Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MATEUS APARECIDO DOS SANTOS

MODELO DE ARQUITETURA PARA APLICATIVOS UTILIZANDO ADOBE FLEX E JAVA

Belo Horizonte
2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

MODELO DE ARQUITETURA PARA APLICATIVOS UTILIZANDO ADOBE FLEX E JAVA

por

MATEUS APARECIDO DOS SANTOS

Monografia de Final de Curso

Prof. Marco Túlio Valente
Orientador

Belo Horizonte
2011

MATEUS APARECIDO DOS SANTOS

MODELO DE ARQUITETURA PARA APLICATIVOS UTILIZANDO ADOBE FLEX E JAVA

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do certificado de Especialista em Informática.

Área de concentração: Ênfase em Engenharia de Software

Orientador(a): Prof. Marco Túlio Valente

Belo Horizonte
2011

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÊNFASE EM ENGENHARIA DE SOFTWARE

Modelo de arquitetura para aplicativos utilizando Adobe Flex e Java

MATEUS APARECIDO DOS SANTOS

Monografia apresentada aos Senhores:

Prof. Marco Túlio de Oliveira Valente - Orientador
DCC - ICEx - UFMG

Prof. Eduardo Magno Lages Figueiredo
DCC - ICEx - UFMG

Belo Horizonte, 02 de dezembro de 2011.

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Santos, Mateus Aparecido dos.

S237m Modelo de arquitetura para aplicativos utilizando Adobe Flex e Java. / Mateus Aparecido dos Santos. – Belo Horizonte, 2011.
xi, 59 f. : il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Marco Túlio de Oliveira Valente.

1. Computação. 2. Engenharia de software. 3. Java (Linguagem de programação de computador). I. Orientador. II. Título.

CDU 519.6*32(043)

Dedico este trabalho primeiramente à minha família e meus amigos, pela paciência e por estarem ao meu lado todo este tempo, me dando forças e acreditando no meu sucesso. E dedico a todos aqueles que contribuíram direta ou indiretamente para a conclusão desta obra.

"E aqueles que foram vistos dançando foram julgados insanos por aqueles que não podiam escutar a música."
(Friedrich Nietzsche)

RESUMO

O principal objetivo do presente trabalho é produzir um modelo de arquitetura para aplicativos de software utilizando Adobe Flex e Java que será implementado no sistema de Gerenciamento da Produção da empresa USIGRA.CNC, com sede na cidade de Nova Serrana Minas Gerais. Para elaboração de tal modelo foi realizado uma revisão literária de conceitos e tecnologias relevantes para o desenvolvimento do modelo arquitetural proposto. Em se tratando de desenvolvimento web o conceito de RIA está amplamente ligado a interfaces de qualidade que proporcionam aos usuários aplicações interativas e que usam de forma eficiente os recursos da arquitetura cliente servidor. O Flex acabou se tornando uma das melhores alternativas, porque, além de gratuito, possui todos os recursos necessários para comunicação com diversas outras linguagens de back-end. Sua fácil estrutura se resume a linguagem MXML, responsável pelo desenvolvimento do layout das aplicações, e o ActionScript que permite a codificação lógica do sistema. No final deste trabalho apresentaremos as principais funcionalidades do sistema de Controle da Produção da empresa e o desenvolvimento de um caso de uso na forma de CRUD que implementa a arquitetura proposta.

Palavras-chave: Adobe Flex, Desenvolvimento WEB, RIA, Back-End, MXML, ActionScript.

ABSTRACT

The main objective of this study is to produce an architectural model for software applications using Adobe Flex and Java to be implemented in the Production Management system of the USIGRA.CNC company, with headquarters in Nova Serrana Minas Gerais. For elaboration of this model, it was carried out a literature review of concepts and technologies relevant to the development of the architectural model proposed. When it comes to Web development the concept of RIA is widely linked to quality of interfaces that provide users with interactive applications and efficiently use the resources of the client server architecture. Flex went on to become one of the best alternative, because in addition to free, it has all the features necessary to communicate with several other back-end languages. Its easy structure comes down to MXML, responsible for developing the layout of the application, and ActionScript that allows the coding logic of the system. At the end of this paper will present the main features of the system of production control of the company and the development of a use case in the form of CRUD that implements the proposed architecture.

Keywords*:* Adobe Flex, Web Development, RIA, Back-End, MXML, ActionScript.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

CSS - Cascading Style Sheets

HTML - *HyperText Markup Language*

IDEs - Integrated Development Environment

RIAs - Rich Internet Application

CRUD - Create, Read, Update, Delete

RUP - Rational Unified Process

J2ME - Java 2 Micron Edition

PDA - Personal Digital Assistant

J2SE - Java 2 Standart Edition

J2EE - Java 2 Enterprise Edition

XML - eXtensible Markup Language

JVM - Java Virtual Machine

MXML - Magic Extensible Markup Language

HTTP AMF - Action Message Format

RPC - Remote Procedure Call

LCDS - Learning Content Development System

Web - *World Wide Web*

Xml - *eXtensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Podemos observar no desenvolvimento de software para a plataforma Web uma divisão de interesses entre aplicações clientes e aplicações servidoras e a forma como as mesmas se comunicam. Nesse ambiente distribuído, podemos ter tecnologias diferentes comunicando-se umas com as outras. Esses interesses possibilitaram o desenvolvimento de ferramentas e tecnologias especialistas para a produção de software, como é o caso da Adobe que desenvolveu uma ferramenta especialista para a camada *front-end* das aplicação chamada Flex.

O Flex é uma estrutura altamente produtiva em se tratando de interfaces com os usuários, que disponibiliza soluções prontas para a confecção de telas interativas e usa amplamente o conceito de RIA (*Rich Internet Aplication* – Aplicações Ricas em Internet), conceito bastante difundido no desenvolvimento de software para a plataforma Web.

A integração é fator crucial para aplicações que usam a internet como ambiente, portanto o Flex oferece bastante recursos para a comunicação com os mais variados tipos de serviços e linguagens de programação disponíveis no mercado.

A plataforma Java de desenvolvimento de software fornece um ambiente estável e evolutivo de suas ferramentas de desenvolvimento, favorecendo desenvolvedores e empresas de produção de software com os mais variados tipos de *frameworks* e IDEs para dar suporte ao desenvolvimento de software. Combinar as tecnologias Java e Flex fornece aos desenvolvedores um meio rápido e eficiente de desenvolver software de qualidade e que utilizam padrões amplamente testados pela comunidade desenvolvedora.

Desenvolver software eficientes e escaláveis utilizando Flex e Java requer uma organização elaborada dos componentes tanto no *front-end* como no *back-end* das aplicações. Isso implica na utilização de padrões e conceitos difundidos pelas boas práticas de programação e utilização de recursos da Engenharia de Software para se obter um produto final que atenda às necessidades dos usuários.

1.1. OBJETIVO GERAL

A partir do tema proposto (Modelo de Arquitetura para Aplicativos utilizando Adobe Flex e Java), o principal objetivo deste trabalho é elaborar um modelo arquitetural para aplicativos de software que utilizam Flex na aplicação *front-end* e Java no *back-end*, que possa ser materializado em uma empresa do setor calçadista de Nova Serrana em Minas Gerais.

1.2. OBJETIVO ESPECÍFICO.

- Apresentar fundamentação teórica dos conceitos da Engenharia de Software;
- Apresentar fundamentação teórica do paradigma de programação orientada por objetos e a plataforma Java;
- Apresentar fundamentação teórica dos conceitos de RIA`s e da tecnologia Adobe Flex;
- Estudar e descrever formas eficientes de integração entre Adobe Flex e Java;
- Apresentar um modelo Arquitetural para aplicativos utilizando Adobe Flex e Java.

1.3. ESTRUTURA DO TRABALHO

O restante desta monografia está organizada descrito a seguir:

- O capítulo 2 apresenta uma revisão de literatura sobre Engenharia de Software, paradigma de programação orientado a objetos, plataforma Java de desenvolvimento de software, RIA e Adobe Flex.

- O capítulo 3 apresenta as formas de integração com o Adobe Flex, o BlazeDS como tecnologia de integração e o modelo arquitetural proposto em auto nível de abstração.
- O capítulo 4 apresenta uma avaliação dos estados realizados no desenvolvimento desta monografia e um estudo de caso para implementar a arquitetura na forma de um CRUD simples.

2. REVISÃO DA LITERATURA

2.1. ENGENHARIA DE SOFTWARE

Desenvolver e implantar softwares nas mais diversas áreas tem se tornado bastante comum. Quando falamos de sistemas de software, temos o conceito de que software é apenas um produto final. Sistemas de software, conceitualmente falando, são bem mais amplos que um mero produto, abrangendo toda a documentação, arquivos de configuração e um conjunto de programas que são necessários para sua execução.

Diante da amplitude e complexidade de um sistema de software, estruturá-lo de uma forma organizada e definir uma metodologia a ser seguida, é de extrema importância para se obter sucesso na implantação de um sistema, desde seu desenvolvimento até a validação pelo usuário.

A partir de 1968, após uma conferência para discutir a denominada “crise do software”, formou-se o conceito de Engenharia de Software. Esta crise era produto direto da introdução (naquela época) de um poderoso hardware de terceira geração, que possibilitou desenvolver softwares cada vez maiores e complexos, inimagináveis para a época. A falta de padronização dificultava ainda mais a manutenção destes no ambiente comercial. Nesta conferência foram levantados vários aspectos referentes à produção de software e a partir destes conceitos foram desenvolvidos, com o passar dos anos, muitas técnicas, métodos e processos que ajudam os desenvolvedores a planejar, executar, implantar e evoluir um projeto de software.

Engenharia de software pode ser definida como uma disciplina da engenharia que aborda todos os aspectos do desenvolvimento de software, desde a concepção até a sua implantação e manutenção (SOMMERVILLE, 2007).

Nas subseções a seguir são apresentadas as principais metodologias para desenvolvimento de software:

2.1.1. METODOLOGIA EM CASCATA

Primeiro modelo publicado do processo de desenvolvimento de software, basicamente o modelo em cascata é um conjunto de técnicas para cada fase particular do processo de software. Cada etapa interage somente com a etapa posterior e esta só é iniciada quando a anterior é finalizada. Quando se finaliza uma etapa, gera-se uma saída, que será o ponto de partida para a próxima etapa (SOMMERVILLE, 2004). A Figura 1 mostra as etapas da metodologia em cascata.

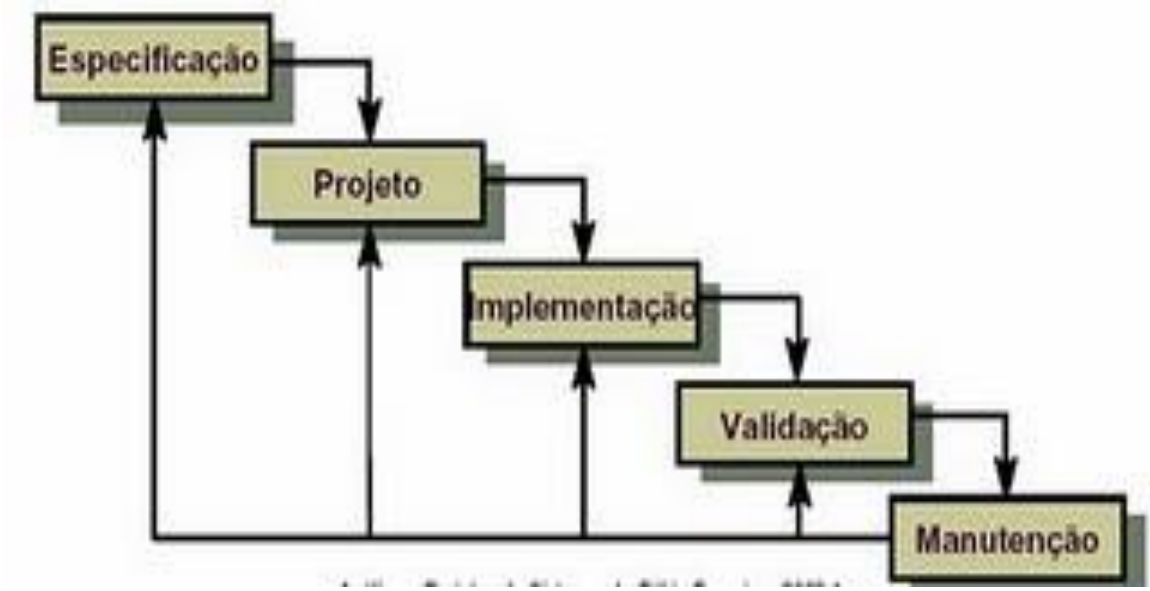

FIGURA 1: Etapas da Metodologia em Cascata
Fonte: Sommerville, 2004

Cada fase do desenvolvimento é muito bem definida no modelo em cascata. Na fase de definição de requisitos são elicitadas todas as funcionalidades do sistema juntamente com o cliente. A partir desta fase, a equipe de análise e projeto elabora as especificações, o sistema é implementado e testado. O cliente só volta a ter contato com a equipe na fase de operação e manutenção.

Um ponto forte a ser considerado da metodologia em cascata é a geração pesada de documentação, fator que auxilia bastante os analistas na especificação de requisitos e modelagem. Dentre estes documentos podemos destacar o documento visão e o documento de especificação de requisitos de software (SRS). Cada documento expressa uma perspectiva diferente das funcionalidades do software. O documento visão apresenta um panorama inicial do sistema num nível de abstração bastante alto, com o objetivo de facilitar o entendimento de todos

envolvidos. Já o SRS é a declaração oficial do que os desenvolvedores do sistema devem implementar e o que é esperado pelo cliente.

