

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

CARLOS AUGUSTO RODRIGUES DIAS

**DESENVOLVIMENTO DE UM SISTEMA PARA GERENCIAMENTO DE CURSOS E
EVENTOS UTILIZANDO A TECNOLOGIA JAVA**

NATAL/RN

2008

CARLOS AUGUSTO RODRIGUES DIAS

DESENVOLVIMENTO DE UM SISTEMA PARA GERENCIAMENTO DE CURSOS E EVENTOS UTILIZANDO A TECNOLOGIA JAVA

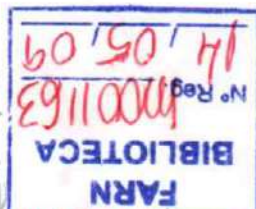
Monografia apresentada ao Curso de
Graduação em Sistemas de Informação
da Faculdade Natalense para o
Desenvolvimento do Rio Grande do
Norte como requisito obrigatório para
obtenção do grau de Bacharel em
Sistemas de Informação

Orientador: Gleydson Lima, M.Sc

NATAL/RN

2008





RB-32.178

004
D531d
0008
EX 01

0003

INPLATAN

CDU 004

RN/FARN/BC

1. Open Source - Monografia. 2. Jkava - Monografia. 3. Hibernata - Monografia I. Título.

Monografia para o Desenvolvimento do Rio Grande do Norte.
Monografia (Bacharelado em Sistemas de Informação) - Faculdade
Orientador: Gleyson Lima (Msc.)

62f.
Natal: FARN, 2008.
Desenvolvimento de um sistema para gerenciamento de cursos e
eventos utilizando a tecnologia Java / Carlos Augusto Rodrigues Dias. -
Dias, Carlos Augusto Rodrigues

Catálogo na Publicação - Biblioteca da FARN
Setor de Processos Técnicos

CARLOS AUGUSTO RODRIGUES DIAS

**DESENVOLVIMENTO DE UM SISTEMA PARA GERENCIAMENTO DE CURSOS E
EVENTOS UTILIZANDO A TECNOLOGIA JAVA**

Monografia apresentada ao curso de Bacharelado em Sistemas de Informações da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em ____ de ____ de ____, pela banca examinadora composta pelos seguintes membros:

Gleydson de A. F. Lima

Gleydson Lima, MSc.
Orientador

Ytalo Rosendo L. Azevedo

Ytalo Rosendo, MSc.
Membro Examinador

Nilda Maria F. de Albuquerque

Nilda Cerqueira, MSc.
Membro Examinador

-volvimento
então, é
neste ponto
que a linguagem
aparece para
a expressão de

relação para
com o mundo
-pensando de
modo de a
relação entre o
com o mundo

Java, 4/10/00

10/10/00

A minha família pelo apoio e dedicação e aos professores
que me ajudaram a obter o conhecimento necessário
para seguir o rumo da minha vida profissional.

RESUMO

Este trabalho consiste no desenvolvimento de um sistema para gerenciamento de cursos e eventos utilizando somente ferramentas *open sources* com a tecnologia JAVA, na qual utilizará o *Hibernate* para realizar o mapeamento objeto relacional das classes para o banco de dados, a linguagem *JSP*, o servidor de aplicações *Tomcat*, e utilizará a biblioteca *JasperReports* para gerar relatórios além de permitir que os usuários administrativos da aplicação efetuem *Login* através de uma base *OpenLDAP (LDAP)*.

Palavras-chave: Open source. Java, Hibernate. JSP. Tomcat. JasperReports. Openldap. LDAP.

ABSTRACT

This work consists in developing a system for management courses and events using only tools open sources with JAVA, which will use the Hibernate to map object relational of classes for the database, the language JSP, Tomcat Application Server and use the JasperReports library to generate reports in addition to allowing users administrative application perform login bases OpenLDAP (LDAP).

Keywords: Open source. Java. Hibernate. JSP. Tomcat. JasperReports. OpenLDAP. LDAP.

SUMÁRIO

1 INTRODUÇÃO	9
1.1 PROBLEMA DE PESQUISA	10
2 OBJETIVOS	11
2.1 GERAL	11
2.2 ESPECÍFICOS	11
3 JUSTIFICATIVA	12
4 REFERENCIAL TEORICO	13
4.1 JAVA	13
4.1.1 JAVA ENTERPRISE EDITION	14
4.1.1.1 SERVLET	15
4.1.1.2 JAVA SERVER PAGES	15
4.1.1.2.1 SINTAXE BÁSICA	16
4.1.1.2.2 VARIÁVEIS PREDEFINIDAS	16
4.1.1.2.3 EXPRESSION LANGUAGE-EL	18
4.1.1.2.3.1 ACESSANDO ATRIBUTOS	18
4.1.1.2.3.2 JSTL	18
4.2 MAPEAMENTO OBJETO RELACIONAL	21
4.2.1 HIBERNATE	21
4.2.1.1 CLASSES PERSISTENTES	23
4.2.1.2 CICLO DE VIDA	24
4.2.1.3 ASSOCIAÇÕES	25
4.2.1.3.1 ONETOMANY	25
4.2.1.3.2 MANYTOONE	26
4.2.1.3.3 ONETOONE	27
4.2.1.3.4 MANYTOMANY	27
4.2.1.4 COLEÇÕES	28
4.2.1.5 MAPEAMENTO POLIMÓRFICO	28
4.2.1.6 BUSCA COM HIBERNATE	28

4.2.1.6.1 HQL (<i>Hibernate Query Language</i>)	29
4.2.1.6.2 CRITERIA	29
4.2.1.7 CACHE	30
4.2.1.8 TRANSAÇÕES	30
4.3 LDAP (<i>Lightweight Directory Access Protocol</i>)	31
4.4 JNDI (<i>Java Naming Directory Interface</i>)	31
4.5 X.500	32
4.6 OPENLDAP	32
4.7 JASPERREPORTS	33
4.7.1 FONTE DE DADOS	34
4.7.2 SUB-RELATÓRIOS	34
4.7.3 IREPORT	35
4.8 TOMCAT	36
3.8.1 INTEGRANDO APACHE E TOMCAT	36
5 METODOLOGIA	38
6 RESULTADOS	39
6.1 MODELAGEM	39
6.1.1 CASOS DE USO	39
6.1.2 DIAGRAMA DE CLASSES	42
6.2 DESENVOLVIMENTO	43
6.2.1 HIBERNATE	43
6.2.2 JSP	47
6.2.3 OPENLDAP	49
6.2.3.1 BIBLIOTECAS QUE IMPLEMENTAM O OPENLDAP	49
6.2.4 IREPORT	52
6.2.5 JASPERREPORTS	55
6.2.6 TOMCAT	58
7 CONCLUSÃO	59
8 REFERÊNCIAS	60

LISTA DE SIGLAS

JSP	<i>Java Server Pages</i>
LDAP	<i>Lightweight Directory Access Protocol</i>
SLAPD	<i>Stand-alone LDAP Daemon</i>
SLURPD	<i>Stand-alone LDAP Update Replication Daemon</i>
JNDI	<i>Java Naming Directory Interface</i>
DAP	<i>Directory Access Protocol</i>
DN	<i>Distinguished Name</i>
JSE	<i>Java Standard Edition</i>
JEE	<i>Java Enterprise Edition</i>
JME	<i>Java Micro Edition</i>
CGI	<i>Common Gateway Interface</i>
EL	<i>Expression Language</i>
JSTL	<i>Java Standard Tag Library</i>
POJO	<i>Plain Old Java Object</i>
HQL	<i>Hibernate Query Language</i>
DNS	<i>Domain Name System</i>
JPA	<i>Java Persistence API</i>

1 INTRODUÇÃO

Com o advento da Internet em que a mesma trouxe inúmeras vantagens tanto para as corporações quanto para os indivíduos onde muitas áreas dos Sistemas de Informação tem se modificado visando abranger toda a infra-estrutura da Web para se integrar num ambiente único e portátil pela necessidade de simplificar a atualização e a manutenção do código-fonte, mantendo em um mesmo local para prover acesso dos diferentes usuários em um ambiente centralizado.

Muitas foram as tecnologias que surgiram para simplificar e centralizar as aplicações Web, na qual a linguagem JAVA se destaca pelo fato de absorver todo seu legado de qualidade comprovado na programação *desktop* para o ambiente cliente – servidor, na qual introduziu o conceito de Orientação a Objetos para Web em que se pode modelar o mundo real com classes e instâncias bem definidos que interagem entre si de modo a gerar o resultado esperado. Na sua estrutura existe o JSP (*Java Server Pages*) que permite desenvolver páginas para internet e produzir aplicações para acessar banco de dados, manipular arquivos, capturar informações a partir de formulários entre inúmeras outras funções encontradas na programação *desktop*. Para tanto, há necessidade de utilização de um servidor de aplicação para realizar o gerenciamento das informações trafegadas entre o cliente e o servidor, onde o *Apache Tomcat* executa tais operações.

Um conceito bastante importante, diz respeito de como os usuários da aplicação terão acesso ao mesmo. A linguagem JAVA oferece total suporte a utilização da forma como se autentica no Sistema Operacional, e com isso introduzir essa autenticação nos sistemas desenvolvidos em JSP, em que consiste em uma conexão LDAP, que é a base de dados da rede onde são armazenadas informações dos clientes, onde as informações podem ser compartilhadas.

Outro ponto importante se concentra na forma como os dados são armazenados na aplicação em que existem métodos que automatizam o procedimento de leitura, gravação, alteração e remoção das informações. Com isso, será utilizado o *Hibernate*, que é um *framework* desenvolvido para fazer a modelagem do objeto da classe com o banco de dados.

Atualmente as informações provenientes dos sistemas que as empresas utilizam estão cada vez mais complexos e necessitam que haja uma fragmentação desses dados para se extrair dados que venham a contribuir com o gerenciamento das grandes e pequenas corporações. Através desse cenário, conclui-se a necessidade da construção de relatórios gerencias. Hoje em dia existem várias ferramentas que colaboram com esse objetivo. Na linguagem de programação JAVA existe o *JasperReports*, que é uma biblioteca padrão para construção de relatórios, que em conjunto com o *JasperReports*, existe o *Ireport* que consiste num programa para se desenvolver o *Layout* do relatório desde o mais simples, até o mais complexo.

O objetivo deste trabalho envolve o estudo e implementação de diversas ferramentas onde o produto final a ser atingido se concentra no desenvolvimento de um sistema para gerenciamento de Cursos e Eventos utilizando a linguagem JAVA como plataforma de desenvolvimento e o servidor de aplicação *Tomcat*, o *framework Hibernate*, para realizar o mapeamento dos dados para um base de dados, a inserção de autenticação utilizando o *LDAP*, e a construção de relatórios com o *JasperReports*.

1.1 - PROBLEMA DE PESQUISA

Como desenvolver uma solução empresarial utilizando ferramentas *open-sources*?

2 OBJETIVOS

2.1 GERAL

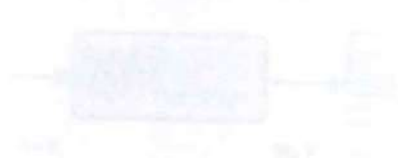
Projeto e implementação de uma aplicação *Web* para gerenciar Cursos e Eventos utilizando a plataforma JEE.

2.2 ESPECIFICO

- Estudar e utilizar o *framework* de Mapeamento Objeto Relacional *Hibernate*;
- Estudar e utilizar a linguagem *Java Server Pages* (JSP);
- Desenvolver relatórios utilizando a biblioteca *Jasper Reports* e a ferramenta *Ireport*;
- Autenticar os usuários administrativos em uma base *LDAP* (*Lightweight Directory Access Protocol*).
- Estudar e utilizar o servidor de aplicação *Tomcat*;

3 JUSTIFICATIVA

Levando em consideração a forma de implementação do sistema em questão, na qual podemos excluir o custo de análise e desenvolvimento, sem a necessidade de adquirir licenças proprietárias de softwares, justifica-se a realização deste trabalho no intuito de desenvolver este projeto utilizando apenas ferramentas *open sources*.



Assinado por:

[Handwritten signature]

4 REFERENCIAL TEÓRICO

4.1 JAVA

JAVA é uma linguagem de programação desenvolvida pela Sun Microsystems na década de 90, e “A par das práticas modernas desenvolvimento de software, *JAVA é orientado objeto*”¹. (SUN MICROSYSTEMS, 2008).

A orientação a objetos consiste em conceder um sistema computacional como um todo orgânico formado por objetos que relacionam entre si (TAFNER; 2001).

Na linguagem de programação JAVA, todo código fonte é escrito primeiro em texto puro e com a extensão .JAVA. Esses arquivos de fonte são compilados em seguida para o .class pelo compilador javac. O arquivo .class não contém código que é nativo para o seu processador; em vez disso contém *bytecodes* (o idioma da máquina virtual JAVA). A ferramenta JAVA então executa a sua aplicação na instância do *Java Virtual Machine*.

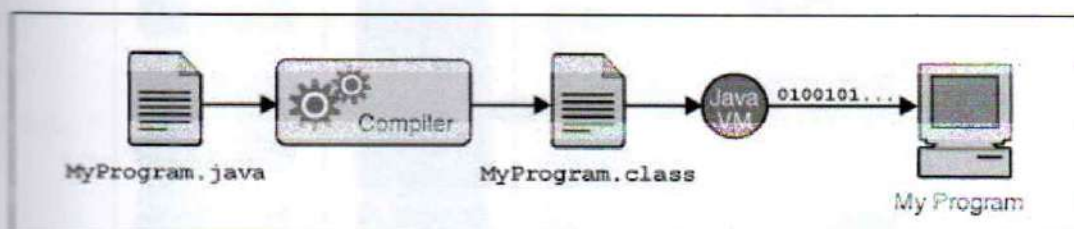


Figura 1: Processo de desenvolvimento com JAVA
Fonte: (SUN MICROSYSTEMS, 2008)

Programas JAVA consistem em partes chamadas classes, estas consistem em partes chamadas métodos que realizam tarefas e retornam as informações ao completarem suas tarefas (DEITEL, 2001)

A linguagem JAVA se divide em três plataformas JSE, JEE, JME.

