UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

SISTEMA DE BANCO DE CURRÍCULOS PARA O SIMULADOR DE EMPRESAS LÍDER

GIULIANO DE ANDRADE

BLUMENAU
2007

2007/1-07

GIULIANO DE ANDRADE

SISTEMA DE BANCO DE CURRÍCULOS PARA O SIMULADOR DE EMPRESAS LÍDER

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação - Bacharelado.

Prof. Maurício Capobianco Lopes, Mestre - Orientador

BLUMENAU
2007

2007/1-07

SISTEMA DE BANCO DE CURRÍCULOS PARA O SIMULADOR DE EMPRESAS LÍDER

Por

GIULIANO DE ANDRADE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Maurício Capobianco Lopes, Mestre – Orientador, FURB
Membro:	Prof. Adilson Vahldick, Especialista – FURB
Membro:	Prof. Ricardo Alencar Azanbuja, Mestre – FURB

Blumenau, 09 de julho de 2007

Dedico este trabalho a todos os meus familiares e amigos que sempre se fizeram presente nas horas que mais precisei.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que sempre esteve presente e me apoiou nas horas de dificuldade.

Aos meus amigos, pelos incentivos e cobranças.

Ao meu orientador, Maurício Capobianco Lopes, por ter acreditado na conclusão deste trabalho.

A autenticidade é a diferença entre os que são e os que tentam ser.

Ângelo Franco

RESUMO

Este trabalho apresenta a implementação de uma base centralizada de currículos, para o jogo de empresas LÍDER, desenvolvida em ambiente web utilizando plataforma Java, o *framework* JavaServer Faces, juntamente com a IDE Sun Java Studio Creator, e o *framework* Hibernate para a camada de persistência. A implementação permite que todas as empresas simuladas do LÍDER acessem uma só base de currículos, permitindo também que uma empresa possa contratar um funcionário de outra empresa, a fim de gerar um ambiente mais competitivo e próximo da realidade.

Palavras-chave: Jogos de Empresas. JavaServer Faces. AJAX.

ABSTRACT

This work presents the implementation of a centralized curriculum database for the LÍDER business game, developed in a web environment using the Java platform, the framework JavaServer Faces, with the IDE Sun Java Studio Creator and the framework Hibernate as the persistence layer. The implementation allows all the simulated enterprises to access a single curriculum base and also enables one enterprise to hire an employee from another enterprise, with the goal of generating a more realistic and competitive environment.

Key-words: Business game. JavaServer Faces. AJAX.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX - *Asynchronous JavaScript and XML*

API – *Application Programming Interface*

CSS – *Cascading Style Sheets*

DAO – *Data Access Object*

DOM – *Document Object Model*

EJB - *Enterprise JavaBeans*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Environment*

JEE - *Java Plataform Enterprise Edition*

JDBC – *Java Data Base Connectivity*

JPA – *Java Persistence API*

JSC – *Sun Java Studio Creator*

JSF - *JavaServer Faces*

JSP - *JavaServer Pages*

MER – Modelo Entidade/Relacionamento

OO – Orientação a Objetos

SGBD – Sistema Gerenciador de Banco de Dados

SO - Sistema Operacional

SQL - *Structured Query Language*

UML – *Unified Modeling Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A sociedade atual convive em um ambiente onde, para que uma pessoa seja competitiva no mercado, esta deve estar em constante aperfeiçoamento. Para esta pessoa se destacar dentro de uma organização, não basta que este aperfeiçoamento seja apenas técnico, pois muitas vezes a capacidade de relacionamento e liderança pode ser até mais importante.

Niveiros (1998) afirma que para um bom desempenho gerencial nestes tempos de mudanças rápidas e constantes, um profundo saber técnico que considere planejamento, custos, produção, controle da qualidade e vendas não é suficiente. Outras habilidades e conhecimentos também passam a ser exigidas, onde a compreensão do elemento humano, sua motivação, sua ação e interação no trabalho são fundamentais; o jogo de empresas LÍDER tem como objetivo estimular este aprendizado.

Segundo Lopes e Wilhelm (2006), o jogo de empresas LÍDER, foi concebido para o treinamento gerencial, tendo como objetivo simular o comportamento humano dentro de uma organização. O jogo tem como principais objetivos: exercitar e fixar estilos de gestão, aumentando a qualidade do processo decisório sobre a gestão de equipes e enfatizar a motivação e liderança eficaz, como fatores chaves de sucesso para desenvolver equipes de alto desempenho.