A flexibilidade para se adaptar a diferentes metodologias da engenharia de software também é uma vantagem a se destacar da metodologia em cascata. Esta caracteristica permite que as empresas desenvolvedoras aproveitem as melhores práticas de um conjunto de outras metodologias, criando um modelo híbrido adaptável a sua realidade.

Um dos principais problemas enfrentados pela metodologia em cascata é a sua divisão inflexível do projeto em estágios distintos e nos custos elevados para a sua execução. Como o relacionamento do cliente é feito apenas nas fases iniciais do desenvovimento, isso causa um congelamento prematuro dos requisitos, dificultando assim a localização e correção de erros no projeto (SOMMERVILLE, 2007).

2.1.2. METODOLOGIA EM ESPIRAL

Diante dos custos elevados e aumento na complexidade, o modelo em cascata estava se tornando um entrave para projetos de software, devido a sua inflexibilidade entre as fases e a sua incapacidade de absorver mudanças nos requisitos. A partir desta visão iniciou-se, dentro da engenharia de software, discussões sobre formas alternativas de desenvolvimentos, que resultou nos modelos interativos.

A metodologia em espiral foi uma das primeiras técnicas baseadas na evolução incremental do software. Em vez de se especificar todas as funcionalidades no início do projeto, são planejadas uma série de interações e, ao final de cada interação, tem-se uma porção da aplicação pronta para o ambiente comercial (VASCONCELOS, 2006). A Figura 2 mostra uma visão geral da metodologia em espiral.

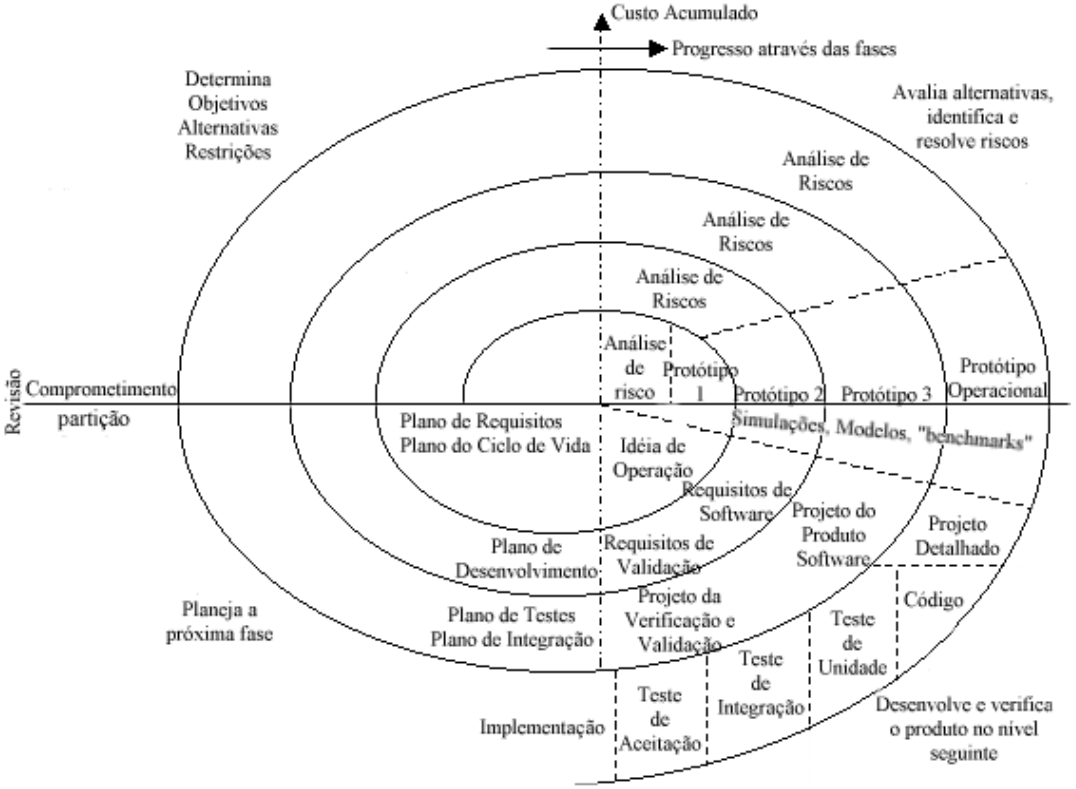

FIGURA 2: Visão Geral da Metodologia em Espiral
Fonte: Sommerville, 2004

2.1.3. METODOLOGIAS ÁGEIS

Com o passar dos anos as metodologias e processos de desenvolvimento de software tradicionais, começaram a ser contestadas pelo fato de estarem burocratizando o desenvolvimento de software, onde muito do tempo gasto para desenvolver um produto de software era desperdiçado com muitas regras e planos, desnecessários para a satisfação dos clientes e acarretando atrasos e elevação de custos. As metodologias ágeis nasceram para lidar com as mudanças contínuas dos requisitos durante o processo de desenvolvimento de software.

De acordo com o manifesto[1] (2011), houve uma discussão em 2001 entre dezessete pesquisadores da área de desenvolvimento de software, com o objetivo de despertar a comunidade desenvolvedora da necessidade de produzir software com custos e prazos menores. Essa discussão resultou no manifesto chamado Manifesto para Desenvolvimento Ágil de Software, no qual foram elaborados quatro valores fundamentais para o desenvolvimento ágil:

- Indivíduos e interações mais que processos e ferramentas;

[1] Para maiores informações acesse: http://agilemanifesto.org/

- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder as mudanças mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

2.1.4. EXTREME PROGRAMMING

A Extreme Programming é uma das mais conhecidas metodologias ágeis. Este tipo de abordagem surgiu na década de 1990, e consiste, basicamente, na abordagem interativa para a especificação, desenvolvimento e entrega do software, tornando o trabalho da equipe de desenvolvimento focada apenas no produto final, sem se preocupar com a documentação do projeto (SOMMERVILLE, 2007).

O processo XP é voltado para o desenvolvimento de software orientado a objetos, com equipes pequenas e que dão suporte ao desenvolvimento incremental. Ou seja, desde o início do projeto são lançados pequenos releases até que se tenha um produto final completo. As práticas pregadas pelo processo são: a integração cliente com a equipe de desenvolvimento; o aceite do usuário a cada release; a simplicidade de se implementar estritamente o que o cliente necessita; a coragem de se produzir quase nenhuma documentação e acreditar que o software será capaz de evoluir com segurança e agilidade (TELES, 2006).

O que torna a XP difícil de ser executada, segundo Beck (2004), é reunir todas as peças e mantê-las unidas, uma vez que envolve a mudança de cultura da própria equipe, a capacidade de cooperação e a integração de todos de forma harmônica. Outro ponto que pode fazer o XP fracassar é a adoção de grandes times de desenvolvedores, clientes desconfiados e tecnologias que não suportam efetivamente as modificações constantes no software.

2.2. PROCESSO PRÁXIS DE DESENVOLVIMENTO DE SOFTWARE

O Práxis é um processo de desenvolvimento de software, com fins educacionais com objetivo de dar suporte ao treinamento de engenharia de

software, com bastante semelhança com processo unificado (Unified Process), RUP[2] (Rational Unified Process).

Segundo Paula Filho (2009), o Práxis é desenhado para suportar projetos realizados por pequenas equipes, com duração de seis meses a um ano. Com isso, pretende-se que ele seja utilizável para projetos de fim de curso de graduação ou similares, ou projetos de aplicação de disciplinas de engenharia de software.

O praxis foi dividido em disciplinas, que por sua vez possuem: tarefas; papéis; produtos de trabalho e orientações. Por padrão o práxis foi dividido em 10 disciplinas, onde estas são classificas em quatro grupo maiores (PAULA FILHO, 2009). O Quadro 1 mostra as disciplinas do Práxis.

QUADRO 1
Disciplinas do Práxis 3.0

Grupo	Disciplina	Sigla	Objetivo
Especificação	Requisitos	RQ	Obter o enuciando completo, claro e preciso dos requisitos de um produto de software.
	Análise	AN	Detalhar, estruturar e validar os requisitos de um produto, em termos de um modelo conceitual do problema.
Solução	Desenho	DS	Definir uma estrutura implementável para um produto de software, que atenda os requisitos especificados para ele.
	Testes	TS	Verificar os resultados os resultados da implementaçã, através do planejamento, desenho e realizações das atividades desse processo.
	Implementação	IM	Realizar o desenho de um sistema em termos de documentação de uso, conforme as tecnologias escolhidas.
Gestão	Gestão da Qualidade	GQ	Verificar e garantir a qualidade em projetos e produtos de software.
	Gestão de projetos	GP	Planejar e controlar os projetos de desenvolvimento de software.
	Gestão de Alterações	GA	Administrar as alterações em requisitos e artefatos dos projetos e produtos.
	Engenharia de	EP	Dar suporte e promover melhorias nos próprios processos de software.

[2] Para maiores informações acesse: http://pt.wikipedia.org/wiki/Rup

Ambiente	Processos		
	Engenharia de Sistemas	ES	Desenvolver o ambiente de sitema em que o produto está incluído.

Fonte: Paula Filho, 2009

2.2.1. DESENVOLVIMENTO ORIENTADO POR CASOS DE USO

Segundo Paula Filho (2009), o ciclo de desenvolvimento de um caso de uso do Práxis 3.0, é composto por uma hierarquia de atividades, onde a maioria leva a um novo estado de desenvolvimento. Na primeira interação de um projeto ele parte de um estado inicial para um estado identificado , então executa-se um conjunto de atividades de primeiro nível:

- Especificação do problema do caso de uso**,** na qual se avança no detalhamento e entendimento dos requisitos expressos no caso de uso, o que é refletido na evolução do modelo do problema. Essa atividade leva ao estado analisado.
- Desenvolvimento da solução do caso de uso**,** na qual o problema é resolvido por meio de um modelo da solução, tendo-se código executável a partir do estado realizado. Essa atividade leva ao estado implementado.
- Conferência da qualidade do caso de uso**,** na qual se procura garantir que, de fato, a solução desenvolvida resolva satisfatoriamente o problema representado pelo caso de uso. Essa atividade leva ao estado completo.

O Quadro 2 mostra os possíveis estados que um caso de uso pode assumir durante seu desenvolvimento. Na sequencia, a Figura 3 ilustra o fluxo das atividade de primeiro nível do ciclo de desenvolvimento de um caso de uso.

QUADRO 2
Estados dos Casos de Uso

Nome	Definição
Identificado	Recebeu nome e descrição sucinta.
Detalhado	Os fluxos de eventos estão completos e consistentes com os atores e interfaces de usuário e de sistema.
Analisado	Os fluxos de eventos foram realizados como interações das colaborações de análise.
Desenhado	As colaborações de uso derivadas estão completamente desenhadas.
Especificado	As colaborações de testes derivadas têm procedimentos e casos de teste foram completamente especificados.
Realizado	As colaborações de desenho interno derivadas foram realizadas, tendo as classes participantes de entidade e controle sido implementadas.
Implementado	As colaborações de desenho interno derivadas estão completamente implementadas, integradas e documentadas.
Testado	Os testes de aceitação foram executados com sucesso por uma equipe independente e os defeitos encontrados foram todos corrigidos.
Validado	Foi aprovado na avaliação dos usuários.
Completo	A linha de base correspondente foi aprovada em auditoria da qualidade.

Fonte: Paula Filho, 2009

Figura 3: Atividades de Alto Nível de Um Caso de Uso

Fonte: Paula Filho, 2009

Os casos de uso no praxis 3.0, são desenvolvidos em interações, onde temos uma interação genérica para interar repetidamente as atividades de desenvolvimeto de um caso de uso.

A Figura 4 ilustra a interação genérica do Práxis 3.0.

