UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CORNÉLIO PROCÓPIO

ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

WILLIAM MARTINS

ADOBE FLEX COMO CAMADA DE APRESENTAÇÃO EM APLICAÇÕES *JAVA*

CORNÉLIO PROCÓPIO

2011

WILLIAM MARTINS

ADOBE FLEX COMO CAMADA DE APRESENTAÇÃO EM APLICAÇÕES *JAVA*

Trabalho acadêmico apresentado para conclusão de Curso de Especialização em *Java*.
Universidade Tecnológica Federal do Paraná Campus Cornélio Procópio.

Orientador: Prof. Dr. Elias Canhadas Genvigir.

CORNÉLIO PROCÓPIO

2011

WILLIAM MARTINS

ADOBE FLEX COMO CAMADA DE APRESENTAÇÃO EM APLICAÇÕES *JAVA*

Monografia apresentada ao Curso de Especialização em *Java*, da Universidade Tecnológica Federal do Paraná, como requisito para obtenção do título de Especialista.

COMISSÃO EXAMINADORA

Cornélio Procópio, 07 de julho de 2011.

Aos meus pais e minha família.

AGRADECIMENTOS

Á Deus por conceder oportunidades na minha vida.

Aos meus pais pelo apoio e incentivo.

Aos professores e colaboradores da UTFPR.

Aos meus amigos.

Aos novos companheiros que tive a oportunidade de conhecer no decorrer do curso.

À minha namorada pela paciência e compreensão.

"A simplicidade é o último grau de sofisticação."

Leonardo da Vinci

RESUMO

O presente trabalho aborda sobre a integração da tecnologia *RIA (Rich Internet Application)*, *Adobe Flex* com *Java* e o uso da ferramenta *BlazeDS* para fazer o processo de troca de informações entre essas duas tecnologias. O termo *RIA* é usado para aplicações que oferecem aos usuários certas facilidades como o uso de aplicações *desktop* em um navegador *web*. O *Adobe Flex* é uma tecnologia *RIA* que oferece componentes prontos que agilizam o desenvolvimento das aplicações contribuindo com o aumento de produtividade, onde estes componentes podem ser editados conforme a necessidade do desenvolvedor que pode usar a linguagem *Java* no seu *back end*. A tecnologia *Java* é uma linguagem muito usada por possuir portabilidade, é usada em grandes projetos de desenvolvimento, tendo dentro de sua tecnologia plataformas diferentes que podem atender de pequenos projetos como projetos muito grandes para grandes empresas.

Palavras Chave: *Adobe Flex, Java, BlazeDS.*

ABSTRACT

This current work on the integration of technology RIA (Rich Internet Application), Adobe Flex with Java and the BlazeDS tool use to make the process of exchanging information between these two technologies. The RIA term is used for applications that provide users with certain features such as the use of desktop applications in a web browser. Adobe Flex is a RIA technology that offers ready-made components that streamline the development of applications contributing to the increase of productivity where these components can be edited as needed by the developer who can use the Java language in its back end. The Java language is widely used for having portability is used in large development projects, and within its different technology platforms that can handle small projects and very large projects for big companies.

Key Words: Adobe Flex, Java, BlazeDS.

LISTA DE SIGLAS

CFCs (Cold Fusion Class)

CSS (Cascating Style Sheets)

ECMA-262 (European Computer Manufacturers Association)

ECMAScript (Linguagem baseada em scripts)

HTML (Hyper Text Markup Language)

HTTP (HyperText Transfer Protocol)

IDE (Integrated Development Environment)

JEE (Java Platform Enterprise Edition)

JME (Java Platform Micro Edition)

JMS (Java Message Service)

JSE (Java Platform Standard Edition)

JVM (Java Virtual Machine)

MVC (Model-View-Controller)

MXML (Macromedia Extentible Mark-up Language)

RIA (Rich Internet Application)

RPC (Remote Procedural Call)

SDK (Software Development Kit)

SWF (Shockwave Flash)

WAR (Web ARchive)

XML (Extensible Markup Language)

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O *Adobe Flex* é uma tecnologia voltada para a área de design com ferramentas que podem ser usadas, em conjunto, para facilitar o desenvolvimento e auxiliar tanto programadores como *designers*. Possui uma *IDE* (*Integrated Development Environment*) para o desenvolvimento de interfaces e inserção de códigos. Esta tecnologia possui duas linguagens próprias, o *MXML* (*Macromedia Extentible Mark-up Language*) e o *ActionScript*, e ainda pode trabalhar com outras linguagens como *Java*.

Java é uma linguagem muito usada hoje principalmente para aplicações que exigem maior robustez. Também é usada em aplicações cliente-servidor, a aplicação cliente possui apenas a interface do programa que transfere todo o processo para o servidor. Para ser usado o *Java,* em conjunto com o *Flex,* é necessário o uso do BlazeDS.

O *BlazeDS* é uma ferramenta desenvolvida pela *Adobe*, que oferece ao usuário uma fácil integração entre o *Java* e o *Flex* realizando a troca de informações entre elas.

Estas tecnologias, usadas em conjunto, podem se tornar uma ferramenta que pode ser usada em empresas a fim de suprir as necessidades de desenvolvimento de software dentro dos padrões da *RIA.*

O trabalho está organizado em capítulos; o Capítulo 2 é mostrado o que é *RIA*, abordando os principais assuntos sobre este termo, sua arquitetura e o desenvolvimento usando a tecnologia *Flex*. No capítulo 3 é apresentada a tecnologia *Flex*, sua funcionalidade, as principais características, as linguagens que são usadas por esta tecnologias e as principais ferramentas que podem ser usadas para facilitar o trabalho com ela.

No capítulo 4 é apontada a ferramenta usada para integrar a aplicação *Flex* com *Java*, o *BlazeDS*. No capítulo 5, uma breve descrição do que é *Java* e sua plataforma JEE (*Java Enterprise Edition*).

No capítulo 6 a integração do *Flex* com *Java*, os procedimentos necessários para que uma aplicação *Flex* funcione corretamente com *Java* e o resultado final obtido deste estudo.

1.1 Objetivos

Como objetivo geral este trabalho está focado no estudo da tecnologia *Flex* aplicada a *Java*, integrando as duas tecnologias usando a ferramenta de serviço de dados *BlazeDS*.

Como objetivo específico visa-se:

- Estudar o conceito de *Rich Internet Application (RIA).*
- Estudar e aplicar o *Adobe Flex* em um sistema usando a ferramenta *BlazeDS* e *Java* .
- Aplicar as funções do *BlazeDS* em uma aplicação *Java* com uma camada de apresentação *Flex*.

1.2 Justificativa

Existem hoje muitos sistemas que oferecem ao usuário certas dificuldades no momento de sua utilização.

A necessidade de facilitar o uso dos sistemas é muito importante no desenvolvimento de *softwares* e hoje a grande novidade são as *RIAs*, as Aplicações Ricas para Internet, existem várias linguagens que oferecem estes serviços que costumam facilitar o uso dos sistemas e deixá-los mais intuitivos.

Uma das tecnologias mais usadas para o desenvolvimento dessas *RIAs* é o *Adobe Flex* que oferece ao desenvolvedor muitos componentes prontos e ajustáveis de acordo com a necessidade e com uma bela aparência, apresentando uma boa opção para a criação de interfaces ricas. Como é uma tecnologia voltada para o *design* do *software* o *Flex* possibilita o uso de linguagens que podem trabalhar em conjunto com as suas funções e processos.

2 RIA (RICH INTERNET APPLICATION / RICH INTERFACE APPLICATION)

RIA (Rich Internet Application ou Rich *Interface Application*) que em português significa Aplicação de Internet Rica, foi criada em 2001 pela *Macromedia* (FAIN et al, 2008). O termo *RIA* segundo VICTORAZZI (2007) é um termo usado pela Adobe com o significado de *Rich Internet Application*, já para a Microsoft trata este como *Rich Interative Application*. Na prática esse termo é usado para aplicações que possuem características e funcionalidades de sistemas *desktop* (FAIN et al, 2008), possuindo conteúdos com mais interatividade.

O principal objetivo da *RIA* é apresentar a flexibilidade e usabilidade de uma aplicação *desktop* executando em ambiente *web*, trazendo ao usuário interatividade para a aplicação (BISSI, 2009).

As aplicações que fazem uso da *RIA* podem oferecer uma interatividade maior as suas funções, alcançada no uso de Ajax, que usa requisições assíncronas entre uma aplicação *web* e o servidor, não é necessário o recarregamento total da página (BUZATTO, 2010).

Além destes itens já apresentados a *RIA* visa aumentar as capacidades das aplicações *web*, como melhorar itens de flexibilidade, usabilidade e eficiência de uma aplicação que tem características *desktop* para ser usada em um ambiente *web* aumentando a interatividade da aplicação com o usuário (BISSI, 2009).

Em uma aplicação *RIA* a maior parte do processamento é executado no servidor da aplicação, com o processamento do cliente sendo executado no navegador. Esse conceito muda o modo de pensar e de desenvolver aplicações para a *web*, apresentando mais interatividade e usabilidade à aplicação (LÓPEZ, 2005).

As aplicações ricas têm como características a agilidade, riqueza e interatividade, em geral, as páginas criadas com esta tecnologia não recarregam a página inteira quando o usuário exige uma resposta do sistema, assim trazem a resposta mais rápida, pois é carregada apenas a parte que interessa no momento, se comportando como uma aplicação *desktop*, além destas, as aplicações ricas também costumam apresentar conteúdos multimídias e animações. De acordo com KNIPP e VALDES (2009) estas aplicações possuem um suporte muito amplo para mídia, áudio,

vídeo entre outras opções multimídias. Tudo isso auxilia o usuário no uso do sistema, tirando a complexidade e proporcionando uma melhor experiência ao usuário no uso da aplicação (ZETIE, 2005).

As restrições apresentadas pelos navegadores *web* como também pela linguagem de marcação HTML (*Hyper Text Markup Language*) é um dos incentivos para que desenvolvedores utilizem este novo tipo de aplicação, que possui vantagens quando comparada com estas, as *RIAs* permitem melhorar o relacionamento do usuário com a aplicação, dando ao usuário o poder de recuperar informações tanto *online* como *off line* e também o uso de conteúdos multimídia, dependendo da tecnologia usada no sistema (LÓPEZ, 2005).

Segundo LÓPEZ (2005) estas aplicações ricas possuem características específicas que são:

- Quando o usuário interage com uma *RIA* ele tem a resposta imediata a sua ação, já que estas não precisam carregar a página totalmente para mostrar a resposta ao usuário, assim carregando somente a parte que interessa ao usuário no momento e não deixando a página toda branca para o carregamento total da página.
- São iniciadas através da ação do usuário quando é inserido um conteúdo na página *web*.
- Os usuários podem interagir com uma aplicação *RIA* como se fosse uma ação em um sistema *desktop*, podendo manusear um objeto no sistema, como redimensionar, animar ou até mesmo arrastar e soltar.
- Apresenta uma interface com uma aparência mais atrativa com menus mais interativos e mais sofisticados que não são usadas em páginas de internet comuns.
- A aplicação não precisa de um navegador específico para ser usada, tem compatibilidade com todos eles, a maioria das *RIAs* são desenvolvidas para serem usadas em navegador.
- Todas usam o modelo do lado do cliente, onde é armazenada a parte gráfica, maximizando a captura de dados na comunicação com o servidor.

- São trocadas informações entre a interface do cliente e o servidor, não sendo necessário recarregar toda a interface a cada troca de informação e somente o carregamento dos dados no módulo em que ele é exibido.
- É usada uma linguagem baseada em XML (*Extensible Markup Language*) na definição da interface de usuário.

Hoje existem muitas tecnologias capazes de criar aplicações *RIAs, como Silverlight, JavaFX, Flex, etc,* e todas elas possuem as características específicas citadas e essas tecnologias se fundamentam em um mesmo conjunto de princípios. Todas usam o modelo do lado do cliente, armazenam a parte gráfica, maximizando a captura de dados na comunicação com o servidor, porque apenas serão trocadas informações entre a interface do cliente e o servidor.

Segundo GUANAIS (2010) as *RIAs* possuem muitas vantagens como:

- Uso de componentes gráficos mais avançados que melhora a experiência visual.
- Uso de recursos multimídia, áudios, gráficos e vídeos.
- A maioria das tecnologias é baseada na linguagem XML tanto na interface gráfica como na transação de dados.
- Pode ser usada em diferentes servidores com linguagens, como .*NET, CORBA, JRun, Tomcat, Java*, entre outras, com maior predominância os baseados em *Java*.
- Permite manejar uma quantidade maior de informações reduzindo o número de transações através do protocolo *HTTP (HyperText Transfer Protocol)*, diminui o consumo de banda utilizada e o uso da memória do servidor onde está sendo executada.
- A aplicação só se comunica com o servidor quando necessário, quando é solicitada alguma informação pelo usuário.
- Pode ser executada em diferentes plataformas e dispositivos com configurações de hardware distintas.
- A camada de apresentação é desvinculada da camada de negócios.
- As *RIAs* podem ser executadas em dois modos, *online* e *off line*.

- Detecta a atualização dos eventos na maioria dos componentes mesmo sem ser atualizada pelo usuário no navegador *web*.

Com o uso de *RIA* é possível criar *web sites* com conteúdos bem distribuidos que usam áudio, vídeo, texto e elementos gráficos, dando maior apoio como, por exemplo, nas vendas com a facilidade do uso e experiência do usuário com a aplicação que será mais valorizada. Facilitando processos que podem ser complicados para o usuário, como cadastros, que podem ser usados para fins de compras online, deixando mais simples a comunicação com a aplicação, as vendas, arquivamento, o tempo de uso da aplicação e garantindo assim o retorno do cliente ao *site* (GUANAIS, 2010).

Esse sistema pode ser usado também para facilitar a administração de uma empresa, mostrando as informações aos colaboradores de forma mais clara, ajudando a tomar decisões mais precisas sobre um projeto ou alteração que venha a ser feita em uma empresa para melhor, pode auxiliar na produtividade, tornando-a mais efetiva, na troca de informações, tomada de decisões e nas metas a serem alcançadas, por isso muitas empresas estão adotando as aplicações *RIAs* (GUANAIS, 2010).

2.1 Arquitetura

As *RIAs* possuem uma arquitetura típica apresentada na figura 1.

