
2.1 JAVA	12
2.1.1 Definição	12
2.1.2 Plataformas Java	12
2.1.2.1 JSE – Java Standard Edition	13
2.1.2.2 JEE – Java Enterprise Edition	13
2.1.2.3 JME – Java Micro Edition	13
2.1.2.3.1 Configurações	14
2.1.2.3.1.1 CDC - Connected Device Configuration	14
2.1.2.3.1.2 CLDC - Connected, Limited Device Configuration	15
2.2 JALBUM	16
2.2.1 Definição	16
2.2.2 Licença e código de fonte	16
2.2.3 JAlbum embutido	16
2.3 PADRÕES DE PROJETO	17
2.3.1 Os padrões	17
2.4 UML - UNIFIED MODELING LANGUAGE	18
2.4.1 Descrição	18
2.4.2 Necessidade	18
2.4.3 Diagramas	19
2.5 HTTP (HIPERTEXT TRANFER PROTOCOL)	20
2.5.1 Protocolo HTTP	20
2.5.2 Funcionamento	20
2.6 JDOM	21
3 DESCRIÇÃO DA APLICAÇÃO	22
3.1 CAPTURAR IMAGEM	22
3.2 LISTAR IMAGENS	23
3.3 EXCLUIR IMAGEM	24
3.4 VISUALIZAR IMAGEM	24
3.5 ENVIAR IMAGEM	25
3.6 VISUALIZAR ÁLBUM	26
4 SOLUÇÃO	28
4.1 ANÁLISE	28
4.1.1 Diagramas	28
4.1.1.1 Diagrama de Casos de Uso	28
4.1.1.2 Diagrama de Classes	29
4.1.1.3 Diagrama de Sequência	30
5 MODELO DA SOLUÇÃO	33
5.1 MÓDULO CLIENTE	33
5.1.1 fCapImagem	34
5.1.2 fEnvImagem	34
5.1.3 fSelImagem	34
5.1.4 lRepImagem	35
5.2 PRINCIPAIS CLASSES E MÉTODOS	35
5.2.1 Pacote <i>mo.app.fone</i>	35
5.2.1.1 AdmFileDir	36

5.2.1.1.1	<i>readFromCameraDir()</i>	36
5.2.1.1.2	<i>writeToCameraDir()</i>	36
5.2.1.1.3	<i>deleteFromCameraDir()</i>	37
5.2.1.2	ClientConvertFile	37
5.2.1.2.1	<i>imageToByte()</i>	38
5.2.1.3	ClientDataSinc	38
5.2.1.3.1	<i>sendImage()</i>	38
5.2.1.4	ClientNameFile	39
5.2.1.4.1	<i>picName()</i>	39
5.2.1.5	ConectFileDir	40
5.2.1.5.1	<i>open()</i>	40
5.2.1.5.2	<i>close()</i>	40
5.2.1.6	ListManager	40
5.2.1.6.1	<i>viewGallery()</i>	41
5.2.1.7	VideoControler	41
5.2.1.7.1	<i>doSnapshot()</i>	41
5.2.1.8	Uso de Threads	42
5.3	MÓDULO SERVIDOR	42
5.3.1	Pacote server	43
5.3.2	Pacote com.app.conf	43
5.3.2.1	ImageManager	43
5.3.2.1.1	<i>byteToImage()</i>	43
5.3.2.2	ServerNameFile	44
5.3.2.2.1	<i>picName()</i>	44
5.3.2.3	ServerDataSinc	45
5.3.2.3.1	<i>SincronizeImg()</i>	45
5.3.2.3.2	<i>sincronizeComment()</i>	46
5.3.3	Pacote com.app.conect	46
5.3.3.1	AlbumManager	48
5.3.3.2	AlbumFactory	50
5.3.3.3	IAlbumManager	51
5.3.3.4	Factory	51
5.3.3.5	BlogManager	51
6	CONCLUSÃO	53
	REFERÊNCIAS	54

ÍNDICE DE FIGURAS

Figura 1 - Compilação Java.....	12
Figura 2 - Plataformas Java: JEE, JSE e JME.....	13
Figura 3 - Diagramas da UML.....	19
Figura 4 - Tela de captura de imagens.....	22
Figura 5 - Tela de listagem de imagens.....	23
Figura 6 - Exclusão de fotos.....	24
Figura 7 - Tela de visualização de imagem.....	24
Figura 8 - Tela de envio de imagem.....	25
Figura 9 - Tela de visualização de álbum.....	26
Figura 10 - Tela de ampliação de imagem.....	27
Figura 11 - Diagrama de Casos de Uso do sistema.....	28
Figura 12 - Diagrama de Classes do sistema.....	29
Figura 13 - Diagrama de Seqüência do Sistema.....	31
Figura 14 - Classes do Módulo Cliente.....	33
Figura 15 - Arquitetura do <i>MIDlet</i> "Fone".....	34
Figura 16 - Método "readFromCameraDir".....	36
Figura 17 - Método "writeToCameraDir".....	37
Figura 18 - Método "deleteFromCameraDir".....	37
Figura 19 - Método "imageToByte".....	38
Figura 20 - Método "sendImage".....	39
Figura 21 - Método "picName".....	39
Figura 22 - Método "open".....	40
Figura 23 - Método "close".....	40
Figura 24 - Método "viewGallery".....	41
Figura 25 - Método "doSnapshot".....	42
Figura 26 - Classes do Módulo Servidor.....	42
Figura 27 - Método "byteToImage".....	44
Figura 28 - Método "picName".....	45
Figura 29 - Método "SincronizeImg".....	45
Figura 30 - Método "sincronizeComment".....	46
Figura 31 - Arquivo de configuração "AlbumConfig.xml".....	47
Figura 32 - Arquitetura básica do Factory Method.....	47
Figura 33 - Implementação do <i>Factory Method</i> no Sistema.....	48
Figura 34 - Classe "AlbumManager".....	49
Figura 35 - Classe "AlbumFactory".....	50
Figura 36 - Classe "IAlbumManager".....	51
Figura 37 - Classe "Factory".....	51
Figura 38 - Classe "BlogManager".....	52
Figura 39 - Exemplo do uso do Factory Method.....	52

INTRODUÇÃO

A necessidade de comunicação é algo inerente ao ser humano e se renova a cada dia. Mesmo nos tempos mais remotos, o homem já se comunicava, seja através de desenhos, gritos ou linguagem corporal. Havia um entendimento entre as partes envolvidas. E com a evolução, com a invenção da escrita e do papel, a informação tornou-se mais acessível para a população.

A informação ganhou mais importância com o passar dos tempos. Nas guerras por exemplo, o sigilo das informações é vital para o triunfo de uma batalha e técnicas de criptografia são usadas para esconder do inimigo o conteúdo de determinadas mensagens. Outro grande passo na evolução tecnológica da informação - depois da invenção da imprensa por Gutenberg - foi o surgimento do computador e das tecnologias sem fio.

O computador permitiu que as informações fossem geradas, processadas e manipuladas numa velocidade e quantidade muito maiores. Não foi apenas o surgimento do computador que nos fez mudar a forma como trabalhamos com a informação. A evolução vista desde o surgimento do primeiro computador é enorme. As máquinas atuais são capazes de gerar, processar, armazenar e transmitir informações em escala muito maior do que os primeiros computadores. O custo e o tamanho dessas máquinas reduziram em proporção diretamente proporcional ao aumento de capacidade computacional. O computador se tornou uma máquina acessível e extremamente popular.

Seguindo no mesmo sentido, as telecomunicações também evoluíram bastante desde o primeiro telefone inventado por Graham Bell. Até pouco tempo atrás, a manipulação de linhas telefônicas era feita a partir de Centrais Telefônicas operadas manualmente por "telefonistas". Surgiram posteriormente telefones que usavam a transmissão de dados através de ondas de rádio. Um passo significativo foi a invenção da telefonia móvel. Desde o seu surgimento há cerca de 2 décadas, essa tecnologia agrega consideráveis avanços. Tanto no que se refere à qualidade dos aparelhos quanto à qualidade de transmissão de voz e dados. Não diferente de outras tecnologias, a telefonia móvel surgiu bastante elitista. Tanto os aparelhos quanto os serviços eram extremamente caros. Atualmente, no entanto, a tecnologia móvel vem se tornando popular. Segundo a Agência Nacional de Telecomunicações (ANATEL) o número de clientes ativos de aparelhos celulares no Brasil cresceu 31% durante o ano de 2006 e a atual base de usuários de celulares no país é de mais de 90 milhões de aparelhos.

Os aparelhos também estão ficando mais sofisticados. É comum encontrarmos aparelhos com recursos como câmera, acesso a Internet, gravador de voz, rádio FM, entre outros.

O auge da evolução da informação ocorreu na década de 80 com o surgimento da Internet. A informação passa a sofrer uma transição, do uso do papel para a manipulação em meio digital. O declínio dos custos de hardware e software e o crescimento extraordinário do acesso comercial aceleraram esta transição. Com computadores mais baratos e com acesso a Internet a informação se dissemina de forma mais fácil. É o meio de comunicação mais abrangente da atualidade, pois, compreende muitas áreas e muitos tipos de mídia e principalmente, é de fácil acesso. Muitas pessoas podem obter e distribuir informações livremente.

A criação de aparelhos de telefonia celular cada vez mais funcionais com o uso de câmeras fotográficas nos permite, por exemplo, compartilhar informações com outras pessoas de forma simples e prática. Seja para fins comerciais ou para lazer. O problema é como compartilhar uma fotografia obtida pela câmera de um aparelho móvel com outras pessoas. Considerando as condições tecnológicas atuais.

A solução proposta neste trabalho consiste em uma aplicação que permite, através de um aparelho celular, mais especificamente da câmera do mesmo, obter a fotografia e enviá-la juntamente com uma breve descrição da mesma para um serviço *Web*. Esse será responsável por publicar o conteúdo recebido. Essa aplicação utiliza recursos das tecnologias móveis de telecomunicações e também usa os recursos computacionais junto com a Internet para publicar e disponibilizar as informações para outros usuários. O Sistema foi proposto para atingir o maior número de pessoas, desde usuários comuns que desejam compartilhar fotos de uma viagem de férias até grandes empresas de notícias que querem dispor suas informações de modo instantâneo e prático pela Internet.