Atualmente o jogo de empresas LÍDER é aplicado em empresas através de cursos/seminários e em disciplinas de cursos de graduação e pós-graduação em diferentes instituições de nível superior. Tendo como base a boa aceitação do jogo por parte dos alunos, este trabalho propõe-se a aplicar melhorias, tornando o jogo mais competitivo e cada vez mais próximo da realidade empresarial. Para isso será implementado um banco de currículos através do qual as empresas farão o processo de recrutamento e seleção dos seus funcionários. Este banco de currículos será comum a todas as empresas, fazendo com que elas tenham que competir pela mão de obra. Esta funcionalidade não existe atualmente, pois cada empresa tem o seu próprio banco de currículos.

Com este banco de currículos, pretende-se simular a disputa pela mão de obra qualificada, pois segundo Almeida (2004), o mercado de trabalho passou a ser mais seletivo e os requisitos para contratação ficaram mais exigentes, principalmente quanto à formação, experiência e comportamento, fazendo com que a disputa por pessoas que façam a diferença na organização, seja cada vez mais acirrada.

Recentemente o jogo de empresas LÍDER foi migrado para a sua versão 9, uma versão

escrita em Java utilizando o *framework* Java Plataform Enterprise Edition (JEE), JavaServer Faces (JSF) e *Asynchronous JavaScript and XML* (AJAX) para deixar a aplicação mais dinâmica e funcional (MICHELUZZI, 2006). É sobre esta versão que este trabalho será implementado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento de um banco de currículos comum para todas as empresas simuladas do Jogo de Empresas Líder.

Os objetivos específicos do trabalho são:

a) permitir aos jogadores avaliar o perfil dos candidatos a emprego quanto à experiência, aspectos motivacionais, faixa salarial desejada, entre outros;
b) permitir a uma empresa fazer ofertas de empregos sobre funcionários de outras empresas simuladas, criando uma competição direta das empresas pelos recursos humanos;
c) criar um ambiente competitivo visando dar mais realismo ao jogo.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos.

O segundo capítulo descreve a fundamentação teórica sobre jogos de empresas, o jogo de empresas LÍDER, a área de recrutamento e seleção e sobre as tecnologias utilizadas para o desenvolvimento deste projeto. Ainda neste capítulo são apresentados alguns trabalhos correlatos.

O terceiro capítulo apresenta o desenvolvimento do projeto como um todo, descrevendo seus requisitos, sua especificação, a implementação, as ferramentas utilizadas e os resultados obtidos com o projeto.

No quarto e último capítulo é apresentada a conclusão do trabalho, como também algumas sugestões de extensões para o mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os fundamentos teóricos necessários para o desenvolvimento do trabalho e alguns trabalhos correlatos.

2.1 JOGOS DE EMPRESAS

Segundo Jacobs e Baun (1987 apud VALDAMERI, 2001), a origem dos jogos de empresas teve inspiração nos jogos militares, onde o objetivo era simular estratégias e táticas em confrontos. Já os jogos de empresas se atentam às disputas de mercado, uma vez que os campos de batalha cedem lugar aos mercados consumidores de bens industrializados.

Esta transição do cenário militar para o empresarial ocorreu de forma natural, já que os objetivos básicos dos jogos militares puderam ser assimilados e aplicados ao ambiente dos homens de negócio, que chefiavam seus exércitos de vendedores e operários.

Há pouco mais de 40 anos que os jogos de empresas computacionais conquistaram com êxito o seu espaço incontestável entre os métodos de ensino, tendo como grande vantagem, a capacidade de acelerar o tempo, comprimindo em poucos dias vários anos de experiência, provendo assim um preparo para as atividades profissionais futuras dos jogadores (NIVEIROS, 1998).

Niveiros (1998) ainda afirma que os jogos de empresas proporcionam meios para que os participantes aprendam a lidar na prática com os problemas que irão encontrar no decorrer de seus trabalhos. Com a compressão do tempo realizada, torna-se possível o aprendizado, mesmo quando as conseqüências de uma decisão tomada estão no futuro ou em partes distantes da organização. Os jogos tornam possível experimentar uma grande variedade de estratégias para o alcance dos objetivos pré-definidos, possibilitando também saber se estes objetivos foram ou não alcançados.