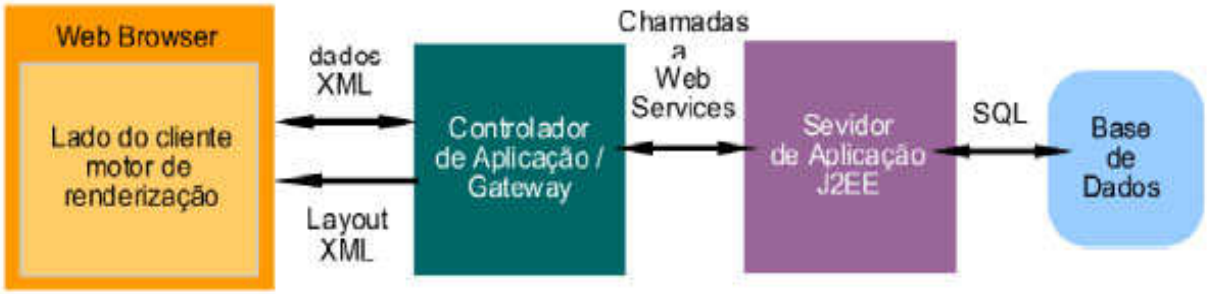

Figura 01 – Arquitetura Típica de *RIAs*.
Fonte: ORACLE, 2007.

No lado do cliente deve-se possuir um navegador que contenha *Rich Client*, que seria o motor para a renderização da interface e necessário para a interpretação

dos dados vindos do servidor. Estes dados também podem ser multimídias. Esta parte diferencia uma *RIA* de uma página normal na *web*, as quais são atualizadas constantemente a cada troca de informação com o servidor. Esta aplicação possui um *gateway* e um controlador de aplicação. O *gateway* é o encarregado de transformar os dados em *XML* que vem do servidor de aplicações, na figura está mostrando um servidor de aplicação *J2EE*, para que seja entendido pelo navegador do cliente. Já o controlador da aplicação é a parte que interage com o cliente (ORACLE, 2007).

É englobado também, pela arquitetura, um gerenciador de dados que, por sua vez, são relacionados ao serviço que a aplicação oferece. Estes dados, algumas vezes, não estão associados a informações próprias e sim a dados que estão disponíveis em serviços na internet (GUANAIS, 2010).

Na arquitetura *RIA* o lado do cliente possui toda a parte gráfica do sistema como o esquema XML ou dados binários de SWF (*Shockwave Flash*), facilitando a comunicação entre cliente e o servidor (GUANAIS, 2010).

3 *ADOBE FLEX*

O Adobe *Flex* surgiu no ano de 2004, criado pela *Macromedia*. Em menos de um ano a *Macromedia* foi comprada pela *Adobe Systems* e a *Adobe* prosseguiu com o desenvolvimento do *Flex* (GRANDBÄCK, 2009).

O Adobe *Flex* é uma ferramenta de código aberto que foi construída usando as classes *ActionScript*, que são usados na criação de interfaces executadas nas páginas *web* através do *Flash Player* e no *desktop* usando o *Runtime Air* (FRAGA, 2009).

Nos últimos anos o Adobe *Flex* ganhou muitos parceiros, como empresas que adotaram a tecnologia como plataforma de desenvolvimento (BISSI, 2009).

Pode ser visto na figura 2 como a arquitetura das tecnologias *Flash* oferecidas pela *Adobe* são divididas, cada ferramenta com um objetivo.

Figura 02 – Camadas da Arquitetura da Tecnologia *Flash*.
Fonte: ADOBE Systems Incorporated, 2009.

O *Flex* é usado para a criação de aplicativos multiplataformas para *web* e *desktop* oferecendo um modelo de linguagens que são baseadas em declarações *XML* (BISSI, 2009).

Esta tecnologia tem se destacado bastante por possuir uma *IDE* que facilita o desenvolvimento, com componentes já prontos e possibilitando ao desenvolvedor, criar ou alterar os componentes já existentes (ADOBE, Learning Resources 2007).

Com o Adobe *Flex* surgiu uma nova ferramenta, o *Flash Catalyst*, que permite a criação de interfaces gráficas, não sendo necessário o uso de códigos, esta ferramenta pode ser de grande ajuda no fluxo de trabalho entre *designers* e desenvolvedores (Adobe: Gumbo Themes, 2009).

3.1 Funcionalidades do *Flex*

O Adobe *Flex* oferece componentes prontos para uso, que facilitam o desenvolvimento das aplicações contribuindo com o aumento de produtividade. Esses componentes são separados em *layout* de container, componentes para a captura de dados, validadores, formação de dados, componentes para apresentação de dados, *data binding*, componentes .*SWC*, *data services*, gerenciamento de *drag and drop* (arrastar de soltar), suporte a imagens e vídeos, ferramenta de dicas, aparência e temas e depuração. De acordo com BISSI (2009) cada uma dessas funcionalidades possui suas características específicas:

- ***Layout de container*:** É a base de todos os containers visuais do *Flex*, ele facilita a manipulação do *layout* os deixando mais simples, o *layout* é colocado dentro de um container que dispõem o visual dos componentes na tela.
- **Componentes para a captura de dados:** Estes são responsáveis pela interação entre o usuário e o sistema, com ele o usuário ira executar ações e informar valores.
- **Validadores:** Como o próprio nome diz, é para validar os dados inseridos pelos usuários, é muito dinâmico, diferente das validações feitas em *Javascript* onde fica limitado com a compatibilidade dos navegadores.
- **Formação de dados:** São simples de manipular e possui um nível satisfatório de formatação de dados não sendo necessário o uso de *Javascript*.
- **Componentes para apresentação de dados:** Estes oferecem um local para a apresentação dos dados, podendo ser uma tabela, lista ou árvores de dados. A manipulação dos dados é simples e também simples de criar, podendo ordenar os dados em colunas.
- ***Data binding*:** Com este recurso pode-se copiar os dados de objetos que estão no lado do cliente para outros objetos automaticamente.

- **Componentes *SWC*:** É possível criar componentes que possam ser reaproveitados em outras aplicações, bastando o desenvolvedor exportá-las no formato *SWC*.
- ***Data services*:** Possibilita à comunicação através de objetos *Java*, *HTTP Post*, *HTTP Get* ou *Web Services* a interação é com objetos que estão do lado do servidor.
- **Gerenciamento de *drag and drop*:** Permite ao usuário o poder de arrastar e soltar um determinado recurso de um lugar para outro na tela.
- **Suporte a imagens e vídeo:** Possibilita a inserção de imagens e vídeos de forma simples.
- **Ferramenta de dicas:** É uma ferramenta para dicas ou ajudas para o usuário, também é chamada de *tooltips*.
- **Aparência de temas:** É um mecanismo para criação de temas ou personalização. Onde depois de ser modificado pode ser alterados em todos os componentes que pertençam aquele estilo, é um mecanismo muito pratico.
- **Depuração:** É um *debugger* que oferece a depuração de códigos no desenvolvimento das aplicações.

Com estas abordagens pode-se notar várias funções que o *Flex* oferece para o desenvolvedor, mostrando-se uma boa opção para ser usado no desenvolvimento de interfaces.

3.2 Características do *Flex*

A tecnologia *Flex* possui algumas características para o seu funcionamento como aplicação *web*. Essas características são:

- ***MVC* (*Model-View-Controller*):** É possível criar aplicações respeitando o modelo *MVC* (BISSI, 2009).

- **Eventos:** É algo que ocorre na interação do usuário ou por dados retornados ao servidor, por exemplo, quando o usuário realiza alguma ação no sistema é ocorrida uma ação. A aplicação *Flex* se comunica com uma ação realizada pelo usuário através de um evento (COLE, 2008). Os eventos mais comuns segundo BISSI (2009) são:

  Clique: Usado quando o usuário clica em algum elemento.

  Alteração: Quando um valor de algum componente é alterado.

  Criação completa: Quando um componente *Flex* é criado.

- **Assíncrono:** Esta é uma característica do *Flex*, pode ser mais bem definido como um evento que pode ser executado juntamente com outro, nenhum depende do outro para ser executado (BISSI, 2009).

3.3 Linguagens usadas pelo *Flex*

O *Flex* possui duas linguagens que são usadas no desenvolvimento de suas aplicações, o *MXML* e o *ActionScript*, a primeira é mais utilizada para o desenvolvimento da interface e a segunda para o controle dos objetos da interface.

3.3.1 MXML (Macromedia Extentible Mark-up Language)

É uma linguagem *XML* que é usada para criar o layout de uma aplicação. Simplifica o desenvolvimento no *SDK (Software Development Kit) Flex* com *tags* que correspondem às classes existentes para a criação do visual (VICTORAZZI, 2007).

De acordo com BISSI (2009) o *MXML* é um sistema de códigos que é baseado em *tags* que são derivadas do *XML*, com isso, ele deve obedecer algumas regras:

- Toda *tag* aberta deve ser fechada.
- Deve ser declarada a versão e o tipo do *XML* usada.
- São diferenciadas letras maiúsculas de minúsculas (*case sensitive*).
- Os valores que são os atributos devem estar entre aspas.

Numa aplicação *Flex* o *MXML* é usado na criação da interface e dos componentes da aplicação, utiliza *tags* como o *HTML* que define a interface do usuário. Pode ser muito bem explorada, por ser bastante conhecida pelos desenvolvedores *web* (BISSI, 2009).

O *MXML* é uma linguagem de marcação que permite a ferramenta um ajuste maior sobre as características de estados, efeitos e transições na interface (ADOBE: Gumbo, 2009).

Seguindo o conceito das aplicações que usam o *Flex*, o *MXML* pode ser a linguagem primaria que vem seguida do uso com o *ActionScript*. As *tags* do *MXML* são usadas diretamente com os objetos em *ActionScript* e todos os atributos que essas *tags* possuírem são remetidas nas propriedades do *ActionScript*. Tudo que pode ser criado com *MXML* no *Flex* pode também ser criado usando o *ActionScript*, porem a facilidade apresentada para o desenvolvimento de componentes usando o *MXML* é bem melhor que no *ActionScript* (BISSI, 2009).

As *tags* em *MXML* são iniciadas com <*mx*> o que se refere ao *nomespace* onde estão localizados os componentes do *Flex* (BISSI, 2009).

Abaixo um exemplo demonstrando como é usado este *nomespace* em conjunto com uma *tag* (BISSI, 2009).

```
<mx:Application xmlns:mx="http://www.adobe.com/2010/mxml">
<mx:Label text="Exemplo"/>
    <mx:Application>
```

Na figura 3 é mostrado um exemplo de um código básico usando a linguagem de marcação *MXML*.

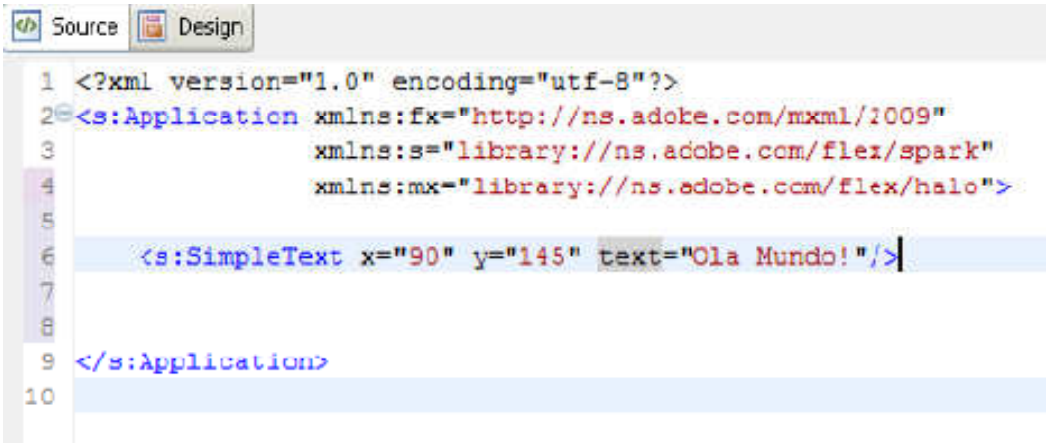

Figura 03 – Código Básico de *MXML* (Olá mundo).

Fonte: COSTA, 2009.

3.3.2 ActionScript

É similar ao *Javascript*, com base no *ECMAScript* (Linguagem baseada em scripts), é uma linguagem usada no lado do cliente para a manipulação e controle dos objetos visuais (VICTORAZZI, 2007).

O *ActionScript* segue as especificações da *ECMA-262 (European Computer Manufacturers Association)*, que permite ao desenvolvedor um maior domínio para a criação de componentes novos e novas implementações de lógica de negócio (BISSI, 2009).

Os aplicativos *Flex* são executados com o *Flash Player Runtime*, é usada a linguagem *ActionScript* que é uma linguagem orientada a objetos, além de *ActionScript*, pode ser usado o *MXML* que é uma linguagem de marcação, semelhante ao *HTML*, baseada em *XML*, á linguagem é a base para a criação de interfaces de usuário, pode ser usada também para a implementação não visual (ADOBE: Gumbo 2009).

Um dos usos do *ActionScript* é facilitar o desenvolvimento de interfaces para usuários e melhorar o suporte a ferramenta (ADOBE: Gumbo 2009).

O *ActionScript* é compilado em um formato de arquivo *SWF* que é executado pelo *Adobe Flash Player*, que exige um *plugin* para ser exibido no navegador (BISSI, 2009).

Um script dentro de um *MXML* deve ser declarado dentro de uma tag *<mx: Script>* com essa *tag* é possível realizar a manipulação de eventos, manipulação de

erros, definição de componentes personalizados, definição de métodos ou propriedades personalizadas na aplicação e a ligação de dados dos objetos para o *Flex* dentro de uma aplicação *MXML*, como é explicado abaixo de acordo com BISSI (2009):

- **Manipulação de Eventos:** A interface do *Flex* é orientada a eventos e estes eventos podem ser tratados com *ActionScript*.
- **Manipulação de Erros:** O *ActionScript* permite ao desenvolvedor a manipulação de erros ocorridos em tempo de execução, podem ser identificados os erros na validação de dados, como por exemplo, uma mensagem enviada para o usuário.
- **Definição de componentes personalizados:** O *Flex* possui alguns componentes padrões que já vem pronto e a partir dos deles é possível criar outros com novos métodos e comportamentos, podendo ser manipulados de acordo com a necessidade da aplicação.
- **Definição de métodos ou propriedades personalizadas na aplicação:** Podem ser redefinidas as propriedades já existentes ou até mesmo criar novos métodos nos componentes de aplicações, adicionando neles comportamentos que sejam necessários em novas aplicações.
- **Ligação de dados dos objetos para o *Flex* dentro de uma declaração MXML:** É muito utilizado para o preenchimento dos dados das classes dos modelos de aplicação. Nada mais é do que uma transferência das informações que foram inseridas pelo usuário nas aplicações *Flex* que estão ligadas com o *Java* ou outra tecnologia no *back-end*.