2 REFERENCIAL TEÓRICO

2.1 JAVA

2.1.1 Definição

Java é uma linguagem de programação orientada a objeto desenvolvida na metade da década de 90 pela *Sun Microsystems* com o objetivo de ser uma linguagem simples e eficiente. Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação (JAVA..., 2007).

Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um "bytecode" que é executado por uma máquina virtual.



Figura 1 - Compilação Java
Fonte: O próprio autor (2007)

A linguagem Java pode ser executada em diversos sistemas operacionais usando para isso uma máquina virtual específica. A Máquina virtual Java (Java Virtual Machine - JVM) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina.

A JVM é responsável pelo gerenciamento dos aplicativos, à medida que são executados. Graças à máquina virtual Java, os programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma versão da JVM, tornando assim essas aplicações independentes da plataforma onde funcionam.

2.1.2 Plataformas Java

Atualmente a plataforma Java está distribuída através das distribuições: JEE, JSE e JME. Pode-se ver Figura 2 a organização dessas três plataformas:

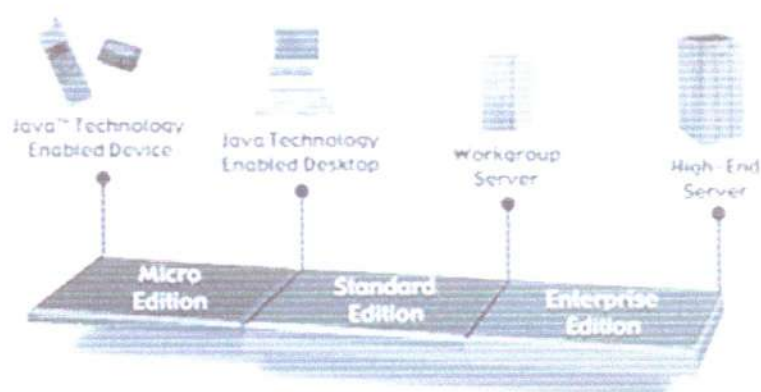


Figura 2 - Plataformas Java: JEE, JSE e JME
Fonte: (PLATAFORMAS ..., 2007)

2.1.2.1 JSE – Java Standard Edition

Projetada para a execução de programas em máquinas simples pessoais e estações de trabalho de acesso individual, ou seja, está direcionada ao mercado de computadores ao qual estamos familiarizados.

2.1.2.2 JEE – Java Enterprise Edition

“É uma plataforma para desenvolvimento de aplicativos empresariais distribuídos.” (ALUR, 2002, p.4).

Adequada a aplicativos baseados em servidores, normalmente utilizada em ambientes corporativos escaláveis, com o objetivo de servir a clientes, fornecedores e funcionários. A plataforma J2EE também inclui recursos para *Web* tais como Servlets, JSP e XML, e recursos para programação “componentizada” (EJB) (MATTOS, 2005).

2.1.2.3 JME – Java Micro Edition

Java Micro Edition inclui uma interface de usuário flexível, segurança robusta, rede embutida protocolo, e suporte a aplicações que usam Internet e que trabalham offline que podem ser baixadas dinamicamente. Aplicações baseadas em JME são portéis e podem ser usadas em muitos dispositivos, considerando as capacidades nativas de cada dispositivo (JMF ..., 2007).

Projetada para dispositivos com recursos limitados de memória, vídeo e processamento. A plataforma JME também oferece um certo grau de portabilidade e permite que uma mesma aplicação funcione em vários dispositivos relativamente diferentes.

Essa portabilidade é alcançada através da utilização da idéia de máquina virtual e de API padronizadas pelos principais fabricantes de dispositivos móveis do mundo. Neste caso, cada dispositivo executa internamente uma máquina virtual Java desenvolvida por seu fabricante, e os aplicativos em Java interagem apenas com a máquina virtual e não com o dispositivo real. Isso garante que programadores Java não precisem conhecer nenhuma API e biblioteca específica de cada dispositivo.

Apesar de os dispositivos de comunicação como celulares, *paggers* e outros dispositivos conectados – como PDAs e microcomputadores – possuírem várias características em comum, eles diferem bastante em relação à forma, às funções e aos componentes internos. Isso exige que eles sejam categorizados e agrupados de maneira que a compatibilidade seja garantida apenas entre os equipamentos semelhantes.

A solução adotada para a plataforma Java foi categorizá-la em duas camadas, chamadas “configuração” e “perfil” (MATTOS, 2005).

2.1.2.3.1 Configurações

A configuração especifica a máquina virtual JVM e alguns conjuntos de APIs básicas que deverão ser utilizados para uma família particular de dispositivos. Atualmente, existem basicamente dois tipos de configurações: *Connected Device Configuration* – CDC (“Configuração para dispositivos conectados”) e *Connected, Limited Device Configuration* – CLDC (“Configuração para dispositivos conectados limitados”).

2.1.2.3.1.1 CDC - Connected Device Configuration

O CDC é destinado a dispositivos com recursos mais avançados e processamento e armazenamento, podendo incluir alguns PDAs. De forma geral, o CDC especifica que uma JVM completa deve ser suportada, assim como definida na especificação da plataforma Java (MATTOS, 2005).

2.1.2.3.1.2 CLDC - *Connected, Limited Device Configuration*

A CLDC está voltada para dispositivos menores que os CDC, cuja limitação se refere aos recursos de memória, processamento e, principalmente, da conectividade com redes e Internet. A CLDC foi criada considerando dispositivos com 128 KB a 512 KB de memória disponível para a plataforma Java.

A conexão limitada refere-se a uma conexão de rede lenta e intermitente. Na maioria dos telefones celulares, por exemplo, a velocidade alcançada é de aproximadamente 9.6 Kbps. Lembrando que estamos tratando de dispositivos com tela reduzida, memória limitada e conexão lenta, aplicações desenvolvidas para CLDC devem ser cautelosas no uso da rede.

A CLDC é baseada em uma JVM reduzida, conhecida como KVM, assim designada pelo fato de seu tamanho ser medido em *kilobytes* em vez de *megabytes*. Enquanto a CLDC é um documento de especificação, a KVM refere-se a uma parte de um software específico (MATTOS, 2005).

Perfil

O perfil está na camada superior a de uma configuração, adicionando APIs e especificações necessárias para o desenvolvimento de aplicações para uma determinada família de dispositivos.

MIDP

MIDP é o mais conhecido perfil do JME baseado na CLDC e KVM, e é o perfil mais desenvolvido e amplamente adotado, principalmente em PDAs, telefones celulares e em outros dispositivos móveis. O MIDP 1.0 foi introduzido pela JSR 37, e ainda está em uso. Já a versão 2.0 foi definida pela JSR 118, aprimorou as capacidades deste produto.

A versão mais atual trouxe alguns aprimoramento: novas APIs de rede suportando TCP *sockets*, UDP datagrama, serialização, inicialização de envio (*push-initialed*) e conexões seguras. Uma API de segurança com mais funcionalidades e policiamento, e APIs para som e eventos de jogos estão sendo adicionada também.

As aplicações MIDP são chamadas de MIDlets. Esses podem usar facilidades de MIDP e as APIs que MIDP herda de CLDC. Junto com o equipamento, os fabricantes fornecem mais dois componentes: classes e aplicações OEM (*Original Equipment*

Manufacturer). As classes OEM são usadas pelo MIDP para acessar funções específicas do aparelho como por exemplo o envio e recebimento de mensagens SMS e acesso a dados persistentes. As aplicações OEM são programas fornecidos pelo fabricante, tal como uma agenda de endereços. As aplicações OEM podem ser acessadas pelo MIDP (MIDP, 2007).

2.2 JALBUM

2.2.1 Definição

JAlbum é uma aplicação para construir álbuns de imagens personalizados para *web*. Suporta o recurso "Arrastar e Largar" de arquivos e pastas, permite seleção de tamanho de imagens, legendas, títulos e seleção de *Skins*. Uma vez criado o álbum, o programa permite publicá-lo diretamente no servidor JAlbum via FTP.

Hoje, JAlbum é um dos mais populares clientes para álbuns de fotografia da *Web*. A variedade de *skins* de álbum e a possibilidade para personalizar a apresentação faz o JAlbum extremamente flexível (DEVELOPER..., 2007).

2.2.2 Licença e código de fonte

As partes genéricas do JAlbum (tudo menos o pacote *se.datadosen.jalbum*) está disponível como fonte aberta sobre da licença LGPL. A licença LGPL permite aos desenvolvedores utilizarem livremente componentes de fonte abertos em projetos não-comerciais e comerciais sem ser forçado a abrir o código fonte deles ou pagamento de *royalties* aos desenvolvedores iniciais. E é principalmente por isso que o JAlbum foi escrito sob licença aberta, para o seu melhoramento não só por parte de seus criadores, mas também de colaboradores aceitos (DEVELOPER..., 2007).

2.2.3 JAlbum embutido

JAlbum é basicamente criado como uma classe de GUI em cima de um *JavaBean* - *AlbumBean*. Essa classe é responsável pela geração do álbum físico e pode ser usada por outras aplicações de Java para a geração de um álbum. Pode-se, por exemplo, utilizar essa classe dentro de um *Servlet*. Intencionalmente, o *JAlbum* foi desenvolvido pra essa finalidade.

2.3 PADRÕES DE PROJETO

2.3.1 Os padrões

Padrões para arquitetura de software são soluções de eficiência já comprovadas e amplamente utilizadas para a resolução de problemas comuns em projeto de software. Estas soluções são desenvolvidas e conhecidas por especialistas e tornam-se padrões por serem reutilizadas em vários projetos e por terem eficácia comprovada.

Alexander (apud GAMMA, 2000, p.19) afirma:

Cada padrão descreve um problema no nosso ambiente e o núcleo de sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira" [AIS+77, pág. x]. Muito embora Alexander estivesse falando acerca de padrões, em construções e cidades, o que ele diz é verdadeiro em relação aos padrões de projeto orientados a objetos. Nossas soluções são expressas em termos de objetos e interfaces em vez de paredes e portas, mas ambos os tipos de padrões representam uma solução para um problema num contexto.