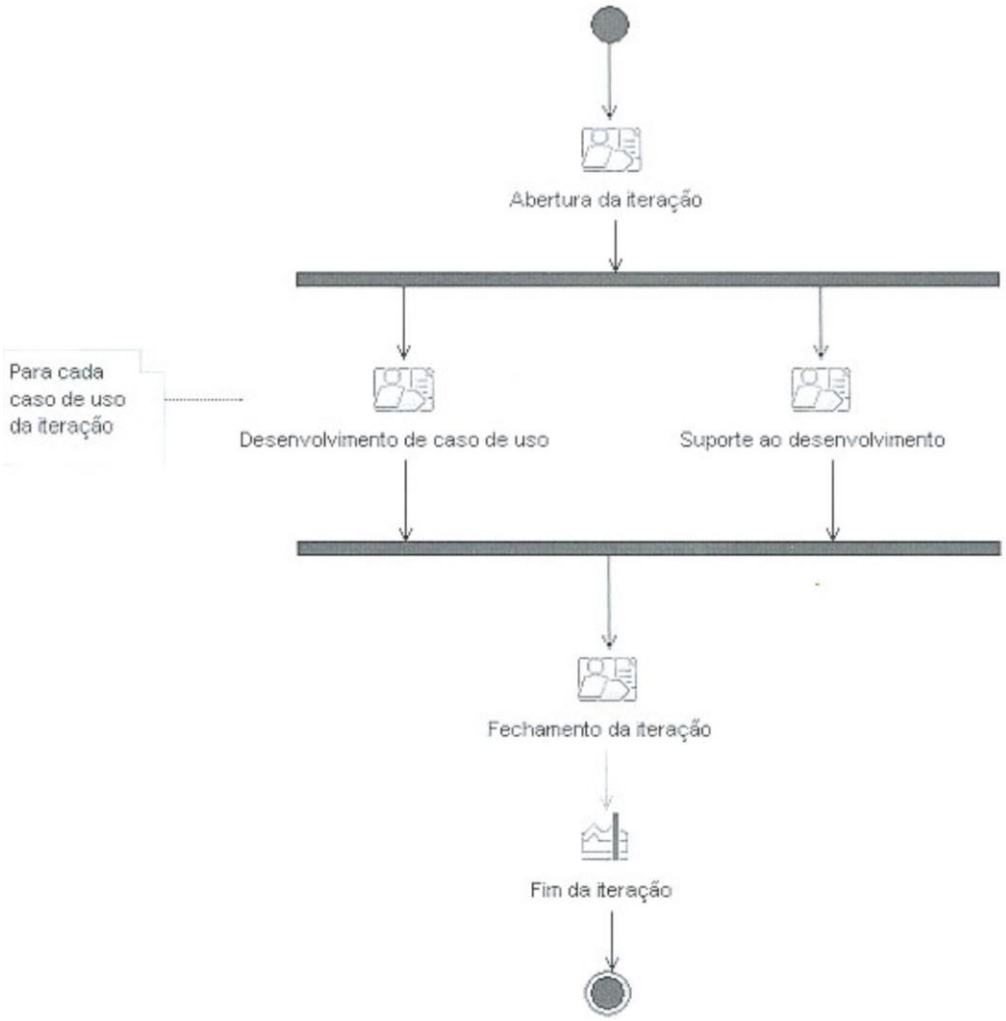

Figura 4: Interação Genérica do Práxis
Fonte: Paula Filho, 2009

2.2.2. CICLO DE VIDA PADRÃO

O processo práxis 3.0 é dividido em fases semelhantes ao processo unificado, salvo pequenas alterações, mostradas no Quadro 3. Basicamente as fases caracterizam o ciclo de vida padrão do Práxis, onde cada fase possui um marco que determina seu fim.

QUADRO 3
Fases do Ciclo de Vida Padrão

Fase	Descrição
Iniciação	Fase que procura atingir o consenso entre as partes interessadas sobre o escopo e objetivos do projeto
Elaboração	Fase na qual se atinge uma arquitetura sólida e uma visão bem fundamentada dos riscos, que permitirá o desenvolvimento estável e bem controlado do restante do projeto.
Construção	Fase na qual se completa o desenvolvimento de uma versão completamente operacional do produto, que atende os requisitos especificados.
Transição	Fase na qual o produto é colocado a disposição de uma comunidade de usuários para testes finais, treinamento e uso inicial.

Fonte: Paula Filho, 2009, p.152.

A Figura 5 mostra o fluxo das fases do ciclo de vida do Práxis 3.0.

Figura 5: Fases do Ciclo de Vida Padrão
Fonte: Paula Filho, 2009

2.3. ARQUITETURA DE SOFTWARE

Com a crescente demanda de sistemas de software cada vez mais complexos, houve a necessidade de se construir software cada vez mais capaz de atender a clientes e domínios diferentes. Isso impulsionou o aparecimento de formas distintas de se construir e estruturar software, em várias perspectivas relacionadas à qualidade de software: disponibilidade; compatibilidade; escalabilidade; manutenabilidade; confiabilidade; segurança entre outros aspectos. Basicamente um software existe nos domínios de sua arquitetura.

Segundo a Wikipédia (2011), embora o termo arquitetura de software seja novo na indústria de software, seus princípios vêm sendo aplicados pela engenharia de software desde os anos 80, onde podemos destacar as primeiras tentativas de capturar e explicar a arquitetura de software de sistemas, que se mostraram imprecisas e desorganizadas. Na década de 90 houve avanços significativos para definir e codificar os aspectos fundamentais desta disciplina.

Mesmo sendo uma disciplina em desenvolvimento, houve bastante evolução na definição de modelos arquiteturais, onde podemos destacar dois modelos clássicos:

- Modelo cliente/servidor: é um modelo computacional que separa clientes (solicita requisições) e servidores (atende requisições), sendo interligados entre si, geralmente utilizando-se redes de computadores para se comunicarem.
- Modelo monolítico: modelo onde a aplicação com suas interfaces, regras de negócios e lógica de acesso a dados ficam concentradas em único programa.

2.4. ORIENTAÇÃO A OBJETOS

A orientação a objetos começou a ser utilizada em maior escala na década de 1980, trazendo uma nova maneira de analisar, projetar e desenvolver sistemas. Esta nova forma aborda o desenvolvimento de sistemas de um modo mais abstrato e genérico. Os grandes benefícios em relação à programação estruturada, que era a mais utilizada até aquele momento, são o reaproveitamento de código e a facilidade de manutenção (SIMOR e DORNELES,2009).

O conceito principal deste paradigma está no objeto, sendo seus processos embasados na representação computacional de objetos reais e de seus relacionamentos. Segundo Simor e Dorneles (2009), a orientação a objetos busca representar, do modo mais fiel possível, situações do mundo real no ambiente computacional.

Os processos do desenvolvimento orientado a objetos, conforme citado, são baseados nos objetos e seus relacionamentos. Porém, é importante destacar a existência de classes, estas definem os objetos presentes no sistema e determinam o comportamento, os possíveis estados e os relacionamentos destes. Desta forma cada objeto é, na realidade, uma instância de uma classe (CEUNES PROGRAMAÇÃO III, s/a).

Baseado em CEUNES Programação III (s/a), os benefícios da utilização da orientação a objetos são a centralização de dados e funções em uma única unidade do sistema (objeto); a reutilização de código (herança) – permite um ganho significativo de tempo e custo no desenvolvimento; o ocultamento de informação (encapsulamento) – uma classe acessa apenas a interface de outra, evitando erros acidentais; manutenabilidade – o sistema não fica acoplado, o que facilita a manutenção.

É comum deparar-se com questionamentos sobre a vantagem da utilização da orientação a objetos em relação à programação estruturada. O Quadro 4 traz um comparativo, onde é possível perceber a metodologia e as técnicas de ambos os paradigmas.

QUADRO 4
Comparativo entre OO e Programação Estruturada

Programação Algorítmica (Procedimental)	Programação OO
• Ênfase: construção de algoritmos; • Utiliza abordagem de refinamentos sucessivos; • Subproblemas são codificados como unidades denominadas procedimentos, subrotinas ou funções; • Unidades de programa podem ser agrupadas em módulos; • Programa resultante consiste de uma coleção de unidades que se comunicam entre si; • Visão *Top-down*; • Forte uso de Decomposição Funcional; • O sistema é composto por dados e funções, tratados separadamente, mas que podem interagir.	• Metodologia desenvolvida objetivando suprir deficiências encontradas em programação algorítmica; • Encapsulamento: combinação de dados (atributos) e funções (métodos) numa única entidade de programa; Objeto: entidade de encapsulamento; • Um programa no paradigma POO consiste de vários objetos que se comunicam (isto é, trocam mensagens) entre si utilizando seus métodos constituintes; • Visão de Objetos cooperativos; • As características de comportamento e informações são modeladas de maneira fortemente relacionadas; • O sistema é composto por objetos, que contém dados e funções (isto é, estão

Problemas: • Programas muito grandes tornam-se muito complexos e difíceis de entender e manter; • Dificuldade em modelar muitos problemas da vida real com enfoque em algoritmos; • É mais fácil simular o funcionamento de sistemas complexos com enfoque em suas partes constituintes do que em termos dos algoritmos utilizados pelo sistema. Exemplo: um automóvel é melhor entendido em termos de direção, freios, etc. do que em termos dos algoritmos que o fazem funcionar; • Linguagens procedimentais não oferecem facilidades para criação de novos tipos de dados que funcionem como os tipos de dados primitivos.	reunidos em um só elemento). Ocultação de Informação: Métodos que fazem parte de um objeto provêem (usualmente) a única forma de acesso aos seus dados; • Campos de um objeto não podem ser acessados diretamente, apenas indiretamente por meio de seus métodos constituintes; • Previne alterações acidentais de dados e facilita a manutenção e depuração dos programas.

Fonte: CEUNES – PROGRAMAÇÃO III. s/a, p. 13 (adaptado).

Diante deste quadro, é possível perceber vantagens significativas da utilização da orientação a objetos, principalmente em sistemas maiores. Várias linguagens de programação seguem este paradigma, C++, C#, Java, Object Pascal, Objective-C, Python, Ruby e Smalltalk são bons exemplos.

2.4.1. PLATAFORMA JAVA

A plataforma Java é um conjunto de tecnologias livres amplamente utilizadas para a construção dos mais variados tipos de sistemas de software. Abrange três versões principais: uma versão voltada para dispositivos móveis, outra voltada para desenvolvimento desktop e outra para aplicações corporativas. A Figura 6 mostra a arquitetura de desenvolvimento Java.

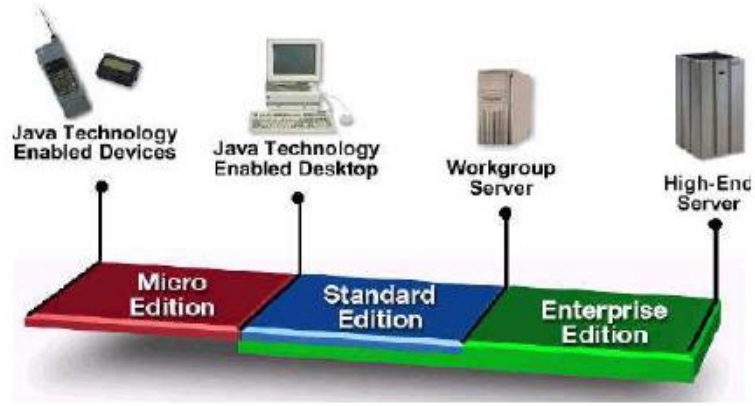

FIGURA 6 – Plataformas de tecnologias Java
Fonte: Coura, 2009

A plataforma Java possuí as seguintes edições:

- *Micro Edition (J2ME)*: Esta versão é voltada para desenvolvimento de aplicações para dispositivos móveis, tais como celulares, PDA's, *smartphones*, dentre outros. Esta plataforma é responsável pela produção de softwares com recursos limitados, ou seja, controla de forma eficaz os recursos disponíveis, para conseguir o melhor desempenho da aplicação;
- *Standard Edition (J2SE)*: Esta versão propicia recursos para o desenvolvimento de aplicações para desktops. Contém todas as API's básicas para o desenvolvimento de aplicações, incluindo suporte para a criação de interfaces gráficas;
- *Enterprise Edition (J2EE)*: Versão voltada para o desenvolvimento de aplicações corporativas e todo o suporte à comunicação em rede. Esta versão abrange a plataforma J2SE e mais um conjunto de especificações e bibliotecas que fornecem recursos para a criação de aplicações.

2.4.2. LINGUAGEM JAVA

O paradigma de orientação a objetos é um dos modelos de programação mais discutidos na área de desenvolvimento. A partir desta discussão, surgiram várias linguagens que suportam este paradigma, tais como o C++, C#, Java, dentre outras. A principal característica destas linguagens é permitir a reutilização de código, facilitando o processo de produção de software.

A linguagem Java surgiu a partir de um projeto desenvolvido pela Sun Microsystem chamado Green Project, que tinha como motivador a criação de tecnologias para a fabricação de dispositivos eletrônicos inteligentes para o consumidor final. Em 1995, a Sun anunciou o Java como plataforma de desenvolvimento, o que despertou o interesse da comunidade comercial, devido à popularização da internet. Com o amadurecimento da linguagem, a sua segunda edição passou a oferecer uma série de recursos que possibilitam desenvolver desde aplicativos corporativos de grande complexidade até softwares voltados para o consumidor final (DEITEL e DEITEL, 2005).

Um grande diferencial da linguagem Java em relação a outras linguagens é o seu processo de compilação: o compilador Java gera os bytecodes (arquivos compilados), que, no momento da execução, serão interpretados pela máquina virtual Java (JVM). A JVM é uma camada que faz a comunicação entre os programas Java e o sistema operacional, garantindo assim sua portabilidade. A Figura 7 mostra o processo de compilação de um programa Java.

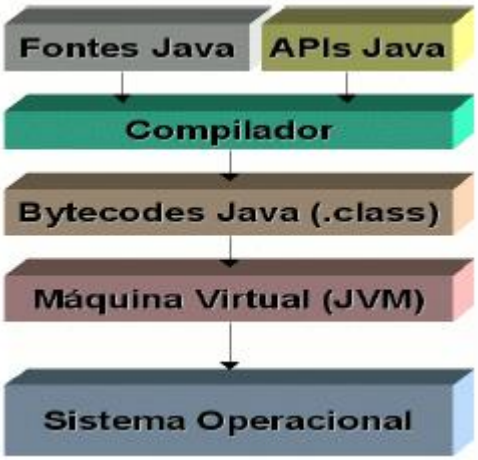

FIGURA 7 – Processo de compilação de um programa Java
Fonte: Ruiz, 2008

2.5. APLICAÇÕES RICAS EM INTERNET

Diante da crescente demanda de aplicações com maior interatividade e riqueza nos conceitos de interface com o usuário, esses conceitos tornaram o desenvolvimento de tais aplicações bastante complexo, abrindo – se então um precedente para as aplicações RIA.

O termo RIA (*Rich Internet Aplication* – Aplicações Ricas em Internet) foi utilizado pela primeira vez em 2002 pela antiga macromídia agora, Adobe (WIKIPEDIA, 2011).

Na verdade, é mais que um conjunto de tecnologias, é um conceito que vem se transformando na evolução das aplicações web tradicionais. É a funcionalidade e interatividade do desktop com a disponibilidade e dinamismo da web (BRIDEE,2007).