Gramigna (1993) comenta que durante o jogo o grupo realiza várias interações e coloca em prática suas habilidades técnicas, da mesma forma que em seu cotidiano. Assim agirá dentro do seu padrão de tomada de decisões e, de acordo com os resultados obtidos, poderá rever e replanejar sua ações a fim de melhorá-las.

2.2 O JOGO DE EMPRESAS LÍDER

O jogo de empresa LÍDER surgiu para o preenchimento de uma lacuna existente na área de desenvolvimento de recursos humanos utilizando meios computacionais, visto que os similares existentes eram manuais e implementados através de formulários e tabelas, limitando-se assim a um número pequeno de variáveis interagindo com modelos, comenta Niveiros (1998).

Niveiros (1998) diz que o objetivo do jogo LÍDER consiste em oferecer a oportunidade de experimentar a aplicação prática da teoria comportamental e sua intenção é aplicar os conceitos da teoria e técnicas de Liderança Situacional descritas por Hersey e Blanchard, a Hierarquia das Necessidades de Maslow e a teoria de Motivação – Higiene de Herzberg.

O cenário do jogo de empresas LÍDER constitui-se no relacionamento do gerente de produção de uma empresa e seus subordinados, sendo que a missão dos participantes é tornar a empresa competitiva, trabalhando com funcionários produtivos e motivados (LOPES, RIBEIRO e WILHELM, 1998).

A função básica do jogo é permitir que os participantes tomem decisões sobre os funcionários a fim de otimizar a produção destes. Através da ficha básica do funcionário, os participantes da simulação podem diagnosticar as potencialidades, as carências e as características deste funcionário. Também através desta ficha, ao contratar um funcionário, é possível analisar antecipadamente as informações deste, como se fosse um currículo (NIVEIROS, 1998).

Lopes (1994) destaca a dinâmica do jogo, enfatizando que o jogador deve entrar com decisões que alteram variáveis do sistema a fim de prever o comportamento humano, tendo como objetivo motivar o funcionário para obter uma melhor produção. Essas decisões podem ser globais ou individuais, onde as globais afetam todos os funcionários da empresa e as individuais são aplicadas a funcionários isoladamente. O jogador pode definir políticas de benefícios, tais como salário, prêmios, treinamentos, planos de saúde, alimentação, entre outros. Os funcionários têm diferentes perfis e, por isso, diferentes reações à postura do líder. Com o processamento da jogada podem ser visualizadas as alterações no comportamento do funcionário. Para o cálculo do processamento do jogo, além das decisões tomadas pelo jogador, levam-se em consideração as necessidades gerais dos funcionários, a maturidade e as situações perturbadoras.

Segundo Micheluzzi (2006), apesar do LÍDER 8 ser uma ferramenta pronta e utilizada

em disciplinas de graduação e pós-graduação, o mesmo possuía algumas limitações. O sistema não permitia, por exemplo, simulação e planejamento de cenários onde o jogador pudesse avaliar as relações de causa e efeito do seu processo decisório, obrigando-o a planejar sem o auxílio do programa. Além disso, estava limitado ao Sistema Operacional (SO) Windows e não podia ser acessado pela Internet. Desta forma, surgiu a necessidade de migrar o software para uma nova tecnologia. Sendo assim, foi desenvolvido o LÍDER 9, sendo que o funcionamento desta nova versão é apresentada a seguir.

A primeira tela apresentada é a tela de login de usuário, conforme ilustra a figura 1.