"Os padrões se referem à comunicação de problemas e soluções." (ALUR, 2002, p.5).

Padrões de projeto podem ser vistos como uma solução que já foi testada para um problema. Desta forma, um padrão de projeto geralmente descreve uma solução ou uma instância da solução que foi utilizada para resolver um problema específico. Padrões de projetos são soluções para problemas que alguém um dia teve e resolveu aplicando um modelo que foi documentado e que você pode adaptar integralmente ou de acordo com necessidade de sua solução.

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e as instancias participantes, seus papéis, colaborações e a distribuição de responsabilidades[...] (GAMMA, 2000, p.20).

Partindo do trabalho de Alexander, profissionais de software começaram a incorporar esses princípios na criação das primeiras documentações de padrões de projetos como um guia para desenvolvedores iniciantes. O resultado prático foi a publicação –*Design Patterns: Elements of Reusable Object-Oriented Software*– em 1995 por Eric Gamma, Richard Helm, Ralph Johnson e John Vlissides - (*gang of four*). Este livro é a referência principal no assunto para a comunidade de software. Nele são descritos 23 padrões de projeto que foram baseados na experiência e observação dos autores.

2.4 UML - UNIFIED MODELING LANGUAGE

2.4.1 Descrição

A *Unified Modeling Language* (UML). é uma linguagem para a elaboração da estrutura de projetos de software orientado a objeto. A UML poderá ser empregada para visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software (BOOCH, 2000).

Deve ficar bem claro, no entanto, que UML não é uma linguagem de programação e sim é uma linguagem de modelagem, cujo objetivo é auxiliar os engenheiros de software a definir as características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. Todas essas características são definidas por meio da UML antes do software começar a ser realmente desenvolvido (GUEDES, 2006, p17).

Essa linguagem sintetiza os principais métodos existentes, sendo considerada uma das mais expressivas para modelagem de sistemas orientados a objetos. Por meio de seus diagramas é possível representar sistemas de softwares sob diversas perspectivas de visualização. Facilita a comunicação de todas as pessoas envolvidas no processo de desenvolvimento de um sistema - gerentes, coordenadores, analistas, designers e desenvolvedores - por apresentar um vocabulário de fácil entendimento (UML, 2007).

2.4.2 Necessidade

Muitos profissionais afirmam que conseguem determinar todas as necessidades de um sistema de informação de cabeça e que sempre trabalharam assim. Qual a real necessidade de se projetar uma casa? Um pedreiro experiente não é capaz de construí-la sem um projeto? Isso pode ser verdade, mas a questão é bem mais ampla, envolvendo fatores extremamente complexos, como levantamento e análise de requisitos, prototipação, tamanho do projeto, complexidade, prazos, custos, documentação, manutenção e reusabilidade, entre outros (GUEDES, 2006).

Além de documentar as necessidades do sistema, evitando que no futuro apareçam surpresas desagradáveis, como acontece no desenvolvimento de sistema sem o devido projeto, a UML provê a partir de diversos diagramas uma forma de documentação, não servindo

apenas para isso, a documentação é apenas uma das vantagens do uso de uma linguagem de modelagem como a UML.

2.4.3 Diagramas

Um diagrama é uma representação gráfica de um conjunto de elementos, geralmente representadas como gráficos de vértices (itens) e arcos (relacionamentos). São desenhados para permitir a visualização de um sistema sob diferentes perspectivas. Nesse sentido, um diagrama constitui uma projeção de um determinado sistema (BOOCH, 2000).

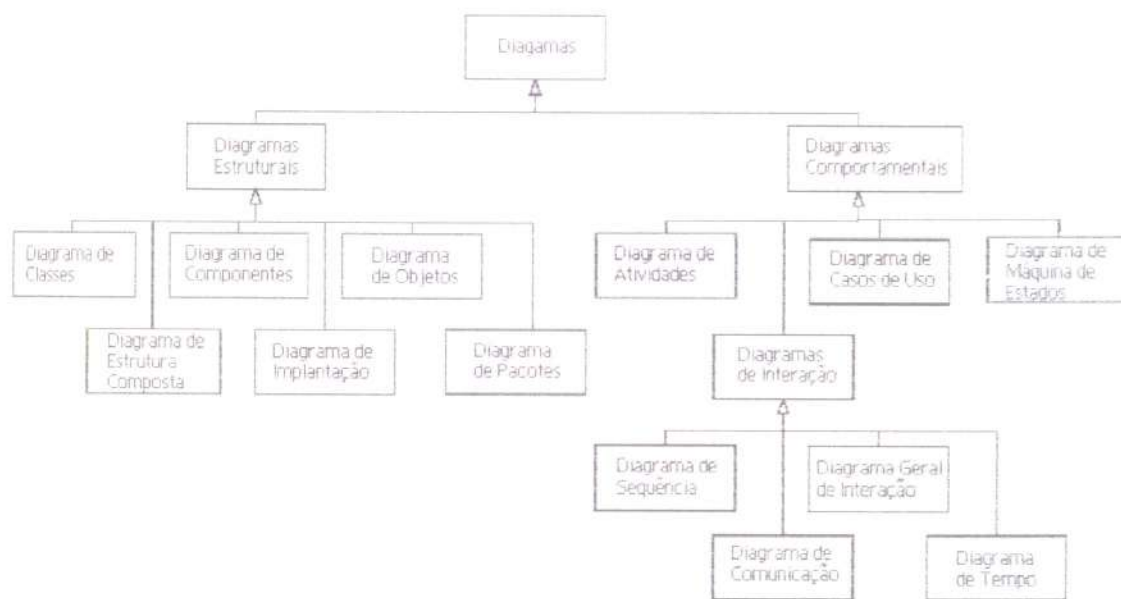


Figura 3 - Diagramas da UML 2.0

Fonte: GUEDES (2006, p.35)

Cada diagrama da UML analisa os sistema, ou parte dele, sob uma determinada ótica, é como se o sistema fosse modelado em camadas, sendo que alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão mais externa, como é o objetivo do Diagrama de Casos de Uso, enquanto outros oferecem uma visão mais profunda do software, apresentando um enfoque mais técnico ou ainda visualizando apenas uma característica específica do sistema ou um determinado processo. A utilização de diversos diagramas permite que falhas sejam descobertas, diminuindo a necessidade de ocorrências de erros futuros (GUEDES, 2006, p.25).

2.5 HTTP (HIPERTEXT TRANSFER PROTOCOL)

2.5.1 Protocolo HTTP

O HTTP (*HyperText Transfer Protocol*) é um protocolo de rede responsável pela transferência de dados e pela comunicação entre cliente e servidor na *World Wide Web* (WWW). Esse protocolo surgiu da necessidade de distribuir informações pela Internet. Para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da *Web* e entendida por todos os computadores ligados à Internet. Com isso, o HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas. Este protocolo tem sido usado pela WWW desde 1990 (PROTOCOLO..., 2007).

A transferência de documentos hipermídia é feita através do protocolo HTTP. Um servidor *web* hospeda o site, enquanto um cliente (um navegador Internet) faz a requisição dos documentos lá contidos. Essa transferência é feita usando, normalmente, a porta 80 do protocolo TCP. Esses documentos são escritos em diversas linguagens. A mais simples de todas é o texto puro, passando pelo conhecido HTML (*HyperText Markup Language*) e chegando a outras linguagens mais recentes e suportadas somente pelos navegadores mais atuais (TORRES, 2001).

2.5.2 Funcionamento

Um sistema de comunicação em rede possui diversos protocolos que trabalham em conjunto para o fornecimento de serviços. Para que o protocolo HTTP consiga transferir seus dados pela *Web*, é necessário que os protocolos TCP e IP (*Internet Protocol* e *Protocolo de Internet*) tornem possível a conexão entre clientes e servidores através de *sockets* TCP/IP.

De acordo com Fielding et al (1999, p. 10 apud HTTP..., 2007)

O HTTP utiliza o modelo cliente-servidor, como a maioria dos protocolos de rede, baseando-se no paradigma de requisição e resposta. Um programa requisitante (cliente) estabelece uma conexão com um outro programa receptor (servidor) e envia-lhe uma requisição, contendo a URI, a versão do protocolo, uma mensagem MIME (padrão utilizado para codificar dados em formato de textos ASCII para serem transmitidos pela Internet) contendo os modificadores da requisição, informações sobre o cliente e, possivelmente, o conteúdo no corpo da mensagem.

O servidor responde com uma linha de *status* (*status line*) incluindo sua versão de protocolo e um código de operação bem sucedida ou um código de erro, seguido pelas informações do servidor, meta-informações da entidade e possível conteúdo no corpo da mensagem. Após o envio da resposta pelo servidor, encerra-se a conexão estabelecida (PROTOCOLO..., 2007).

2.6 JDOM

JDOM (*Java Document Object Model*) é uma API de código fonte aberto especializada no processamento de arquivos XML (*Extensible Markup Language*) que especificamente foi projetado para a plataforma de Java de forma que isto pode tirar proveito de suas características de linguagem. JDOM integra o DOM (*Document Object Model*) e SAX (*Simple API for XML*). É uma simplificação da especificação da W3C Document Object Model.

3 DESCRIÇÃO DA APLICAÇÃO

A aplicação apresentada neste texto consiste em uma ferramenta que permite ao usuário compartilhar fotos obtidas a partir da câmera digital de um celular através de um serviço *web* acessível pelo próprio aparelho. A aplicação oferecerá minimamente as seguintes funcionalidades:

- Capturar Imagem
- Listar Imagens
- Ver Imagem
- Excluir Imagem
- Enviar Imagem
- Visualizar álbum

3.1 CAPTURAR IMAGEM

Esta funcionalidade faz parte do módulo cliente da aplicação e fará o controle da câmera do dispositivo. A partir dessa funcionalidade o usuário poderá escolher entre três opções: *Capturar*, *Sair* ou *Ver fotos*.



Figura 4 - Tela de captura de imagens
Fonte: O próprio autor (2007)

O comando *Capturar* irá obter uma foto a partir da câmera do dispositivo. A imagem capturada é gravada na memória dispositivo, em seguida, exibida em outra tela – *Tela de Visualização da Imagem*. Nessa segunda tela, o Usuário poderá escolher a opção *Enviar Imagem*.