JSE – *Java Standard Edition*, arquitetura voltada para o desenvolvimento de aplicativos desktop;

JEE – *Java Enterprise Edition*, arquitetura de desenvolvimento para servidores, contém bibliotecas para o acesso a sistemas que utilizam o protocolo *TCP-IP* e que podem funcionar de forma distribuída;

JME - Java Micro Edition, ambiente de desenvolvimento para dispositivos móveis e portáteis. (SUN MICROSYSTEM; 2008).

4.1.1 JAVA ENTERPRISE EDITION

A plataforma *Java Enterprise Edition*, oferece um modelo para se desenvolver aplicações em ambientes corporativos, voltada para operações multi-camadas, baseadas em componentes que são executados em um servidor de aplicações, na qual são divididas em três camadas: as máquinas clientes (*Cliente-Side Presentation*), o servidor JEE (*Server-Side Presentation*) e a máquina do banco de dados. (*Server-Side Bussiness Logic*).

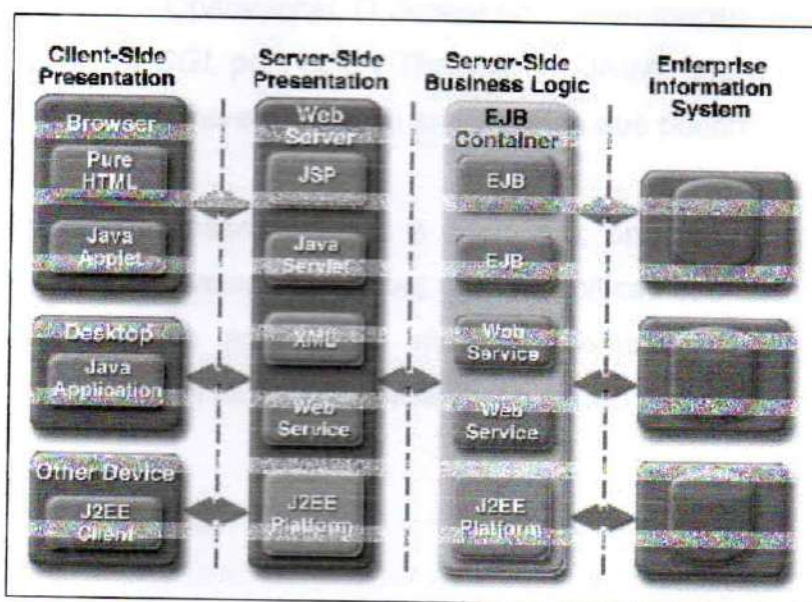


Figura 2: Modelo da arquitetura JEE multicamadas
Fonte: (SUN MICROSYSTEM, 2008)

Na camada Cliente é onde se define o suporte aos componentes de visualização das operações que serão executados no cliente.

A camada *Server-Side Bussiness Logic* se refere à hospedagem das aplicações em um **Contêiner Web**³ onde existem componentes de negócios que são executados no servidor e também trata do ciclo de vida dos componentes *Web*.

E a camada *Server-Side* trata dos componentes *Web* e contem uma serie de especificações, como *Web Service*, *XML*, e em maior importância, encontramos a ***Servlet*** e a ***Java Server Pages***. (PAWLAN; et al, 2002).

4.1.1.1 SERVLET

Servlet é uma classe da linguagem de programação *JAVA* usada para ampliar a capacidade dos servidores que hospedam aplicações acessadas via modelo de programação cliente-servidor.

A mesma funciona semelhante aos *Scripts CGI*⁴ (*Common Gateway Interface*) que são pequenos programas que interpretam os parâmetros passados pelo navegador e geram a página depois de processá-los gerando a cada requisição um novo processo no Sistema Operacional. O *Servlet* possui um mecanismo de geração de conteúdo diferente do *CGI*, pois utiliza *Threads* em *JAVA*, que é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente.

No *servlet* existem classes básicas e interfaces, onde as principais são a *javax.servlet* para escrever classes genéricas não específicas das aplicações *WEB*, pois *Servlet* não se aplica apenas ao modelo Cliente-Servidor, e a classe *javax.servlet.http* onde contem as classes específicas para o desenvolvimento *WEB*.

4.1.1.2 JAVA SERVER PAGES

Java Server Pages é uma tecnologia baseada em *JAVA* que simplifica o processo de desenvolvimento de sites dinâmicos na qual permite inserir conteúdo estático (que podem ser escrito em qualquer formato baseado em texto, como por exemplo, *HTML*) e conteúdo dinâmico em uma página *WEB*, onde é fornecido através de um navegador *WEB* cliente, utilizando toda a capacidade da tecnologia *Java Servlet*.

⁴ - *CGI (Common Gateway Interface)* Aplicação servidora utilizada geralmente para processar solicitações do navegador (browser) através de formulários *HTML*, enviando o resultado em páginas dinâmicas *HTML*.

³ - *Container Web* é uma aplicação dedicada a servir aplicações *Web* para usuários na Internet.

Uma página JSP depois de compilada transforma-se em uma *Servlet* no Servidor, o que vai de acordo com (FIELDS, et. al, 2000) onde expõe que atualmente a maioria das implementações de JSP se baseia em *Servlets*.

Segundo a definição da Especificação JSP 1.2, uma página JSP é um documento baseado em texto que descreve como processar uma Solicitação (*Request*) para criar uma Resposta (*Response*).

Uma das principais características da linguagem JSP está na separação do conteúdo estático do dinâmico, ou seja, a lógica da aplicação é separada do código HTML, que é responsável pela interface com o cliente. Assim, é possível alterar o *design* da página *WEB* sem influência alguma na lógica da aplicação. (FIELDS, et. al, 2000).

3.1.1.2.1 SINTÁXE BÁSICA

A sintaxe que o JSP utiliza a presença de dois tipos genéricos de dados: estáticos, e os próprios elementos sintáticos. Os dados estáticos são todos os que não podem ser interpretados por serem desconhecidos, pois não existem relação com o JSP, como por exemplo, o texto da página e arquivos de configuração *XML*. Os elementos sintáticos são os componentes da sintaxe JSP, onde os mesmo podem ser expressos de três formas distintas:

Diretivas que são elementos utilizados para obter informações globais e que não dependem de qualquer solicitação (`<%@Diretiva %>`);

Ações são informações empregadas em tempo de execução (`<jsp:Ação>`);

Scripting que são os que estabelecem um mecanismo que une os dados estáticos e os dinâmicos onde podem ser `<%! Declaração %>` que são utilizadas pela própria página e são executados uma única vez, `<% Código %>` empregados com códigos de programação e são executados a cada nova solicitação da página, e por ultimo `<%= Expressão %>` que são utilizados na geração de dados exibidos ao usuário. (NICK, 2003).

4.1.1.2.2 VARIÁVEIS PREDEFINIDAS

Uma variável predefinida é conhecida como um objeto implícito, devido ao fato de poderem ser utilizadas por nome dentro do código JAVA em uma JSP, sem jamais ter de declará-las. (NICK, 2003).

As variáveis predefinidas disponíveis na linguagem JSP são:

Request	Usada para ler atributos do cliente;
Response	Usada para enviar dados ao cliente
Session	Usado para armazenar variáveis de sessão;
Out	- Recuperado a partir de <i>response</i> para envio de dados;
Application	- Permite troca de dados entre páginas JSP;
Exception	É o objeto que é resultante de um de erro numa aplicação JSP;
Config	Representa o objeto <i>ServletConfig</i> da página JSP. Define métodos para acessar qualquer informação de inicialização advindas de um servlet.
PageContext	Encapsula características dependentes de implementação no objeto <i>javax.servlet.jsp.PageContext</i> .
Page	Segundo (NICK, 2003) Essa variável corresponde a referencia " <i>this</i> " no servlet gerado. Esta é a mais extensa em atributos que no total são 12 onde pode haver a ocorrência mais de uma diretiva em uma mesma página que são: <i>language</i> , <i>extends</i> , <i>import</i> , <i>session</i> , <i>buffer</i> , <i>autoFlush</i> , <i>isThreadSafe</i> , <i>isErrorPage</i> , <i>errorPage</i> , <i>contentType</i> , <i>PageEncoding</i> .

Quadro 1: Variáveis pré-definidas na linguagem JSP
Fonte: Elaboração própria (2008)

4.1.1.2.3 LINGUAGEM DE EXPRESSÃO – EL

Na versão anterior do JSP (1.2), existiam duas maneiras de passar valores aos atributos de uma *taglib*: estaticamente ou através de expressões (também chamadas de *request-time expressions* – ou simplesmente RT). (GEISS, 2008)

Embora flexível, o uso de expressões RT compromete a manutenção das páginas JSP, pois seu uso requer conhecimentos da linguagem JAVA – o que dificulta o trabalho dos designers envolvidos. A *Expression Language (EL)* visa eliminar esse problema, facilitando o acesso aos objetos e aos seus métodos. Podem se apresentar de duas 3 formas distintas, **literal** (representa pela expressão entre `{` e `}`, com **operadores** (+, -, *, /, %, lógicos e comparativos) e com **palavras reservadas** (*and* (&&), *equals* (==), maior (>), maior igual (>=), *or* (||), *not* (!), negação (!=), menor (<), menor igual (<=), *div* (/) e *mod* (%)) (VILLAFUERTE, et al, 2008).

4.1.1.2.3.1 ACESSANDO ATRIBUTOS

Apesar de avaliar expressões literais simples ser útil, EL mostra sua importância ao acessar e processar atributos e propriedades em escopos diferentes. O acesso ao atributo é simples com EL (simplesmente referenciada pelo nome). As propriedades, os métodos, e as disposições do *JavaBean* podem ser acessadas usando o nome do atributo". notação, como por exemplo: `{usuario.sobreNome}`. (VILLAFUERTE, et al, 2008)

4.1.1.2.3.2 JSTL (Java Standard Tag Library)

Devido à necessidade de dividir as operações de programação e design para se obter JSPs bem organizadas internamente surgiu o conceito criar páginas dinâmicas bastante complexas sem escrever código JAVA dentro delas, onde se pensou em fornecer tags que tornem fáceis tarefas que exigiriam várias linhas de código na aplicação.

JSTL consiste em uma coleção de bibliotecas, tendo cada uma um propósito bem definido, que permitem escrever páginas JSPs sem código JAVA, levando

assim a uma maior legibilidade do código e interação entre programadores e *designers* onde uma página JSTL é uma página JSP contendo um conjunto de tags JSTLs.

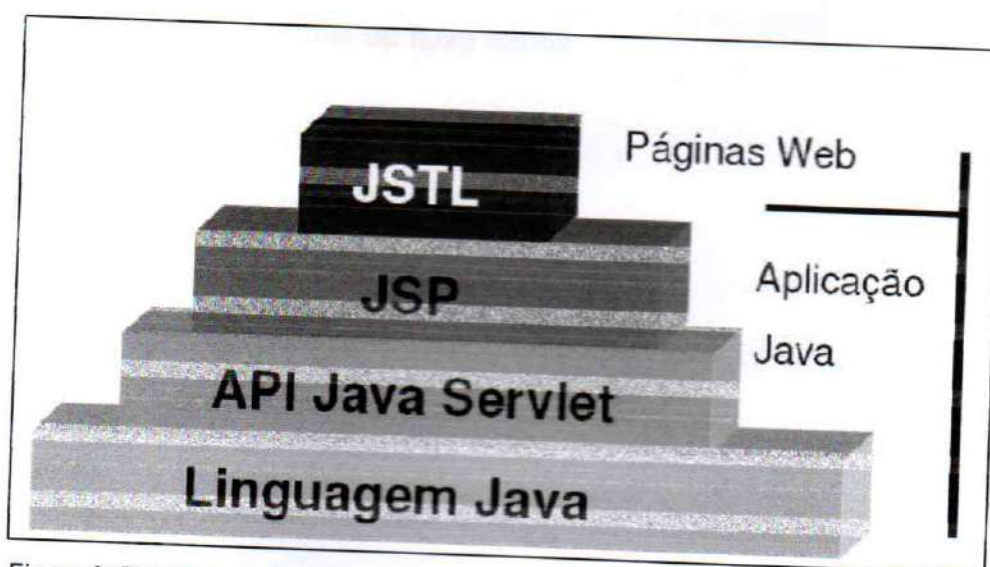


Figura 3: Estrutura do JSTL
Fonte: (NETO, 2008)

Possui varias *tags* largamente utilizadas para controle de fluxo, gerenciamento de url, formatação de textos e datas, manipulação de XML, e acesso a banco de dados.

Tag	Utilização
<i>c</i>	Controle de fluxo
<i>fmt</i>	Formatação de texto
<i>x</i>	XML
<i>sql</i>	Acesso a dados de banco de dados

Quadro 2 – Variáveis do JSTL
Fonte: Elaboração própria (2008)

Exemplo da utilização do JSTL:

```
<c:set value="100" var="count"/>
=
request.setAttribute("count",100);
```

Para que o compilador JSP não compreenda que a *tag c:set* seja um código *HTML* inválido, é preciso colocar uma tag informando ao compilador que tudo que tiver prefixo *c*, é uma *tag library*:

```
<@taglib prefix="c" uri="arquivo_tld"/>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"%>
```

Como exemplo da tag de controle de fluxo temos:

```
<c:forEach items="${list}"
begin="5" end="20" step="4"
var="item">
<c:out value="${item}"/>
</c:forEach>
```

Fonte: (LIMA, 2008)

Geralmente esta tag é mais utilizada para a repetição. Permite acesso por interação aos *arrays*, instâncias de *java.util.Collection*, *java.util.Iterator*, *java.util.Map* e *java.util.Enumeration*. Percorre cada elemento expondo o valor atual da repetição no código da página JSP contido na tag. (VILLAFUERTE, et al, 2008).

O atributo *fmt* pode é utilizado para formatar datas de acordo com a *locale*,

Exemplo:

```
<fmt:formatNumber type="currency" value="${salary}"/>
<fmt:message key="welcome" />
<fmt:formatDate value="${data}"/>
<fmt:formatDate value="${data}" dataStyle="full"/>
```

Fonte: (LIMA, 2008)

4.2 MAPEAMENTO OBJETO RELACIONAL

O conceito de mapeamento relacional esta totalmente ligada à forma de como interligar os dados advindos do modelo de programação de orientado a objetos para uma base relacional onde os dados se encontram estruturados unicamente na forma de tabelas.