De acordo com COSTA (2009) o código de *ActionScript* é 90% parecido com a sintaxe em *Java*, como pode ser observado na figura 4.

```
1  package justjava
2  {
3      import spark.components.Panel;
4
5      public class ClasseDemo extends Panel
6      {
7
8          public var controle:Boolean = false;
9          public var titulo:String = "Just java 2009";
10         public var count:int = 0;
11         public function ClasseDemo()
12         {
13             super();
14             this.title = titulo;
15                 if(controle){
16                     this.startDrag();
17                 }
18         }
19     }
20 }
```

Figura 04 – Sintaxe de *ActionScript*.

Fonte: COSTA, 2009.

3.4 Tecnologias *Flex*

O *Adobe Flex* é um pacote que possui algumas tecnologias para a construção de aplicações voltadas para a *web* como afirma VICTORAZZI (2007), este pacote é composto por:

- *Adobe Air*
- *Adobe Flash*
- *Adobe Flash Builder*
- *Adobe Flash Catalyst*
- *Adobe Flex Charting*
- *Adobe Flex SDK*
- *Adobe Flex Data Services*

3.4.1 Adobe Air

Adobe Air significa *Adobe Integrated Runtime*, esta ferramenta torna possível a execução da aplicação *Flex* em *desktop off line*. O *Air* não é um *plugin* para o navegador, mas sim uma aplicação nativa que permite acesso aos recursos do sistema e possui um construtor de banco de dados em *SQLITE* (ADOBE: Air Faq, 2009).

Pela falta de segurança apresentada pelos navegadores e não possuir o recurso de arrastar e soltar (*drag and drop*) e por às vezes não estar conectado na internet o *Air* é a solução para aplicações *Flash* em *desktop* (BISSI, 2009).

3.4.2 *Adobe Flash*

Hoje o *Flash* é muito conhecido por ser uma das ferramentas mais usadas na *web* para apresentar dados ricos e mídias em *sites*. Foi criado em 1996 pela macromedia. É importante o destaque do *flash* porque as aplicações feitas em *Flex* são criadas em *ActionScript*, este que é compilado em *SWF* que é executado no *Flash* Player (GRANDBÄCK, 2009).

O *Flex* em comparação com o *Flash* pode ser exaltado pelo ambiente que é exibido para a criação de aplicações mais complexas, exceto para animações, que é a especialidade do *Flash*, o objetivo o qual foi construído. O *Flex* permite a criação de aplicações para o lado servidor/cliente, bando de dados e aplicações avançadas em UI e *MXML*. As aplicações em *Flex* exigem o *Adobe Flash Player 9* (ADOBE Flash, 2009).

3.4.3 Adobe Flash Builder

É um ambiente de programação, *design* e testes, com um compilador, *debugger* e pode ser usado com o *Eclipse*, é uma *IDE* para aplicações *Flex*. Ele possui editores para *ActionScript*, *CSS* (*Cascating Style Sheets*) e *MXML* (VICTORAZZI, 2007).

Este ambiente de programação pode ser visualizado na figura 5.

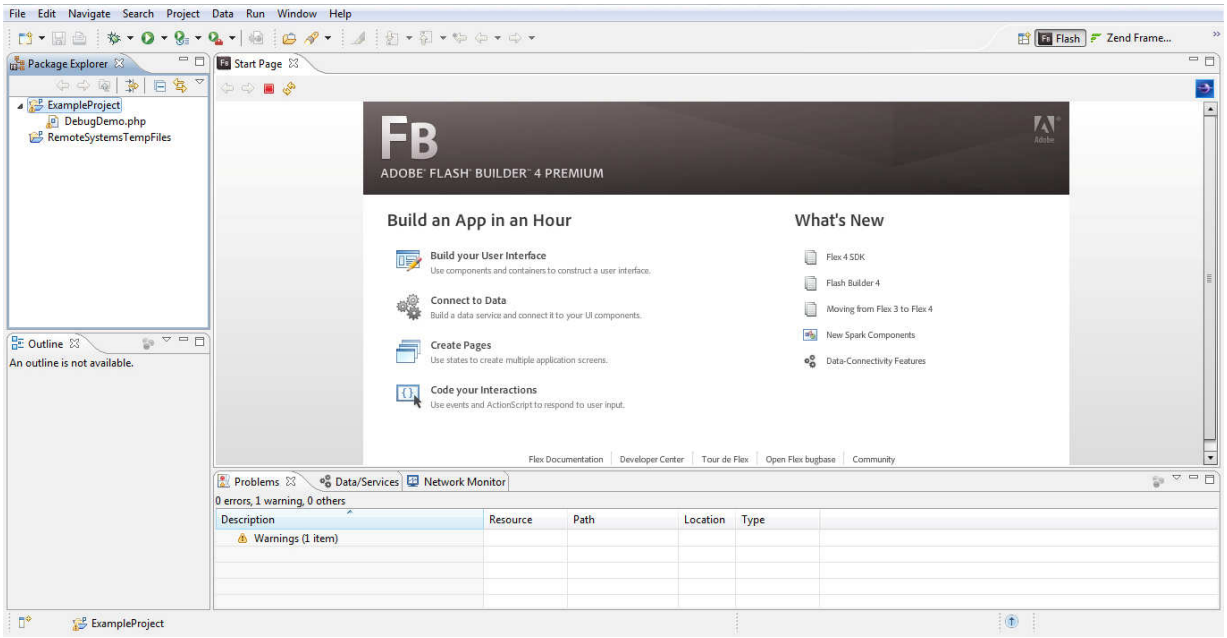

Figura 05 – Ambiente *Adobe Flash Builder*.

Adobe Flash Builder está na versão 4, é uma aplicação comercial para o desenvolvimento de *Flex* desenvolvido com o *Eclipse* que comporta uma codificação inteligente que auxilia o usuário, depuração interativa e um *design* visual de *layouts*, que acelera o desenvolvimento de aplicativos em *Flex* (BISSI, 2009). O usuário pode criar uma aplicação e continuar a mesma a qualquer momento com esta ferramenta. O *Adobe Flash Builder* permite ao desenvolvedor adicionar funcionalidades a gráficos e importar elementos do *Flash Catalyst* (ADOBE Flash Builder, 2011).

Ele apresenta um ambiente de desenvolvimento amigável no formato de arrastar e soltar os componentes mostra visualmente os componentes *Flex* que estão à disposição para o desenvolvedor, separados por abas e agrupamentos. O desenvolvedor pode optar por desenvolver visualmente e/ou usando códigos (BISSI, 2009).

Para testes o *Flash Builder* possui um monitor chamado *FlexUnit* que permite que o desenvolvedor possa visualizar os pedidos e as respostas entre o cliente e o servidor, podendo medir o desempenho ou depurar os aplicativos. O *FlexUnit* é uma ferramenta para testes de aplicativos *Flex* e *ActionScript* semelhante ao *JUnit* usado em *Java* (GRANDBÄCK, 2009).

3.4.4 Adobe Flash Catalyst

Adobe Flash Catalyst é uma ferramenta usada para o *design*, ele permite ao desenvolvedor, criar interfaces ou modelos com conteúdo interativo, sem ser necessário escrever qualquer tipo de código. Foi lançado na versão beta no ano de 2009. O principal objetivo é ser um ambiente para o desenvolvimento de *designers* e para arquitetos de *RIAs* que pode ser entregue a desenvolvedores *Flex*, que mais tarde pode inserir funcionalidades e ações a esse sistema utilizando o *Flash Builder.*

Esta ferramenta surgiu para que os *designers* possam ter mais preponderância no desenvolvimento do sistema, podendo criar modelos do sistema com maior rapidez e com maiores funcionalidades (GRANDBÄCK, 2009).

O *Flash Catalyst* oferece ao desenvolvedor um ambiente fácil para o trabalho e que facilita no seu trabalho, pois é de fácil aprendizagem, seu ambiente de trabalho é muito parecido com outros softwares utilizados por *designers*, como *Photoshop*, *Creative Suite, Ilustrator, FireWorks*, etc. Os projetos criados no *Catalyst* podem ser facilmente editados e desenvolvidos com o *Flash Builder* (ADOBE Flash Catalyst 2009).

3.4.5 Adobe Flex Charting

Usado para facilitar a visualização de uma quantidade de dados em gráficos em duas dimensões (VICTORAZZI, 2007).

3.4.6 Adobe Flex SDK

Base da tecnologia para a criação de aplicações. É um conjunto de ferramentas, compilador, *debugger*, duas linguagens de programação *MXML* e *ActionScript* e os códigos fontes das classes para o desenvolvedor (VICTORAZZI, 2007).

O *Adobe Flex SDK* é um kit para desenvolvimento em *Flex*, ele possui um compilador, ferramentas, bibliotecas de componentes e documentação que ajudam no

desenvolvimento de interfaces, além de outros utilitários que ajudam no desenvolvimento. O kit é gratuito, mas é necessário o uso de um editor para que seja ganho uma boa produtividade na tecnologia (BISSI, 2009).

Atualmente na versão 4 da *SDK*, com relação a anterior teve uma ótima melhora na produtividade, melhorias no compilador e nos recursos da linguagem, fornece ferramentas que podem ser usadas tanto por programadores como *designers* que podem colaborar entre si no desenvolvimento (BISSI, 2009).

3.4.7 Flex Data Services

Serviço usado para troca de mensagens com um servidor, podendo ser compatível com *JEE* ou com um *Servlet*. Pode receber os dados em tempo real, através de um sistema *streaming*, mantendo a comunicação entre o cliente e o servidor (VICTORAZZI, 2007).

Pode ser observado como é a arquitetura do *Flex Data Service* na figura 6.

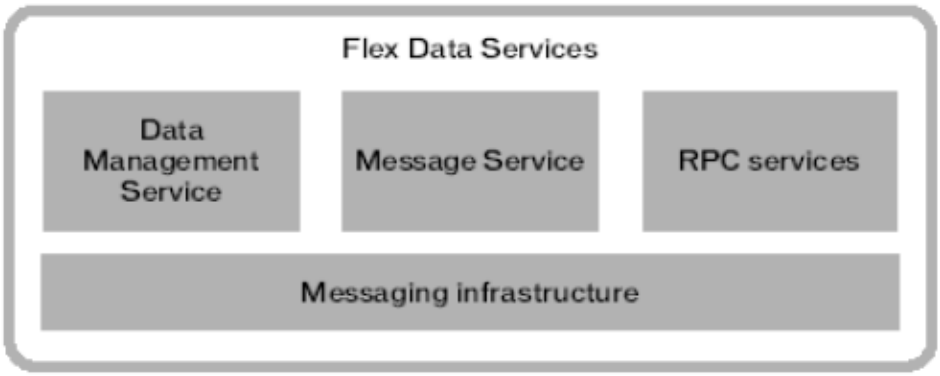

Figura 06 – Arquitetura do *Adobe Flex Data Service*.
Fonte: VICTORAZZI, 2007.

Estes são serviços para a interação das aplicações *Flex* que estão sendo usadas no cliente com a aplicação no servidor possibilitando a distribuição dos dados do servidor *web* para as aplicações cliente como também atualizá-las (BISSI, 2009).

São oferecidas três funcionalidades por estes *Data Services*, são *RPC Services Messaging Service* e *Services Adapters*. Na figura 7, pode ser observada cada característica dentro de suas funcionalidades.

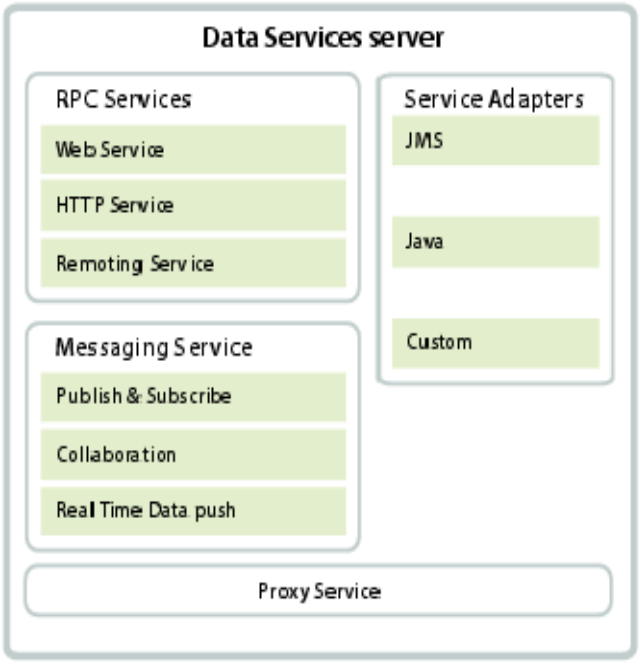

Figura 07 – Características Básicas de um *Data Services.*

Fonte: ADOBE SYSTEMS, 2008a.

O *Data Management Service* (Serviço Gerenciador de Dados) permite a redução da quantidade de códigos para o desenvolvimento de uma *RIA*. Ele é escrito na maioria das vezes usando *XML*, pode ser criada uma aplicação rapidamente e permite a cooperação entre programadores e designers. Oferece serviços como paginação sob demanda, sincronização de dados entre cliente e servidor, entre outras funcionalidades que melhoram esta comunicação (ADOBE Livecycle Data Service, 2008).

Com o *Data Management Service* pode-se gerenciar dados utilizando uma combinação com o lado do cliente com as funcionalidades do lado do servidor, para a distribuição de dados entre vários clientes. O lado do cliente possui um objeto *ActionScript* que funciona em conjunto com o lado do servidor de dados (ADOBE Livecycle Data Service, 2007).

Quando este serviço de dados é usado, as alterações nos dados no cliente *Flex* são automaticamente atualizadas, com este serviço pode ser escrito código para a sincronização de dados entre o aplicativo e o cliente *Flex* (ADOBE Livecycle Data Service, 2008).

De acordo com ADOBE LIVECYCLE DATA SERVICE (2008) este serviço possui algumas características para ajudar o desenvolvedor para que ele não insira uma grande quantidade de códigos no lado do cliente:

- Mantém todo o controle de todos os itens criados, atualizados e excluídos pelo usuário no lado do cliente.
- Mantém o controle do valor original dos dados.
- Realiza chamadas *RPC* (*Remote Procedural Call*) para alterações como criar, atualizar ou excluir.
- Controla possíveis conflitos que podem surgir no processo de atualização dos dados.