Através do comando *Sair*, o Usuário poderá abandonar a Aplicação. Já o comando *Ver fotos* possibilita ao Usuário visualizar todas as imagens que estão salvas no diretório padrão do dispositivo.

3.2 LISTAR IMAGENS

Esta funcionalidade disponibiliza para o Usuário uma lista de todas as imagens que estão gravadas na memória do dispositivo e é acessada a partir do comando *Ver fotos* da tela de captura de imagens. O Usuário poderá escolher uma foto para ser enviada para o serviço de publicação de imagens ou excluída.

Na tela de *Lista de Imagens*, as imagens são dispostas em uma lista ordenada alfabeticamente. Os nomes serão criados automaticamente pela aplicação no momento da captura da imagem e seguirá o seguinte padrão: “prefixo predefinido + composição de data atual”. Como exemplo temos: a imagem P30-10-2007_235753.jpg.



Figura 5 - Tela de listagem de imagens
Fonte: O próprio autor (2007)

Nesse caso o prefixo é: “P” e a composição de data é a data atual, começando pelo dia e sequencialmente vai até milisegundos. Essa funcionalidade é importante, pois permite que as imagens sejam organizadas em função das suas datas de captura.

Ainda na tela de *Lista de Imagens* o Usuário terá as seguintes opções: *Ver*, *Excluir*, *Voltar* e *Sair*. O comando *Ver* dará uma visualização da imagem selecionada na lista de imagens. A partir do comando *Excluir*, o Usuário poderá excluir permanente uma imagem selecionada.

3.3 EXCLUIR IMAGEM



Figura 6 - Exclusão de fotos
Fonte: O próprio autor (2007)

Na tela de *Lista de Imagens*, as imagens são descritas em uma lista ordenada alfabeticamente. A partir dessa lista o Usuário poderá selecionar uma imagem e escolher a opção *Excluir*. O Sistema exibe uma tela de confirmação para que o Usuário se assegure de que deseja realmente executar tal funcionalidade.

Se o Usuário confirmar o comando, a Aplicação irá apagar a imagem definitivamente do dispositivo. A imagem é apagada da memória do dispositivo. Essa funcionalidade permite que o Usuário descarte imagens e libere espaço na memória do aparelho.

3.4 VISUALIZAR IMAGEM

Essa funcionalidade pode ser acessada a partir do comando *Visualizar Imagem* da tela de *Lista de Imagens*. Quando esse comando é acionado, a imagem selecionada na lista é exibida em uma outra tela – *Visualização de Imagem*. Essa funcionalidade é importante pois permite ao Usuário visualizar uma imagem antes de enviá-la para publicação no site ou excluí-la. Isso garante que o Usuário tenha certeza de qual imagem ele está manipulando visto que apenas o nome da imagem não é suficiente para identificá-la com segurança.



Figura 7 - Tela de visualização de imagem
Fonte: O próprio autor (2007)

Nessa tela, o Usuário poderá escolher as opções *Voltar*, *Sair* ou *Enviar* para a imagem que está sendo exibida. O comando *Voltar* levará o Usuário para a tela de *Lista de Imagens*. Já o comando *Sair* encerra a aplicação. O comando *Enviar* executa a funcionalidade *Enviar Imagem*.

3.5 ENVIAR IMAGEM

A funcionalidade *Enviar Imagem* é umas das principais funcionalidades do sistema. É através dela que o Usuário poderá enviar uma imagem e sua descrição para um servidor *web*. A implementação dessa funcionalidade está dividida nos dois módulos da aplicação, ou seja, uma parte da funcionalidade é implementada no módulo cliente e outra parte no módulo servidor.

Na aplicação no dispositivo móvel, a funcionalidade *Enviar Imagem* poderá ser acessada a partir da tela *Lista de Imagens* ou *Visualizar Imagem*. A tela *Enviar Imagem* exibe a imagem que será enviada bem como um campo de texto para que o Usuário possa digitar uma descrição para a imagem. Essa tela oferece ainda a opção *Enviar* para que a imagem e descrição sejam enviadas para o serviço *web*. A aplicação se encarrega de converter a imagem com a descrição e enviá-las para o serviço *web*. Antes do envio, a imagem e a sua descrição serão codificadas em *bytes* e enviadas para o módulo servidor da aplicação através da *Internet* – protocolo HTTP.



Figura 8 - Tela de envio de imagem
Fonte: O próprio autor (2007)

O módulo servidor da Aplicação tem como responsabilidade receber a imagem e a descrição codificadas, interpretar essas informações e disponibilizá-las num site *web*.

É possível também definir as configurações visuais do álbum digital - cores, estilos de borda, disposição de imagens e outros - através de estilos. A página principal exibe, além das fotos em miniatura, algumas informações do álbum como autor, título, data da última atualização, etc.



Figura 10 - Tela de ampliação de imagem
Fonte: O próprio autor (2007)

A página de detalhes será responsável pela visualização de uma determinada foto no modo ampliado. Essa tela terá ainda uma barra de controle que permitirá que o Usuário avance ou retroceda na visualização das outras fotos do álbum.

O comando *Página Principal* leva o Usuário de volta à página principal onde poderá escolher uma foto através das miniaturas exibidas. O comando *Próximo* exhibe a próxima imagem da fila no modo ampliado e o comando *Anterior* faz o inverso, mostra a foto anterior à foto corrente também no modo ampliado.

A imagem ampliada será exibida com resolução de 640x480 *pixels*. A descrição da imagem é exibida abaixo da imagem.

4 SOLUÇÃO

4.1 ANÁLISE

Uma das primeiras fases de Engenharia de um Software consiste no Levantamento de Requisitos (as outras etapas, sugeridas por muitos autores, são: Análise de Requisitos, Projeto, Codificação, Testes e Implantação). Nesta etapa, o Analista busca compreender as necessidades do Usuário e o que ele deseja que o Sistema realize [...] Durante a Análise de Requisitos, uma linguagem de modelagem auxilia a entender questões que não foram concebidas durante as entrevistas iniciais (GUEDES, 2005).

O primeiro passo na realização da Análise dessa aplicação foi identificar quais os Casos de Uso seriam oferecidos.

4.1.1 Diagramas

4.1.1.1 Diagrama de Casos de Uso

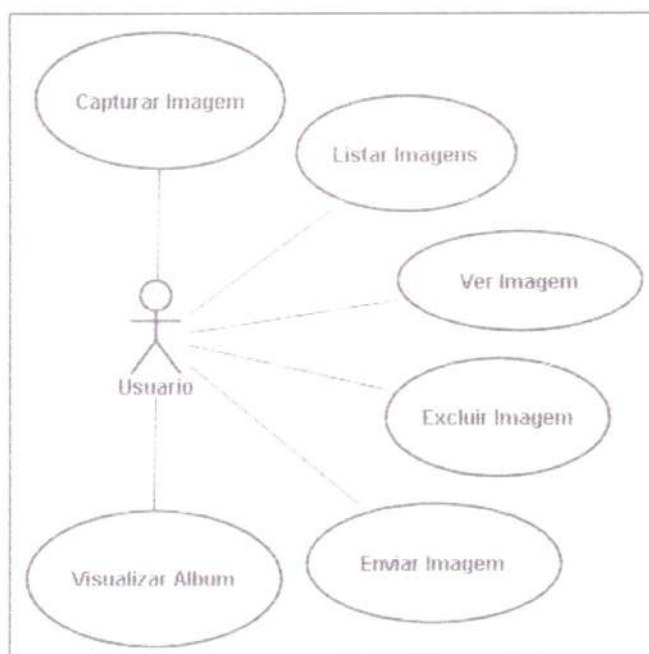


Figura 11 - Diagrama de Casos de Uso do sistema
Fonte: O próprio autor (2007)

Através do Diagrama de Casos de Uso é possível identificar as funcionalidades do Sistema e os seus principais Usuários.

A partir dos requisitos apresentados na seção anterior, foram identificados as funcionalidade apresentadas no Diagrama de Caso de Uso acima. O Sistema terá um Ator

Usuário que fará uso dos seguintes serviços: *Capturar Imagem*, *Listar Imagens*, *Ver Imagem*, *Excluir Imagem*, *Enviar Imagem* e *Visualizar Álbum*.

4.1.1.2 Diagrama de Classes

O principal enfoque desse diagrama é permitir a visualização das classes que irão compor o Sistema com seus atributos e métodos e demonstrar a relação entre as classes. É um dos diagramas mais importantes da UML.