FIGURA 8: Abstraindo o Conceito RIA
Fonte: Bridee, 2007

As aplicações RIA estão a cada dia ganhado mais mercado. Diversos aplicativos, tais como planilhas eletrônicas e gestores de conteúdo, já abordam conceitos de aplicativos ricos em internet e estão cada vez mais semelhantes aos softwares desktop (SCHMITZ, 2008).

Entendendo melhor o conceito de RIA, é uma nova forma de desenvolver software 100% web, através da utilização de componentes ricos em interface. Isso implica na utilização de diversos recursos, tais como: transições, efeitos, arrastar, soltar, entre outros. Esses recursos podem ser utilizados de forma bem simples, ao contrário do modelo HTML+JavaScript, onde era preciso programar bastante para alcançar qualquer tipo de funcionalidade da RIA (SCHMITZ, 2008).

2.6. A ADOBE FLEX

Adobe Flex é um framework de código aberto criado pela Adobe, que utiliza tecnologias para desenvolvimento da camada de apresentação. Ele possibilita a criação de interfaces com visual totalmente inovador e com boa integração com a camada de aplicação.

Segundo Schmitz (2008), o Adobe Flex é um framework de desenvolvimento de aplicações RIA, contendo diversos tipos de componentes visuais e uma poderosa linguagem de programação chamada Action Script 3.0. Com este conjunto, o Flex torna-se uma ferramenta essencial para a criação de software para a web, utilizando a fundo o conceito RIA.

A forma como o Flex funciona é bem semelhante ao Adobe Flash, que é uma ferramenta de animação, geralmente utilizado nos mais diversos sites da Web. Da mesma forma que o Flash, o Flex também cria seu arquivo final com extensão SWF[3]. Para que se dê o funcionamento do arquivo gerado é necessária uma compilação, Na Figura 9 podemos ver claramente como isso acontece (SCHMITZ, 2008).

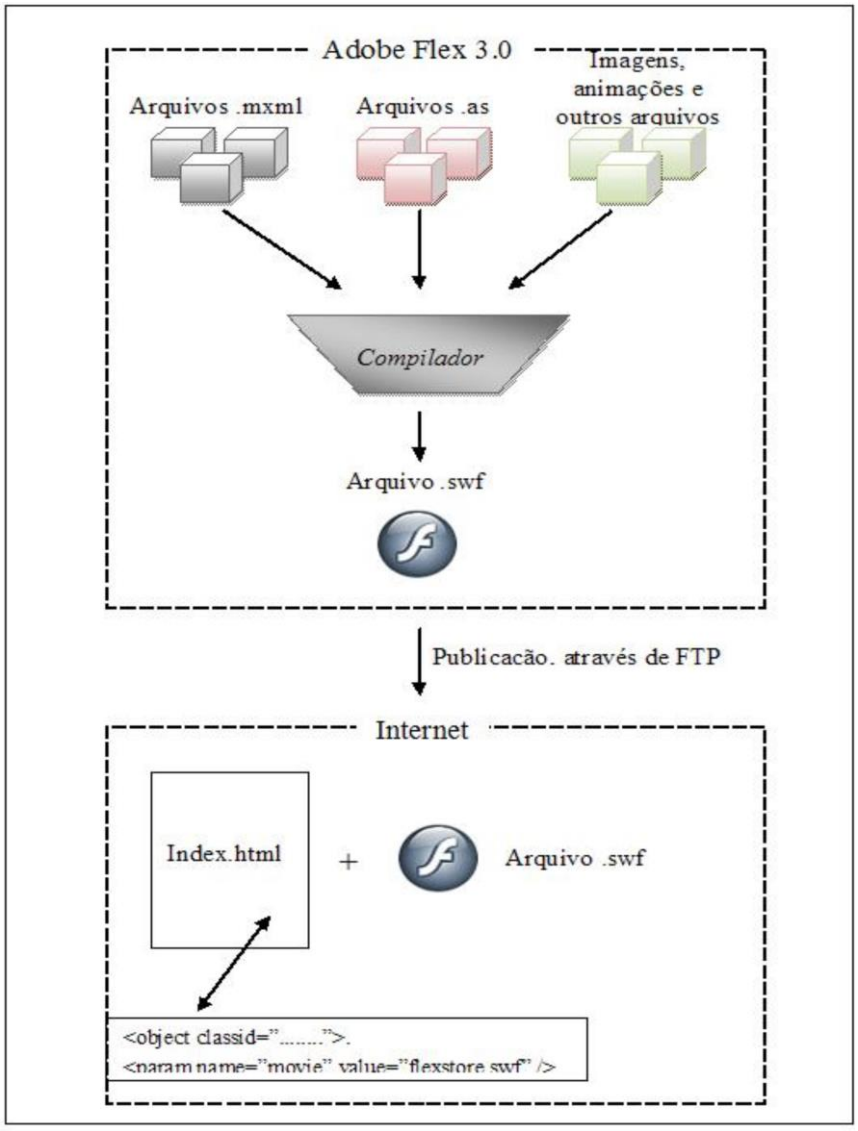

FIGURA 9 – Compilação do Adobe Flex
Fonte: Schmitz, 2008

[3] Para maiores informações acesse: http://pt.wikipedia.org/wiki/SWF

2.6.1 ESTRUTURA DE DESENVOLVIMENTO DO ADOBE FLEX

O Flex oferece um modelo moderno de linguagem de programação baseado em padrões que oferecem suporte a modelos comuns de design. MXML, é uma linguagem declarativa baseada em XML, usada para descrever comportamentos e layout de interface com usuário, e ActionScript 3.0, uma linguagem de programação orientada por objetos, a qual é usada para criar a lógica de cliente, segundo o site oficial da Adobe[4].

Existem outros recursos que também fazem parte do Flex como: folhas de estilos (CSS), arquivos de configuração, imagens, animações, entre outros. É de responsabilidade do compilador agrupar tudo em um único arquivo swf.

2.6.1.1 MXML

A linguagem MXML tem bastante semelhança com XML[5], pois, usa marcações de texto. Sua estrutura de desenvolvimento é praticamente a mesma do HTML, mas, por utilizar padrões da XML é uma linguagem mais organizada e consequentemente mais estruturada que o HTML. Sua sintaxe é bem parecida com a do HTML por utilizar tags de início e fim representadas pelos sinais de maior e menor.

Segundo Schmitz (2008), MXML é uma linguagem de marcação de texto usada principalmente para inserir componentes visuais no seu aplicativo. Assim como a linguagem HTML, onde você pode usar tags para criar uma página web, a linguagem MXML possui diversos componentes prontos que podem ser inseridos em seu projeto, bastando apenas observar algumas regras da linguagem XML, como:

- Todas as tags devem estar em forma hierárquica;
- Todas as tags devem ter seu fechamento;
- Tags únicas também possuem fechamento.