RPC Service são serviços para aplicações que realizam chamadas para acessar dados externos. Tal tecnologia permite ao aplicativo realizar solicitações assíncronas que processam requisições para retornar os dados para o lado do cliente. (ADOBE Systems, 2008a).
Para isto existem tecnologias para acessar estes dados que são:

- Serviços *HTTP GET* ou *HTTP POST*.
- Objetos *Java*.
- Serviços *SOAP* juntamente com *WebServices*.

Ainda Costa (2009) afirma que o *Flex* pode ser usado com qualquer linguagem que pode ser usada no servidor ou como são chamadas, linguagens *Server*.

O *Messaging Service* permite aos aplicativos clientes que se comuniquem de forma assíncrona com o servidor. Esse tipo de comunicação é realizado para a transferência de mensagens de um servidor interno ou externo, tem suporte a dois tipos de comunicação, sendo elas por tópicos ou filas (BISSI, 2009). Veja na figura 8 como é feita a comunicação de um cliente *Flex* com um produto *Java* usando o *Data Service* com *JMS (Java Message Service) Server*.

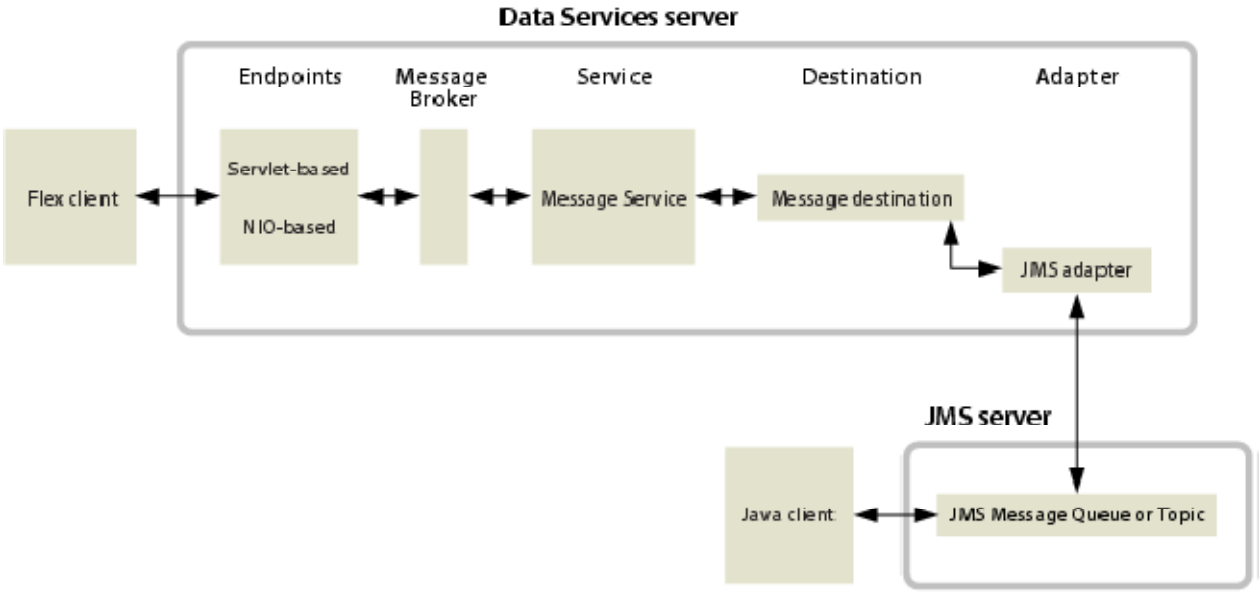

Figura 08 – Integração entre *Messaging Services*.
Fonte: Adobe Systems, 2008b.

Devido à exigência de maior complexidade no momento da criação em servidores *JMS* e por ser mais lenta o que o *RPC Service* a comunicação por serviços de mensagens é pouco utilizada (BISSI, 2009).

O *Flex* pode ser usado em conjunto com outras ferramentas que auxiliam no desenvolvimento, a *Adobe* possui uma vasta família de produtos que melhoram o processo de desenvolvimento.

Para conectar o lado do cliente ao servidor é necessário usar um *Data Service*, existem várias ferramentas capazes de realizar esta função, mas segundo BISSI (2009) as mais populares são:

- *LiveCycle Data Services.*
- *BlazeDS.*
- *Granite Data Services.*

4 BLAZEDS

Uma característica que ajuda muito no uso do *Flex* é a integração com diversas linguagens no *back-end*. No *back-end* do *Flex* pode ser usado: *Ruby on Rails*, *dotNet, PHP, Java, Cold Fusion*. Isso permite ao desenvolvedor a flexibilidade em usar a linguagem que mais lhe agrade (BISSI, 2009).

O *BlazeDS* fornece um conjunto de serviços que possibilita a conexão de um aplicativo do lado cliente com os dados do lado do servidor, podendo ser passado esses dados do servidor para vários clientes em tempo real (ADOBE Systems Incorporated, 2008).

O *BlazeDS* é baseado na tecnologia de troca de mensagens remota *Java* que permite aos desenvolvedores conectar facilmente ao *back-end*, realizando a troca de dados em tempo real com o *Adobe Flex* e aplicativos *Adobe AIR* (JORDAHL, 2007). O *BlazeDS* permite ao *Flex* usar a *tag* <*mx:RemoteObject*> para fazer chamadas *RPC* no servidor, tradução automática de valores de objetos para o *ActionScript* e os aplicativos chamam *CFCs* (*Cold Fusion Class*) em vez de classes *Java* (JORDAHL, 2007).

As aplicações *Flex* integradas ao *Java* utilizando o *BlazeDS* é possível realizar chamadas diretas aos métodos dos objetos *Java* que estão do lado do servidor. Esta tecnologia facilita a comunicação com a interface que é desenvolvida com *Flex* (ADOBE: *BlazeDS*, 2010).

As aplicações que usam o *BlazeDS* caracteristicamente são aplicações *Adobe Flex* ou *AIR*, comunicando com o servidor usando os serviços *RemoteObject, HTTPService, Webservice*, etc, que são parte do kit de desenvolvimento de software *Flex* (ADOBE *Systems Incorporated,* 2008).

O *BlazeDS* possui novas funcionalidades, como a comunicação em tempo real com o servidor, coletando os dados para serem mostrados ao usuário, levando os dados do servidor para o cliente usando o *HTTP*, tornando mais ágil a aplicação (ADOBE: *BlazeDS*, 2010).

O *BlazeDS* oferece aos desenvolvedores o serviço de *Flex Remoting* e *Messaging*. Com o *Flex Remoting* é possível obter um formato de dados binários em série que é chamada de *ActionScript Message Format* ou *AMF*, que garante um meio

mais rápido e eficiente de transporte de dados para as aplicações ricas, dando melhor desempenho nos aplicativos (BISSI, 2009).

O *Flex Remoting* ajuda os desenvolvedores com a conexão aos dados e a lógica de negócios no *back-end*. Essa e mais funcionalidades são possíveis com o *BlazeDS* que possui muitos recursos para a integração com os dados e é distribuído gratuitamente (ADOBE, 2009).

O servidor do *BlazeDS* executa uma aplicação *web* em um servidor de aplicação *JEE* com todas as suas funcionalidades, a vantagem é que ele obedece a especificação *JEE* (BISSI, 2009).

Pode observar na figura 9 a arquitetura do *BlazeDS*, que é dividida entre uma aplicação cliente e um servidor de aplicação *JEE*.

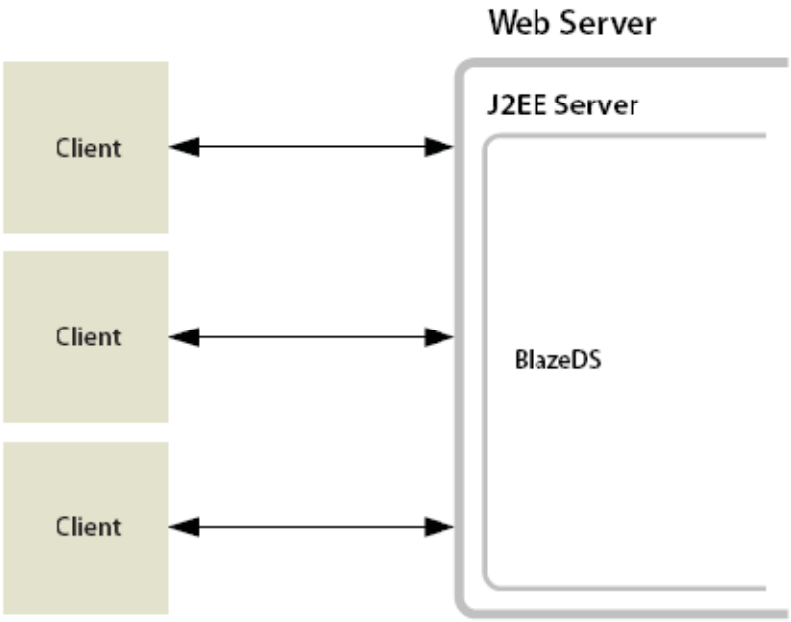

Figura 09 – Arquitetura Resumida do *BlazeDS*.
Fonte: ADOBE SYSTEMS, 2008a.

De acordo com BISSI (2009), as vantagens do *BlazeDS* são:

- Interfaces ricas para o usuário.
- *IDE* que possibilita o aumento do desenvolvimento de aplicações com maior capacidade.

- As aplicações feitas em *Flex* possuem um comportamento parecido com as em *desktop*.
- Pode se conectar com mais de um tipo de *back-end*, dando a possibilidade de o usuário escolher a linguagem que será usada no servidor.
- Linguagem orientada a objetos.
- Reduz o tráfego da rede e do servidor.
- As aplicações criadas não são modificadas com os navegadores diferentes uma vez que são executados com o *Flash Player*.
- Interage com vídeo e áudio.
- Não é baseado em *HTML*
- É totalmente baseada em padrões de arquiteturas.
- A página não é recarregada completamente quando o usuário submete a alguma funcionalidade como no *HTML*.

Segundo BISSI (2009), as desvantagens do *BlazeDS* são:

- Para ser executada é necessário o *plugin* para o *Flash Player* na máquina do cliente.
- Para executar o *Flash* sem problemas é necessária uma configuração mínima de *hardware*.
- Por usar uma linguagem não comum, o *ActionScript*, a curva de aprendizagem é maior.
- Muitas vezes o arquivo *SWF* é armazenado em *cache* no navegador, e quando é atualizada não aparece, pois é mostrado o que estava armazenado anteriormente.
- O botão voltar exibido pelo navegador só funciona se for configurado na aplicação para que tenha tal funcionalidade.

5 *JAVA*

Hoje a tecnologia *Java* é muito vasta e possui várias divisões entre as plataformas existentes para serem usadas (BISSI, 2009; GONÇALVES, 2006).

A linguagem *Java* é uma linguagem interpretada, o seu código fonte é transformado em *bycode* pelo compilador que é interpretada pela Máquina Virtual *Java* ou *JVM* (*Java Virtual Machine*) (BISSI, 2009).

Essa facilidade torna o *Java* uma linguagem com portabilidade e maior flexibilidade (GONÇALVES, 2006). Um sistema criado em *Java* pode ser executado em qualquer sistema operacional sem nenhuma mudança no código ou funcionalidade (BISSI, 2009).

A tecnologia *Java* hoje é dividida entre três plataformas ou ambientes de desenvolvimento que são o *JSE* (*Java Platform Standard Edition*), *JME* (*Java Platform Micro Edition*) e a plataforma que será usada neste trabalho, a *JEE* (*Java Platform Enterprise Edition*) (BISSI, 2009).

5.1 *JEE (Java Enterprise Edition)*

O *JEE* é voltado para aplicações corporativas *web* que necessitam de um alto desempenho que suporte uma grande quantidade de usuários acessando ao mesmo tempo (GONÇALVES, 2006). Foi criada para desenvolver aplicações para servidores, sistemas distribuídos, multicamadas entre outros, com maior apreensão na questão da segurança (BISSI, 2009). Versão designada para criação de aplicações de grande porte (ANDREAZZA, 2008).

O *JEE* na verdade são regras ou padrões que devem ser seguidos pelos programadores que desenvolverão o sistema usando esta plataforma (BISSI, 2009).

Está tecnologia possui uma arquitetura em multicamadas que permite uma grande flexibilidade no *design* onde facilita o desenvolvimento das aplicações podendo dar aos desenvolvedores e aos *designers* que trabalhem cada um com o seu papel. Está facilidade na verdade aumenta a dificuldade no desenvolvimento, no teste, na implantação e na administração (BISSI, 2009).

O *JEE* possibilita a criação e desenvolvimento de diversas tecnologias como: *JSP, EJB, JavaMail, JMS, Web Services, JSF,* etc. Como pode ser visto na figura 10.

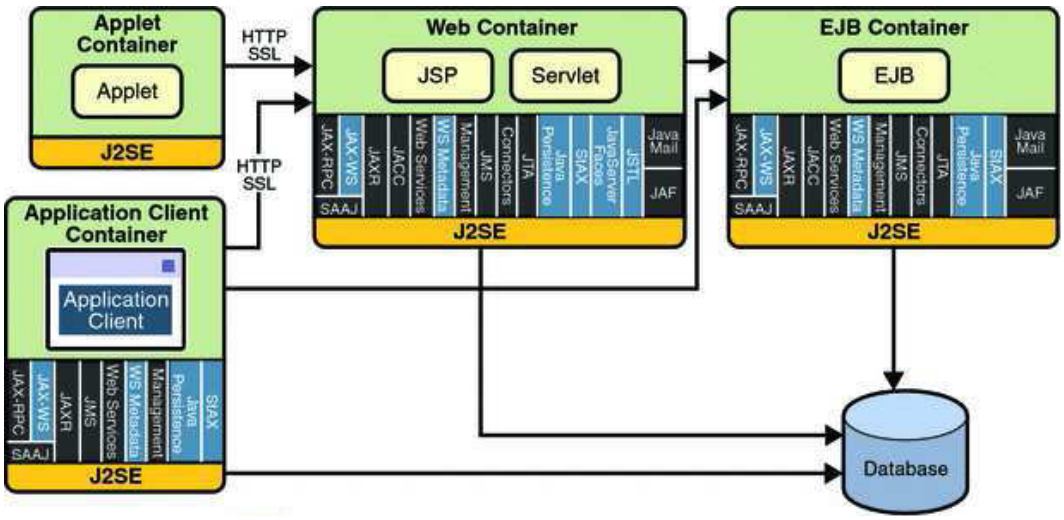

Figura 10 – arquitetura da Plataforma *JEE.*

Fonte*:* BISSI, 2009.