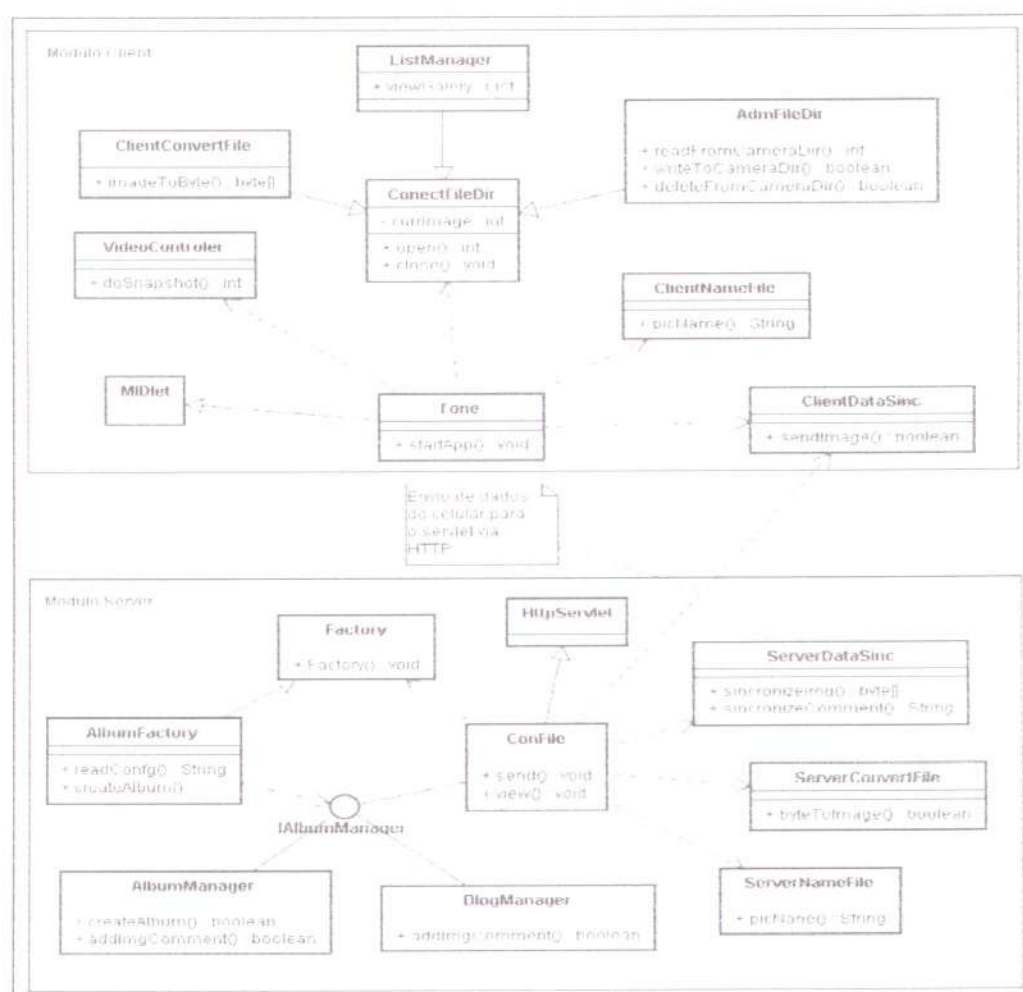


Figura 12 - Diagrama de Classes do sistema

Fonte: O próprio autor (2007)

O Sistema é composto por dois módulos: Cliente e Servidor. O primeiro irá funcionar em um dispositivo móvel e enquanto o segundo em um servidor *Web*.

No módulo Cliente a classe mais importante é *Tone*. Essa classe é uma extensão da classe *MIDlet* que utiliza todas as outras classes do pacote, inclusive a classe *ConectFileDir*. Essa última é superclasse de outras três classes.

No módulo Servidor a classe mais importante é *ConFile* que é uma extensão da classe *HttpServlet*. No servidor *Web* essa classe faz uso de todas as classes do módulo inclusive a interface *IAlbumManager*.

4.1.1.3 Diagrama de Seqüência

O Diagrama de Seqüência preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Em geral, pode-se usar o Diagrama de Caso de Uso e o Diagrama de Classes como artefatos de entrada para a elaboração dos Diagramas de Seqüência.

mensagens entre os objetos dos dois módulos do Sistema definem as responsabilidades de cada classe. As trocas de mensagens são iniciadas pelo ator *Usuário*.

5 MODELO DA SOLUÇÃO

Como já dito anteriormente, o Sistema de Foto Álbum tem o objetivo de, por meio de um dispositivo móvel, atualizar um site *web* previamente publicado com fotos e mensagens. A aplicação é composta de duas partes: o *Módulo Cliente*, que é o sistema que alimenta o servidor através de dados enviados a partir de um dispositivo móvel e o *Módulo Servidor* que funciona no servidor, responsável por receber os dados e publicá-los no site.

A manipulação da câmera fotográfica do dispositivo móvel foi implementada a partir do suporte oferecido pela configuração CLDC 1.1 e pelo *profile* MIDP 2.0. Durante o desenvolvimento da aplicação foi utilizado um emulador de aparelho celular. Para o funcionamento correto o dispositivo móvel deverá atender minimamente aos seguintes requisitos: suporte a JSR 75, JSR 135 e JSR 179 (FILECONNECTION..., 2007).

5.1 MÓDULO CLIENTE

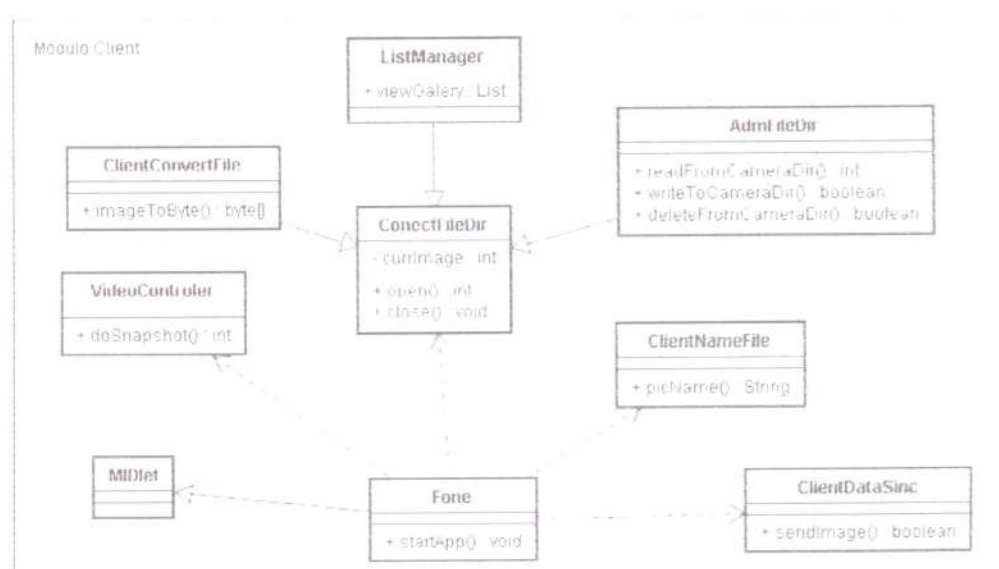


Figura 14 - Classes do Módulo Cliente
Fonte: O próprio autor (2007)

O módulo cliente do Sistema – Midlet *Fone* – é composto por 4 componentes visuais. São eles:

- *fCapImagem*
- *fEnvImagem*
- *fSellImagem*
- *fReplImagem*

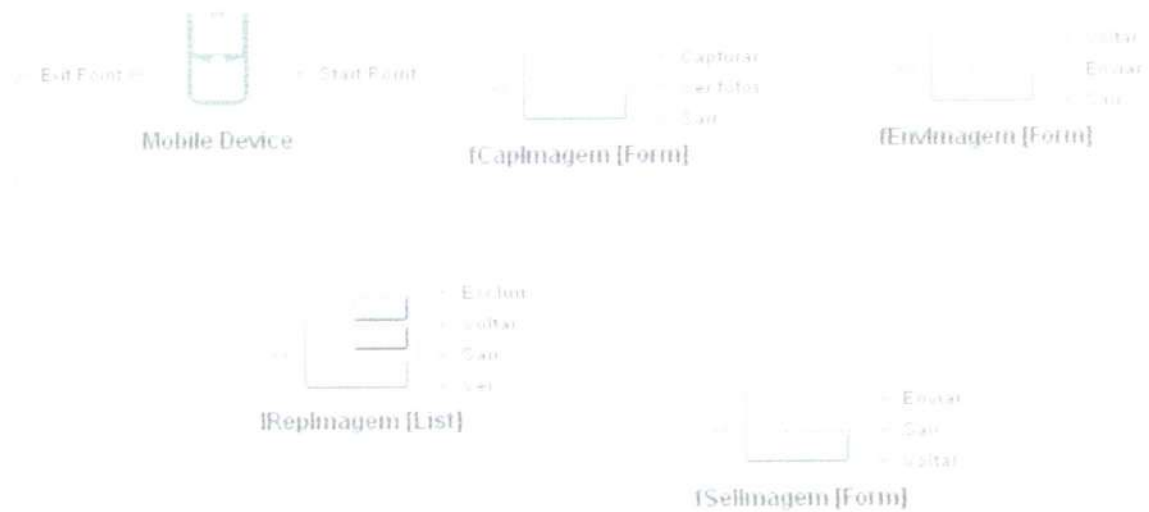


Figura 15 - Arquitetura do MIDlet "Fone"
Fonte: O próprio autor (2007)

5.1.1 fCapImagem

É um formulário que implementa o controle da câmera do dispositivo. Contém 3 comandos: *Capturar* que obtém uma imagem da câmera, salva o arquivo fisicamente na memória do celular e a envia para o segundo formulário. O comando *Ver Todos* que encaminha o Usuário para uma outra tela, a fim de escolher uma imagem que está no repositório padrão de imagens do celular, disposta na tela em forma de lista e o comando *Sair* que encaminha o Usuário para o início da Aplicação.

5.1.2 fEnvImagem

Formulário responsável por mostrar a imagem e disponibilizar um campo de texto para digitação da descrição da foto escolhida. O comando *Enviar* desse formulário dispara a sincronia com o servidor, enviando a imagem e a sua descrição. A sincronização é feita utilizando uma conexão HTTP. Já o comando *Voltar* envia a aplicação para o formulário de captura de imagem, onde poderá escolher outra imagem. O comando *Sair* reinicia a aplicação.

5.1.3 fSellImagem

No formulário de seleção de imagem o Usuário poderá visualizar a imagem previamente escolhida em uma lista de imagens obtidas do repositório padrão do aparelho. A

foto é identificada pelo seu nome exibida logo abaixo, possui um comando *Voltar* que quando acionado encaminha o *Usuário* para a lista de imagens e um comando *Sair* que leva o *Usuário* ao início da Aplicação.

5.1.4 IRepImagem

É uma lista, alimentada pelas fotos que estão no repositório padrão do aparelho, identificadas pelo seu nome. Além dos comandos *Voltar* que encaminha o *Usuário* para a tela de captura da imagem e do comando *Sair* que reinicia a aplicação, esse formulário possui dois outros comandos *Ver* e *Excluir*. O comando *Ver* encaminha o usuário para a tela de seleção de imagens afim de assegurar que aquela é mesmo a imagem escolhida e o comando *Excluir* exclui definitivamente a imagem do repositório.

5.2 PRINCIPAIS CLASSES E MÉTODOS

O *Módulo Cliente* está distribuído em dois pacotes de classes, são eles: *fone* onde fica o *MDlet* central e o *mo.app.fone* onde se localizam as classes utilitárias do Sistema. São elas:

- *AdmFileDir*
- *ClientConvertFile*
- *ClientDataSinc*
- *ClientNameFile*
- *ConectFileDir*
- *ListManager*
- *VideoControler*

5.2.1 Pacote *mo.app.fone*

Neste pacote estão as classes desenvolvidas para controlar a câmera digital, administrar o diretório de imagens, gerar nomes das imagens e sincronizar dados com o servidor *web*.