[4] Para maiores informações acesse: http://www.adobe.com/br/products/flex/overview/
[5] Para maiores informações acesse: http://pt.wikipedia.org/wiki/Xml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
                xmlns:cf="coldfusion.service.mxml.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Declarations>
        <cf:Document id="docConvert" cfServer="10.0.2.2" cfPort="8300"
            servicePassword="SERVICE_PASSWORD" serviceUserName="SERVICE_USERNAME"
            action="generate" format="PDF" result="handleResult(event)"
            content="{sourceText.htmlText}"  />
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            import mx.rpc.events.ResultEvent;
            private function handleResult(event:ResultEvent):void {
                if(docConvert.format == "PDF"){
                    swf.unloadAndStop();
                    docConvert.format = "FlashPaper";
                    pdfPath.text = event.result.toString();
                    docConvert.execute();
                }else{
                    swf.source = event.result.toString() + "?POPUP_ENABLED=false";
                    docConvert.format = "PDF";
                    btRender.enabled = true;
                }
            }
        ]]>
    </fx:Script>
    <mx:HDividedBox width="100%" height="100%">
        <mx:VBox label="Create" width="450" height="550">
            <mx:RichTextEditor title="Content" id="sourceText"  width="100%" height="60%" />
            <mx:Button label="Render" id="btRender" click="{docConvert.execute()}" />
        </mx:VBox>
        <mx:VBox>
            <mx:SWFLoader id="swf" width="100%" height="78%" />
            <s:TextInput id="pdfPath" width="100%" />
        </mx:VBox>
    </mx:HDividedBox>
</s:Application>
```

FIGURA 10: Exemplo de código MXML
Fonte: http://www.mxstudio.com.br/wp-content/uploads/2010/08/03082010pc1.png

De acordo com a Figura 10, podemos observar o que foi mencionando nos parágrafos anteriores.

2.6.1.2 ACTION SCRIPT

Com a crescente demanda de aplicativos RIA`s, e o consagrado e amplamente utilizado paradigma de programação OO, a Adobe se viu na necessidade de capacitar o Flex com uma linguagem de programação robusta para a programação da lógica do lado do cliente (*front-end*). Além de utilizar recursos

nativos da plataforma Flash[6] disponibiliza para os programadores os recursos do paradigma OO.

De acordo com SCHIMTZ (2010), a tecnologia Flash evoluiu a ponto de estar em mais de 98% dos navegadores disponíveis no mercado. O Flash além de desenhar animações, evoluiu em vários outros conceitos, tais como uma linguagem de programação base. Todas as animações dos banners flash e interação e comportamento das aplicações Flex são feitas com Action Script.

O processo de desenvolvimento dos aplicativos Flex possui um padrão inicialmente você desenha a sua aplicação, podendo utilizar o modo design. Depois de montar a sua aplicação, você pode inserir no mesmo arquivo MXML o seu código Action Script, ou criar classes separadas do arquivo MXML e invocá-las quando necessário (SCHMITZ, 2008).

[6] Para maiores informações acesse: http://www.adobe.com/br/flashplatform/

3. MODELO ARQUITETURAL PROPOSTO

Comunicação é um recurso fundamental para as aplicações na plataforma web, estudar formas de integração entre Adobe Flex e Java é efetivamente fundamental para elaborar um modelo arquitetural de desenvolvimento de software utilizando tais tecnologias.

3.1 FORMAS DE INTEGRAÇÃO DO FLEX

Como a tecnologia Flex é basicamente especialista na camada de apresentação das aplicações, os aplicativos desenvolvidos em Flex precisam se comunicar com os mais variados tipos de serviços e linguagens de programação disponíveis no mercado, para persistir e manipular dados.

Segundo (SOUZA, 2010), dentre as principais formas de comunicação disponíveis para integração com Flex, podemos destacar:

- HTTP Service: através do protocolo HTTP[7], o Flex realiza chamadas gets e posts para transferir e enviar dados para algum script ou página, onde o mapeamento e conversão de tipos é feito manualmente. Os dados podem ser transportados em três distintos formatos: XML, JSON[8] e texto comum.
- WebService: envia e recebe mensagens SOAP[9] sobre HTTP, para se comunicar com Web Services, que por sua vez permitem a integração de serviços de diferentes aplicações desenvolvidas em diversas plataformas. O mapeamento dos tipos é feita automaticamente, facilitando assim a integração e desenvolvimento.
- Remote Objects: técnica também chamada de Remoting, envia e recebe dados AMF[10] (*Action Mensage Format*). Não é um protocolo e sim um formato binário nativo para ActionScript, permitindo assim a troca de dados fortemente tipos, que favorece o desenvolvimento e aumenta a performance na comunicação.

[7] http://pt.wikipedia.org/wiki/HTTPHTTP#M.C3.A9todos
[8] http://www.json.org
[9] http://www.w3.org/TR/soap/
[10] http://help.adobe.com/pt_BR/as3/mobile/WS4bebcd66a74275c34219bc0c12431922a0c-7ffb.html

Em caso de comunicação Remoting, é necessário a existência de um Gateway para realizar toda a infraestrutura de comunicação, tanto no cliente quanto no servidor. Os Gateways também chamados de Data Services, são na verdade frameworks de integração do Flex com as mais variadas tecnologias de *back-end*, que transportam dados no formato AMF entre o cliente e servidor (ADOBE, 2011).

Com base nas informações obtidas no site da Adobe podemos destacar a existência de muitos Gateways no mercado, tanto desenvolvidos pela Adobe como de terceiros, livres de licença e com licença de uso. A Figura 11 ilustra algumas opções para Gateways e tecnologias de *back-end*.

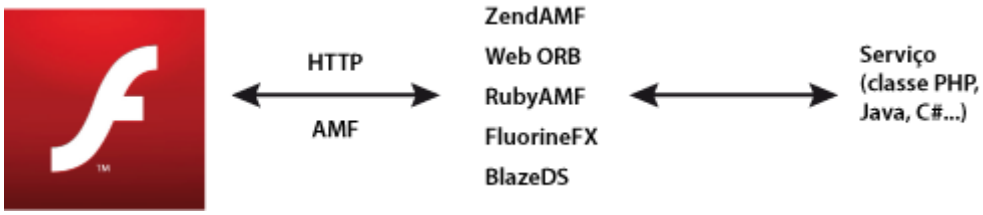

FIGURA 11 – Gateways e Plataformas de *Back-End*
Fonte: http://help.adobe.com/pt_BR/as3/mobile/WS4bebcd66a74275c34219bc0c12431922a0c-7ffb.html

3.2. INTEGRAÇÃO FLEX E JAVA

A integração é elemento fundamental para qualquer aplicativo RIA, portanto Flex e Java possuem inúmeras formas de integração, as quais foram citadas na seção anterior. Cada uma com vantagens e desvantagens uma em relação à outra, necessitando assim de uma análise de cada contexto ou domínio para a escolha de uma tecnologia de integração, que possa atender requisitos de qualidade. A Figura 12 mostra as possibilidades de integração Flex e Java.

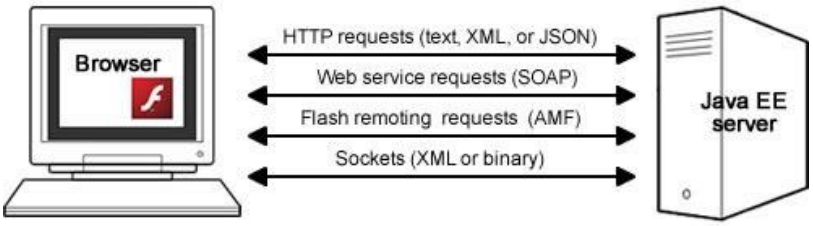

FIGURA 12 – Tecnologias para Integração Flex e Java
Fonte: http://www.adobe.com/devnet/flex/articles/flex_java_architecture.html#articlecontentAdobe_numbered header_2

O uso de AMF reduz o tamanho dos dados e melhora a velocidade de transmissão. Como o AMF é um

> formato nativo para o tempo de execução, enviar dados AMF para o tempo de execução evita serialização e deserialização com alto consumo de memória no lado do cliente. O gateway remoto lida com essas tarefas. Ao enviar um tipo de dado ActionScript para um servidor, o gateway remoto lida com a serialização no servidor. O gateway também envia o tipo de dado correspondente para você. Esse tipo de dado é uma classe criada no servidor que expõe um conjunto de métodos que podem ser chamados a partir do tempo de execução (ADOBE, 2011).

Usar AMF para integrar Java e Flex se tornou bastante interessante tendo em vista a facilidade de uso e, conforme a Adobe[11], aplicativos utilizando esta forma de integração se comunicam até dez vezes mais rápidos, do que os formatos utilizando textos como XML e SOAP. Alguns dos principais Data Services ou Gateways para integração de Flex e Java são:

- LiveCycleDS: É solução final da Adobe para Data Services, onde são disponibilizados recursos como gerenciamentos de dados, serviços de mensagens e chamadas RPC[12] (*Remote Procedure Call*) e recursos de suporte.
- BlazeDS: É um Data Service baseado em Java para comunicação Flex e Java que usa transferência de dados via AMF. Fornecido gratuitamente pela adobe. É uma alternativa livre para LiveCycleDS, mas não suporta todas as suas características especialmente a parte de gerenciamento de dados.