Persistência automática e transparente de objetos de um aplicativo JAVA para tabelas em um banco de dados relacional, utilizando meta-dados que descrevem o mapeamento entre os objetos e o banco de dados. Em essência, transforma dados de uma representação para a outra. (BAUER, 2005).

Este modelo esta embasado na forma de como transformar os dados de um objeto em uma linha de uma tabela de um banco de dados, ou de forma inversa, com a transformação de uma linha da tabela em um objeto na aplicação.

4.2.1 HIBERNATE

O Hibernate é um framework open-source para Mapeamento Objeto Relacional para aplicativos escritos na linguagem de programação JAVA⁵ onde com o mesmo podemos efetuar todas as operações básicas comumente realizadas em banco de dados como criar e alterar tabelas, inserir, atualizar e excluir dados, tratar restrições, projeções e junções, agrupamentos, ordenação e agregação, mediante o uso de arquivos (XML) e bibliotecas para estabelecer tais operações. (GALHARDO; LIMA, 2007).

O Hibernate gera as consultas de dados e libera o desenvolvedor do trabalho manual da conversão dos dados resultante, mantendo o programa portátil para quaisquer bancos de dados.

⁵ - Não apenas em JAVA, mais em outras linguagens também.

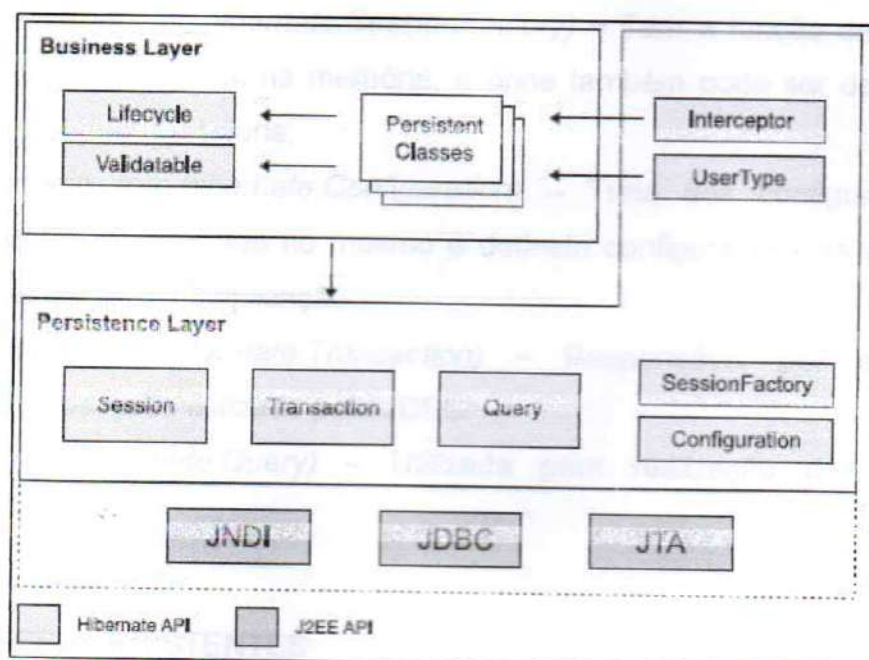


Figura 4: Arquitetura do Hibernate
Fonte: (BAUER, 2005)

A arquitetura do Hibernate é formada basicamente por um conjunto de interfaces. A camada de negócio aparece acima da camada de persistência por servir como uma cliente da camada de persistência.

Na interface **Session**, **Transaction** e **Query**, é onde se realiza as operações básicas de criação, atualização, remoção e consulta no banco de dados;

Na interface **Configuration**, encontra-se o arquivo de configuração do Hibernate;

A interface **Interceptor**, **Lifecycle** e **Validatable** são responsáveis por realizar a interação entre os eventos do Hibernate e a aplicação;

E por último, as interfaces que permitem a extensão das funcionalidades de mapeamento do Hibernate são a **UserType**, **CompositeUserType**, **IdentifierGenerator**.

Além disso, o Hibernate também interage com APIs do JAVA já existentes como JNDI, JDBC e JTA.

Na camada de persistência encontramos:

Session (*org.hibernate.Session*) – É responsável pela comunicação entre a aplicação e a persistência, através de uma conexão JDBC. Com ele é possível criar, remover, atualizar e recuperar objetos persistentes;

SessionFactory (*org.hibernate.SessionFactory*) – Tem a função de manter o mapeamento objeto relacional na memória, e onde também pode ser denominado como fábrica de objetos Sessions;

Configuration (*org.hibernate.Configuration*) – Trata das configurações de inicialização do hibernate, onde no mesmo é definido configurações de acesso ao banco de dados utilizado na aplicação.

Transaction (*org.hibernate.Transaction*) – Responsável por abstrair o mecanismo de transações utilizado pelo JDBC.

Query (*org.hibernate.Query*) – Utilizada para realização de consultas. (BAUER, 2005).

4.2.1.1 CLASSES PERSISTENTES

Uma classe persistente é aquela que implementa as entidades domínio do negócio de forma que associa a cada tabela do banco de dados a um POJO (*Plain Old Java Object*), que é um objeto JAVA que não precisa implementar nenhuma interface ou estender nenhuma classe. Como exemplo de um POJO, podemos citar a figura 5.

O mesmo deve possuir todos os atributos e possuírem métodos *get()* e *set()* e também um método específico para a chave primária. (GALHARDO; LIMA, 2008).

```

public class Curso {
    private int idCurso;
    private int cargaHoraria;
    private String ementa;
    private String objetivoGeral;
    private String objetivoEspecifico;
    private String conteudoProgramatico;
    private String disciplinas;
    private String areaCurso;
    public int getIdCurso() {
        return idCurso;
    }
    public void setIdCurso(int idCurso) {
        this.idCurso = idCurso;
    }
    public int getCargaHoraria() {
        return cargaHoraria;
    }
    public void setCargaHoraria(int cargaHoraria) {
        this.cargaHoraria = cargaHoraria;
    }
    public String getEmenta() {
        return ementa;
    }
    public void setEmenta(String ementa) {
        this.ementa = ementa;
    }
    public String getObjetivoGeral() {
        return objetivoGeral;
    }
}

```

Figura 5: Exemplo de um POJO
 Fonte: Dados da Pesquisa

4.2.1.2 CICLO DE VIDA

O Hibernate sempre que precisa transferir dados da memória para o banco de dados e mutuamente é necessário haver uma interação entre a aplicação e a camada de persistência, onde se isso é possível invocando o gerenciador de persistência e as interfaces de consultas. E quando isso ocorre à aplicação precisa ter conhecimento sobre o ciclo vida da persistência, que podem ser de três estados: *Transient*, *Persistent* e *Detached*.

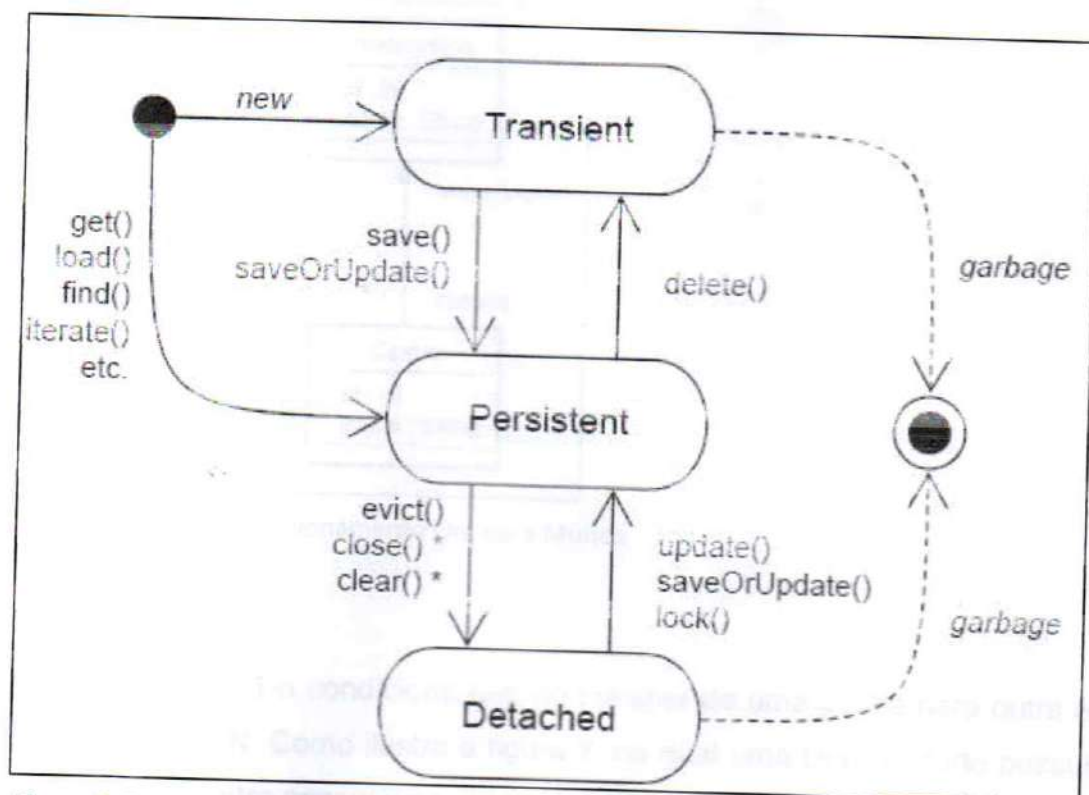


Figura 6: Estado de um objeto e a transição entre Hibernate e Aplicação
Fonte: (BAUER, 2005)

Objetos Transientes: São objetos que suas instâncias não estão nem estiveram associados a algum contexto persistente. Eles são instanciados, utilizados e após a sua destruição não podem ser reconstruídos automaticamente;

Objetos Persistentes: São objetos que estão associados a um contexto persistente e estão sempre associados a um objeto Session;

Objetos Detached: São objetos que tiveram suas instâncias associadas a um contexto persistente. São objetos em um estado intermediário, nem são transientes nem persistentes. (GALHARDO; LIMA, 2007).

4.2.1.3 – ASSOCIAÇÕES

Associações definem como manipular as relações entre as tabelas, na qual podem ocorrer de 4 tipos distintos.

4.2.1.3.1 @ONETOMANY – Um para Muitos (1-n)

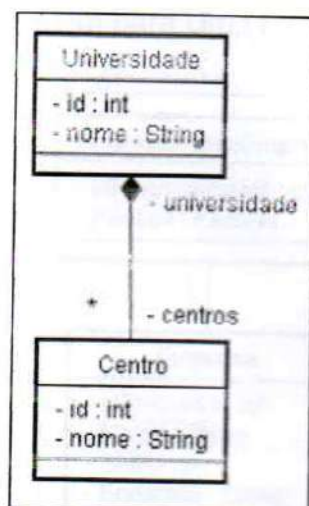


Figura 7: Exemplo de relacionamento Um para Muitos
Fonte: (LIMA, 2008)

Um relacionamento 1-n condiciona que no máximo de uma tabela para outra é 1 e na ordem inversa é N. Como ilustra a figura 7, na qual uma Universidade possui vários Centros e um Centro possui uma única Universidade.

4.2.1.3.2 @MANYTOONE – Muitos para Um (n-1)

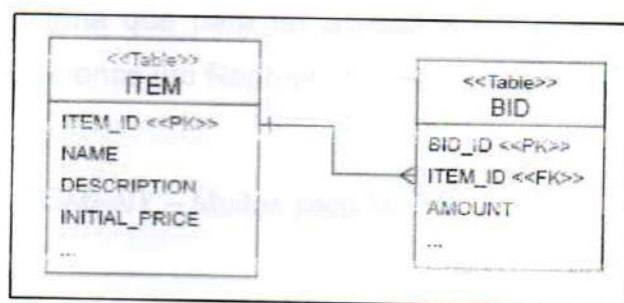


Figura 8: Exemplo de relacionamento Muitos para Um
Fonte: (BAUER, 2005)

Um relacionamento n-1 trata que a entidade que possui vários objetos, esta associada a apenas um único conjunto de valor de uma entidade.

Como ilustra a figura 8 em que na tabela *ITEM* (muitos) possui um único *BID* (Oferta), e um *BID* possui em vários *ITEMs*.

4.2.1.3.3 @ONETOONE – Um para Um (1-1)

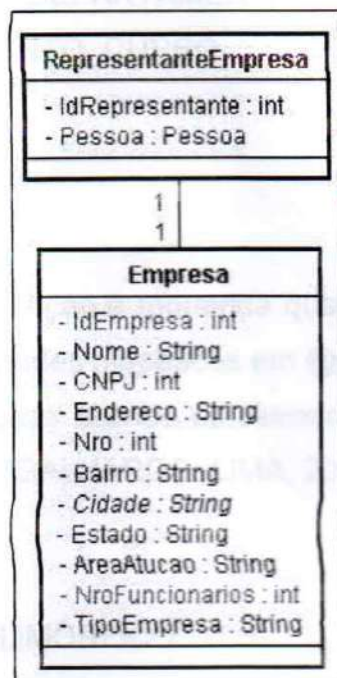


Figura 9: Exemplo de relacionamento Um para Um
Fonte: Dados da Pesquisa

Em um relacionamento 1-1 condiciona que **uma** entidade possui apenas **um** objeto na tabela mapeada no relacionamento. Como ilustra a figura 8, na qual a regra de negocio confirma que para ter acesso a um determinado sistema, uma empresa deve possuir apenas um RepresentanteEmpresa.

4.2.1.3.4 @MANYTOMANY – Muitos para Muitos (n-n)

No relacionamento n-n indica que uma entidade possui muitos objetos na tabela relacionada e de forma inversa, com a mesma definição. Com isso ocorre a criação de uma entidade fraca entre essas entidades para guardar o relacionamento entre essas duas através de suas chaves primárias.