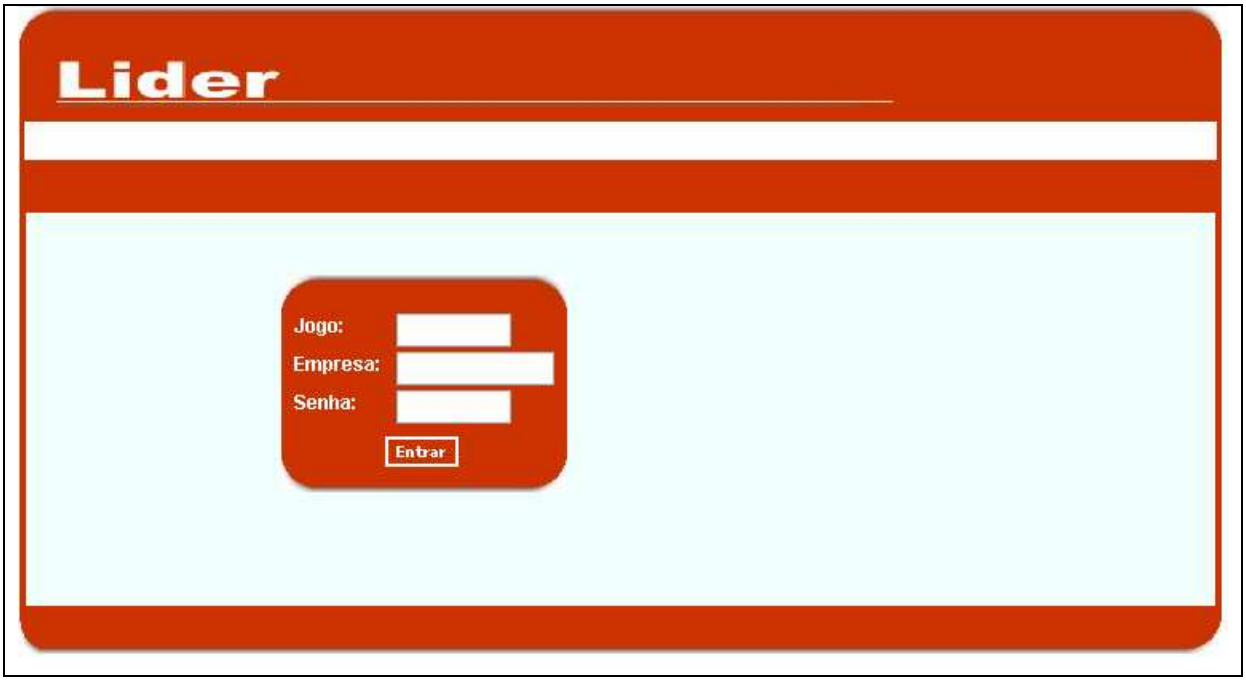


Figura 1 – Tela de login do Jogo de Empresas Líder 9

Logando no sistema com um usuário que possui perfil de administrador, as opções de menu são as ilustradas na figura 2.

Figura 2 – Menu de administrador do Jogo de Empresas Líder 9

A primeira opção do menu permite que o administrador faça o cadastro de um candidato. A figura 3 demonstra a tela de cadastro de candidatos.