- WebORBJava: solução livre para a comunicação Flex e Java, possui bastante funcionalidades inclusive o gerenciamento de dados. Uma curiosidade é que WebORB[13] possui soluções Data Services para diversas tecnologias de back-end como PHP e .NET.
- GraniteDS: O Data Service Granite[14] é uma solução gratuita com bastante aceitação pelos desenvolvedores, fornecendo suporte

[11] http://opensource.adobe.com/wiki/display/blazeds/Overview
[12] http://pt.wikipedia.org/wiki/Chamada_de_procedimento_remoto
[13] http://www.dehats.com/drupal/?q=node/33
[14] http://www.graniteds.org/confluence/pages/viewpage.action?pageId=229378

para serviços de mensagens, RPC e geração de código entre outros recursos.

3.3. TECNOLOGIA DE INTEGRAÇÃO

Escolher um Gateway AMF é uma tarefa importantíssima quando falamos de aplicativos utilizando Flex e Java. Como temos inúmeras opções com características distintas, precisamos levar em consideração muitos fatores para escolher a tecnologia de integração. É necessário ter em mente os requisitos da aplicação em foco, para escolhermos a tecnologia que melhor favorece a realização dos requisitos.

Questões relacionadas como: continuidade, no caso dos Gateways livres; formas de comunicações disponíveis; compatibilidade com servidores de aplicação e facilidade de uso pelos desenvolvedores estimulam a escolha.

A Adobe, por ser a criadora do Flex possui uma vasta comunidade de usuários que utilizam seus Data Services, que os colocam em posição de destaque em relação aos demais, pelo fato de serem desenvolvidos pela mesma empresa criadora do Flex. BlazeDS e LiveCycleDS são os Data Services que a adobe mantém. Basicamente LCDS é uma solução paga e consequentemente mais completa que o BlazeDS, que é distribuído livremente pela Adobe.

A principal diferença entre os dois Data Services é a capacidade do LCDS de gerenciamento de dados que pode oferecer entre outros recursos a sincronização de dados entre cliente e servidor em tempo real segundo (KNIGHT, 2009). A Figura 13 mostra os serviços disponibilizados por cada Data Service.

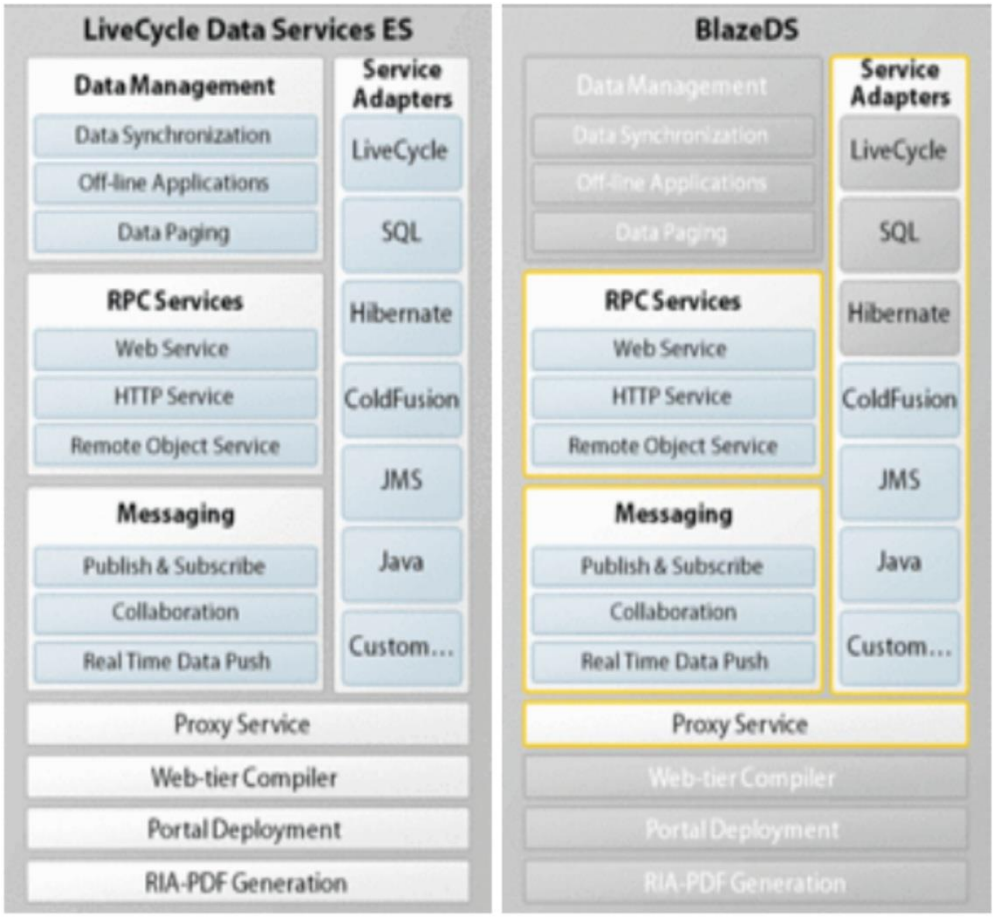

FIGURA 13 – Tecnologias para Integração Flex e Java
Fonte: http://gregsramblings.com/2008/03/27/livecycle-ds-vs-livecycle-es-clearing-up-the-confusion/

3.3.1. BLAZEDS

Como vimos anteriormente o BlazeDS é um Data Service *open source*, provido de inúmeros recursos para a troca de dados em aplicativos Flex e Java, com bastante aceitação no mercado e com uma série de recursos que destacaremos no Quadro 5 disponibilizado no site da adobe.

QUADRO 5
Características do BlazeDS

Característica	Descrição
Serviços Disponíveis	
Remoting	Remoting fornece uma chamada e um modelo de resposta para acessar dados externos a partir de aplicações Flex ou Ajax. Os desenvolvedores podem criar aplicativos que fazem solicitações assíncronas a serviços remotos, incluindo serviços de HTTP,

	serviços Web baseados em SOAP e invocação direta de objetos nativos Java.
Serviços de Proxy	O serviço de proxy ajuda a contornar as limitações do browser.
Clustering software	Suporte a mensagens de aplicações que usam clustering para garantir que as mensagens recebidas por um servidor em um cluster são encaminhadas para todos os clientes inscritos e ligados a outros servidores.
Action Message Format (AMF)	Fornece um formato binário muito compacto para serialização e desserialização de dados e invocação de método remoto.
Integração	
Arquitetura aberta para adaptação	Arquitetura aberta para se integrar com JMS, EJB, componentes ColdFusion e outras fontes de dados.
Java Message Service (JMS).	Aplicações Java podem publicar mensagens para Flex e código Java pode responder às mensagens que as aplicações Flex enviarem.
Biblioteca para Cliente Ajax.	Permite combinar Flex e Ajax. API também suporta o acesso à funcionalidade JMS para código Java.
Do lado do servidor, framework de integração de componentes.	Fornece integração de frameworks, tais como Spring ou EJB, para clientes Flex.
Mensagens	
Publicar e escrever mensagens sobre HTTP	Permite o envio de dados para serem construídos de uma maneira confiável embora preservando o modelo de implementação web.

Fonte: http://opensource.adobe.com/wiki/display/blazeds/Features

3.4. MODELO ARQUITETURAL

O modelo arquitetural elaborado com o estudo das tecnologias Flex, BlazeDS e Java tem o objetivo de possibilitar o desenvolvimento de aplicativos com boas práticas de desenvolvimento. A organização da estrutura de pacotes, tanto no *front-end* quanto no *back-end* foram desenvolvidas levando em consideração a divisão de interesses de cada componente do aplicativo, como também organização e reusabilidade. Isso possibilita aos componentes da aplicação um baixo acoplamento.

A Figura 14 mostra como é organizada a estrutura de pacotes tanto no *front-end* quanto no *back-end*.

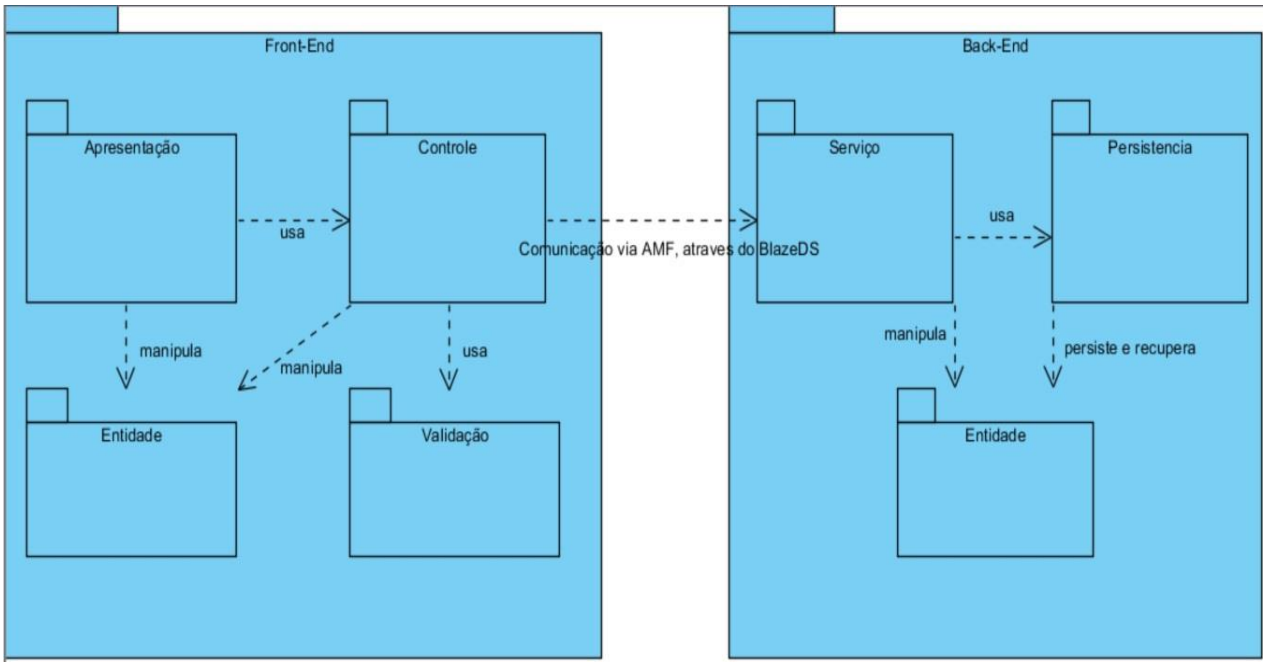

FIGURA 14 – Visão Geral da Arquitetura
Fonte: Próprio autor, 2011

3.4.1. FRONT-END

A arquitetura do *front-end* foi organizada nos seguintes pacotes:

- Apresentação: abriga todos os arquivos mxml da aplicação, componentes desse pacote abstraem o interesse de interface com o usuário, onde só sabem desenhar telas entre outras funcionalidades. Nada de lógica, seu principal objetivo é coletar e mostrar dados.
- Validação: componentes com responsabilidades de realizar os mais variados tipos de validação de dados. A validação de dados é muito importante em se tratando de comunicação cliente/servidor, pois os dados enviados para o servidor precisam estar checados com fim de evitar comunicação desnecessária. O que acarreta perda de desempenho da aplicação.
- Controle: nesse pacote fica concentrada a lógica de toda a aplicação front-end. Componentes de controle usam componentes

de validação de dados e comunicam com o *back-end* para transmitir e receber dados via Remoting.

- Entidade: encapsula toda entidade manipulada pela aplicação. Para cada entidade Java existente no *back-end* existe uma entidade Action Script espelho no front-end.

Esses pacotes são os pacotes de maior importância lógica da aplicação *front-end*, no entanto, isso não quer dizer que só existirá esses pacotes. Em aplicativos Flex, scripts, eventos e imagens são usados com frequência, organizar estes recursos em pacotes distintos melhora muito a organização da aplicação. A Figura 15 mostra a arquitetura do *front-end* com sua lógica de comunicação dos pacotes.

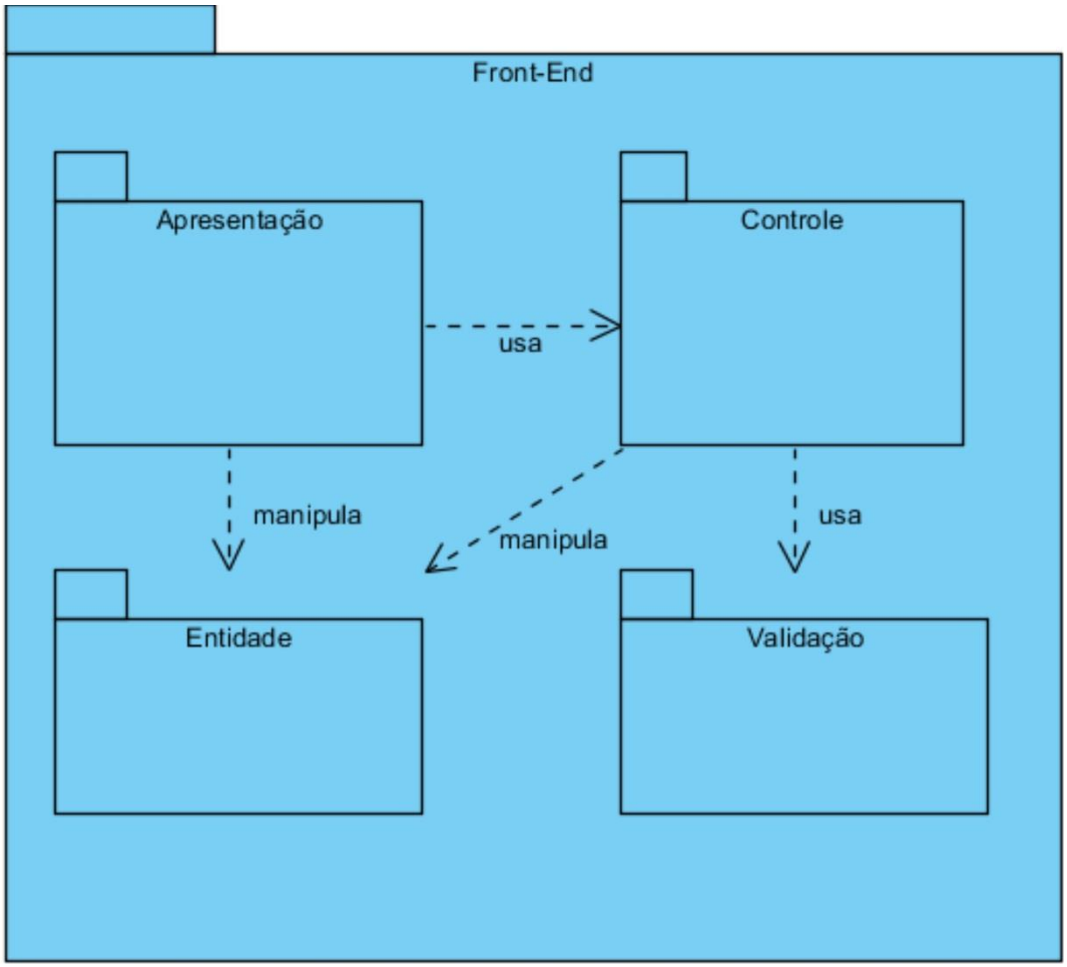

FIGURA 15 – Arquitetura do *Front-End*
Fonte: Próprio autor, 2011

3.4.2. BACK-END

A arquitetura back-end foi organizada nos seguintes pacotes:

- Serviço: encapsula toda a lógica da aplicação *back-end* e a disponibiliza para a aplicação *front-end*. É basicamente uma fachada com objetivo de fornecer um ponto de acesso para a aplicação *back-end*.
- Persistência: responsável por abrigar componentes que cuidam da lógica de persistência de dados. Esse pacote pode conter um framework de persistência objeto-relacional ou o padrão DAO[15] (*Data Acces Object*).
- Entidade: encapsula toda entidade persistente manipulada pela aplicação.

A divisão dos pacotes da aplicação *back-end* levou em consideração a separação de interesses para cada componente. A Figura 16 ilustra os pacotes da aplicação *back-end* com sua respectiva lógica de comunicação.

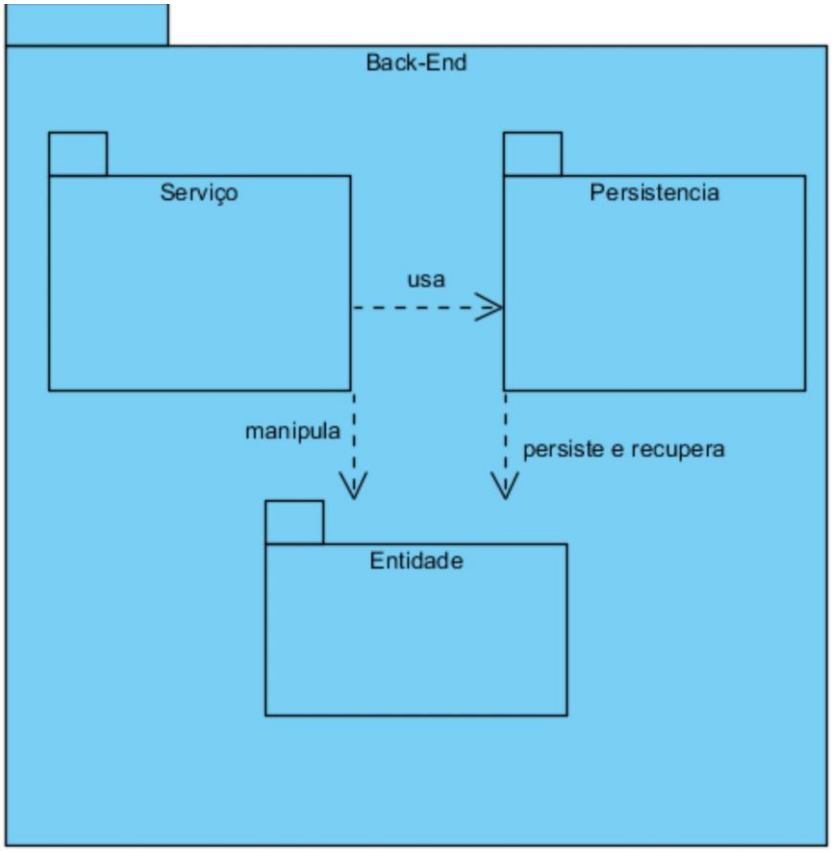

FIGURA 16 – Arquitetura do *Back-End*
Fonte: Próprio autor, 2011

[15] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

4. AVALIAÇÃO

Como resultado dos estudos realizados durante o desenvolvimento da monografia e da constante necessidade de desenvolvimento de software com qualidade em se tratando de usabilidade e reusabilidade, foi gerado um modelo arquitetural para desenvolvimento de aplicativos utilizando as tecnologias Adobe Flex e Java. Esse modelo foi também utilizado no desenvolvimento de um aplicativo para gerenciamento da produção de uma empresa do pólo calçadista de Nova Serrana Minas Gerais.

4.1. DOMÍNIO DA APLICAÇÃO

A aplicação alvo da arquitetura elaborada é um sistema de gerenciamento da produção da empresa Usigrav.cnc, que se situa na cidade de Nova Serrana e atua no setor de terceirização para a indústria calçadista.