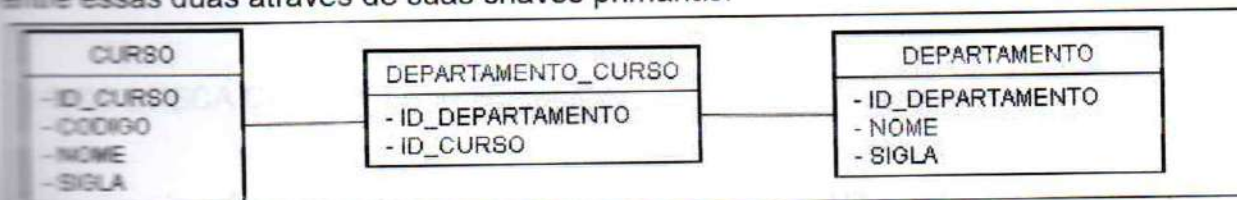


Figura 10: Exemplo de relacionamento Muitos para Muitos
Fonte: (LIMA, 2008).

Como ilustra a figura 10, em que um CURSO possui vários DEPARTAMENTOS, e um DEPARTAMENTO possui vários CURSOS, gerando assim a tabela DEPARTAMENTO_CURSO.

4.2.1.4 COLEÇÕES

Uma tabela para a coleção é requerida quando a mesma contém valores ou referências para outras entidades mapeadas em `@ManyToMany`.

Coleções são mapeadas usando os elementos `<set>`, `<list>`, `<map>`, `<bag>`, `<array>` e `<primitive-array>` (GALHARDO; LIMA, 2007).

4.2.1.5 MAPEAMENTO POLIMÓRFICO

Polimorfismo é uma característica de linguagens orientada a objetos como JAVA. Uma associação polimórfica é uma associação que pode se referir a instâncias de uma subclasse da classe que foi especificado no arquivo de configuração do Hibernate, na qual suporta três estratégias de mapeamento de herança:

- Uma tabela por hierarquia de classes, onde cada classe concreta é mapeada para uma tabela diferente no banco de dados.
- Uma tabela por subclasse, que todas as classes são mapeadas em uma única tabela;
- Uma tabela por classe concreta (polimorfismo implícito), que mapeia cada tabela, inclusive a classe mãe, para tabelas diferentes. (GALHARDO; LIMA, 2007).

4.2.1.6 BUSCA COM HIBERNATE

Para realizar consultas no banco de dados, o Hibernate disponibiliza três maneiras: HQL (*Hibernate Query Language*), *Criteria Query API* e SQL nativo.

4.2.1.6.1 HQL (*Hibernate Query Language*)

É uma linguagem para recuperação de dados baseada em SQL, com a diferença de que ela não trabalha com as tabelas do banco, e sim com os objetos da aplicação, por isso a mesma é orientada a objetos.

HQL suporta polimorfismo, herança e associações, suporta ainda agregação, ordenação, paginação, sub-consultas, *outer joins*, chamadas de funções SQL e projeção, que permite a realização de buscas por apenas os dados necessários, não sendo necessário carregar todo o objeto e suas associações.

```
from Aluno aluno where aluno.matricula >= 35
from Endereco end where
    ( end.rua in ("Bernardo Vieira", "Prudente de Moraes") )
    or ( end.numero between 1 and 100 )
from Professor p where p.nome like "João%"
from Aluno aluno order by aluno.nome asc
```

Figura 11: Exemplo consulta utilizando HQL
Fonte: (GALHARDO, LIMA; 2008)

4.2.1.6.2 CRITERIA

É um mecanismo orientado a objetos que permite criar consultas a partir de um conjunto de objetos e definir critérios em tempo de execução.

O Criteria é uma abordagem de consulta programática, diferente do HQL que é uma linguagem de SQL mais orientado a objeto. Através do Criteria é possível programar a consulta através de manipulação dinâmica de um conjunto de critérios.

Por exemplo:

```
c.addCriteria(Expression.eq("nome", "Joao")); (GALHARDO; LIMA, 2008)
```

4.2.1.7 – CACHE

Cache é uma técnica utilizada para aprimorar o desempenho da aplicação no contexto do banco de dados, onde é feito uma copia local dos dados evitando assim o acesso ao banco de dados toda vez que for feita uma consulta no mesmo.

Suporte a cachê de objetos e de consultas.

Citar um exemplo de utilização. Por exemplo: resultado de um concurso.

Tipo de Cache: **escopo de transação, escopo de processo, e escopo de cluster** (GALHARDO; LIMA, 2007).

4.2.1.8 TRANSAÇÕES

Uma transação é uma unidade de execução indivisível (ou atômica), ou seja, todas as etapas pertencentes a uma transação são completamente finalizadas ou nenhuma delas termina. A partir deste modelo definimos que uma operação só vai ser finalizada (gravada no banco de dados) se durante todo o processo não houver nenhum problema (falta de energia, por exemplo), ou seja, na ocorrência de erro em alguma das operações dentro da transação todas as operações realizadas desde o início podem ser revertidas e as alterações no banco de dados desfeitas, garantindo assim, a unicidade do processo

Para que um banco de dados garanta a integridade dos seus dados deve possuir quatro características:

Atomicidade: O banco de dados deve garantir que todas as transações sejam indivisíveis ou atômicas.

Consistência: Após a execução de uma transação, o banco de dados deve continuar consistente, ou seja, deve continuar com um estado válido.

Isolamento: Mesmo que várias transações ocorram paralelamente, nenhuma transação deve influenciar nas outras.

Durabilidade: Após a finalização de uma transação, todas as alterações feitas por ela no banco de dados devem ser duráveis, mesmo havendo falhas no sistema após a sua finalização. (GALHARDO; LIMA, 2007).

4.3 LDAP (*Lightweight Directory Access Protocol*)

LDAP significa *Lightweight Directory Access Protocol*, ou seja, protocolo leve de acesso a diretórios sobre LANs (*Local Area Network*), o que vai de acordo com OLIVEIRA, et. al. (2004, p.2) que diz que o LDAP pode ser utilizado para integração de informações gerenciadas, para que usuários de rede, aplicações autônomas e ferramentas de gerência possam acessar as informações, utilizando protocolos básicos, de maneira padronizada.

Serviço de diretório é um banco de dados otimizado para ler, navegar e procurar. Os diretórios tendem a conter descritivos, atributos/características e um suporte sofisticado para filtros. Os serviços de diretórios geralmente não suportam complicadas transações de registros, grandes volumes de dados, encontradas atualmente em sistema de banco de dados. (LDAP, 2008).

Um serviço de diretório LDAP é um processo de servidor que utiliza o protocolo TCP para conexões em sua porta, que por padrão é a 389, para realizar operações em um diretório que obedece a um esquema estabelecido pelo administrador da rede. O diretório é constituído por entradas que possui um identificador distinto, denominado *DN (Distinguished Name)*, e que podem possuir diversos pares de atributo e valor. Estes atributos obedecem a um *Object Class*, que por sua vez é padronizado em *RFC* e que foi previamente definido no esquema do diretório LDAP, quando o administrador de rede realizou a configuração. (OLIVEIRA, et al., 2004).

4.4 JNDI (*Java Naming Directory Interface*)

A interface de diretório de nomes *JNDI*, prover uma atribuição de nomes e diretórios e funcionalidades para aplicações escritas na linguagem de programação JAVA. É projetado para ser independente de qualquer diretório ou serviço específico da implementação do serviço. (SUN MICROSYSTEMS, 2008).

Dentro de um sistema em que componentes e serviços podem estar espalhados por diversas camadas, é necessário haver um lugar central unificado para esses componentes e serviços se registrarem.

É isso que o *JNDI* é, e ele permite que todas as várias partes da aplicação localizem umas as outras de uma maneira muito simples e consistente. (NICK, 2003;) Como o *JNDI* é independente de qualquer implementação específica, as aplicações podem usá-lo para acessar múltiplos serviços de nomeação e diretório, incluindo os serviços de nomeação e diretório existentes como *LDAP*, *NDS*, *DNS*⁶ e *NIS*. Isso é permite que aplicações JEE coexistam em aplicações e sistemas legados. (PAWLAN et al, 2002).

4.5 X.500

X.500 é o diretório padrão OSI, que define uma lista completa de serviços, incluindo um modelo de informações, um diretório de nomes, um modelo funcional, e uma *framework* de autenticação X.500 que também define o Directory Access Protocol (DAP) utilizados pelas máquinas clientes para acesso ao diretório.(HOWES, 1995).

4.6 OPENLDAP

O *Open LDAP* é um Servidor de LDAP *Open Source* desenvolvido para Sistemas Unix.

Consiste essencialmente de dois serviços: ***slapd*** e ***slurpd***.

Slapd (*Stand-alone LDAP Daemon*):

⁶ - *Domain Name System* é um sistema que armazena informações sobre nome de *hosts* e domínios, um tipo de banco de dados distribuído na rede.

O *slapd* é um serviço *LDAP* autônomo (desenhado para correr como um servidor único) responsável por escutar conexões nas portas definidas, que podem ser uma ou mais (tipicamente é usada a porta 389). A autonomia permite ao servidor tomar vantagem sobre o *cache*, gerir os acessos concorrentes à Base de Dados e conservar os recursos do sistema.

Slurpd (*Stand-alone LDAP Update Replication Daemon*).

O *slurpd* é um Serviço *LDAP* autônomo de atualização e replicação de dados entre as Bases de Dados dos vários servidores. Permite propagar as alterações de uma Base de Dados *slapd* para outra. (ARAUJO, 2008).

4.7 – JASPER REPORTS

O *Japer Reports* é uma biblioteca para a geração de relatórios *open source* e escrita em JAVA. (ABRANTES, SENGER, 2006).

O Processo de desenvolvimento de um relatório se inicia com a codificação da definição do relatório em um arquivo *XML* usando tags e atributos definidos no arquivo *jasperreports.dtd*. O próximo passo está na criação do arquivo *.jrxml* que é a extensão do arquivo onde fica o *template* do relatório e também onde é definida toda a estrutura do relatório como por exemplo, os textos, imagens, cálculos, parâmetros entre outros. Logo após esse processo, como ilustra a figura 11, é preciso compilar o arquivo *.jrxml*, onde o resultado da compilação é gerado um arquivo com a extensão *.jasper*, que representa uma classe em JAVA compilada que representa um objeto da classe *JasperReports*.

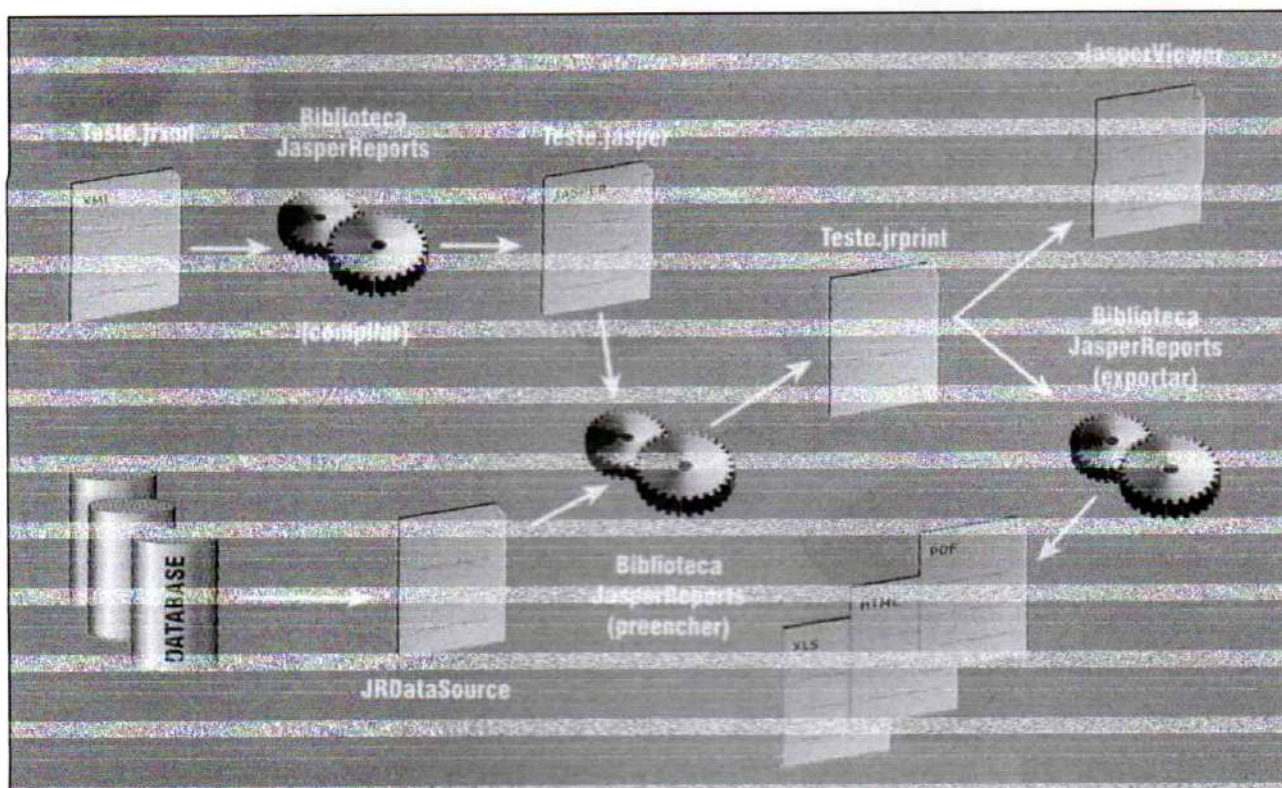


Figura 12: Processo de desenvolvimento de um relatório com JasperReports.
Fonte: (ABRANTES, SENGER, 2006).

4.7.1 FONTE DE DADOS

Para a construção de um relatório é necessária a existência de uma origem de dados para que o relatório possa ser preenchido.

A biblioteca *JasperReports* oferece sustentação a vários tipos de origem de dados e usa uma relação chamada *JRDataSource*, que é uma “representação abstrata de uma origem dos dados de *JasperReports*” (JASPERREPORTS, 2008). A fonte de dados pode ser uma conexão ao banco de dados e uma consulta em SQL.

4.7.2 SUB-RELATÓRIOS

Sub-relatório é um elemento que faz a ligação com outro relatório, ou seja, é um relatório dentro de outro. (ABRANTES, SENGER, 2006).

Sub-relatórios são úteis quando é criado um tipo de detalhe-mestre de um relatório, ou quando a estrutura de um único relatório, não é suficiente para

descrever a complexidade do documento desejado. Um Sub-relatório é, de fato, um relatório normal incorporado em outro relatório. Qualquer modelo de relatório pode ser usado como um Sub-relatório quando incorporados em outro relatório. Tal como outros elementos do relatório, o elemento do Sub-relatório tem uma expressão que é avaliada no momento da execução, a fim de obter a fonte do objeto Sub-relatório. (JASPERREPORTS, 2008).