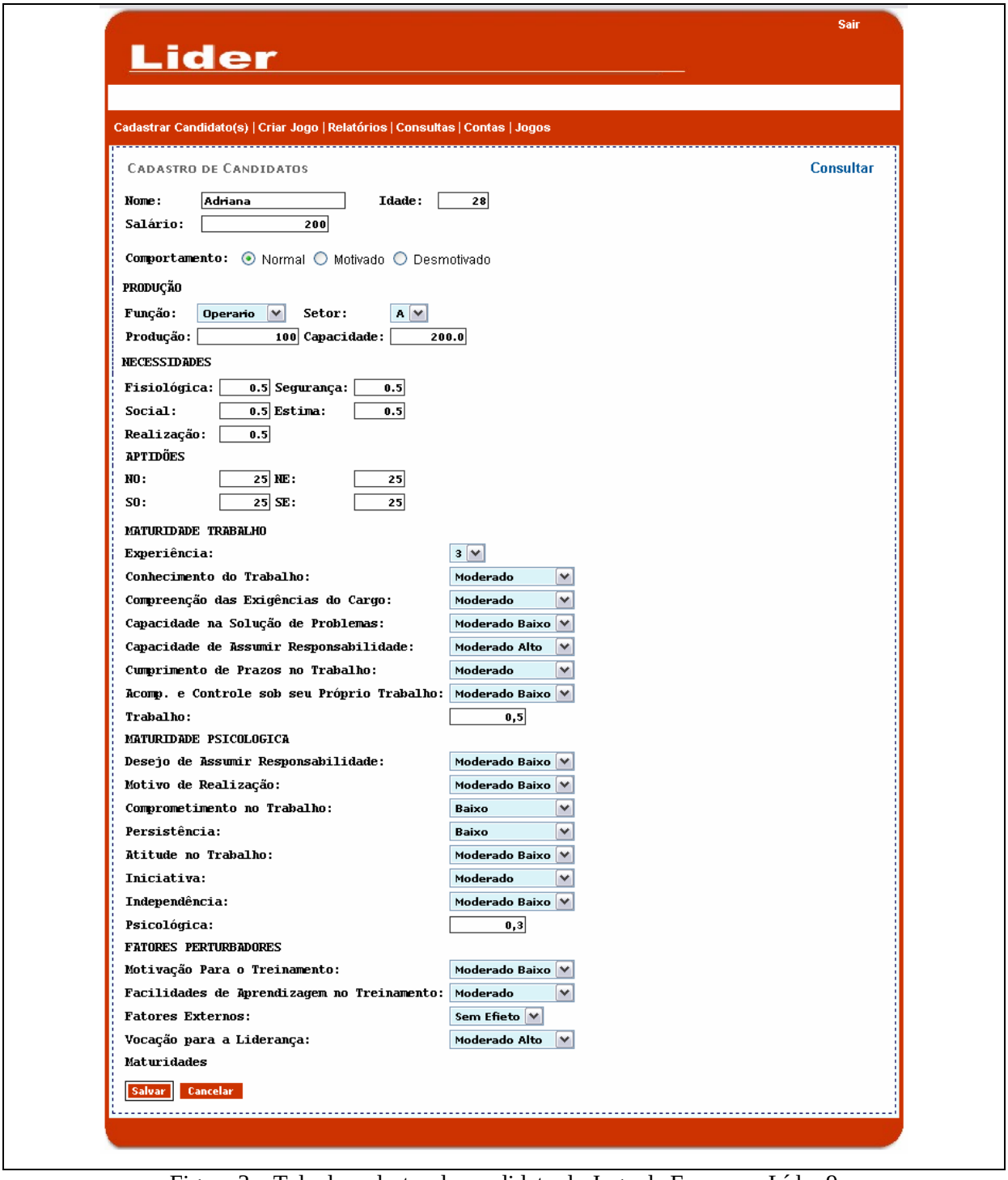


Figura 3 – Tela de cadastro de candidato do Jogo de Empresas Líder 9

Este cadastro exige que todos os campos da tela sejam preenchidos e quando o administrador escolher uma função para o candidato, o sistema automaticamente sugere um salário e define em que setor o candidato deve ser enquadrado. Caso a função for Inovação ou Qualidade, o candidato não pode possuir um Setor. A mesma coisa acontece com o campo Capacidade, ou seja, a capacidade de um candidato é calculada e definida com base no campo Produção, sem que o usuário precise entrar com os dados. O cálculo da Maturidade no Trabalho e da Maturidade Psicológica é feito automaticamente à medida que o administrador

altera os campos relacionados a elas. Após a confirmação do cadastro o sistema disponibiliza o candidato no sistema permitindo que o mesmo possa ser contratado por uma empresa.

Acessando o menu Criar Jogo, o sistema possibilita ao administrador cadastrar uma ou várias empresas que irão compor o jogo, conforme ilustra a figura 4. Para isto é necessário selecionar todos os candidatos que farão parte da(s) empresa(s). Na linha correspondente a cada candidato, existe a opção de visualizar os detalhes deste candidato, conforme pode ser visto na figura 5. Depois dos candidatos selecionados, existe a necessidade de informar a quantidade de empresas que farão parte do jogo, sendo que o sistema apresenta uma tabela com uma sugestão de nome para cada empresa, que pode ser alterada pelo administrador. Em seguida é solicitado que seja criada uma senha inicial para os jogadores logarem no sistema e informar se o jogo está integrado ao Jogo de Empresas Virtual[1] ou não.