5.2.1.1 AdmFileDir

A classe *AdmFileDir* é responsável por administrar o repositório de fotos do dispositivo móvel. Possui três métodos. São eles:

5.2.1.1.1 readFromCameraDir()

Método responsável por ler uma imagem a partir do diretório de fotos do dispositivo móvel. Recebe como parâmetro um objeto *FileConnection*, de acordo com a JSR75 representa o endereço físico da imagem. O retorno do método é um objeto *ImageItem* contendo a imagem solicitada.

```
public static ImageItem readFromCameraDir(FileConnection currImage){
    byte [] b;
    ImageItem imi = null;
    try {
        if (!currImage.exists()) {
            System.out.println("Arquivo de Imagem não existe");
        }

        InputStream fis = currImage.openInputStream();
        int fileSize = (int)currImage.fileSize();
        b = new byte[fileSize];
        int length = fis.read(b, 0, fileSize);
        fis.close();
        if (b != null) {
            Image im = Image.createImage(b, 0, b.length);
            imi = new ImageItem("", im, Item.PLAIN, "");
        }
    } catch (Exception e) {
        System.out.println("Erro ocorrido: " + e.getMessage());
    }
    return imi;
}
```

Figura 16 - Método "readFromCameraDir"

Fonte: O próprio autor (2007)

5.2.1.1.2 writeToCameraDir()

Método responsável por salvar uma imagem no diretório de fotos do dispositivo móvel. Recebe como parâmetro um objeto *FileConnection*, de acordo com a JSR75 representa o endereço físico da imagem e um vetor do tipo *byte* que representa a imagem convertida em *bytes*. O retorno do método é um objeto *boolean*, representando *true* para sucesso e *false* para falha.

```
public static boolean writeToCameraDir(byte[] myImage, FileConnection fconn) {
    boolean result = false;
    try {
        if (!fconn.exists()) {
            fconn.create();
        }
        OutputStream out = fconn.openOutputStream();
        out.write(myImage);
        out.close();
        result = true;
    } catch (Exception e) {
        result = false;
        System.out.println("Erro ocorrido: " + e.getMessage());
    }
    return result;
}
```

Figura 17 - Método "writeToCameraDir"

Fonte: O próprio autor (2007)

5.2.1.1.3 deleteFromCameraDir()

Método responsável por excluir uma imagem a partir do diretório de fotos do dispositivo móvel. Recebe como parâmetro um objeto *FileConnection* que de acordo com a JSR75 representa o endereço físico da imagem. O retorno do método é um objeto *boolean*, representando *true* para sucesso e *false* para falha.

```
public static boolean deleteFromCameraDir(FileConnection fconn){
    boolean result = false;
    try {
        if (!fconn.exists()) {
            fconn.close();
        }else{
            fconn.delete();
            result = true;
        }
    } catch (Exception e) {
        result = false;
        System.out.println("Erro ocorrido: " + e.getMessage());
    }
    return result;
}
```

Figura 18 - Método "deleteFromCameraDir"

Fonte: O próprio autor (2007)

5.2.1.2 ClientConvertFile

A tarefa de converter uma imagem em vetor de *bytes* é feita por essa classe. Possui apenas um método:

5.2.1.2.1 *imageToByte()*

Método responsável por converter uma imagem do diretório de fotos em um vetor do tipo *byte*. Recebe como parâmetro um objeto *FileConnection*, de acordo com a JSR75 representa o endereço físico da imagem. O retorno do método é um vetor do tipo *byte* contendo a imagem solicitada convertida em *bytes*.

```
public static byte[] imageToByte(FileConnection currImage){
    byte [] b = null;
    ImageItem imi = null;
    try {
        if (!currImage.exists()) {
            System.out.println("Arquivo de Imagem não existe");
        }
        InputStream fis = currImage.openInputStream();
        int fileSize = (int)currImage.fileSize();
        b = new byte[fileSize];
        int length = fis.read(b, 0, fileSize);
        fis.close();
    } catch (Exception e){
        System.out.println("Erro ocorrido: " + e.getMessage());
    }
    return b;
}
```

Figura 19 - Método "imageToByte"
Fonte: O próprio autor (2007)

5.2.1.3 ClientDataSinc

Esta é uma das classes mais importantes do sistema, é ela que faz a sincronia de dados com o *Servlet* controlador do servidor *Web*, enviando a imagem convertida em *bytes* e sua descrição. Também possui apenas um método:

5.2.1.3.1 *sendImage()*

Método responsável por enviar, através de uma conexão *http*, uma imagem e sua descrição para um endereço *Web*. Recebe como parâmetro um vetor de *bytes* representando a imagem convertida, uma *String* com o comentário da imagem e uma *String* contendo o a URL do *Servlet* na Internet. O retorno do método é um valor *boolean*, representando *true* para sucesso e *false* para falha.

```

public boolean sendImage(byte[] imagem, String comment, String url){
    boolean result = false;
    try{
        HttpURLConnection con = null;
        con = (HttpURLConnection) Connector.open(url);
        OutputStream saida = con.getOutputStream();
        DataOutputStream dout = new DataOutputStream(saida);

        dout.writeUTF(comment);
        dout.writeInt(imagem.length);
        dout.write(imagem);

        dout.close();
        saida.close();
        con.close();
        result = true;
    } catch (Exception e) {
        System.out.println("Erro ocorrido: " + e.getMessage());
        result = false;
    }
    return result;
}

```

Figura 20 - Método "sendImage"

Fonte: O próprio autor (2007)

5.2.1.4 ClientNameFile

Essa classe administra a nomenclatura dos arquivos de foto que são salvos no repositório de fotos do aparelho através de um único método:

5.2.1.4.1 *picName()*

Método responsável por gerar chaves de nomes de acordo com as regras especificadas no levantamento de requisitos. O retorno do método é uma *String* contendo o nome da imagem.

```

public static String PicName(){
    long dtMili = System.currentTimeMillis();
    Date dt = new Date(dtMili);
    Calendar cal = Calendar.getInstance();
    cal.setTime(dt);
    int day = cal.get(Calendar.DAY_OF_MONTH);
    int month = cal.get(Calendar.MONTH);
    int year = cal.get(Calendar.YEAR);
    int hour = cal.get(Calendar.HOUR_OF_DAY);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);
    int milesimo = cal.get(Calendar.MILLISECOND);

    String file = "P"+day+"-"+month+"-
    "+year+"_"+hour+minute+second+milesimo+".jpg";
    return file;
}

```

Figura 21 - Método "picName"

Fonte: O próprio autor (2007)

5.2.1.5 ConectFileDir

Essa classe faz o controle da conexão de acesso a imagens do dispositivo. Já as classes *AdmFileDir*, *ClientConvertFile* e *ListManager* são especializações dessa classe.

A classe *ConectFileDir* possui o seguinte método:

5.2.1.5.1 open()

Método responsável por abrir uma conexão a um arquivo físico do dispositivo. Recebe como parâmetro uma *String* contendo o nome do arquivo e outra contendo o caminho do diretório de fotos do dispositivo. O retorno do método é um objeto *FileConnection* que representa a imagem solicitada.

```
Public FileConnection open(String imageName, String photosDir){
    try {
        currImage = (FileConnection) Connector.open(photosDir + imageName);
        if (!currImage.exists()) {
            System.out.println("Arquivo de Imagem não existe");
            //throw new IOException("Arquivo de Imagem não existe");
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return currImage;
}
```

Figura 22 - Método "open"
Fonte: O próprio autor (2007)

5.2.1.5.2 close()

Método sem retorno, responsável por fechar uma conexão a um arquivo físico do dispositivo.

```
public void close(){
    try {
        currImage.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Figura 23 - Método "close"
Fonte: O próprio autor (2007)

5.2.1.6 ListManager

Possuindo apenas um método, esta classe manipula uma lista de imagens, exibindo os nomes das imagens que estão gravados no repositório.

5.2.1.6.1 *viewGallery()*

Método responsável por listar todas as imagens existentes no diretório padrão de fotos do dispositivo. Recebe como parâmetro um objeto *FileConnection*, de acordo com a JSR75 representa o endereço físico do diretório. O retorno do método é um objeto do tipo *List* representando a lista com os nomes das imagens que estão no diretório.

```
public static List viewGallery(FileConnection photosDir) {
    List lRepImagem = null;
    Enumeration en;
    try {
        en = photosDir.list();
        lRepImagem.deleteAll();
        while (en.hasMoreElements()) {
            String fileName = (String)en.nextElement();
            if (fileName.charAt(fileName.length()-1) == '/') {
            } else {
                lRepImagem.append(fileName.toString(), null);
            }
        }
    } catch (Exception e) {
        System.out.println("Erro ocorrido: " + e.getMessage());
    }
    return lRepImagem;
}
```

Figura 24 - Método "viewGallery"
Fonte: O próprio autor (2007)

5.2.1.7 VideoControler

A função de capturar imagens da câmera digital do dispositivo é feita por esta classe, utilizando um método dessa classe:

5.2.1.7.1 *doSnapshot()*

Método responsável por capturar uma imagem da câmera do dispositivo. Recebe como parâmetro um objeto *VideoControl*, de acordo com a JSR135 representa o controlador de vídeo instanciado na aplicação. O retorno do método é um objeto *ImageItem* contendo a imagem capturada.

É uma aplicação *Web* acessada a partir do dispositivo móvel. Basicamente o Sistema possui três pacotes de classes fracamente acopladas. São eles:

- *server*
- *com.app.conf*
- *com.app.connect*

5.3.1 Pacote *server*

É nesse pacote que se encontra o *Servlet* controlador *ConFile.java* que é usado na conexão com o equipamento móvel. Essa classe utiliza as classes utilitárias dos pacotes *com.app.conf* e *com.app.connect*.

5.3.2 Pacote *com.app.conf*

Possui três classes com funções que: administram a conversão dos dados recebidos pelo *Servlet*, fazem a sincronização dos dados recebidos (imagem e sua descrição) e finalmente possui uma classe para administração da nomenclatura dos arquivos de foto.

5.3.2.1 *ImageManager*

Tem o objetivo de converter os dados e transformá-los em imagem do tipo “*jpg*” e salvá-la em um diretório. Possui apenas um método:

5.3.2.1.1 *byteToImage()*

Método responsável por converter um vetor de *bytes* em uma imagem e salvá-la em um diretório de fotos do servidor. Recebe como parâmetro um vetor de *bytes* contendo a imagem convertida em *bytes*, uma *String* contendo o nome da imagem e uma *String* contendo o caminho do diretório onde a imagem deverá ser gravada. O retorno do método é um objeto *boolean*, representando *true* para sucesso e *false* para falha.