4.7.3 – IREPORT

O *Ireport* é uma ferramenta visual que usa o *JasperReports* internamente e facilita muito o trabalho do desenvolvedor na criação de relatórios, dos mais simples aos mais complexos. No *Ireport* é possível editar visualmente um arquivo de *template* de relatório (*.jrxml*), e gerar o arquivo *.jasper*, entre outras operações. (ABRANTES, SENGER; 2006), como ilustra a figura 13, onde mostra a tela inicial do *Ireport*:

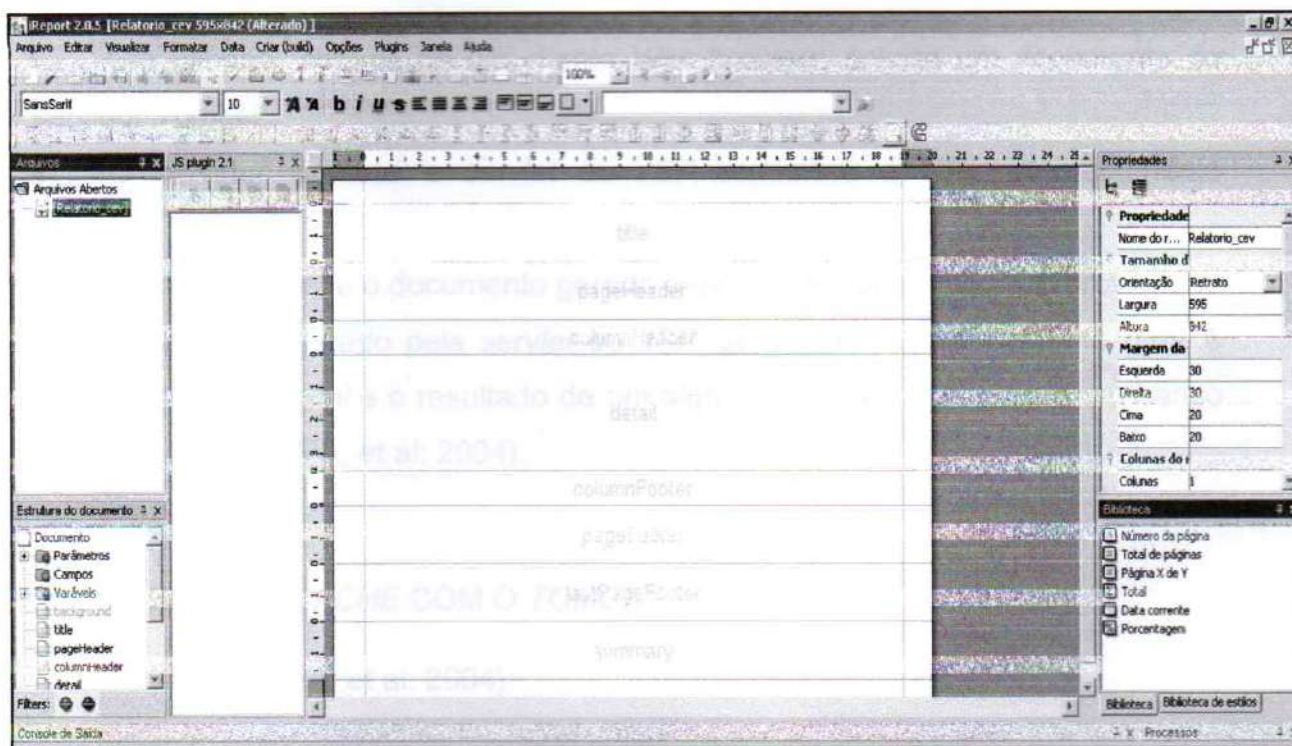


Figura 13: Tela inicial do Ireport.
Fonte: Dados da pesquisa

4.8 TOMCAT

O *Apache Jakarta Tomcat* é um subprojeto do projeto *Jakarta*. Ele é um *container Web open-source*, desenvolvido e mantido pela Fundação *Apache*. Ele permite a execução de aplicações voltadas para *Web* centrando nas tecnologias *Servlet* e *Java Server Pages* (DEVMEDIA, 2008).

Desde 1995, quando foi criado o projeto *Apache*, o objetivo de estender o uso do *Apache* em pequenas, médias e grandes corporações, instituições de pesquisa, estudantes, ambientes *Intranet* tem sido uma evolução constante devido à disponibilidade e confiabilidade do *Apache*, utilizado em 50,4% dos servidores *Web* no mundo⁷.

O servidor *Tomcat* tem a capacidade de converter automaticamente qualquer página JSP em um *servlet* equivalente. Ou seja, o *Tomcat* é capaz de criar código-fonte em JAVA a partir de um documento *html*. A criação dinâmica do código funciona da seguinte forma:

- Um usuário, através de um cliente *Web-browser*, solicita um documento do servidor *Tomcat*;
- O servidor executa a *servlet* ou *JSP* específica à *URL* do documento recebido.
- O usuário recebe o documento gerado e exibe o resultado no *Web-browser*.
- O conteúdo gerado pela *servlet* ou *JSP*, geralmente um documento *html*, é formado por *tags html* e o resultado de um algoritmo JAVA ou acesso a um banco de dados. (OLIVEIRA, et al; 2004).

4.8.1 INTEGRAR APACHE COM O TOMCAT

Segundo (OLIVEIRA, et al. 2004):

⁷ - http://news.netcraft.com/archives/Web_server_survey.html

O Apache atende apenas requisições de páginas estáticas. Para atender requisições dinâmicas, será utilizado o *Tomcat* para atender especificamente as requisições de páginas dinâmicas. A integração entre *Tomcat* e *Apache* permite que este atenda requisições de páginas estáticas (código *html*, imagens, etc.) enquanto aquele atende requisições dinâmicas (*Servlets* e *Java Server Pages*).

Para realizar a integração entre o *Tomcat* e o *Apache* é necessário um elemento connector, sendo adotado nesse projeto o *mod_jk*.

O módulo JK2 é uma reescrita do módulo JK que além de adicionar novas funcionalidades simplificou muito sua configuração. Apesar de ser compatível com o Apache 1.3 o JK2 foi escrito tendo o Apache 2 em vista, pois ele é mais adequado para servidores multi-threads. O JK2 incrementa a modularidade e possui uma clara separação entre camadas físicas e de protocolo. (LIMA, 2003).

5 METODOLOGIA

Tecnológica

Como o objetivo deste estudo envolve a implementação de uma aplicação, defini-se assim, como sendo uma pesquisa Tecnológica.

Segundo MEDEIROS, Juliano Rafael; *apud*, Jung (2004), a presente pesquisa científica é relatada como tecnológica, a qual utiliza conhecimentos básicos, tecnologias existentes e conhecimentos tecnológicos que tem como objetivo um novo produto ou processo.

Exploratória

A pesquisa exploratória é o primeiro passo de todo trabalho científico. São finalidades de uma pesquisa exploratória, sobretudo quando bibliográfica, proporcionar maiores informações sobre determinado assunto; facilitar a delimitação de um tema de trabalho; definir os objetivos ou formular as hipóteses de uma pesquisa ou descobrir novo tipo de enfoque para o trabalho que se tem em mente. (ANDRADE, 1998).

Estudo de Caso

De acordo com Jung (2004), "Tanto nas engenharias como na computação são necessários estudos anteriores para que se possa determinar como está funcionando o processo ou como é o processo, para que posteriormente seja recomendado, desenvolvido e implantado um novo sistema, produto ou técnica de otimização".

6 RESULTADOS

O sistema em questão visa automatizar as etapas de realização e controle para instituições/empresas que ofereçam cursos e/ou eventos ao público em geral, onde dentre suas funções podemos destacar:

- Cadastro de Consultores;
- Cadastro de Cursos/Eventos;
- Gerenciamento de Inscrições – Pessoa Física e Jurídica;
- Geração de Lista de Presença;
- Certificado de Participação;
- Relatórios Gerenciais;
- Entre outros.

6.1 MODELAGEM

6.1.1 CASOS DE USO

Um caso de uso define um conjunto de instâncias de casos de uso, no qual cada instância é uma seqüência de ações realizada por um sistema que produz um resultado de valor observável para determinado ator.

Os Casos de Uso têm por finalidade especificar o comportamento do sistema, sendo uma descrição de um conjunto de ações. Ele especifica o comportamento do sistema, sem ser necessário descrever como esse comportamento será implementado. A vantagem disso é a possibilidade do desenvolvedor se comunicar com o cliente sem se preocupar com os detalhes de como será implementado determinado comportamento do sistema.

O caso de uso foi utilizado para descrever todas as operações realizadas pelos usuários que irão utilizar o sistema de gerenciamento de cursos e eventos

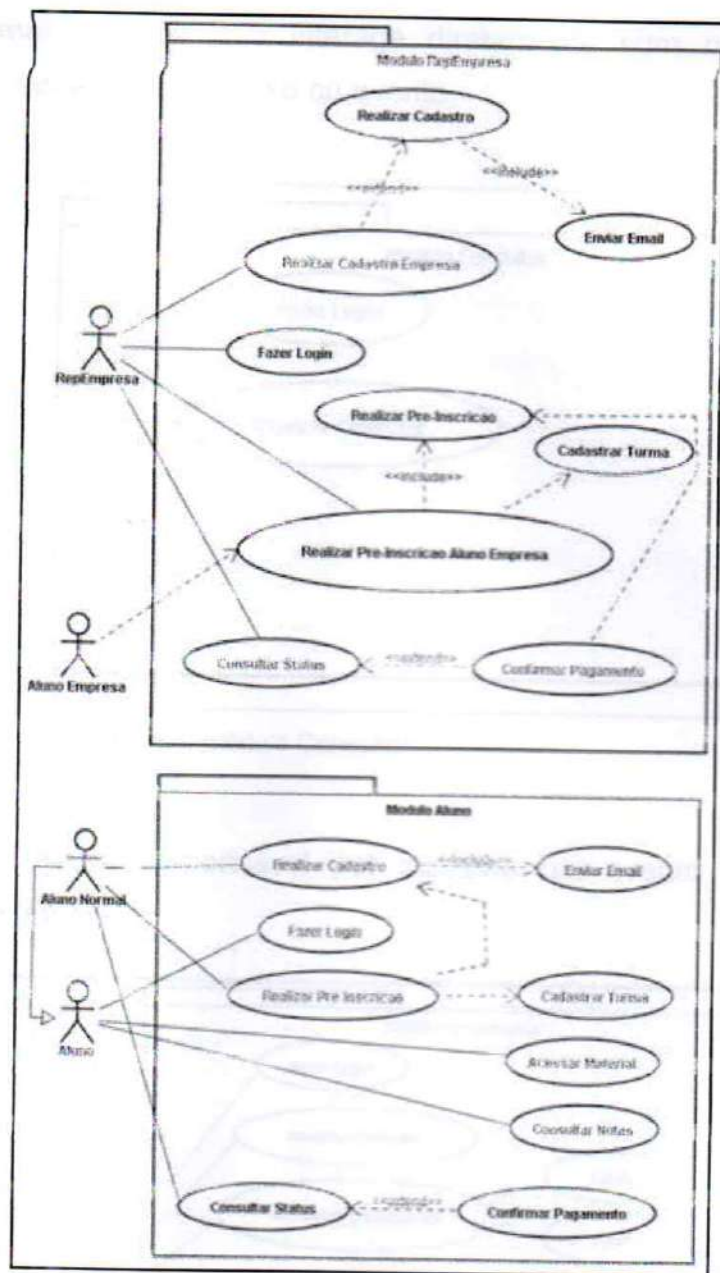


Figura 14.1: Casos de Uso: módulos Aluno e RepEmpresa
Fonte: Dados da pesquisa

Atores:

RepEmpresa (Representa da Empresa): Pessoa responsável por realizar inscrições de vários alunos em determinados cursos ou eventos pertence a algum setor de uma empresa.

AlunoEmpresa: Pessoa identificada por possuir relacionada a si uma empresa que oferecerá o curso ou evento.

AlunoNormal: Pessoa que interage diretamente com o sistema efetuando a inscrição em determinado curso ou evento.

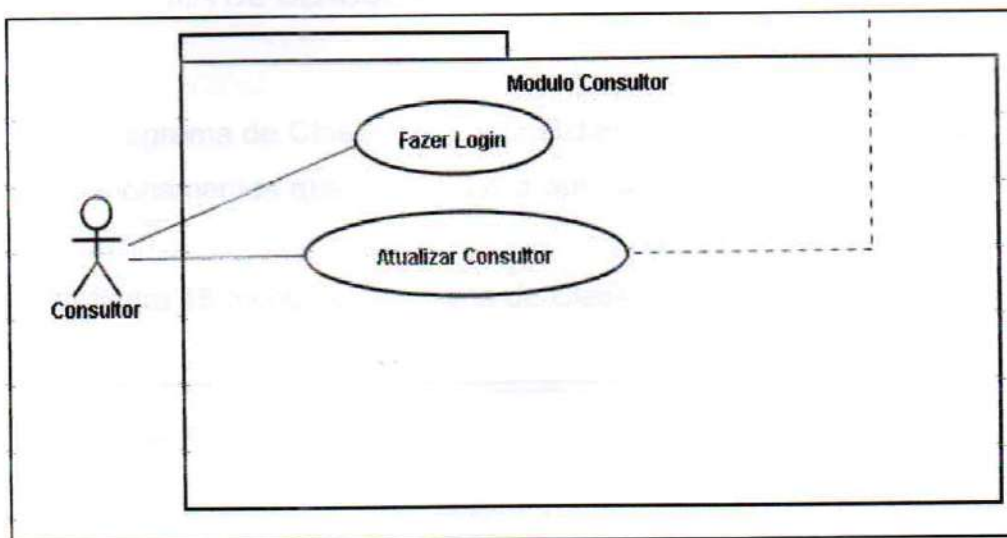


Figura 14.2: Casos de Uso: módulo Consultor
Fonte: Dados da pesquisa

Consultor: Pessoa responsável por ministrar um evento ou uma determinada disciplina de um curso.