A Usigrav.cnc é uma empresa inaugurada em março de 2005 na cidade de Nova Serrana. A empresa percebeu um amplo mercado no segmento de desenvolvimento de matrizes, clichês e desenvolvimento de arte final para a indústria calçadista, que predomina na cidade.

4.1.1. APLICAÇÃO ALVO

Com um trabalho diferenciado de seus concorrentes, em termos de qualidade e relacionamento com seus clientes a empresa viu a demanda de seus produtos aumentarem rapidamente. Para atender a demanda, sua capacidade de produção aumentou significativamente. Com o aumento da produção abriu-se um precedente, controlar de forma eficiente a produção da empresa. O Sistema de Controle da Produção está sendo proposto para além de controlar a produção da empresa, gerar informações relevantes para melhorar a administração e traçar metas futuras, em um mercado que está cada vez mais concorrido.

A empresa possui vários processos e subprocessos internos dos quais a maioria são gerenciados manualmente através de papeis ou até mesmo da conversa informal entre os funcionários. A empresa também usa uma planilha de controle de serviços que é gerenciada pelo software Excel. Essa planilha faz o controle de todos

os serviços a serem desenvolvidos e a distribuição destes serviços para os funcionários.

Na forma que vem sendo realizadas todas as atividades dos processos de produção da empresa, percebe-se facilmente o desperdício de tempo para realizar atividades simples e uma grande dependência de seus funcionários para realizar tais atividades, fato que não é do interesse da empresa. Outro fator preocupante é a não geração de informação estratégica sobre as atividades dos processos internos, fato que é imprescindível para uma boa administração e planejamento de metas futuras.

O Principal benefício deste sistema é coordenar a forma com que os funcionários irão desempenhar as atividades do processo de produção da empresa. Se o sistema falhar ou não ser construído, a empresa perderá com despesas operacionais, tempo com pessoal, e possivelmente perderá crédito com clientes descontentes. Outro benefício do sistema será uma maior visibilidade do processo de produção da empresa.

O sistema será responsável pelo gerenciamento da produção da empresa Usigrav.cnc, que inclui: serviços solicitados pelos clientes, desenhos (artes), projetos de usinagens, usinagens, uma política de mensagens internas, funcionários, clientes, material (matéria prima), máquinas de usinagens CNC, consertos (reparos em usinagens já realizadas), uma política de geração de relatórios estáticos e dinâmicos (controle de usinagem, listas de prioridades: desenho, usinagem e consertos, lista de corte de material, entre outros).

4.1.2. PRINCIPAIS FUNCIONALIDADES

Com base no levantamento de requisitos inicial realizado na empresa foram identificadas as principais funcionalidades do sistema proposto. O Quadro 6 mostra as funções com uma breve descrição.

QUADRO 6

Principais Funcionalidades do Sistema

Função	Descrição
Gerenciar Funcionários	Controle das atividades realizadas por cada funcionário. Controle de visibilidade do sistema e operações de criação, exclusão e atualização de

	funcionários entre outros recursos.
Gerenciar Clientes	Controlar um cadastro de clientes, com operação de criação, exclusão, atualização dos dados dos clientes.
Gerenciar Serviços	Controlar todo serviço solicitado pelos clientes da empresa e manter o cadastro dos serviços disponibilizando operações de criação, exclusão e atualização.
Gerenciar Orçamentos	Cadastrar e gerenciar todo orçamento solicitado pelos clientes da empresa.
Gerenciar Mensagens	Serviço de troca de mensagens entre os funcionários da empresa.
Gerenciar reuniões	Serviço de anotações diárias para auxiliar as reuniões da empresa. As observações anotadas diariamente entram ou não em pautas de reuniões da empresa.
Gerenciar Material	Gerenciar todo material utilizado na produção dos clichês e matrizes produzidos pela empresa.
Gerenciar Pendências de Serviços	Todo serviço pode gerar uma pendência como: alterações, consertos e erros dos serviços realizados. Manter e controlar um cadastro de pendências.
Gerenciar Relatórios	Gerar uma série de relatórios dinâmicos e estáticos, que serão detalhados no documento de requisitos de software do sistema.

Fonte: Próprio autor, 2011

Com base nos requisitos descritos no quadro anterior, foi gerando um diagrama de contexto dos principais casos de uso do sistema. A Figura 17 ilustra os casos de usos.

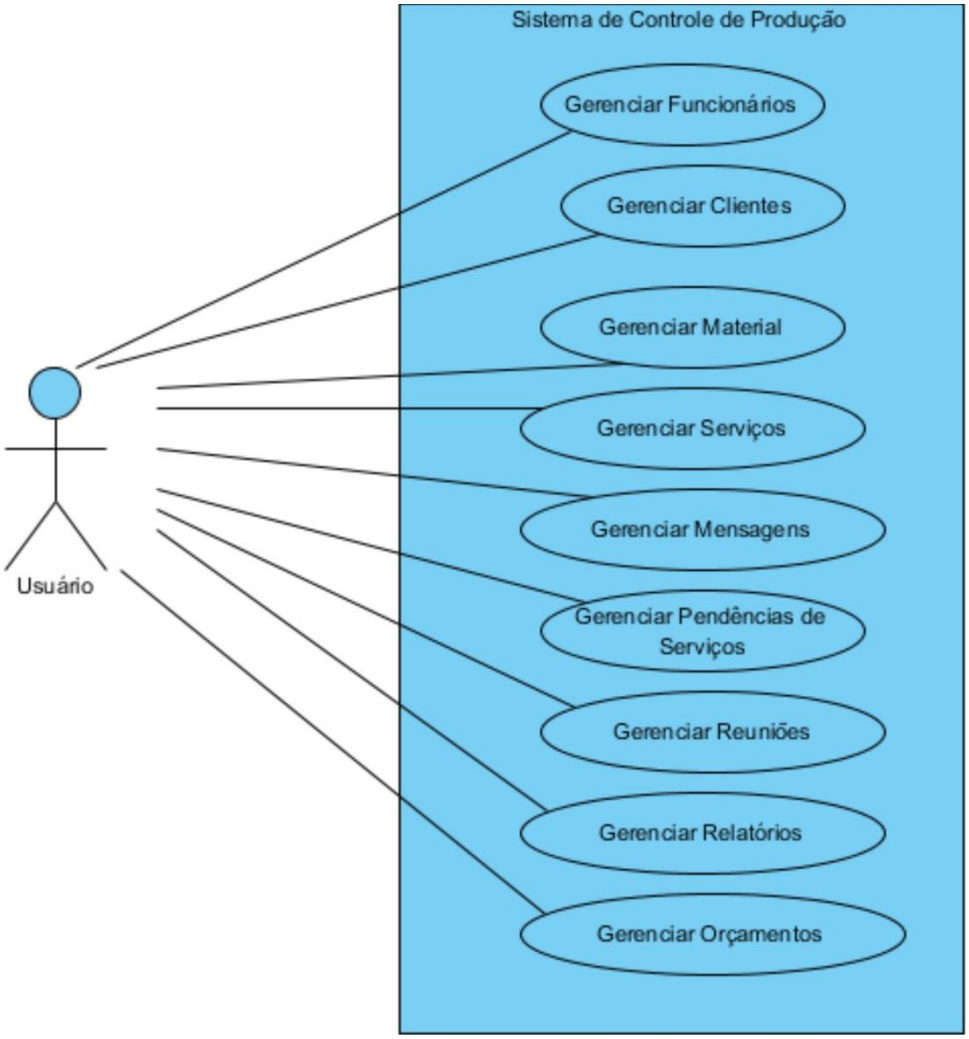

FIGURA 17 – Diagrama de Casos de Uso do Sistema
Fonte: Próprio autor, 2011

4.2. PROJETO E ARQUITETURA

O projeto a ser desenvolvido terá a plataforma web como seu ambiente, pelo fato dessa arquitetura gerar inúmeros benefícios tais como a não necessidade de instalação do sistema nos computadores dos usuários, onde somente será necessário um navegador web para rodar a aplicação que ficará em um computador servidor. Esse modelo favorece a manutenção do sistema e a disponibilidade do mesmo para os usuários.

No desenvolvimento web as tecnologias Flex e Java se destacam pelos inúmeros benefícios estudados no processo de desenvolvimento dessa monografia. A plataforma Java além de fornecer um ambiente completo de desenvolvimento, conta com uma linguagem robusta que utiliza amplamente os conceitos de orientação a objetos, oferece também um vasto conjunto de frameworks dos mais

variados propósitos para auxiliar o desenvolvimento de sistemas. Isso a torna uma plataforma bastante utilizada e difundida pelos desenvolvedores de software, que favoreceu a sua escolha como plataforma servidora para a aplicação a ser desenvolvida.

Em se tratando de desenvolvimento de software para rodar no navegador web a tecnologia Flex é conhecida como uma ferramenta poderosa para desenvolver interfaces de qualidade. Além de disponibilizar inúmeros componentes para facilitar a usabilidade de software, conta com uma poderosa linguagem orientada a objetos, disponibiliza também uma infraestrutura de integração sólida e diversas opções para comunicação da aplicação cliente com a aplicação servidora. O Flex usa amplamente o conceito de RIA, que pode entre outros recursos reduzir significativamente o número de sincronizações entre cliente e servidor e aumentar a interatividade da aplicação cliente.

Entre os principais benefícios da utilização de aplicativos utilizando a tecnologia RIA para interação com usuários podemos destacar:

- Riqueza das interfaces, que podem contar com recursos dos mais variados tipos como, por exemplo, arrastar e soltar entre outros, dando assim um visual diferenciado aos aplicativos;
- Melhor resposta, a interface é mais reativa às interações dos usuários por não necessitar submeter solicitações a cada interação;
- Equilíbrio entre cliente e servidor, a divisão do processamento entre cliente e servidor favorece a escalabilidade do servidor, que pode responder mais solicitações de mais clientes;
- Comunicação assíncrona, permite ao cliente não esperar por uma resposta do servidor, pois não existe sincronismo nas mensagens entre cliente e servidor;
- Otimização da rede, o cliente é dotado de inteligência capaz de decidir quais dados serão enviados para o servidor, diminuindo o tráfego da rede.

Esses recursos fazem do Flex a tecnologia escolhida para o desenvolvimento da aplicação cliente.

4.2.1. IMPLEMENTANDO A ARQUITETURA

Com o objetivo de validar a arquitetura proposta em termos de organização e comunicação de seus pacotes foi feito um aplicativo de teste na forma de um CRUD simples para aplicar a arquitetura. O CRUD manipula uma entidade funcionário, tanto na *back-end* quanto no *front-end* da aplicação de exemplo.

Na aplicação *back-end* podemos destacar a classe Funcionário Servico, que é encarregado de disponibilizar os métodos remotos a serem acessados pela aplicação *front-end* e de comunicar-se com a classe FucionarioDAO para persistir e recuperar os dados dos funcionários. A Figura 18 mostra os métodos da classe FuncionarioServico.