```

public boolean byteToImage( byte[] data, String picPath, String picName){
    boolean result = false;
    String fImagem = picPath + picName;
    Image imagem = Toolkit.getDefaultToolkit().createImage(data);
    MediaTracker mediaTracker = new MediaTracker(new Container());
    mediaTracker.addImage(imagem, 0);
    try { mediaTracker.waitForID(0);
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
    // define a qualidade da imagem
    int qualidade = 100; // 100% - define as medidas do thumbnail
    int largura = 640;
    int altura = 480;
    double thumbRatio = (double)largura / (double)altura;
    int larguraImagem = imagem.getWidth(null);
    int alturaImagem = imagem.getHeight(null);
    double imageRatio = (double)larguraImagem / (double)alturaImagem;
    if(thumbRatio < imageRatio){
        altura = (int)(largura / imageRatio);
    } else{
        largura = (int)(altura * imageRatio);
    }
    // Desenha a imagem original para o thumbnail e redimensiona
    BufferedImage thumbImage = new BufferedImage(largura, altura,
    BufferedImage.TYPE_INT_RGB);
    Graphics2D graphics2D = thumbImage.createGraphics();
    graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
    graphics2D.drawImage(imagem, 0, 0, largura, altura, null);
    // Salva a nova imagem
    try { BufferedOutputStream out = new BufferedOutputStream(new
    FileOutputStream(fImagem));
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(thumbImage);
        param.setQuality((float)qualidade / 100.0f, false);
        encoder.setJPEGEncodeParam(param);
        encoder.encode(thumbImage);
        out.close();
        result = true;
    } catch (Exception e) {
        result = false;
        System.out.println("Erro: " + e.getMessage());
    }
    return result;
}

```

Figura 27 - Método "byteToImage"

Fonte: O próprio autor (2007)

5.3.2.2 ServerNameFile

Possui um método que cria um nome para os arquivos de foto, usando a data atual para isso:

5.3.2.2.1 *picName()*

Método responsável por gerar chaves de nomes. O retorno do método é uma *String* contendo o nome da imagem.

```
public String PicName() {  
  
    long dtMili = System.currentTimeMillis();  
    Date dt = new Date(dtMili);  
    Calendar cal = Calendar.getInstance();  
    cal.setTime(dt);  
    int day = cal.get(Calendar.DAY_OF_MONTH);  
    int month = cal.get(Calendar.MONTH);  
    int year = cal.get(Calendar.YEAR);  
    int hour = cal.get(Calendar.HOUR_OF_DAY);  
    int minute = cal.get(Calendar.MINUTE);  
    int second = cal.get(Calendar.SECOND);  
    int milésimo = cal.get(Calendar.MILLISECOND);  
  
    String file = "PIC"+day+"-"+month+"-  
    "+year+"_"+hour+minute+second+milésimo+".jpg";  
  
    return file;  
}
```

Figura 28 - Método "picName"
Fonte: O próprio autor (2007)

5.3.2.3 ServerDataSinc

Uma das classes mais importantes do *Módulo Servidor*, essa classe possui dois métodos que tratam da sincronia de dados com o dispositivo móvel, é com ela que podemos recuperar a imagem e sua descrição a partir de uma conexão HTTP.

5.3.2.3.1 SincronizeImg()

Método responsável por sincronizar o envio da imagem a partir do dispositivo móvel. Recebe como parâmetro um objeto *HttpServletRequest* que representa a requisição feita ao *Servlet* do servidor *Web*. O retorno do método é um vetor do tipo *byte* contendo a imagem. Esse método pode disparar uma exceção do tipo *IOException*.

```
public byte[] sincronizeImg(HttpServletRequest request) throws IOException{  
    int tamanho = 0;  
    ArrayList lista = null;  
    InputStream inp = request.getInputStream();  
    DataInputStream dinp = new DataInputStream(inp);  
    tamanho = dinp.readInt();  
    byte[] dados = new byte[tamanho];  
    int i = 0;  
    while(i < tamanho){  
        dados[i] = dinp.readByte();  
        i++;  
    }  
    dinp.close();  
    inp.close();  
    return dados;  
}
```

Figura 29 - Método "SincronizeImg"
Fonte: O próprio autor (2007)

5.3.2.3.2 *sincronizeComment()*

Método responsável por sincronizar o envio do comentário da imagem a partir do dispositivo móvel. Recebe como parâmetro um objeto *HttpServletRequest* que representa a requisição feita ao *Servlet* controlador do servidor *Web*. O retorno do método é uma *String* contendo a descrição. Esse método pode disparar uma exceção do tipo *IOException*.

```
public String sincronizeComment(HttpServletRequest request) throws IOException{
    InputStream inp = request.getInputStream();
    DataInputStream dinp = new DataInputStream(inp);

    String descricao = dinp.readUTF();

    dinp.close();
    inp.close();

    return descricao;
}
```

Figura 30 - Método "sincronizeComment"
Fonte: O próprio autor (2007)

5.3.3 Pacote *com.app.connect*

Este pacote contém as classes de geração do álbum digital. A geração do álbum é feita pela classe *AlbumManager*. O Sistema utiliza o padrão de projetos *Factory Method* que permite fazer com que o sistema tenha recursos pra não só gerar foto álbuns, mas pra gerar *Blogs*, e outros mais, mudando apenas um componente de software. Conforme discorre Gamma: "Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O *Factory Method* permite adiar a instanciación de uma classe para subclasses" (GAMMA, 2000, p.112).

A função do *Factory Method* é definir uma interface para a criação de objetos e permitir que as classes derivadas decidam qual a classe será instanciada. Este padrão permite que a classe escolha a instanciación para as suas classes derivadas. É a implementação de uma fábrica de objetos. No nosso sistema existe um arquivo *AlbumConfig.xml* que está no pacote "connect".

```

<?xml version="1.0" encoding="UTF-8"?>

<config>
  <factory>
    <tipo>JAlbum</tipo>
    <!--
    <tipo>JBlog</tipo>
    <tipo>JAlbum</tipo>
    -->
  </factory>
</config>

```

Figura 31 - Arquivo de configuração "AlbumConfig.xml"
 Fonte: O próprio autor (2007).

Esse arquivo é responsável por especificar que tipo de objeto será criado pela fábrica de objetos, seja criação de foto álbuns ou Blog, sendo esse último proposto a ser feito em trabalhos futuros, no aperfeiçoamento do sistema.

No nosso sistema existe a necessidade de disponibilizar mais de um serviço de geração de foto álbum, o ideal seria que essa mudança fosse feita com o mínimo de impacto no código, do contrário teríamos que alterar o código pra podermos mudar a geração de foto álbum para a geração de Blogs, por exemplo.

A classe *ConFile* é responsável por criar e administrar o álbum. Obrigatoriamente tal classe deve conhecer a classe *Factory* que é uma classe abstrata e que contém o método de criação de objetos. A classe *AlbumFactory* é uma especialização da classe *Factory* que determina qual classe de controle deverá ser instanciada.

AlbumFactory que é a nossa fábrica de objetos, retornará um objeto do tipo *AlbumManager* ou *BlogManager*, dependendo do conteúdo do arquivo *AlbumConfig.xml* que estipula o tipo de objeto a ser criado. A classe *ConFile* também deve conhecer a interface *IAlbumManager*. As classes *AlbumManager* e *BlogManager* implementam essa interface.

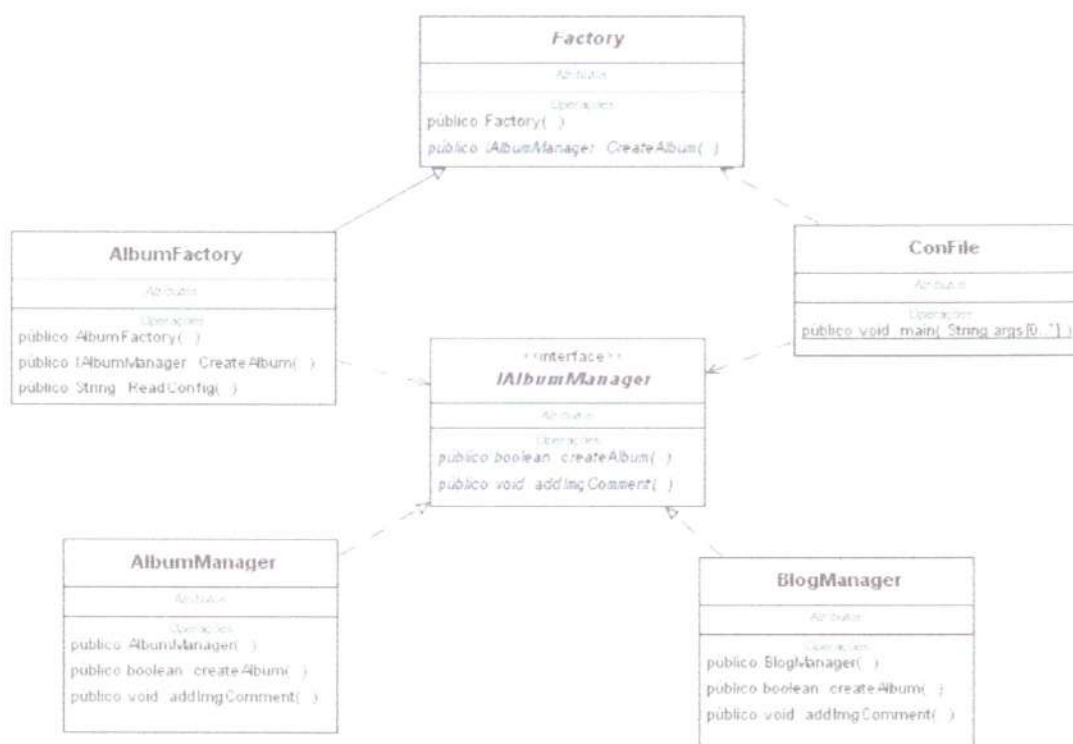


Figura 32 - Implementação do *Factory Method* no Sistema
 Fonte: O próprio autor (2007)

5.3.3.1 AlbumManager

A classe *AlbumManager* tem a função de gerar o foto álbum e gravar a descrição de uma foto específica, utilizando para isso uma biblioteca fornecida pelo *Ialbum*. Possui dois métodos: *createAlbum()* e *addImgComment()*.

O método *createAlbum()* recebe como parâmetro três *Strings*: o caminho da pasta de fotos, o caminho da pasta destino do álbum e o tipo de álbum a ser criado, e tem um retorno do tipo *boolean* indicando o sucesso da operação.

O método *addImgComment()* recebe como parâmetro também três *Strings*: a descrição da foto, o diretório das fotos e o nome dela.