A plataforma *JEE* possui algumas características que são listadas por BISSI, (2009):

- Controle transacional flexível.
- Suporte para *web services*.
- Reusabilidade de recursos e componentes.
- Modelo de segurança unificado.
- Divisão da estrutura em camadas bem definidas.
- Programação orientada a componente.

O *JEE* é uma plataforma para o desenvolvimento de aplicações distribuídas. Esta tecnologia facilita o uso de recursos computacionais distribuídos, como acesso a banco de dados, componentes *web*, uso de mensagens assíncronas, etc. (TEMPLE, MELLO, CALEGARI, SCHIEZARO, 2004).

O *JEE* não apresenta muitas diferenças para desenvolvedores que estão acostumados a já utilizar outras plataformas *Java*, não apresentando muitas

dificuldades com relação à mudança de plataforma (TEMPLE, MELLO, CALEGARI, SCHIEZARO, 2004).

A arquitetura da *JEE* é apresentada em várias camadas, sendo que cada camada possui um componente que é munido em um container, em uma aplicação pode ter vários containers interagindo entre si, como pode ser visto na figura 11. (TEMPLE, MELLO, CALEGARI, SCHIEZARO, 2004).

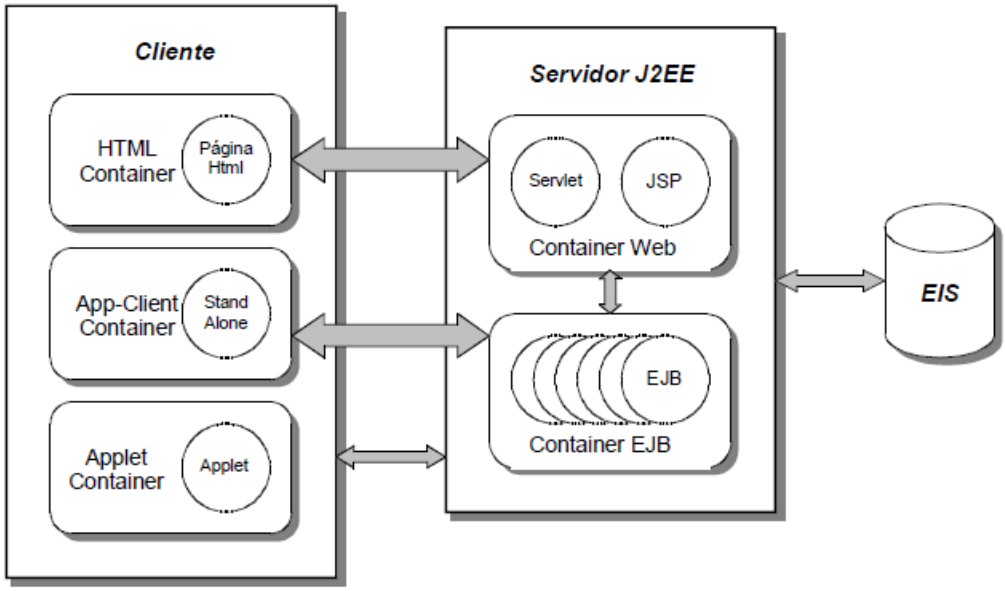

Figura 11 – Interação entre Camadas, Componentes e *Containers*.
Fonte: TEMPLE, MELLO, CALEGARI, SCHIEZARO, 2004.

6 INTEGRAÇÃO ADOBE *FLEX* E *JAVA*

Para o desenvolvimento de uma aplicação usando *Flex* com *Java* é necessário uma ferramenta que realize a troca de informações entre essas tecnologias, a ferramenta escolhida é o *BlazeDS* como serviço de dados. Esta é a ferramenta usada como ponte entre a interface criada no *Flex* e o servidor em *Java*.

6.1 Configuração do *BlazeDS no projeto Java*

O *BlazeDS* possui três tipos de arquivos para desenvolvedores os quais podem ser encontrados no *site* do fabricante, sendo um com o código fonte, pois é uma

ferramenta com código aberto, outra opção que também possui um servidor integrado para testar a ferramenta e outra opção que é o pacote que contém os arquivos *WAR* (*Web ARchive*) que são os necessários para inserir no projeto para que funcione corretamente.

Este pacote *WAR* possui os arquivos que são usados no projeto *Java* para que as classes e serviços do *Java* sejam executados na interface ou lado cliente criado com o *Flex*.

Primeiramente é necessário adicionar as bibliotecas do *BlazeDS* na biblioteca do projeto *Java*, que estão no arquivo .*WAR*, este arquivo deve ser descompactado, nele existe todos os arquivos necessários para que seja configurado a ponte de comunicação entre a aplicação *Flex* e *Java*, esses arquivos devem ser inseridos na biblioteca, os arquivos do *BlazeDS* estão na pasta *WEB-INF/lib* que foram extraídas do arquivo *WAR*, os arquivos nesta pasta são arquivos com extensão *.jar*, o resultado final deve ficar como na figura 12, os arquivos inseridos no projeto são essenciais para que a aplicação possa usar as funções que o *BlazeDS* disponibiliza.

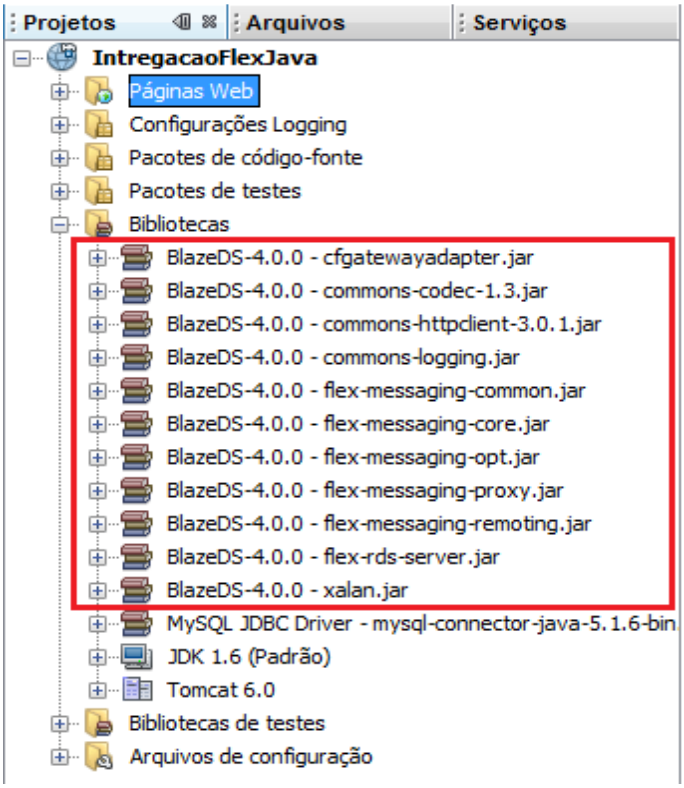

Figura 12 – Arquivos do *BlazeDS* Adicionados a Biblioteca *Java*

Com as bibliotecas *BlazeDS* já inseridas no projeto é necessário ainda inserir os arquivos de configuração que estão no arquivo .*WAR* do *BlazeDS*, que foi descompactado para a inserção da biblioteca. Estes arquivos são arquivos de configuração do *BlazeDS*.

Esses arquivos de configuração estão dentro da pasta *WEB-INF\flex* que foi descompactada no arquivo .*WAR*, estes arquivos possuem a extensão *XML*. Dentro da pasta onde estão os arquivos, todos devem ser selecionados, no projeto da aplicação *web* deve ser criado uma pasta dentro da pasta *WEB-INF*, o nome da pasta no projeto foi nomeada como *Flex*, dentro desta pasta devem ser colocados todos os arquivos copiados, como na figura 13.

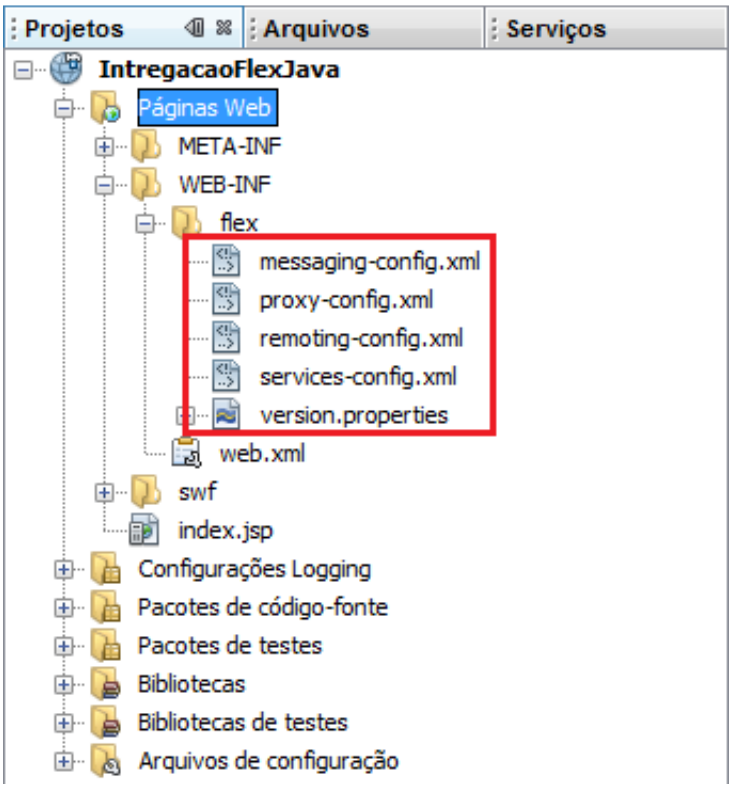

Figura 13 – Arquivos *XML* de Configuração Remota do *BlazeDS*

Esses arquivos de configuração são essenciais para que o compilador do *Flex* saiba quais os serviços estão configurados no *BlazeDS*. Esses serviços são implementados em classes *Java* e configurados nos arquivos de configuração do *BlazeDS* para que a aplicação *Flex* saiba quais serviços podem ser usados.

Estes serviços são criados no projeto *Java*, no caso deste projeto foi criado serviços para cada classe criada, como é uma aplicação de cadastro, os serviços criados são de gravar, atualizar, excluir, inserir. Pode ser visto a figura 14 os serviços criados.

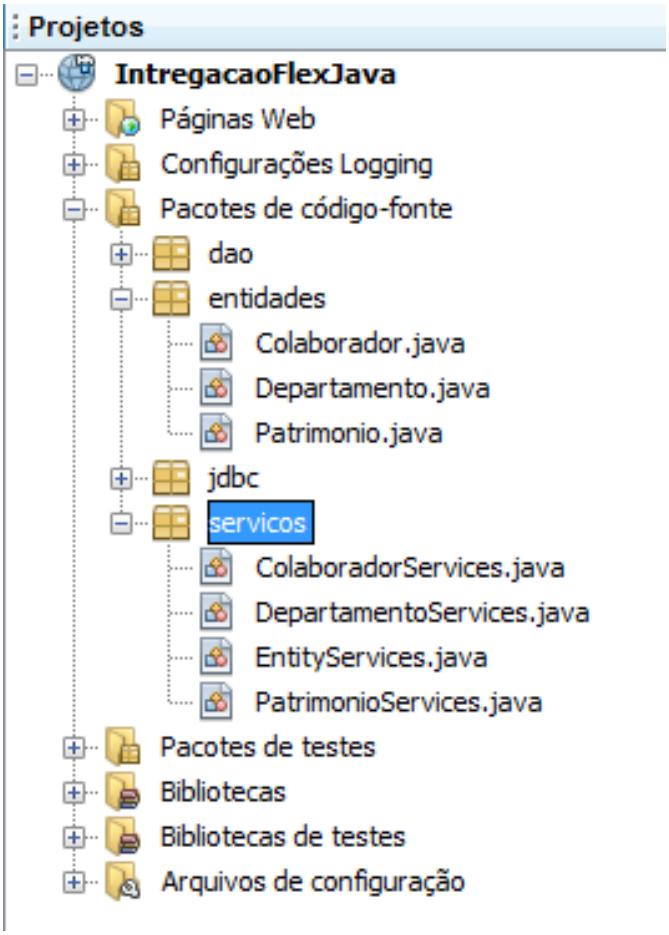

Figura 14 – Serviços.

Cada serviço como mencionado possui alguns métodos que serão usados na interface *Flex*, a figura 15 exibe o código de um modelo de serviço.

```
package servicos;

        import dao.Dao;
        import dao.DepartamentoDao;
        import entidades.Departamento;
        import java.util.List;


        public class DepartamentoServices extends EntityServices<Departamento> {

            @Override
            public void save( Departamento obj ) throws Exception {
                Dao<Departamento> dao = new DepartamentoDao();
                dao.save( obj );
                dao.closeConnection();
            }

            @Override
            public void update( Departamento obj ) throws Exception {
                Dao<Departamento> dao = new DepartamentoDao();
                dao.update( obj );
                dao.closeConnection();
            }

            @Override
            public void delete( Departamento obj ) throws Exception {
                Dao<Departamento> dao = new DepartamentoDao();
                dao.delete( obj );
                dao.closeConnection();
            }
```

Figura 15 – Modelo de Serviço Departamento.