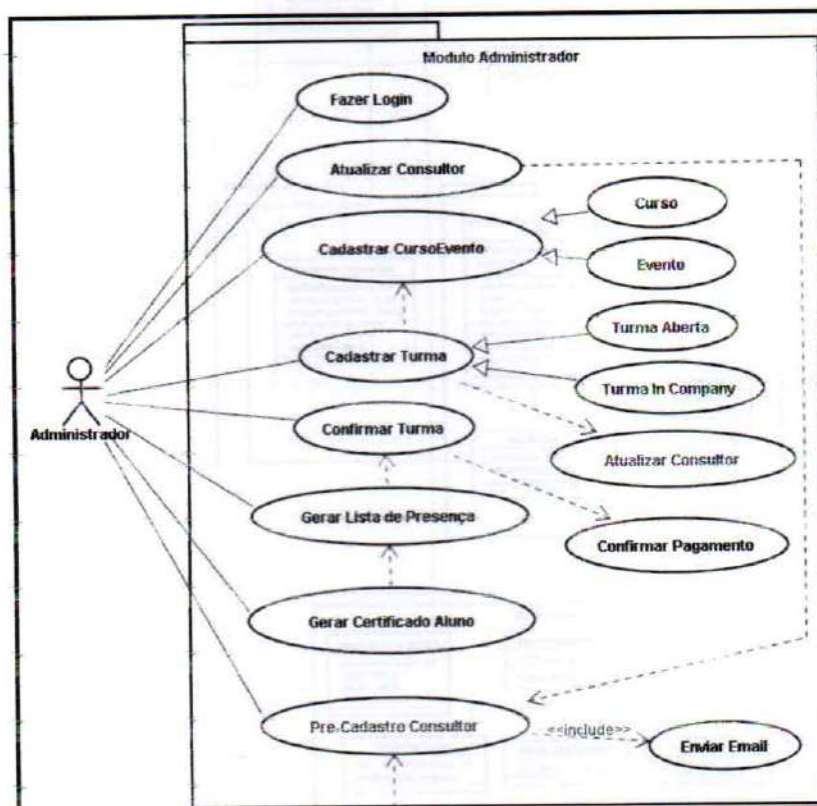


Figura 14.3: Casos de Uso: módulo Administrador
Fonte: Dados da pesquisa

Administrador: Pessoa responsável por gerenciar todos os cadastros do sistema.

6.1.2 DIAGRAMA DE CLASSES

O Diagrama de Classes tem a finalidade de visualizar as classes, interfaces e seus relacionamentos que integrarão a aplicação, mostrando os aspectos estáticos da mesma.

A figura 15 mostra o Diagrama de Classe implementado para este trabalho:

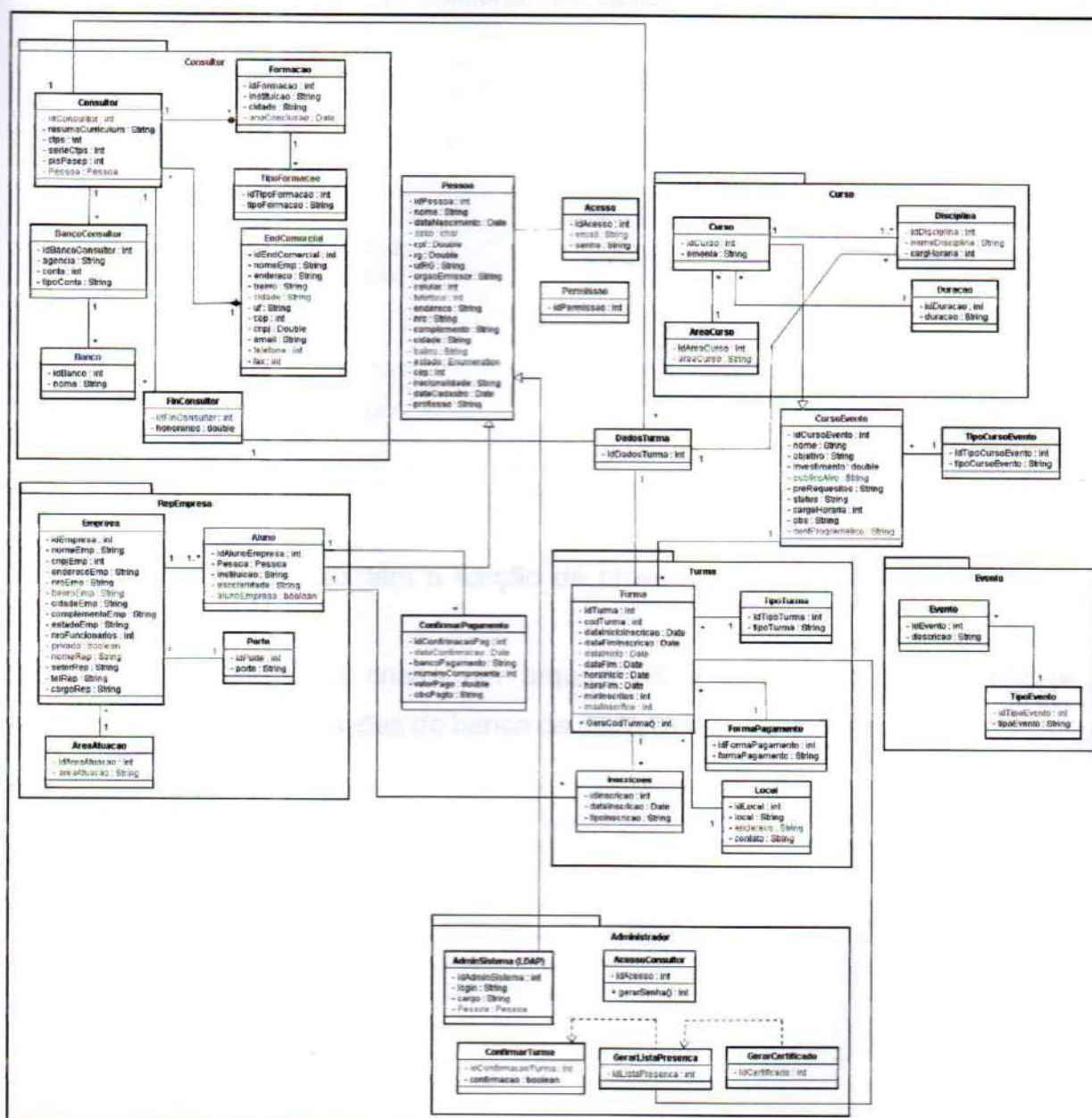


Figura 15: Diagrama de Classe
Fonte: Dados da pesquisa

6.2 DESENVOLVIMENTO

6.2.1 HIBERNATE

Para o processo de armazenamento dos dados da aplicação, se faz necessário a implementação de uma estratégia de persistência que permita vincular as classes JAVA (orientadas o objeto) ao banco de dados *PostgreSQL*. Para isso, foi implementada uma solução baseada no *framework JPA (Java Persistence API)*, o *Hibernate*.

O primeiro passo foi adicionar as bibliotecas do *Hibernate* para realizar o mapeamento:

```
antlr-2.7.5H3.jar
asm.jar
asm-attrs.jar
cglib-2.1.jar
commons-collections-2.1.1.jar
commons-logging-1.0.4.jar
dom4j-1.6.jar
ehcache-1.1.jar
ejb3-persistence.jar
hibernate3.jar
hibernate-annotations.jar
jta.jar
jtds-1.2.jar
log4j-1.2.9.jar
```

Essas bibliotecas têm a função de prover o serviço de persistência dos na aplicação.

Em seguida, foi criado um arquivo XML para definir as estratégias de mapeamento, configurações do banco de dados e classes mapeadas:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"hibernate-configuration"
<session-factory name="java:comp/env/hibernate/SessionFactory">
  <!-- properties -->
  <property name="connection.driver_class">
    org.postgresql.Driver</property>
  <property name="connection.url">jdbc:postgresql://localhost:5432/gce</property>
  <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
  <property name="show_sql">true</property>
  <property name="connection.username">root</property>
  <property name="connection.password">l2carlos</property>
  <property name="connection.pool_size">10</property>
  <property name="hibernate.hbm2ddl.auto">update</property>
  <!--<property name="hibernate.hbm2ddl.auto">create</property>-->
  <property name="cache.use_second_level_cache">true</property>
  <!-- classes mapeadas -->
  <!-- Dominio -->
  <mapping class="br.org.iel.rn.dominio.Acesso"/>
  <mapping class="br.org.iel.rn.dominio.DadosTurma"/>
  <mapping class="br.org.iel.rn.dominioCursoEvento"/>
  <mapping class="br.org.iel.rn.dominio.Pessoa"/>
  <mapping class="br.org.iel.rn.dominio.TipoCursoEvento"/>
  <!-- Empresa -->
  <mapping class="br.org.iel.rn.dominio.empresa.AreaAtuacao"/>
  <mapping class="br.org.iel.rn.dominio.empresa.Empresa"/>
  <mapping class="br.org.iel.rn.dominio.empresa.Aluno"/>
  <mapping class="br.org.iel.rn.dominio.empresa.Porte"/>
  <!-- Administrador -->
  <!--<mapping class="br.org.iel.rn.dominio.administrador.AdminSistema"/>-->
  <!-- Consultor -->
  <mapping class="br.org.iel.rn.dominio.consultor.Banco"/>
  <mapping class="br.org.iel.rn.dominio.consultor.BancoConsultor"/>
  <mapping class="br.org.iel.rn.dominio.consultor.Consultor"/>
  <mapping class="br.org.iel.rn.dominio.consultor.EndComercial"/>
  <mapping class="br.org.iel.rn.dominio.consultor.FinConsultor"/>
  <mapping class="br.org.iel.rn.dominio.consultor.Formacao"/>
  <mapping class="br.org.iel.rn.dominio.consultor.TipoFormacao"/>
  <!-- Curso -->
  <mapping class="br.org.iel.rn.dominio.curso.AreaCurso"/>
  <mapping class="br.org.iel.rn.dominio.curso.Curso"/>
  <mapping class="br.org.iel.rn.dominio.curso.Disciplina"/>
  <mapping class="br.org.iel.rn.dominio.curso.Duracao"/>
  <!-- Evento -->
  <mapping class="br.org.iel.rn.dominio.evento.Evento"/>
  <mapping class="br.org.iel.rn.dominio.evento.TipoEvento"/>
  <!-- Turma -->
  <mapping class="br.org.iel.rn.dominio.turma.FormaPagamento"/>
  <mapping class="br.org.iel.rn.dominio.turma.LocalTurma"/>
  <mapping class="br.org.iel.rn.dominio.turma.Inscricoes"/>
  <mapping class="br.org.iel.rn.dominio.turma.TipoTurma"/>
  <mapping class="br.org.iel.rn.dominio.turma.Turma"/>
</session-factory>
</hibernate-configuration>

```

Figura 16: Arquivo de configuração hibernate.cfg.xml

Fonte: Dados da Pesquisa

A partir desse arquivo que o *Hibernate* reconhece as classes que serão mapeadas para o banco.

O seguinte trecho do código tem a função de fornecer as propriedades da conexão onde é visualizado o banco de dados, o endereço do servidor onde está hospedado a aplicação, a senha de acesso a base, entre outras configurações que podem ser adicionadas:

name="connection.driver_class">org.postgresql.Driver	Nome da classe do <i>driver JDBC</i> do banco de dados que está sendo utilizado;
name="connection.url">jdbc:postgresql://localhost:5432/gce	É a <i>URL</i> de conexão específica do banco que está sendo utilizado;
name="dialect">org.hibernate.dialect.PostgreSQLDialect	Implementação do dialeto <i>SQL</i> específico do banco de dados a ser utilizado. Usado para identificar as particularidades do banco de dados;
name="show_sql">true	Utilizado para definir se os <i>SQLs</i> gerados pelo Hibernate devem ou não ser exibidos (<i>true</i> <i>false</i>).
name="connection.username">root	Nome de usuário com o qual o Hibernate deve se conectar ao banco;
name="connection.password">12carlos	Senha do banco de dados;
name="connection.pool_size">10	Tamanho do <i>pool</i> de conexões.

Quadro 3 – Parâmetros do arquivo de configuração hibernate.cfg.xml
Fonte: Elaboração própria (2008)

Por último, foi feito as anotações nas classes da aplicação, onde são definidos as chaves primárias das classes e os relacionamentos das mesmas. Como mostra a figura 5, onde foi criada a classe consultor junto com seus atributos.