```
@Service("funcionarioService")
@RemotingDestination(channels={"my-amf"})
public class FuncionarioServico {

        FuncionarioDao funcionarioDao = new FuncionarioDao();

        @RemotingInclude
        public Funcionario save(Funcionario funcionario) throws DaoException  {
                return funcionarioDao.save(funcionario);
        }

        @RemotingInclude
        public Funcionario update(Funcionario funcionario) throws DaoException {
                return funcionarioDao.update(funcionario);
        }

   @RemotingInclude
        public void remove(Funcionario funcionario) throws DaoException {
        funcionarioDao.remove(funcionario);
        }

   @RemotingInclude
        public List<Funcionario> loadAll() throws DaoException {
        return funcionarioDao.loadAll();
   }

   @RemotingInclude
        public Funcionario loadById(Funcionario funcionario) throws       DaoException {
                return funcionarioDao.loadById(funcionario);
        }
}
```

FIGURA 18 – Classe FuncionarioServico da Aplicação *Back-End*
Fonte: Próprio autor, 2011

A Figura 19 mostra uma visão geral da aplicação back-end na perspectiva de comunicação das classes e seus respectivos pacotes e a Figura 20 mostra o projeto da aplicação *back-end* feita no Eclipse com seus pacotes e classes.

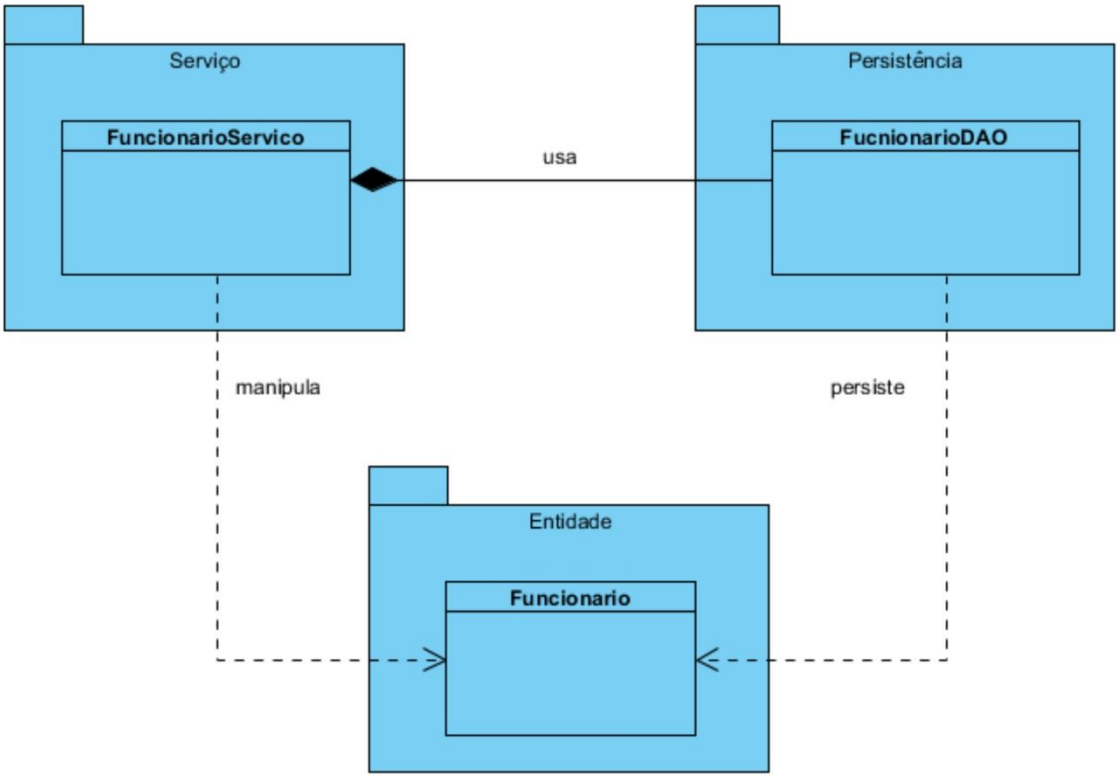

FIGURA 19 – Comunicação das Classes com seus Respectivos Pacotes
Fonte: Próprio autor, 2011

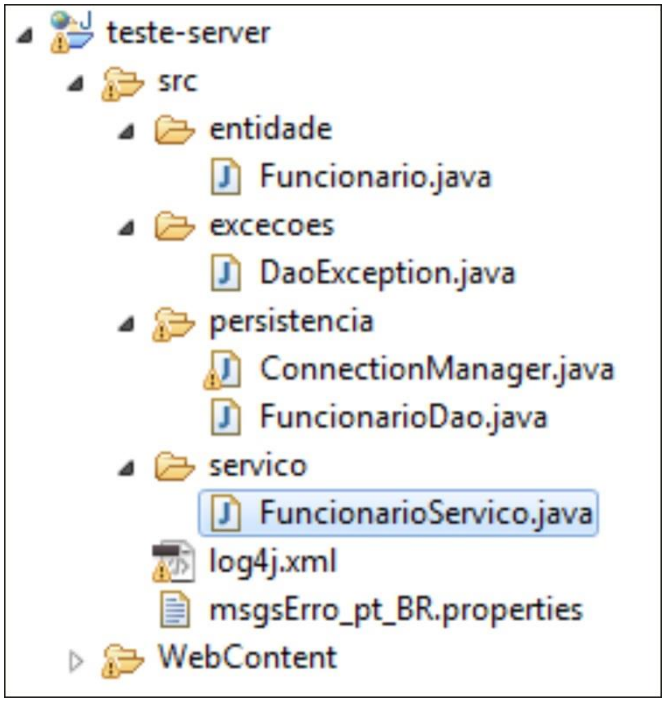

FIGURA 20 – Classes e Pacotes da Aplicação *Back-End*
Fonte: Próprio autor, 2011

Na aplicação *front-end* foi montada a arquitetura levando em consideração a divisão de interesses de cada componente para favorecer o desacoplamento dos componentes da aplicação.

Os principais pacotes da aplicação *front-end* são:

- Apresentação: responsável pelos arquivos MXML da aplicação que são na verdade as telas que serão exibidas para os usuários com objetivo de coletar e mostrar os dados manipulados pela aplicação;
- Controle: as classes deste pacote têm um papel fundamental na arquitetura da aplicação *front-end*, pois usam as classes de validação para realizar validações de dados e fazem a comunicação com a aplicação *back-end*. É a ponte entre os arquivos MXML com a aplicação servidora;
- Validação: os componentes deste pacote abstraem o interesse de validação de dados, para garantir que os dados enviados para a aplicação back-end estejam validados. O principal objetivo dessas classes é garantir a transferência eficiente de dados entre cliente e servidor para que não haja comunicação inválida, otimizando a comunicação entre o cliente e o servidor;
- Entidade: responsável por abrigar as classes que representam os tipos de dados definidos no domínio da aplicação. Um detalhe importante deste pacote é que ele é gerado automaticamente pelo BlazeDS. Para toda entidade Java persistente no *back-end* existe uma classe Action Script espelho no *front-end*;
- Serviço: pacote que abstrai toda a infraestrutura de comunicação da aplicação *front-end* com a aplicação *back-end*. Todos os métodos disponíveis para acesso remoto são disponibilizados nas classes deste pacote. As classes são geradas automaticamente pelo *framework* de integração.

Complementando a organização arquitetural da aplicação *front-end*, podemos destacar algumas regras básicas que se aplicam à mesma:

- Para todo arquivo MXML existe uma classe controladora para gerenciar suas solicitações.

- Para toda entidade manipulada pelo sistema existe uma classe que garante a validade dos dados da mesma.
- Para toda entidade existe uma classe de serviço que garante acesso remoto às operações básicas de alteração, inclusão, exclusão, visualização e operações mais complexas que a entidade deve possuir.

A Figura 21 mostra a organização dos principais pacotes, com suas respectivas classes e relacionamentos.

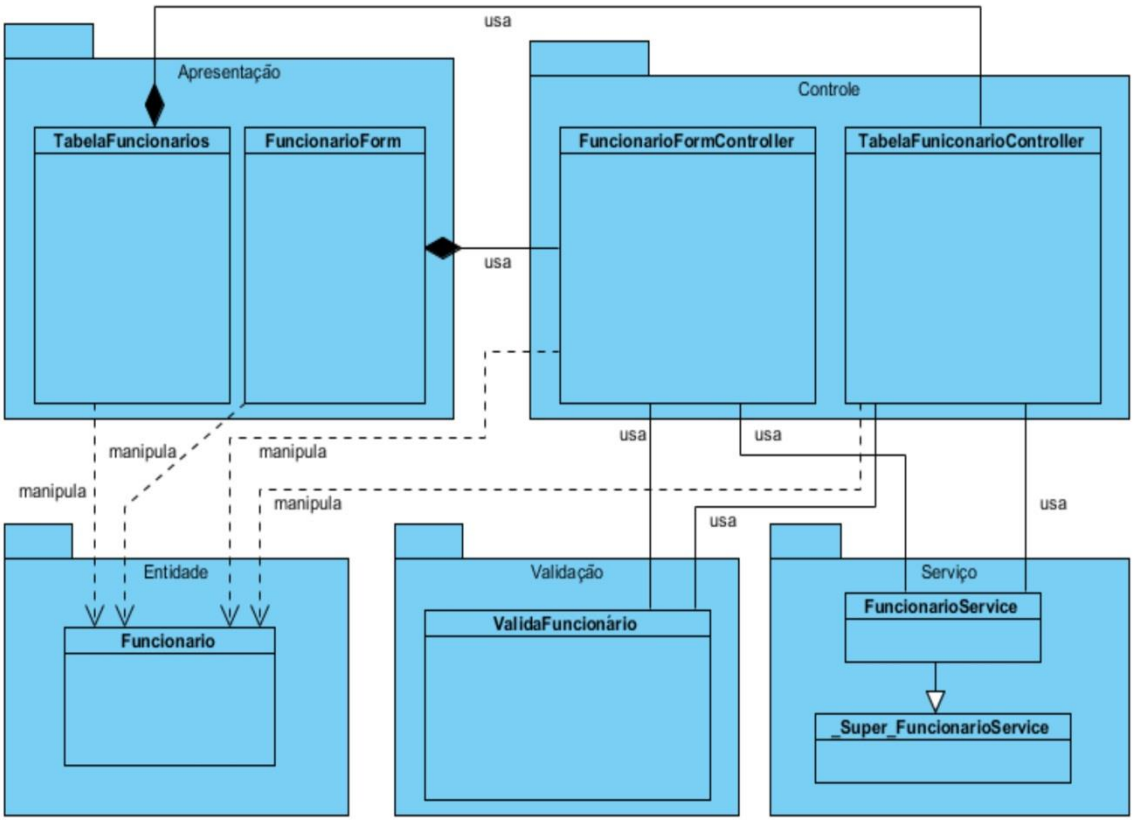

FIGURA 21 – Classes e Pacotes da Aplicação *Front-End*
Fonte: Próprio autor, 2011

A Figura 21 ilustra claramente a visibilidade de cada pacote onde podemos destacar os componentes do pacote apresentação que visualizam componentes do pacote controle e entidade. Os componentes de controle que visualizam componentes dos pacotes entidade, validação e serviço.

Os componentes dos pacotes entidade e validação não acessam nenhum componente. Já o pacote serviço é o limite da aplicação *fronte-end*, tendo acesso remoto aos métodos da classe FuncionarioServico da aplicação *back-end*.

A Figura 22 mostra o projeta da aplicação *front-end* com seus pacotes e classes.

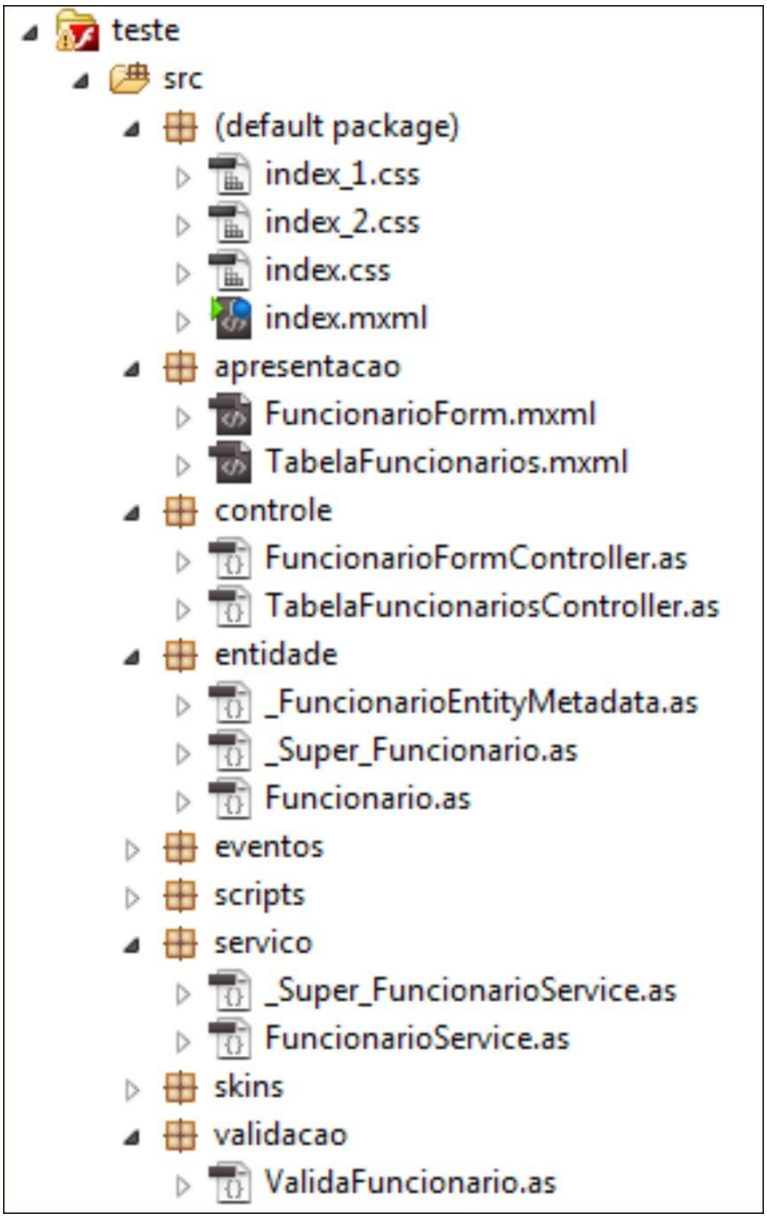

FIGURA 22 – Classes e Pacotes da Aplicação *Front-End*
Fonte: Próprio autor, 2011

5. CONCLUSÃO

Atualmente, observa-se uma crescente demanda dos usuários por aplicativos com alto grau de usabilidade e uma crescente onda de desenvolvimento de aplicativos que tem a Web como seu ambiente operacional. Estes fatores favoreceram o surgimento de conceitos como RIA, que vem crescendo e evoluindo consideravelmente pelo fato de disponibilizarem recursos elevados de interfaces com o usuário e muitas opções de integração com plataformas consolidadas de desenvolvimento de software, além de fornecer um mecanismo de comunicação mais eficiente entre cliente e servidor.

Devido a estes fatores podemos concluir que o Adobe Flex como ferramenta de desenvolvimento de aplicativos RIA`s oferece os recursos necessários para integração com linguagens robustas de *back-end*, possibilitando um desenvolvimento de software com menor esforço. O Flex como foi estudado oferece também recursos avançados para criar aplicações interativas e com boa usabilidade.

O estudo das formas de integração do Flex com linguagens de *back-end* possibilitou escolher formas eficientes de integração do Flex com a plataforma Java. Com isso, foi proposto um modelo arquitetural para o desenvolvimento de aplicativos utilizado as duas tecnologias.

5.1. CONTRIBUIÇÕES DA MONOGRAFIA

Esta monografia além de apresentar revisão bibliográfica de conceitos essenciais para o desenvolvimento de software apresentou um estudo de formas eficientes de integração do Flex e da plataforma Java. Esse estudo possibilitou a proposição de um modelo arquitetural que poderá ser materializado em uma empresa do polo calçadista de Nova Serrana em Minas Gerais para resolver problemas operacionais no gerenciamento de sua produção.

5.1. TRABALHOS FUTUROS

Como continuidade dos trabalhos realizados nesta monografia, implementar a arquitetura proposta em sistemas para testar sua eficiência, em

situações reais, seria uma boa opção para trabalhos complementares desta monografia.

REFERÊNCIAS

ADOBE. *Flash Remoting*. Disponível em: Adobe Flash Platform: http://help.adobe.com/pt_BR/as3/mobile/WS4bebcd66a74275c34219bc0c12431922a0c-7ffb.html. Acesso em: 10 Nov. 2011,

BRIDEE, Erko. **Web 2.0 e RIA**. Disponível em: <http://www.slideshare.net/erko/palestra-web-20-ria>. Acesso em: 27 Out. 2011.

CARVALHO, Ana Elizabete Souza de; TAVARES, Helena Cristina; CASTRO, Jaelson Brelaz. **Uma estratégia para implantação de uma gerência de requisitos visando a melhoria dos processos de software.** Centro de Informática, Universidade Federal de Pernambuco. Disponível em: <http://www.inf.puc-rio.br/wer01/Pro-Req-3.pdf>. Acesso em: 01 jul. 2011.

CEUNES – Programação III. **Apostila introdução a orientação a objetos**. UFES/CEUNES. São Mateus-ES. s/a. Disponível em: <http://www.ceunes.ufes.br/downloads/2/mariateixeira-EC.Programa%C3%A7%C3%A3o%20III.Cap%C3%ADtulo%201.2009.2.pdf >. Acesso em: 04 jun. 2010.

COURA, Michele dos Santos. **Ambiente de aprendizagem de lógica de programação**. Faculdade de Engenharia de Guaratinguetá Especialização em Informática Empresarial. Guaratinguetá – SP. 2006. Disponível em: <http://www.feg.unesp.br/ceie/Monografias-Texto/CEIE0605.pdf>. Acesso em: 13 jun. 2010.

DEITEL, H. M.; DEITEL P. J. **Java como programar**. 6 ed. São Paulo: Pearson Prentice Hall, 2005.

FLATSCHART, Fábio. **ActionScript 3.0**: interatividade e multimídia no Adobe Flash CS5. Rio de Janeiro: Brasport. 2010. Disponível em: < <http://books.google.com/books?id=hGdqdnZujdwC&printsec=frontcover&dq=actionscript+3.0&hl=ptBR&ei=vOSuTszoCYXVgQfUpbXqDw&sa=X&oi=book_result&ct=result&resnum=5&ved=0CEkQ6AEwBA#v=onepage&q&f=false>. Acesso em: 28 outubro 2011.

KNIGHT, R. **InfoQ**: *Blaze data services or livecycle data services?,* Disponível em: http://www.infoq.com/articles/Blaze-LiveCycle Acesso em 11 de Novembro de 2011

RUIZ, Evandro Eduardo Seron. **Introdução ao Java. 2008**. Disponível em: <http://dfm.ffclrp.usp.br/~evandro/ibm1030/intro_java/java_basics.html>. Acesso em: 13 jun. 2010.

SCHMITZ, Daniel. **Adobe Flex Builder 3.0: Conceitos e Exemplos.** 1° Edição. Rio de Janeiro: Brasport. 2008.

SCHMITZ, Daniel. **Dominando Flex e Java**. 2010

SINDINOVA. **Nova Serrana:** A Cidade Que Tem Sempre os Pés no Chão. Disponível em: http://www.sindinova.com.br/index.php?option=com_content&view=article&id=1&Itemid=3&lang=br. Acesso: 16 Nov, 2011.

SIMOR, Fernando W.; DORNELES, Carina F. **Um estudo de caso para análise comparativa entre programação Estruturada e Orientada a Objetos.** Instituto de Ciências Exatas e Geociências - Universidade de Passo Fundo. Passo Fundo,RS. 2010. Disponível em: <http://www.upf.br/computacao/images/stories/TCs/arquivos_20091/Fernando_Simor.pdf>. Acesso em: 04 jun. 2010.

SOMMERVILLE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Pearson. 2004.

SOMMERVILLE, Ian. **Engenharia de Software.** 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

SOUZA, P. (10 de 12 de 2010). *Arquitetura Básica de Rich Internet Applications com Adobe Flex.* Disponível em: http://rectius.com.br/blog/?p=39 Acesso em 10 de Nov de 2011

VASCONCELOS, Alexandre Marcos Lins de. et al. **Introdução a engenharia de software e a qualidade de software.** Universidade Federal de Lavras, Lavras - MG. 2006. Disponível em: <ttp://www.facape.br/jocelio/es/apostilas/Mod.01.MPS_Engenharia&QualidadeSoftware_ V. 28.09.06.pdf>. Acesso em: 02 set. 2011.

TELES, Vinícios Manhães. **Extreme Programming:** Aprenda como encantar seus usuários desenvolvendo softwares com agilidade e alta qualidade. São Paulo: Novatec, 2006.

WIKIPEDIA. **Rich Internet Application**. Disponível em: <http://en.wikipedia.org/wiki/Rich_Internet_Application>. Acesso em: 15 de Outubro 2011.

WIKIPÉDIA.. **Internet Rica.** Disponível em: http://pt.wikipedia.org/wiki/Internet_rica#Benef.C3.ADcios Acesso em 18 Nov, 2011