Estes serviços devem ser configurados para que o projeto em *Flex* possa usá-lo, a configuração é feita no arquivo *remoting-config.xml* que foi adquirido no arquivo com extensão WAR, o arquivo do *BlazeDS*. O serviço deve ser declarado no arquivo *remoting-config.xml* com um identificador para que a aplicação *Flex* o use, na figura 16 está a configuração do serviço da classe Departamento, mostrando que possui um id e as propriedades onde esta localizado o serviço no projeto *Java*.

```
<destination id="departamentoServices">
    <properties>
        <source>servicos.DepartamentoServices</source>
    </properties>
</destination>
```

Figura 16 – Configuração do Serviço Departamento no *remoting-config.xml.*

Para que a comunicação entre *Flex* e *Java* seja um sucesso o arquivo *web.xml*, padrão em projetos de aplicações *web* usando *Java*, deve ser configurado com os arquivos do *BlazeDS*, para que seja concluída a configuração e que ele seja reconhecido no projeto *Java*. A figura 17 mostra como deve ficar o arquivo *web.xml.*

web.xml

Geral | Servlets | Filtros | Páginas | Referências | Segurança | XML

```
9         <!-- Http Flex Session attribute and binding listener support -->
10    <listener>
11        <listener-class>flex.messaging.HttpFlexSession</listener-class>
12    </listener>
13
14    <!-- MessageBroker Servlet -->
15    <servlet>
16        <servlet-name>MessageBrokerServlet</servlet-name>
17        <servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>
18        <init-param>
19            <param-name>services.configuration.file</param-name>
20            <param-value>/WEB-INF/flex/services-config.xml</param-value>
21       </init-param>
22        <load-on-startup>1</load-on-startup>
23    </servlet>
24
25    <servlet>
26        <servlet-name>RDSDispatchServlet</servlet-name>
27                <display-name>RDSDispatchServlet</display-name>
28        <servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>
29                <init-param>
30                        <param-name>useAppserverSecurity</param-name>
31                        <param-value>true</param-value>
32                </init-param>
33        <load-on-startup>10</load-on-startup>
34    </servlet>
35
36    <servlet-mapping id="RDS_DISPATCH_MAPPING">
37        <servlet-name>RDSDispatchServlet</servlet-name>
38        <url-pattern>/CFIDE/main/ide.cfm</url-pattern>
39    </servlet-mapping>
40
41    <servlet-mapping>
42        <servlet-name>MessageBrokerServlet</servlet-name>
43        <url-pattern>/messagebroker/*</url-pattern>
44    </servlet-mapping>
45
46    <welcome-file-list>
47        <welcome-file>index.jsp</welcome-file>
48    </welcome-file-list>
49</web-app>
50
```

Figura 17 – Mapeamento do *BlazeDS* no Arquivo *web.xml.*

O último arquivo a ser configurado do *BlazeDS* no projeto é o arquivo *commons-logging.properties*, para configurá-lo é necessário adicionar uma pasta nos diretórios de pacote do projeto *Java*, para realizar este procedimento o desenvolvedor deve ir nas propriedades do projeto e inserir uma pasta especifica para este arquivo de configuração.

Neste caso está sendo usado a *IDE Netbeans* 6.9.1, nesta é necessário clicar com o botão direito no nome do projeto e escolher a opção propriedades, dentro de propriedades na categoria código fonte, adicionar uma pasta, como na figura 18, pode ser notado que o nome que foi dado é configurações *logging*.

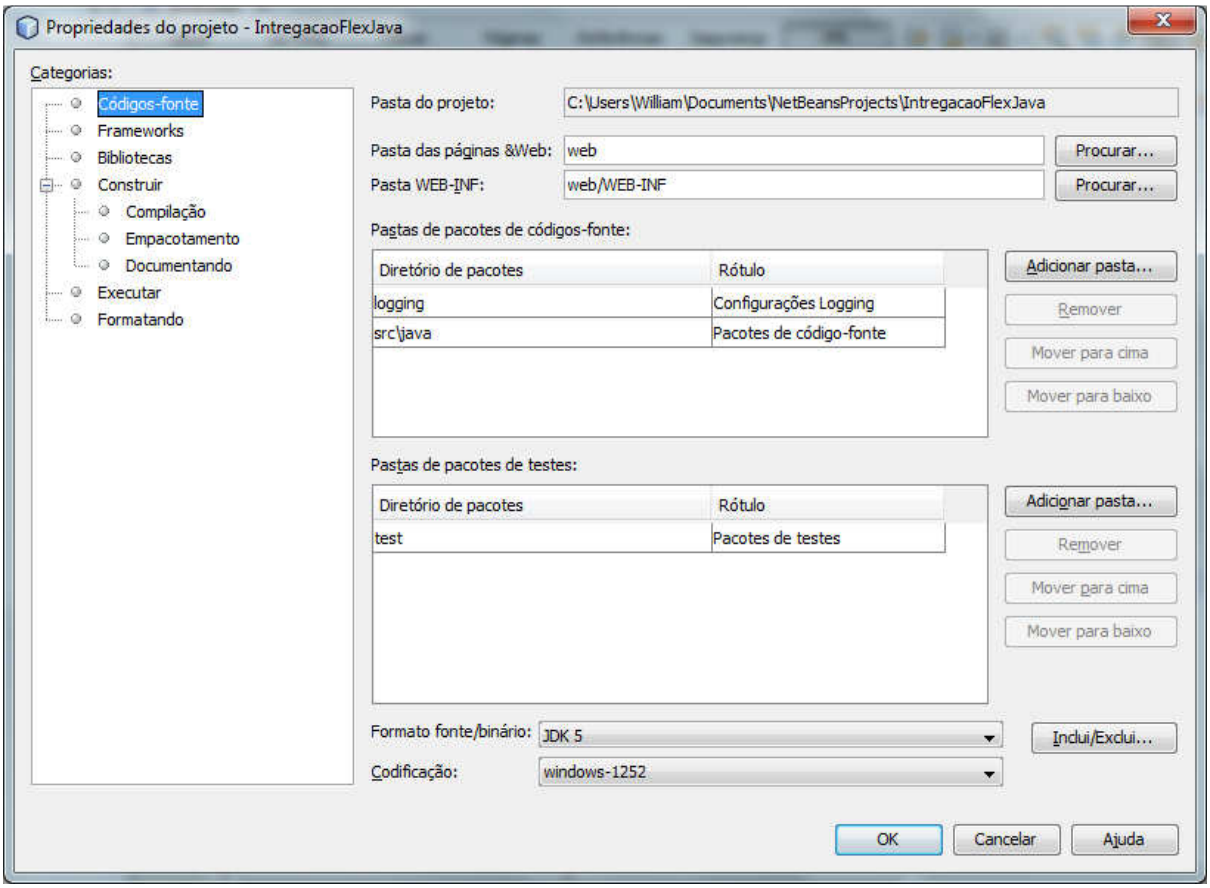

Figura 18 – Configurações *Logging*.

Após criar este pasta, deve ser colado o arquivo *commons-logging.properties* que está localizado dentro da pasta onde foi descompactado o arquivo com extensão

WAR do *BlazeDS*, está dentro da pasta *WEB-INF/.* A estrutura do projeto ficara como na figura 19.

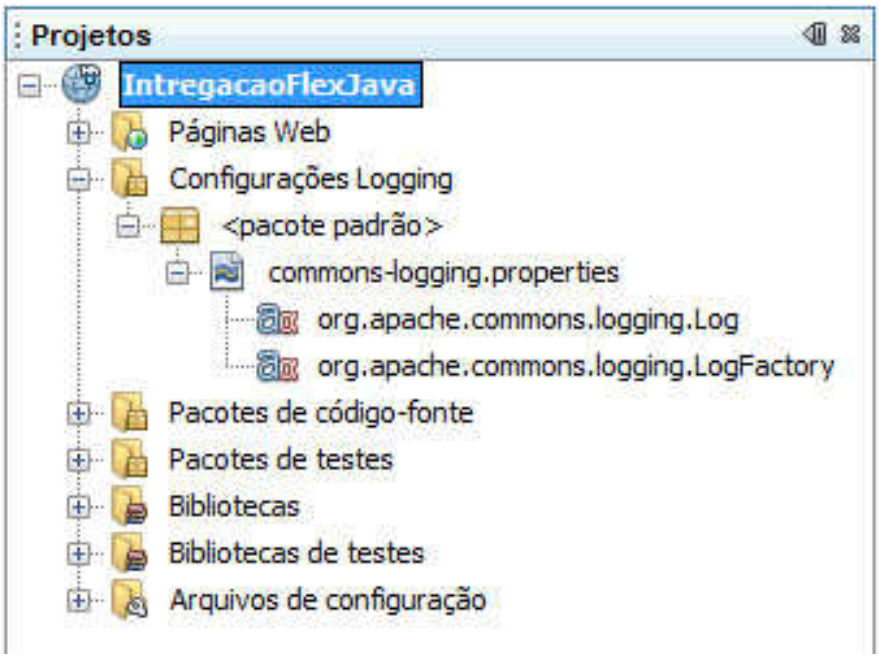

Figura 19 – Estrutura do Projeto *Java* com a Pasta *logging*.

O projeto *Java* está configurado com os arquivos necessários para o funcionamento do *BlazeDS*, o projeto criado em *Java* é um projeto que busca mostrar as funcionalidades mais simples que podem ser feitas entre uma aplicação *Java* e uma aplicação *Flex*.

6.2 Estrutura da Aplicação

A aplicação *Java* pode ser melhor visualizada na figura 20, um diagrama de classes da aplicação, um controle básico de patrimônio, onde um departamento possui colaboradores e patrimônios.

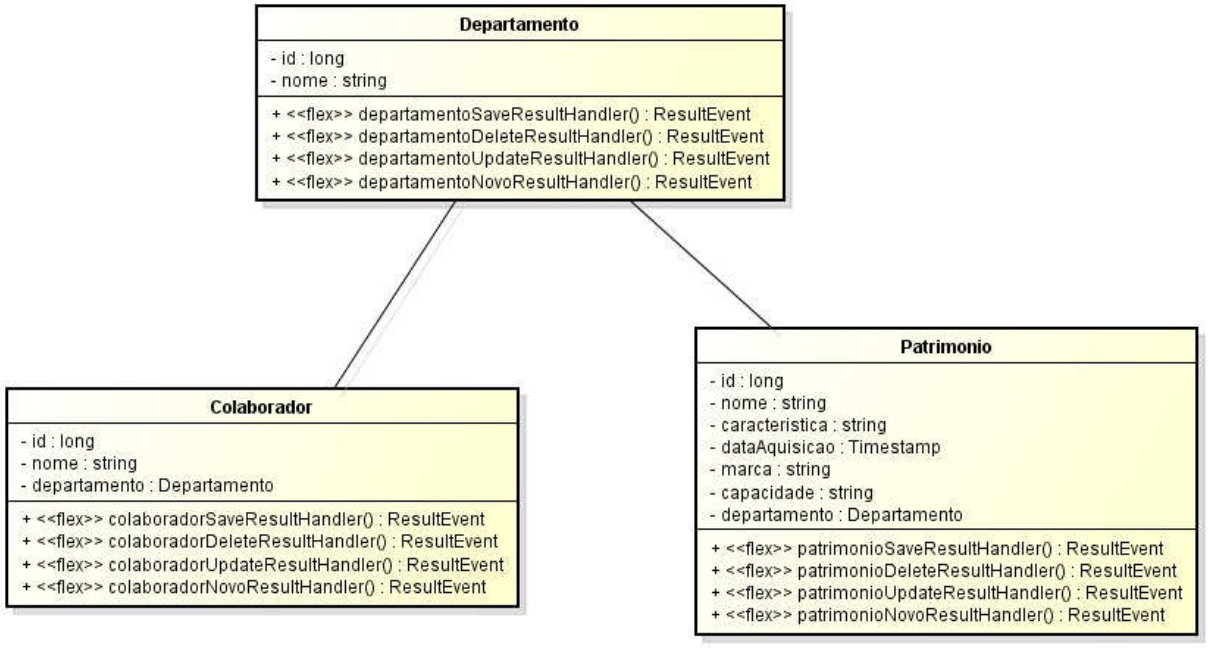

Figura 20 – Diagrama de Classes.

Foram criadas três classes em *Java* e atribuídas a cada uma seus respectivos atributos. Os métodos ou funções serão desenvolvidos no ambiente de desenvolvimento de *Flex*, que neste projeto foi será usado o *Adobe Flash Builder 4.*

Nesta ferramenta pode ser desenvolvida a interface, arrastando os componentes necessários e redimensionando e editando suas propriedades de acordo com a necessidade e com a aparência que o desenvolvedor estiver planejando utilizar. Na figura 21, pode ser observado este ambiente de desenvolvimento e a interface do projeto *Flex* que será usado na integração com *Java*.

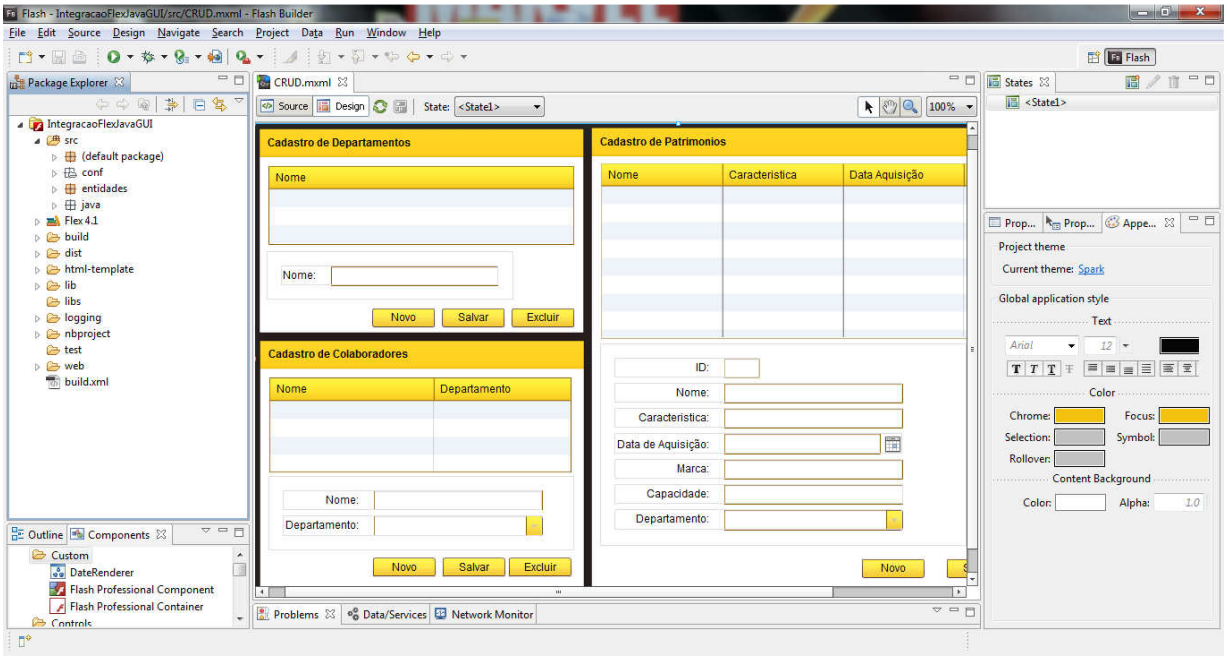

Figura 21 – Interface do Projeto, Criada no *Adobe Flash Builder 4*.

Para simplificar, o projeto terá apenas uma tela que contem o cadastro de todas as classes existentes, com todas as funções, salvar, excluir, novo. Estas telas contem todos os dados que serão inseridos e o usuário poderá manipulá-los.

Para que os projetos tanto de *Java* como em *Flex* sejam unificados, os projetos devem estar na mesma pasta *root*, ou seja, os projetos devem estar localizados no mesmo lugar de origem. No projeto *Java* para simplificar foi criado uma pasta com o nome *SWF*, onde todos os arquivos referentes ao *Adobe Flash Builder* estarão nela.