```

package com.app.conect;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import se.datadosen.jalbum.AlbumBean;
import se.datadosen.jalbum.AlbumObject;
import se.datadosen.jalbum.AlbumObjectFactory;
import se.datadosen.jalbum.Config;

public class AlbumManager implements IAlbumManager {
    public AlbumManager() {}

    public boolean createAlbum(String PDirFoto, String PDirAlbum, String PAlbumType) {
        boolean result = false;
        if(PDirFoto == "") {
            PDirFoto = "C:\\\\fotos\\";
            //String dirFoto = "C:\\\\fotos\\";
        }
        if(PDirAlbum == "") {
            PDirAlbum = "C:\\\\album\\";
            //String dirAlbum = "C:\\\\album\\";
        }
        if (PAlbumType == "") {
            PAlbumType = "Mamba";
        }
        Config.getConfig().setProgramDirectory("C:/Arquivos de programas/JAlbum");
        File jalbumDir = new File("C:/Arquivos de programas/JAlbum");
        se.datadosen.util.SmartResourceBundle.setDefaultDir(jalbumDir);
        AlbumBean engine = new AlbumBean();
        Map vars = new HashMap();
        vars.put("printing", "none");
        engine.setUserVariables(vars);
        engine.setTextFileComment(true);
        engine.setDirectory(PDirFoto);
        engine.setSkin(PAlbumType);
        engine.setOutputDirectory(PDirAlbum);
        engine.setAppendImages(true);
        try {
            engine.makeAlbum();
            result = true;
        } catch (Exception e) {
            result = false;
            //System.out.println("Erro - " + e.getStackTrace());
        }
        return result;
    }

    public void addingComment(String PComment, String PDirFoto, String PFileName) {
        boolean result = true;
        AlbumBean engine = new AlbumBean();
        //engine.setDirectory(PDirFoto);
        if((PComment != null) && (PFileName != null)){
            File pathPai = new File(PDirFoto);
            File pathFile = new File(PDirFoto + PFileName);
            AlbumObjectFactory factory = new AlbumObjectFactory(engine);
            AlbumObject objPai = factory.createInstance(pathPai);
            AlbumObject obj = factory.createInstance(pathFile, objPai);
            obj.setName("AlbumTeste");
            try {
                obj.setComment(PComment);
            } catch (Exception ex) {
                //ex.printStackTrace();
            }
        }
    }
}

```

Figura 33 - Classe "AlbumManager"

Fonte: O próprio autor (2007)

5.3.3.2 AlbumFactory

A classe *AlbumFactory* herda as características da classe abstrata *Factory* reescrevendo assim seus métodos como é o caso do método *CreateAlbum()* que retorna um objeto do tipo *IAlbumManager*. Por sua vez é uma interface que é implementada pelos objetos *Produto* o método *ReadConfig()* é próprio da classe não sendo herdado da classe *Factory*, este método lê configurações a partir de um arquivo xml, para a leitura do arquivo *AlbumConfig.xml* que estipula o tipo de objeto a ser criado é usada uma outra biblioteca, a *jdom.jar* que proporciona uma maior facilidade na manipulação de arquivos XML. De acordo com o retorno do método *ReadConfig()* a classe retorna um objeto álbum.

```
package com.app.conect;

import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class AlbumFactory extends Factory {

    public AlbumFactory() {
    }

    public IAlbumManager CreateAlbum() {

        IAlbumManager p = null;
        String resp = ReadConfig();
        if(resp.equals("JAlbum")){
            p = new AlbumManager(); }
        if(resp.equals("JBlog")){
            p = new BlogManager(); }
        return p; }

    public String ReadConfig(){
        File f = new File("src/com/app/conect/AlbumConfig.xml");
        SAXBuilder sb = new SAXBuilder();
        Document d = null;
        try {
            d = sb.build(f);
        } catch (Exception ex) { }
        Element mural = d.getRootElement();
        List elements = mural.getChildren();
        Iterator i = elements.iterator();
        String resp = null;
        while (i.hasNext()) {
            Element element = (Element) i.next();
            resp = element.getChildText("tipo"); }
        return resp;
    }
}
```

Figura 34 - Classe "AlbumFactory"
Fonte: O próprio autor (2007)

5.3.3.3 IAlbumManager

A classe *IAlbumManager* é uma interface que é implementada pelas classes *AlbumManager* e *BlogManager* sendo o último a ser desenvolvido em trabalhos futuros. No diagrama de elementos básicos do padrão *Factory Method* ela seria o *Product* e as classes que implementarem essa interface seriam os *ConcretProducts*. Nela são declarados os métodos a serem usados.

```
package com.app.conect;

public interface IAlbumManager {

    boolean createAlbum(String PDirFoto, String PDirAlbum, String PAlbumType);
    void addImgComment(String PComment, String PDirFoto, String PFileName);
}
```

Figura 35 - Classe "IAlbumManager"
Fonte: O próprio autor (2007)

5.3.3.4 Factory

Essa classe é superclasse da classe *AlbumFactory*. É uma classe abstrata que declara o método de construção dos objetos – *Factory Method*. No diagrama de elementos essenciais ela seria o *Factory*.

```
package com.app.conect;

public abstract class Factory {

    public Factory() {
    }

    public abstract IAlbumManager CreateAlbum();
}
```

Figura 36 - Classe "Factory"
Fonte: O próprio autor (2007)

5.3.3.5 BlogManager

Esta classe é similar a classe *AlbumManager* e também implementa a interface *IAlbumManager*. No entanto, essa classe é responsável por gerar conteúdo para *Blogs*. Essa classe não foi implementada e sua codificação é uma das sugestões de trabalhos futuros.

```
package com.app.conect;

public class BlogManager implements IAlbumManager{

    public BlogManager() {
    }

    public boolean createAlbum(String PDirFoto, String PDirAlbum, String
PAlbumType) {
        boolean a = false;
        System.out.println("metodo CreateAlbum da classe BlogManager");
        return a;
    }

    public void addImgComment(String PComment, String PDirFoto, String PFileName)
    {
        System.out.println("metodo addimgComment da classe BlogManager");
    }
}
```

Figura 37 - Classe "*BlogManager*"

Fonte: O próprio autor (2007)

6 CONCLUSÃO

O desenvolvimento de aplicações *JME (Java Micro Edition)* para envio de imagens utilizando servidores *Web* e protocolo *HTTP (Hypertext Transfer Protocol)* para a geração de foto álbuns é complexo, mas prático e funcional.

Neste trabalho foi desenvolvida uma Aplicação Java constituída de dois módulos, um trabalhando em um dispositivo móvel e o outro funcionando no servidor *Web*. Com a finalidade de criar um foto álbum digital que é acessado através de um site *web* a partir de um navegador comum.

Várias dificuldades foram encontradas durante o desenvolvimento da aplicação, dentre elas: escassez de material de pesquisa, especificamente da parte móvel do Sistema, dificuldade no trabalho com emuladores de celulares com o uso do *NetBeans IDE*, escassez de material referente a ferramentas de geração de álbuns digital utilizando linguagem Java e dificuldade no uso dos recursos multimídia do dispositivo móvel.

O desenvolvimento dessa aplicação proporcionou uma visão mais ampla na modelagem de projetos utilizando *UML (Unified Modeling Language)* e utilização de padrões de projeto, experiência na utilização de programação direcionada a dispositivos móveis, e experiência na geração de álbuns digitais utilizando a ferramenta *JAAlbum*.

REFÊRENCIAS

ALUR, D. **Core J2EETM**: as melhores praticas e estratégias de design. Rio de janeiro: Campus, 2002.

BOOCH, G. **UML**: Guia do usuário. Rio de janeiro: Campus, 2000.

DEVELOPER information. Disponível em: <<http://jalbum.net/developer.jsp>>. Acesso em 10 set. 2007.

FILECONNECTION APIs. Disponível em: <<http://jcp.org/en/jsr/detail?id=75>>. Acesso em 10 de ago. 2007.

GAMMA, E. **Padrões de projeto**: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GUEDES, G. **UML**: Uma abordagem prática. São Paulo: Novatec, 2006.

JAVA (Linguagem de programação). Disponível em: <[http://pt.wikipedia.org/wiki/Java_\(linguagem_de_programa%C3%A7%C3%A3o\)](http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o))> Acesso em 03 set. 2007.

JME at a Glance. Disponível em: <<http://java.sun.com/javame/index.jsp>>. Acesso em 13 ago. 2007.

MATTOS, E. **Programação Java para Wireless**. São Paulo: Digerati Books, 2005.

PLATAFORMAS Java. Disponível em: <www.javaman.com.br>. Acesso em 11 set. 2007.

PROTOCOLO http. Disponível em: <<http://pt.wikipedia.org/wiki/HTTP>>. Acesso em 10 set. 2007.

TORRES, G. **Redes de computadores curso completo**. São Paulo: Axcel Books, 2001.

UML. Disponível em: <<http://pt.wikipedia.org/wiki/UML>>. Acesso em 26 nov. 2007.