```

package br.org.iel.rn.dominio.consultor;

import br.org.iel.rn.dominio.*;
import br.org.iel.rn.dominio.evento.Evento;
import java.util.Collection;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
import org.hibernate.annotations.Cascade;
import org.hibernate.annotations.CascadeType;

@Entity
@Table(name = "Consultor")
@PrimaryKeyJoinColumn(name = "idPessoa")
public class Consultor extends Pessoa implements java.io.Serializable {

    private String resumoCurriculum;
    private int ctps;
    private String serieCtps;
    private int pisPasep;
    private int fax;

    @OneToMany(mappedBy = "consultor")
    @Cascade(CascadeType.ALL)
    private Collection<DadosTurma> dadosTurmas;

    @OneToMany(mappedBy = "consultor", fetch = FetchType.LAZY)
    @Cascade(CascadeType.ALL)
    private Collection<Formacao> formacoes;

    @OneToMany(mappedBy = "consultor")
    @Cascade(CascadeType.ALL)
    private Collection<BancoConsultor> bancoConsultores;

    @OneToMany(mappedBy = "consultor", fetch = FetchType.LAZY)
    @Cascade(CascadeType.ALL)
    private Collection<FinConsultor> finConsultores;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "endComercial")
    private EndComercial endComercial;

    public String getResumoCurriculum() {
        return resumoCurriculum;
    }

    public void setResumoCurriculum(String resumoCurriculum) {
        this.resumoCurriculum = resumoCurriculum;
    }

    public int getCtps() {
        return ctps;
    }
}

```

Figura 17: Classe consultor mapeada com anotações do Hibernate
 Fonte: Dados da pesquisa

```
@OneToMany(mappedBy = "consultor")
@Cascade(CascadeType.ALL)
private Collection<DadosTurma> dadosTurmas;
```

O exemplo anterior usa um mapeamento Um para Vários (*@OneToMany*), ou seja, um consultor possui varias turmas vinculadas. O *@Cascade(CascadeType.ALL)* indica com que ação em cascata o relacionamento será tratado, caso do *.ALL* especifica a junção de *delete* e *save-update*;

6.2.2 JSP

Para que a aplicação fosse implementada, foi utilizada a linguagem *JSP* (Java Server Pages) que faz parte da tecnologia *JEE* (Java Enterprise Edition) onde os códigos gerados são executados no servidor de aplicação.

Uma página *Jsp* permite a inserção de código *html*, em que consiste a criação das páginas visualizadas pelos *browsers*.

GCE - Consultor

: Consultor :

- » [Nova Consultor](#)
- » [Gerar Acesso](#)

::Curso :

- » [Nova Curso](#)

: Evento :

- » [Nova Evento](#)

: Gerenciamento de Turmas :

- » [Nova Turma](#)
- » [Confirmar Turma](#)
- » [Turmas Abertas](#)

Logado no Sistema como: null

Novo Consultor

Informações Pessoais

Nome		Sexo	☐ Masculino ☐ Feminino	Data de Nascimento	
CPF		RG		Orgão Emissor	
UF	AC	CTPS		Serie CTPS	
DIS PASEP		Nacionalidade			
Email					

Informações de Contato

Rua				N.º	
Complemento		Bairro		Cidade	
Estado	AC	Cep			
Celular		Telefone		Fax	

Endereço Corporativo

Empresa		CNPJ		Sector	
Endereço		Bairro		Cidade	
Estado	AC	Telefone		Fax	
Email					

Formação

Resumo do Curriculum

Cadastrar >>
Limpar

Figura 18: Tela de cadastro de consultor
Fonte: Dados da pesquisa

O exemplo anterior mostra a tela de cadastro de consultor do sistema de cursos e eventos, onde a JSP de nome *cadConsultor1.jsp* tem como ação a submissão dos dados para a Jsp *addConsultor.jsp* referida na seguinte linha de código:

```
<form action="addConsultor1.jsp" method="get">
```

A JSP *addConsultor.jsp* tem com principal função, recuperar todas informações digitadas no tela *cadConsultor1.jsp* e sem seguida submeter os dados para o banco de dados, onde pode ser visualizada a seguir:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="java.text.Format"%>
<jsp:useBeanid="consultor"
class="br.org.iel.rn.dominio.consultor.Consultor" />
<jsp:useBeanid="endComercial" class="br.org.iel.rn.dominio.consultor.EndCome
rcial" />
<jsp:setProperty name="consultor" property="*" />
<jsp:setProperty name="endComercial" property="*" />
<html>
  <head>
    <%
      SimpleDateFormat data_dia = new SimpleDateFormat("dd/mm/yyyy");
      java.util.Date hoje = new java.util.Date();
      String data = data_dia.format(hoje);
    %>
    <title></title>
  </head>
  <body>
    <%
      GeralDAO daoend = new GeralDAO();
      consultor.setEndComercial(endComercial);
      daoend.create(endComercial);
      GeralDAO daocon = new GeralDAO();
      consultor.setDataCadastro(hoje);
      consultor.setEndComercial(endComercial);
      consultor.setEstado(request.getParameter("estado"));
      daocon.create(consultor);
      response.sendRedirect("cadConsultor2.jsp");
    %>
  </body>
</html>
```

Como visto na modelagem, um consultor possui um endereço comercial, por isso a aplicação importa os dados da classe *endComercial*:

```
<jsp:useBeanid="endComercial" class="br.org.iel.rn.dominio.consultor.EndCome
rcial" />
```

Em seguida, é instanciado um objeto da classe `endComercial` e setado essa informação junto com o consultor que esta sendo criado, e por ultimo, os dados são persistidos para o banco de dados: `daocon.create(consultor);`

6.2.3 OPENLDAP

A implementação do controle de acesso é importante para distinguir os usuários que terão acesso a aplicação. O *OpenLDAP* é uma solução baseada no software da "Universidade de Michigan" (os quais fornecem a informação necessária para criação de implementações *LDAP*) o qual cumpre os objetivos essenciais de um servidor de diretórios *LDAP* além não possuir custo de licença de uso e obedecer a padronização estabelecida na *RFC*⁸ 2251.

6.2.3.1 BIBLIOTECAS QUE IMPLEMENTAM O OPENLDAP

Para que a aplicação forneça as condições de implementar o controle de acesso através da base *OpenLDAP*, se faz necessário adicionar no ambiente de desenvolvimento as bibliotecas do *JNDI* (*JAVA Naming Directory Interface*) que funcionará como provedor de serviço (*services provider*) do *LDAP*.

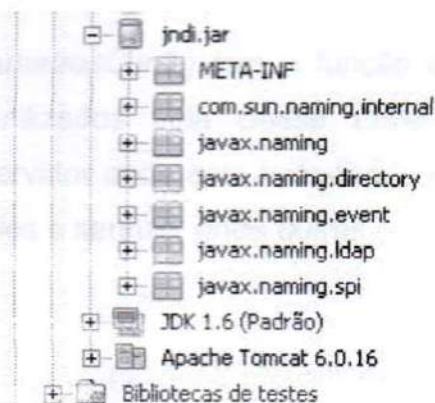


Figura 19: Bibliotecas JNDI para acesso do LDAP
Fonte: Sun Microsystems

⁸ RFC é um documento que descreve os padrões de cada protocolo da Internet, onde os RFC são documentos públicos, qualquer pessoa tem acesso a eles, podendo ler, comentar, enviar sugestões e relatar experiências sobre o assunto.

O passo seguinte se constituiu na criação de duas classes JAVA na aplicação: **LDAPParmetrosConfig** e **LDAPServiceLocator**.

LDAPParmetrosConfig.

```
public class LDAPParmetrosConfig {
    public static String INITIAL_CTX = "com.sun.jndi ldap.LdapCtxFactory";
    public static String SERVIDOR = "ldap://localhost:389";
    public static String SEARCHBASE = "ou=ti, dc=krlos, dc=com, dc=br";
    public static String ADMIN_DN = "cn=admin, dc=krlos, dc=com, dc=br";
    public static String ADMIN_PW = "123456";
    public static String BASE_DN = "dc=krlos, dc=com, dc=br";
}
```

<code>public static String INITIAL_CTX</code>	Implementação do Initial context para LDAP
<code>public static String SERVIDOR</code>	Servidor LDAP
<code>public static String SEARCHBASE</code>	Base de busca
<code>public static String ADMIN_DN</code>	Nome distinto do admin
<code>public static String ADMIN_PW</code>	Senha
<code>public static String BASE_DN</code>	"top level" do diretório (base DN)

Quadro 4: Parâmetros da classe LDAPParmetrosConfig.
Fonte: Elaboração própria, 2008

A classe **LDAPParmetrosConfig** tem a função de fornecer os parâmetros necessários que serão utilizados pela classe **LDAPServiceLocator** como por exemplo, o endereço do servidor onde esta hospedado o serviço LDAP, a base de busca dos nomes de usuários e senhas, entre outras.

LDAPServiceLocator:

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;

public class LDAPServiceLocator {
    protected static LDAPServiceLocator instance;
    private LDAPServiceLocator() {
```

```

        super();
    }
    /**
     * Obtém a mesma instancia de LDAPServiceLocator para todas as chamadas
     * @return um objeto LDAPServiceLocator
     */
    public static LDAPServiceLocator getInstance() {
        if (instance == null) {
            instance = new LDAPServiceLocator();
        }
        return instance;
    }
    public DirContext getContext() {
        Hashtable env = new Hashtable(2);
        // Especifica a fabrica de INITIAL CONTEXT
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            LDAPPParametrosConfig.INITIAL_CTX);
        // Especifica o IP/Nome e a porta do servidor LDAP
        env.put(Context.PROVIDER_URL, LDAPPParametrosConfig.SERVIDOR);
        // Usado quando o servidor LDAP não permite busca como anonymous
    };

    DirContext ctx = null;
    try {
        // Obtém um Initial Context
        ctx = new InitialDirContext(env);
    } catch (NamingException e) {
        System.out.println("Não foi possível obter um contexto LDAP");
        e.printStackTrace();
    }
    return ctx;
}
}

```

A classe **LDAPServiceLocator** é responsável por implementar a principal interface do serviço de nomes LDAP: a *Context*. É através dessa interface que é realizada a busca dos usuários que irão utilizar o sistema. A operação mais importante é a de *lookup*, em que se procura por um determinado nome no serviço de nomes e o objeto é retornado.

Para a criação de um *InitialContext* tem que informar o parâmetro da fabrica do *InitialContext* visualizado na seguinte linha:

```

env.put(Context.INITIAL_CONTEXT_FACTORY,
    LDAPPParametrosConfig.INITIAL_CTX);

```

Onde foi definido na classe na **LDAPPParametrosConfig**.

6.2.4 IREPORT

Para utilização do *Ireport*, foi usada a classe consultor para a criação da parte gráfica do relatório.

Para esse trabalho foi utilizada a versão 2.0.5 do *Ireport*. O primeiro passo é definir as configurações da fonte de dados utilizada nos relatórios.

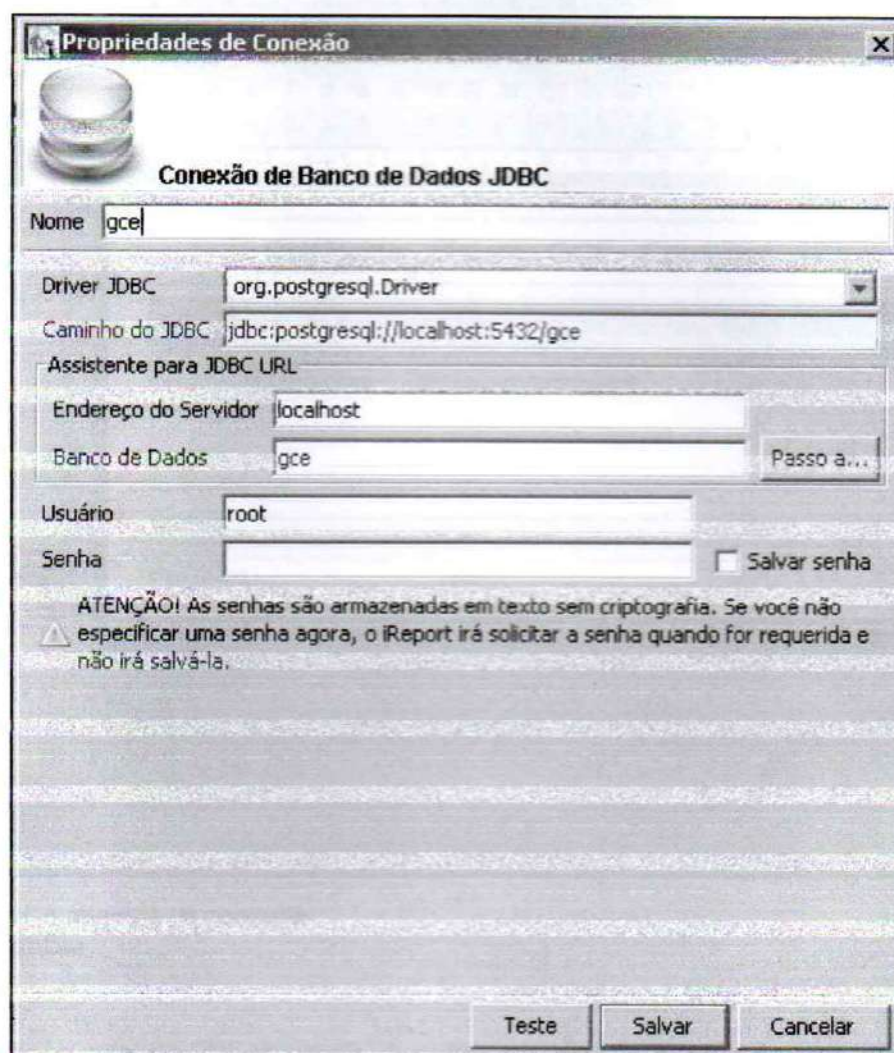


Figura 21: Propriedade da conexão com o banco de dados
Fonte: Dados da pesquisa

Como na modelagem da aplicação, existe a tabela consultor, que herda de pessoa os seus atributos, foi criado um relatório para mostrar o Nome do Consultor, sua Idade e o Resumo do seu Curriculum, definida na seguinte *Query SQL*:

```
select p.nome, p.datanascimento, c.resumocurriculum
from consultor as c, pessoa p
where p.idpessoa = c.idpessoa;
```

Em seguida, foi criado um novo relatório e inserido essa *Query*, onde depois de finalizado, foi gerado o arquivo com relatório gráfico.

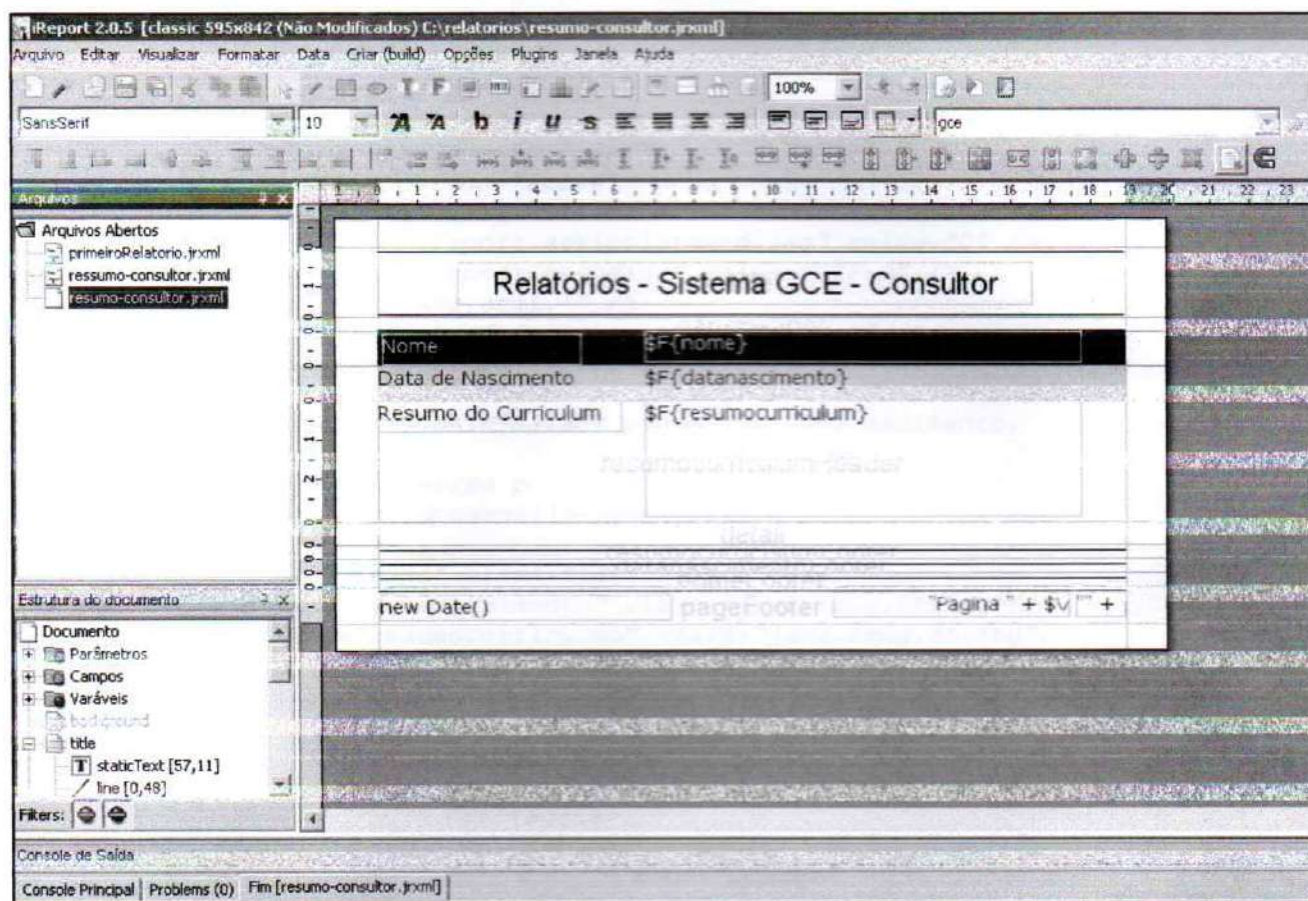


Figura 22: Ireport com o relatório de consultor

Fonte: Dados da pesquisa

Quando o projeto do relatório é salvo no disco, é gerado um arquivo com extensão “.JRXML”, que corresponde ao projeto do relatório. O projeto pode ser também compilado, a fim de gerar o arquivo com extensão “.JASPER”, que será salvo no disco e estará pronto para receber parâmetros e dados na operação de preenchimento do relatório. Na figura 20 é mostrada o exemplo do arquivo JRXML gerado pelo *Ireport*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Created with iReport - A designer for JasperReports -->
```

```

<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport
    name="resumo-consultor"
    columnCount="1"
    printOrder="Vertical"
    orientation="Portrait"
    pageWidth="595"
    pageHeight="842"
    columnWidth="535"
    columnSpacing="0"
    leftMargin="30"
    rightMargin="30"
    topMargin="20"
    bottomMargin="20"
    whenNoDataType="NoPages"
    isTitleNewPage="false"
    isSummaryNewPage="false">
    <property name="ireport.scriptlethandling" value="0" />
    <property name="ireport.encoding" value="UTF-8" />
    <import value="java.util.*" />
    <import value="net.sf.jasperreports.engine.*" />
    <import value="net.sf.jasperreports.engine.data.*" />

    <queryString><![CDATA[select p.nome, p.datanascimento,
c.resumocurriculum
from consultor as c, pessoa p
where p.idpessoa = c.idpessoa]]></queryString>

    <field name="nome" class="java.lang.String"/>
    <field name="datanascimento" class="java.lang.String"/>
    <field name="resumocurriculum" class="java.lang.String"/>

    <group name="nome" >
        <groupExpression><![CDATA[$F{nome}]]></groupExpression>
        <groupHeader>
            <band height="25" isSplitAllowed="true" >
                <rectangle>
                    <reportElement
                        mode="Opaque"
                        x="0"
                        y="0"
                        width="535"
                        height="25"
                        forecolor="#8080FF"
                        backcolor="#000000"
                        key="rectangle"
                        isRemoveLineWhenBlank="true"/>
                    <graphicElement stretchType="NoStretch">
                        <pen lineWidth="0.0" lineStyle="Solid"/>
                    </graphicElement>
                </rectangle>
                <staticText>
                    <reportElement
                        x="3"
                        y="2"
                        width="143"
                        height="23"
                        forecolor="FFFFFF"
                        backcolor="FFFFFF"
                        key="staticText"

```

```

isRemoveLineWhenBlank="true"/>
<box>
  <topPen
lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
  <leftPen lineWidth="0.0" lineStyle="Solid"
lineColor="#000000"/>
  <bottomPen lineWidth="0.0"
lineColor="#000000"/>
  <rightPen lineWidth="0.0" lineStyle="Solid"
lineColor="#000000"/>
</box>

<textElement>
  <font fontName="Verdana" size="14"/>
</textElement>
<text><![CDATA[None]]></text>
</staticText>
<textField isStretchWithOverflow="false" pattern=""
isBlankWhenNull="true" evaluationTime="Now" hyperlinkType="None"
hyperlinkTarget="Self" >

  <reportElement
    x="191"
    y="0"
    width="313"
    height="23"
    forecolor="#FFFFFF"
    backcolor="#FFFFFF"
    key="textField"
    isRemoveLineWhenBlank="true"/>
  <box>
    <topPen
lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
    <leftPen lineWidth="0.0" lineStyle="Solid"
lineColor="#000000"/>
    <bottomPen lineWidth="0.0"
lineColor="#000000"/>
    <rightPen lineWidth="0.0" lineStyle="Solid"
lineColor="#000000"/>
  </box>
  (...)

```

Figura 20: XML gerado pelo Ireport (arquivo do JRXML)
 Fonte: Dados da pesquisa

6.2.5 JASPERREPORT

A inclusão de relatórios em aplicações empresarias é de fundamental importância no contexto dos Sistemas de Informações.

A arquitetura JAVA permite abstrair os dados dos bancos de dados e transformá-las em informações valiosas para as empresas, onde existem ferramentas que auxiliam este trabalho, tal como a biblioteca de classe JAVA chamada JasperReports, que com o auxílio da ferramenta *Ireport* proporciona o desenvolvimento de relatórios para aplicações escritas em JAVA. As classes

pertencentes a biblioteca do *JasperReports*, vêm com um conjunto de métodos que dão suporte ao desenvolvimento do relatório.

O próximo passo é habilitar a aplicação a abrir o relatório gerado no *lreport* através da biblioteca do *JasperReports*. Para isso, foi criada uma classe JAVA chamada *RelatorioJasperConsultor*.

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperRunManager;

public class RelatorioJasperConsultor extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        Map m = new HashMap();

        try {
            Class.forName("com.postresql.jdbc.Driver");

            Connection c =
                DriverManager.getConnection("jdbc:postgresql://localhost:5432/gce", "root",
                    "12carlos");

            InputStream i =
                getClass().getClassLoader().getResourceAsStream("resumo-consultor.jasper");

            byte r[] = JasperRunManager.runReportToPdf(i, m, c);

            response.setContentType("application/pdf");

            OutputStream o = response.getOutputStream();

            o.write(r);
        } catch (SQLException se) {
            se.printStackTrace();
        } catch (ClassNotFoundException cnf) {
            cnf.printStackTrace();
        } catch (JRException jre) {
            jre.printStackTrace();
        }
    }
}
```

A seguinte classe é responsável por buscar no diretório, o relatório criado e compilado no *Ireport*, com a respectiva configuração da base de dados, onde o relatório foi exportado para um arquivo de extensão ".PDF":

```
response.setContentType("application/pdf");
```

Relatórios - Sistema GCE - Consultor	
Nome	CARLOS AUGUSTO RODRIGUES DIAS
Data de Nascimento	13/11/1984
Resumo do Curriculum	Trabalha a mais de dois ano na area de TI onde ja ministrou varios treinamentos pro SENAC/RN
Nome	ANTONIO DIAS RODRIGUES
Data de Nascimento	17/02/1962
Resumo do Curriculum	Formado pela USP/SP, possui especialização em Harvard. Ja ministrou aula na POLY. Atualmente é pesquisador do nucleo de biologia da UFRN.
Nome	SANDRA GOMES DE MELO
Data de Nascimento	15/08/1975
Resumo do Curriculum	Formada pela FGV, ja ministrou palestra na Congresso Brasileiro de Recursos Humanos. Possui uma empresa de consultoria em RH prestando serviço a emopresas como SADIA
30/11/08 21:08	
Pagina 1 of 1	

Figura 23: Relatório gerado pela aplicação em formato PDF

Fonte: Dados da pesquisa

6.2.6 TOMCAT

O que motivou a escolha do *Tomcat* como *contêiner Web* ou servidor de aplicação se deu pelo fato da sua simplicidade de instalação e integração com o ambiente de desenvolvimento, além da rapidez e segurança na execução dos dados enviados ao servidor.

Outro ponto de fundamental importância deve-se ao fato do servidor *Tomcat* ser *open-source* e gratuito, além de permitir a execução de aplicações voltadas para *Web* centrando nas tecnologias *Servlet* e *Java Server Pages*.

8 CONCLUSÃO

Neste trabalho foi desenvolvida uma aplicação Web para desenvolvimento de cursos e eventos utilizando a tecnologia JAVA. Por ser um escopo considerável para uma monografia, ainda falta concluir algumas funcionalidades. No decorrer da pesquisa deste projeto foram encontradas várias dificuldades, como por exemplo, implementar uma base *LDAP* teste, devido à grande complexidade da instalação no sistema operacional *Linux*, que demandava certo conhecimento técnico.

Concluimos que apesar de utilizar várias ferramentas para se atingir o produto final, está comprovado que é possível desenvolver qualquer aplicação utilizando apenas *softwares* livres. E isso se deve ao fato da plataforma JAVA oferecer a portabilidade e centralização das aplicações atuais voltadas para *Web*, para tanta, basta estudar e aprender os novos conceitos impostos pela arquitetura.

9 REFERÊNCIAS

ABRANTES, Ana; SENGER, Yara. **Relatórios na Web passo a passo**; Java Magazine: Devmedia; v. 37; p.52; 2006.

ANDRADE, Maria Margarida de. **Introdução à Metodologia do trabalho científico**; 3. ed. São Paulo: Atlas, 1998.

ARAUJO, Correia. **Análise da ferramenta "Open LDAP" e performance do protocolo LDAP**, Disponível em: <<http://paginas.fe.up.pt/~jrualo/estagio/files/openLDAP.pdf>>. Acesso em: 07 junho. 2008.

BAUER, Christian; KING, Gavin. **Hibernate in Action**; Manning, 2005.

BOMFIM, Junior; FRANCISCO, Tarcizio. **Tecnologia Java na Internet**, São Paulo:Erica, 2002.

CERQUEIRA, Alisson Gomes. **Implementação de módulos PAM e NSS para autenticação segura e distribuída**; Disponível em: <<http://bazar.ginix.ufla.br/index.php/MonosARL/article/viewDownloadInterstitial/33/65>>. Acesso em: 28 maio. 2008.

DEVMEDIA; **CONHEÇA o Apache Tomcat**. Disponível em: <<http://www.devmedia.com.br/articles/viewcomp.asp?comp=4546>>. Acesso em: 03 maio. 2008.

FIELDS, Duane K. e Kolb; MARK, A. **Desenvolvendo com Java Server Pages**; Ciência Moderna; 2000.

GALHARDO, Rafaela; LIMA. Gleydson; **Introdução ao Hibernate (Tutorial)**; Disponível em: <<http://www.jeebrasil.com.br/mostrar/5>>. Acesso em 23 maio. 2008.

GALHARDO, Rafaela; LIMA. Gleydson; **Mapeando associações com Hibernate**; Disponível em: <<http://www.jeebrasil.com.br/mostrar/12>>. Acesso em 23 maio. 2008

GALHARDO, Rafaela; LIMA. Gleydson; **Transações com Hibernate (Tutorial)**; Disponível em: <<http://www.jeebrasil.com.br/mostrar/3>>. Acesso em 23 maio. 2008

GEISS, L de Carvalho. **JSTL – JSP Standard Tag Library**; disponível em: <http://wwwiti.cs.uni-magdeburg.de/iti_ti/SS04/Uebung9SS04.pdf>. Acesso em: 31 maio. 2008.

HM Deitel, PJ Deitel. **Java como programar**; 3ª Edição; Porto Alegre: Bookman; 2001.

HOWES; Timothy A. **The Lightweight Directory Access Protocol: X.500 Lite**; Disponível em: <<http://www.kingsmountain.com/directory/doc/ldap/ldap.html>>. Acesso em: 29 maio. 2008.

JASPERFORGE. **Jasperreports Tutorial**. Disponível em: <http://www.jasperforge.org/jaspersoft/opensource/business_intelligence/jasperreports/tutorial.html>. Acesso em: 30 maio. 2008.

JUNG, Carlos Fernando. **Metodologia para pesquisa e desenvolvimento**. Rio de Janeiro: Axcel Books, 2004.

LDAP; **Mas afinal o que é LDAP**. Disponível em: <<http://www.ldap.org.br/>> Acesso em: 30 maio. 2008.

LIMA, Gleydson. **Ambiente de produção com Apache, JBoss e mod_jk2**; Mundo Java; 2003.

MONICA, Pawlan et al. **Tutorial do J2EE: A Série Java Enterprise Edition**; Rio de Janeiro: Campos; 2002.

MOREIRA, Silvio Costa. **Utilização das ferramentas Jasperreports e Ireport para a geração de relatório em aplicações Java**. Disponível em: <<http://www.ulbra-to.br/ensino/43020/artigos/relatorios2005-2/Arquivos/SilvioCM-EstagioSupervisionadoemSistemasdeInformacao.pdf>>. Acesso em: 27 maio. 2008.

NETO, Plácido Antonio de Souza. **JSTL: Uma forma padrão de desenvolver página WEB Dinâmica**; Disponível em: <<http://www.cefetrn.br/~placido/downloads/jstl.pdf>>. Acesso em: 27 maio. 2008.