6.3 Configuração *Flex usando Adobe Flash Builder 4*

Para que um projeto *Flex* seja criado com intenção de integração com *Flex*, o desenvolvedor deve definir algumas opções especificas no momento da criação, como na figura 22 está destacado.

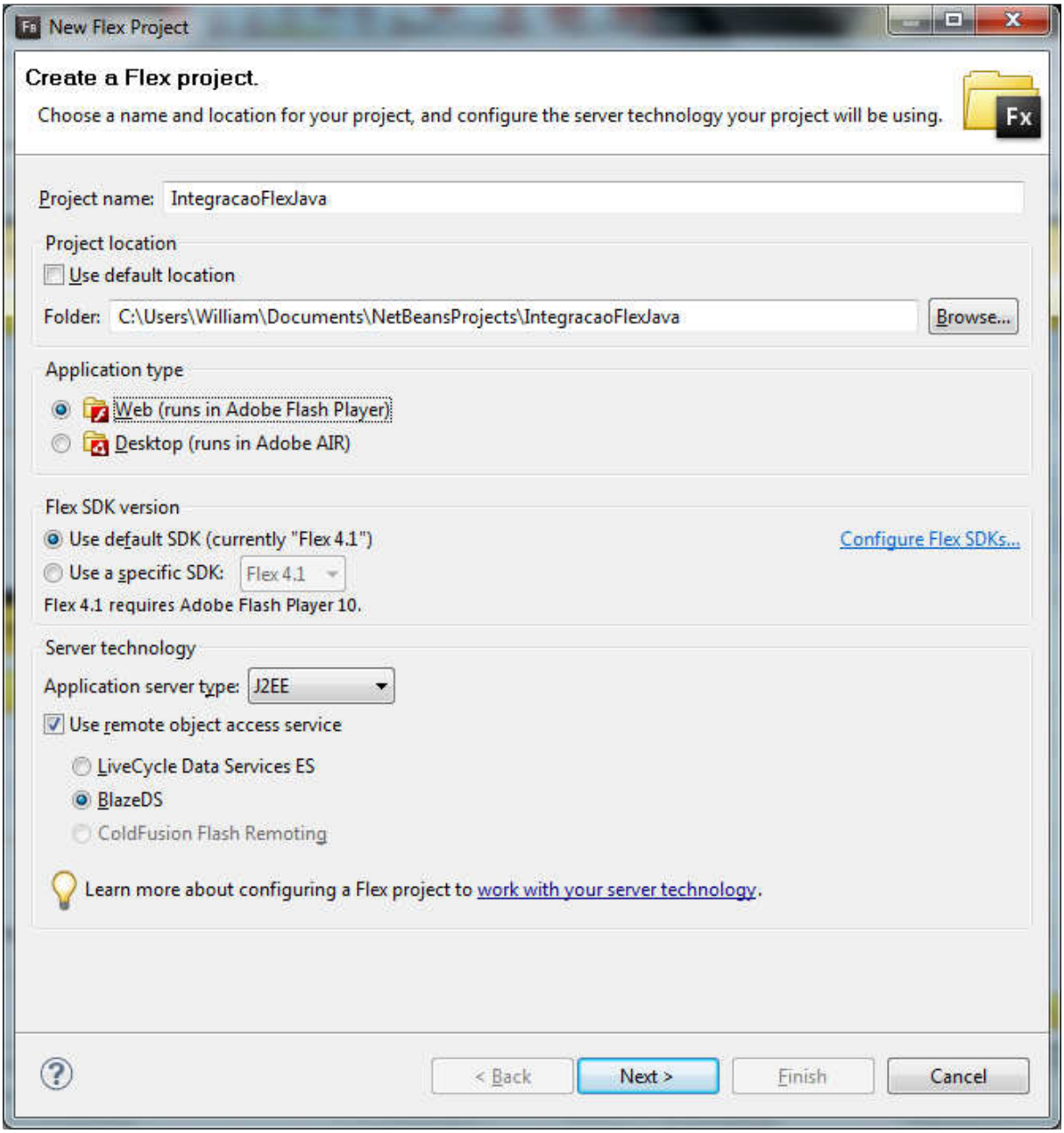

Figura 22 – Especificidades de Criação de um Projeto *Flex* com *Java* e *BlazeDS*.

Na figura 22 pode ser notado que o local onde será armazenado o projeto *Flex* é o mesmo onde está armazenado o projeto *Java*, o tipo de aplicação é *web*, o qual foi designado este projeto e o tipo de servidor pode ser visto que está selecionado a opção *J2EE*, ou seja, a plataforma *Java* que trabalha com sistemas *web*, por fim o *BlazeDS* como serviço de acesso remoto a objetos.

Para que a aplicação *Java* tenha comunicação com a aplicação em *Flex*, esta ultima deve possuir classes com a mesma estrutura das classes em *Java*. Estas classes terão um papel importante, pois são elas que irão receber a informação antes que seja passada para o servidor, ou para as classes do servidor.

Essas classes criadas em *Flex* são classes *ActionScript* que receberão a informação, onde o *BlazeDS* serializará para passar para o servidor *Java* que receberá a informação convertida para um formato que o servidor entenderá e trabalhara com estas informações e assim também (vice-versa), pois as mensagens enviadas pelo servidor *Java* também serão serializadas e passadas em um formato que o *ActionScript* e a aplicação *Flex* possam trabalhar. Pode ser visto na figura 23 a semelhança entre uma classe *Java* e uma classe criada no *Flex* com *ActionScript*.

Classe emActionScript

```
package entidades {

        [Bindable]
        [RemoteClass(alias="entidades.Departamento")]
        public class Departamento {

            private var _id: Number;
                private var _nome: String;

                public function get id(): Number {
                        return _id;
                    }

                public function set id( valor: Number ): void {
                        _id = valor;
                    }

                public function get nome(): String {
                        return _nome;
                    }

                public function set nome( valor: String ): void
                        _nome = valor;
                    }

                public function toString(): String {
                        return _nome;
                    }

            }

    }
```

Classe em Java

```
package entidades;

/**
 *
 * @author William
 */
public class Departamento {

        private Long id;
        private String nome;

        public Long getId() {
            return id;
        }

        public void setId( Long id ) {
            this.id = id;
        }

        public String getNome() {
            return nome;
        }

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


    }
```

Figura 23 – Comparação de Classe *ActionScript* e *Java*.

Na figura 24, pode ser visto que as classes possuem os mesmos atributos, na classe *ActionScript* existe um diferencial, a *tag Bindable* e *RemoteClass*.

A *tag Bindable* informa que esta classe esta ligada a classe, no caso do exemplo, Departamento esta relacionada aos componentes gráficos, todas as mudanças que ocorrer nesta classe será refletida automaticamente no componente. Já a *tag RemoteClass* que dizer que a classe selecionada é uma classe remota que os objetos dela serão enviados via *AMF*. Neste caso a alias esta com a localização da classe departamento como está no projeto *Java*, se a estrutura fosse diferente a origem também seria ou se fosse outro nome de classe, o alias apenas indica onde está localizado a classe que a classe *ActionScript* está referenciando.

Para criar a interface usando o *Adobe Flash Builder 4* é muito prático, pois é possível arrastar e soltar os componentes que serão usados. No caso deste trabalho foi criado uma interface com as três tabelas em uma tela só, nesta tela foi inserido os campos em que serão trabalhados, como pode ser visto na figura 24, onde foram usados os componentes do tipo *datagrid, form, button, textinput*, etc.

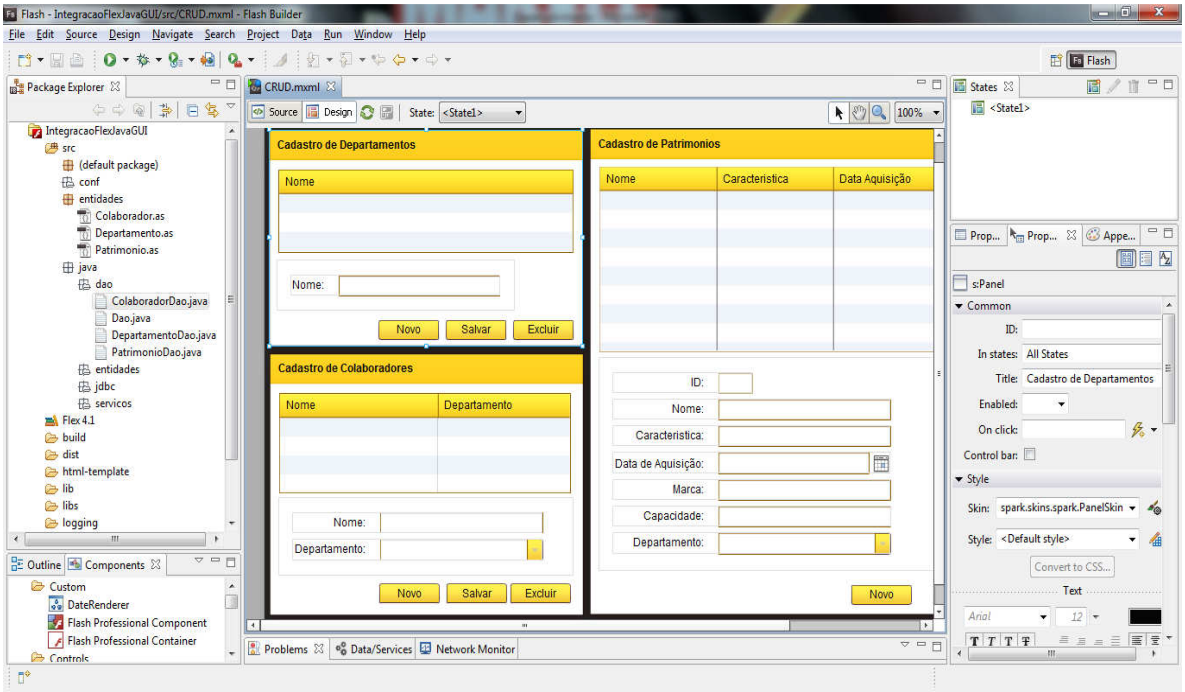

Figura 24 – Interface *Flex*.

Com a interface já criada é necessário que ela trabalhe junto com *Java*, ou seja, que tenha comunicação entre as duas linguagens. Para que isso ocorra o código deve conhecer que existe outra aplicação no qual será trabalhado junto com ela para isso é usado o código *<s:RemoteObject,* que através da comunicação com *BlazeDS* (no arquivo *remoting-config.xml*) onde foram configurados os serviços, busca no projeto em *Java* algo relacionado ou serviço no qual está invocando. A figura 25 exibe um trecho do código do projeto onde o serviço é invocado pela aplicação *Flex*.

```
<s:RemoteObject
                id="servDepartamento"
                        destination="departamentoServices"
                                        showBusyCursor="true">

                <s:method
                                name="save"
                                                fault="faultHandler(event)"
                                                                result="departamentoSaveResultHandler(event)"/>

                <s:method
                                name="update"
                                                fault="faultHandler(event)"
                                                                result="departamentoUpdateResultHandler(event)"/>

                <s:method
                                name="delete"
                                                fault="faultHandler(event)"
                                                                result="departamentoDeleteResultHandler(event)"/>

                <s:method
                                name="listAll"
                                                fault="faultHandler(event)"
                                                                result="departamentoListAllResultHandler(event)"/>

        </s:RemoteObject>
```

Figura 25 – *RemoteObject* Aplicação *Flex*.

Pode ser visualizado na figura 25 como foi feito o *remoteObject*, nele devem ser colocados os métodos presentes no serviço criado em *Java*, como pode ser visto na figura 15 na página 42, onde mostra o código do serviço em *Java*, este mostra que as mesmas funções implementadas em *Java* são inseridas nesta codificação também.

Para que os dados que estão armazenados no banco de dados sejam carregados, é necessário que uma função seja implementada para que cada vez que a

estrutura da aplicação seja executada, carregue os dados automaticamente quando é iniciada, na figura 26 é apresentada a interface executando com os dados carregados e na figura 27 o código que carrega estas informações na interface.

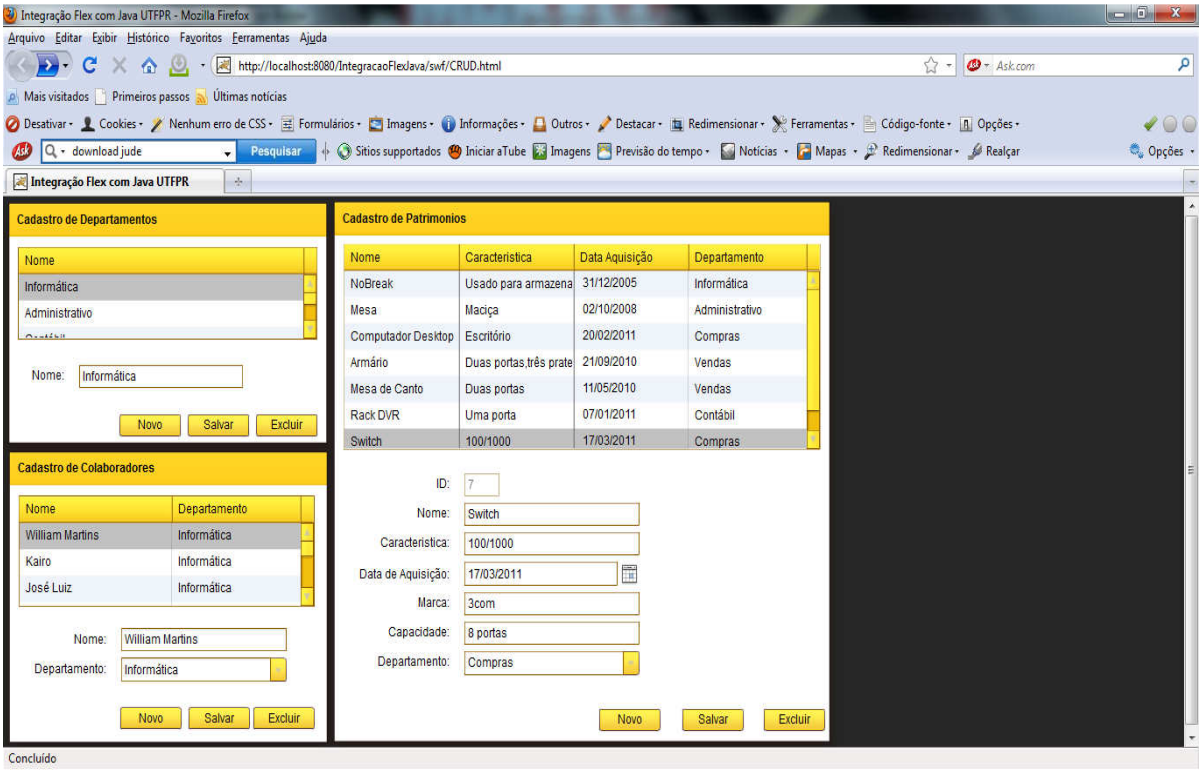

Figura 26 – Interface Executando.

```
private function creationCompleteHandler( event: FlexEvent ): void {
        servDepartamento.listAll();
        servColaborador.listAll();
        servPatrimonio.listAll();
    }
```

Figura 27 – Função *creationComplete*.

Na figura 25, da pra notar no código que para o serviço departamento foi estabelecido um *id*, este *id* esta sendo usado aqui também na função *creationComplete*, na Figura 30, onde é invocado o método *listAll( )* que esta no serviço departamento, por isso *servDepartamento.listAll( );* onde o *servDepartamento* é o *id* registrado na

implementação do *remoteObject* na figura 16. Como em departamento, todas as outras classes tiveram o mesmo tratamento e para todas foram criadas as mesmas funções.

Ainda a interface possui uma particularidade, pois existe um *comboBox* que carrega os dados de outra tabela ou base de dados, neste projeto a função usada para que o *comboBox* que busca os dados registrados em Departamento, exige uma codificação também que usa o *dataProvider* para exibir o resultado da tabela Departamento como uma lista ou *ArrayCollection*, como na figura 28.

```
private function departamentoListAllResultHandler( event: ResultEvent ): void {

        tabelaDepartamento.dataProvider = event.result;
        comboDepartamentoColaborador.dataProvider = event.result as ArrayCollection;
        comboDepartamentoPatrimonio.dataProvider = event.result as ArrayCollection;

    }
```

Figura 28 – Código para Exibir os Dados no *comboBox*.

Na figura 28, é apresentado o código para ser exibido na tela de cadastros de Colaborador como para Patrimônio. Após inserir este código a aplicação mostra todos os dados contidos na tabela como uma lista, na figura 29, são exibidos estes dados no *comboBox*.

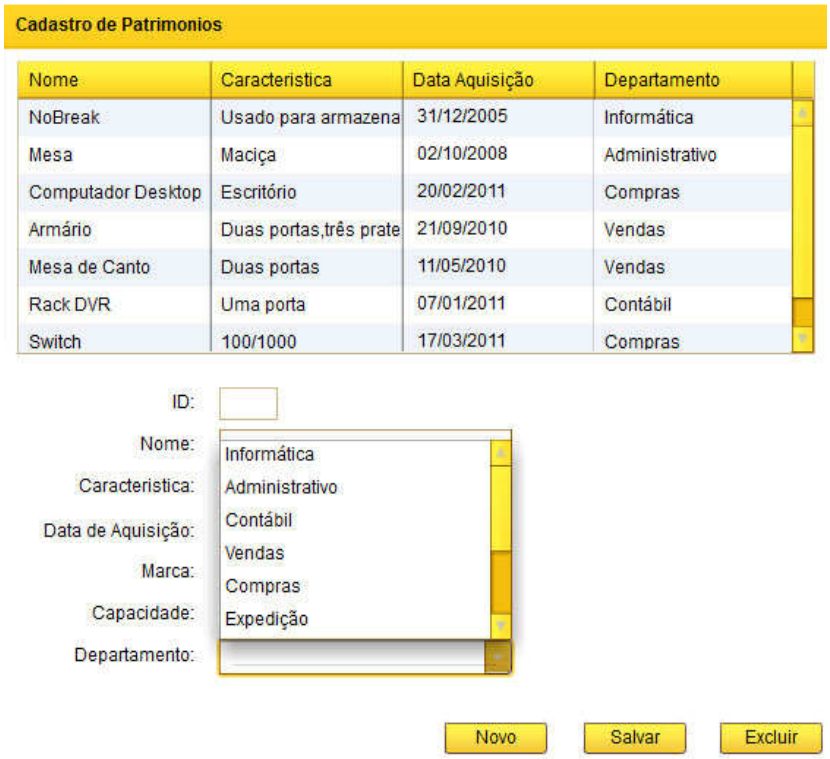

Figura 29 – Cadastro de Patrimônios, Exibindo Dados no *comboBox*.

Os botões Novo, Salvar, Excluir, também possuem uma implementação para fazer suas devidas funções, na figura 30, pode ser visto o código do botão salvar contido no cadastro departamento.

```
private function btnSalvarDepartamentoClickHandler( event: MouseEvent ): void {

        if ( departamentoSelec ) {

                departamentoSelec.nome = fieldNomeDepartamento.text;

                servDepartamento.update(departamentoSelec);

            } else {

                    var o: Departamento = new Departamento();

                    o.nome = fieldNomeDepartamento.text;

                    servDepartamento.save(o);

                }
```

Figura 30 – Função do Botão Salvar do Cadastro de Departamentos.

Com cada botão tendo suas funções implementadas, o programa está finalizado, os dados podem ser manipulados, inseridos, excluídos, editados e consultados.

6.4 Resultado Final da Aplicação

O resultado final pode ser visto nas próximas figuras, a figura 31 mostra o cadastro de colaboradores, a figura 32 mostra o cadastro de departamento, a figura 33 mostra o cadastro de patrimônios e por fim, na figura 34 a interface em *Flex* executando em um navegador.

Figura 31 – Tela Cadastro de Colaboradores.

Figura 32 – Tela Cadastro de Departamentos.

Figura 33 – Cadastro de Patrimônios.

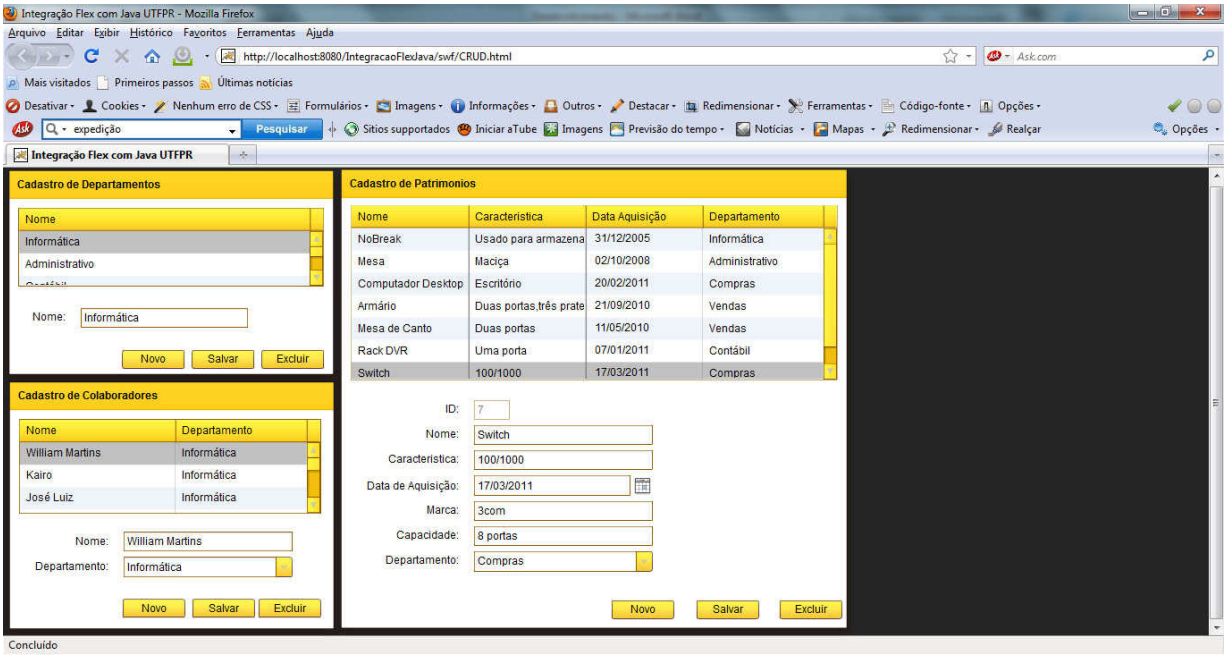

Figura 34 – Aplicação *Flex*.

CONCLUSÃO

Como o uso de softwares tem sido cada vez mais comum em vários ramos, tanto a tecnologia de processos quanto a de interface desses programas vem evoluindo. Todavia, nenhuma é melhor que a outra, todas possuem suas particularidades.

O Adobe *Flex* permite ao desenvolvedor a escolha da linguagem a se trabalhar, pois esta tecnologia permite o trabalho conjunto com outras linguagens ou tecnologias, como *Java* que é uma das linguagens mais usadas no mundo. Com suas características voltadas a criação de grandes aplicações corporativas é uma linguagem segura.

Este trabalho foi focado nessas duas tecnologias e a integração da mesma em uma única aplicação, com o desenvolvimento desta aplicação pode ser notado a fortes características *RIA* presentes no Adobe *Flex* e a rapidez que os processos são feitos e o armazenamento e a rápida resposta obtida tanto na atualização do banco de dados quanto na inserção de dados. Como a aplicação foi um exemplo simples das funções básicas mais utilizadas em uma aplicação, seria interessante o desenvolvimento de algo mais robusto que atendesse a um propósito em especifico.

Para trabalhos futuros seria conveniente o desenvolvimento de um sistema que integrasse estas duas tecnologias que usasse mais a fundo as funcionalidades do Adobe *Flex*, como a edição dos componentes para um uso especifico em uma aplicação ou um sistema mais robusto com maiores funcionalidades. Já em *Java* nesta mesma linha de trabalho, usar mais a fundo funcionalidades da tecnologias *Java* ou até mesmo alguma de suas tecnologias disponíveis trabalhando junto com *Flex*.

REFERENCIAS

ADOBE: BlazeDS. **BlazeDS Confluence.** 2010.
Dísponivel em: <http://opensource.adobe.com/wiki/display/BlazeDS/BlazeDS> Ultimo acesso em: 03/2011.

ADOBE Flash. **Adobe Labs Adobe Flash Platform Technologies.** 2010.

ADOBE Flash Catalyst. **Adobe Labs Adobe Flash Catalyst.** 2010.

ADOBE Gumbo. **Gumbo Flex SDK.** 2010.

ADOBE Learning Resources. **Developing Flex Applications.** Adobe Flex 3: Developer's Guide. 2007.

ADOBE Systems. **BlazeDS Developer Guide.** San Jose: Adobe Systems, 2008.

ADOBE Systems Incorporated. **The business benefits of rich internet applications for enterprises.** 2009.

ADOBE Systems Incorporated. **BlazeDS Developer Guide**. 2008.

ADOBE, **Flex Builder Overview.** 2011. Disponível em: <http://www.adobe.com/br/products/flex/overview>. Ultimo acesso em: 03/2011.

ADOBE Systems. Adobe **LiveCycle Data Services ES 2.6 Developer Guide.** San Jose: Adobe Systems, 2008b. Disponível em: <http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds/help.html?content=jms_messaging_2.html>. Ultimo acesso: 03/2011.

ADOBE LiveCycle Data Services ES. **Developer's Guide.** 2007. <http://blogs.adobe.com/flexdoc/files/flexdoc/lcds_dev_guide_6-06.pdf> Acessado em: 03/2011.

ADOBE Livecycle Data Services ES 2.6, **Developer Guide** 2008. <http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds/lcds_26_devguide.pdf> Acessado em 03/2011.

ANDREAZZA, Jean Daniel Henri Merlin. **Sistema de Orçamento Financeiro Pessoal com DWR e Integração PHP.** Especialização Java. Universidade Tecnológica Federal do Paraná. 2008.

BISSI, Wilson. Camadas **de apresentação web para Java: seu uso na prática.** Curso de pós-graduação em desenvolvimento de sistemas orientados a objetos em Java. Centro universitário de Maringá. 2009.

BUZATTO, David. **Formalização de um Modelo de Processo de Reengenharia Centrado no Usuário para Conversão de Aplicações Desktop em RIAs.**Universidade Federal de São Carlos. 2010 .

COLE, Alaric. **Learning Flex 3: Getting up to speed with rich internet applications**, 2008.

COSTA, Igor. **Criando experiências ricas na internet com Adobe Flex e Java**, JustJava 2009. 2009.

FAIN, Yakov. RASPUTNIS Victor. TARTAKOVSKY, Anatole. **Rich Internet Applications with Adobe Flex and Java.** SYS-CON Books/2007.

FRAGA, Rodrigo. **Interfaces de qualidade com Adobe Flex.** Java Magazine, Grajaú, v. 68, n. 7, p.20-30, maio 2009.

DEITEL, Paul J. e DEITEL Harvey M. **Ajax, Rich Internet Applications, and Web Development for Programmers.** New Jersey: Prentice Hall, 2008.

GONÇALVES, Edson. **Dominando NetBeans.** 2006.

GRANDBÄCK, Carl David. **Rich Internet Application (RIAs): A Comparison Between Adobe Flex, JavaFX and Microsoft Silverlight.** Master of Science Thesis in the Programme Software Engineering and Technology. Department of Computer Science and Engineering. Chalmers University of Technology. University of Gothenburg. 2009.

GUANAIS, Kaio Araújo. **Aplicações Ricas de Internet.** Curso de Sistemas de Informação. Instituto de Estudos Superiores da Amazônia (IESAM), 2010.

JORDAHL, Tom. **BlazeDS.** Adobe Systems Inc. 2007.

KNIPP, Eric. VALDES, Ray. **Navigating the Ajax vs. 'Heavy RIA' Dilemma.** Gartner, Inc. 2009.

LÓPEZ, Xavier Farré. **Rich Internet Application.** Trabalho de Conclusão de Curso. Universidade Politécnica da Catalunia, Espanha, 2005.

TEMPLE, André. MELLO, Rodrigo Fernandes. CALEGARI, Danival Taffarel. SCHIEZARO, Maurício. **Jsp, Servlets E J2ee**. 2004.

VALDES, Ray. **Key Issues in Rich Internet Application Platforms and User Experience.** Gartner, Inc. 2009.

VICTORAZZI, Nelson Rogério. **Ria – Rich Internet Applications**. Universidade Federal do Rio Grande do Sul, instituto de informática. Curso de Especialização em Web e Sistemas de Informação. 2007.

ZETIE, Carl. **The Rise of Rich Internet Applications.** Forrester Research, Inc. 2005.