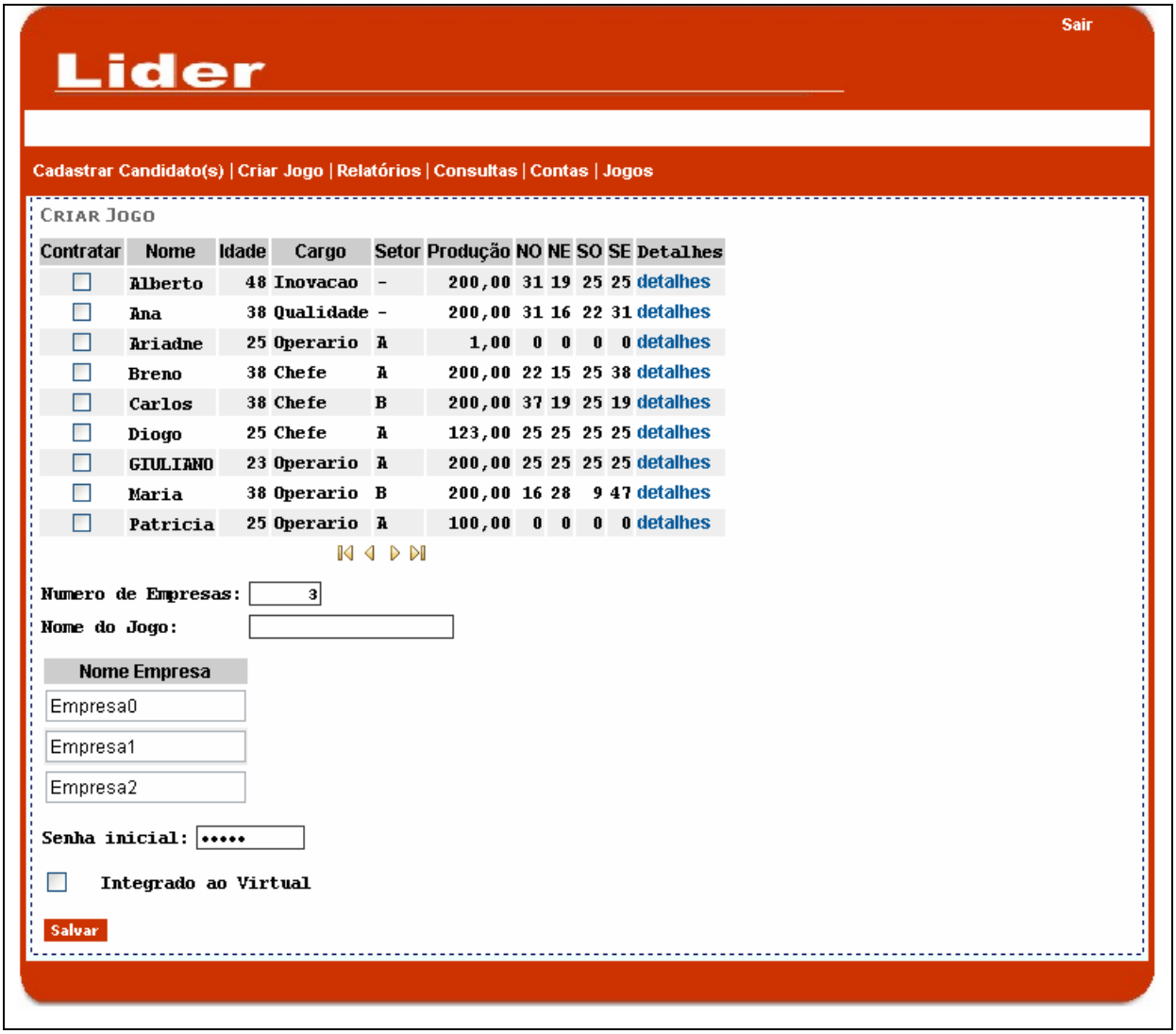


Figura 4 – Tela Criar Jogo do Jogo de Empresas Líder 9

[1] Jogo de empresas para gestão econômico financeira.

Figura 5 – Tela de detalhes do candidato do Jogo de Empresas Líder 9

Com as empresas criadas o administrador já pode distribuir os jogos com as suas respectivas senhas para cada jogador. Tendo essas informações, através da tela de login ilustrada na figura 1 o jogador pode logar no sistema.

Logando no sistema com um usuário que possui perfil de jogador, as opções de menu são as ilustradas na figura 6.

Figura 6 – Menu de jogador do Jogo de Empresas Líder 9

O menu Simulação acessa a tela de simulação, ilustrada na figura 7. Ela permite que o jogador visualize os dados dos períodos processados, apenas selecionando o período de interesse no *combobox* período. Além disso, a principal finalidade dessa tela é prover a simulação e processamento de um período no jogo, permitindo ao jogador visualizar em tempo real o impacto de suas decisões.

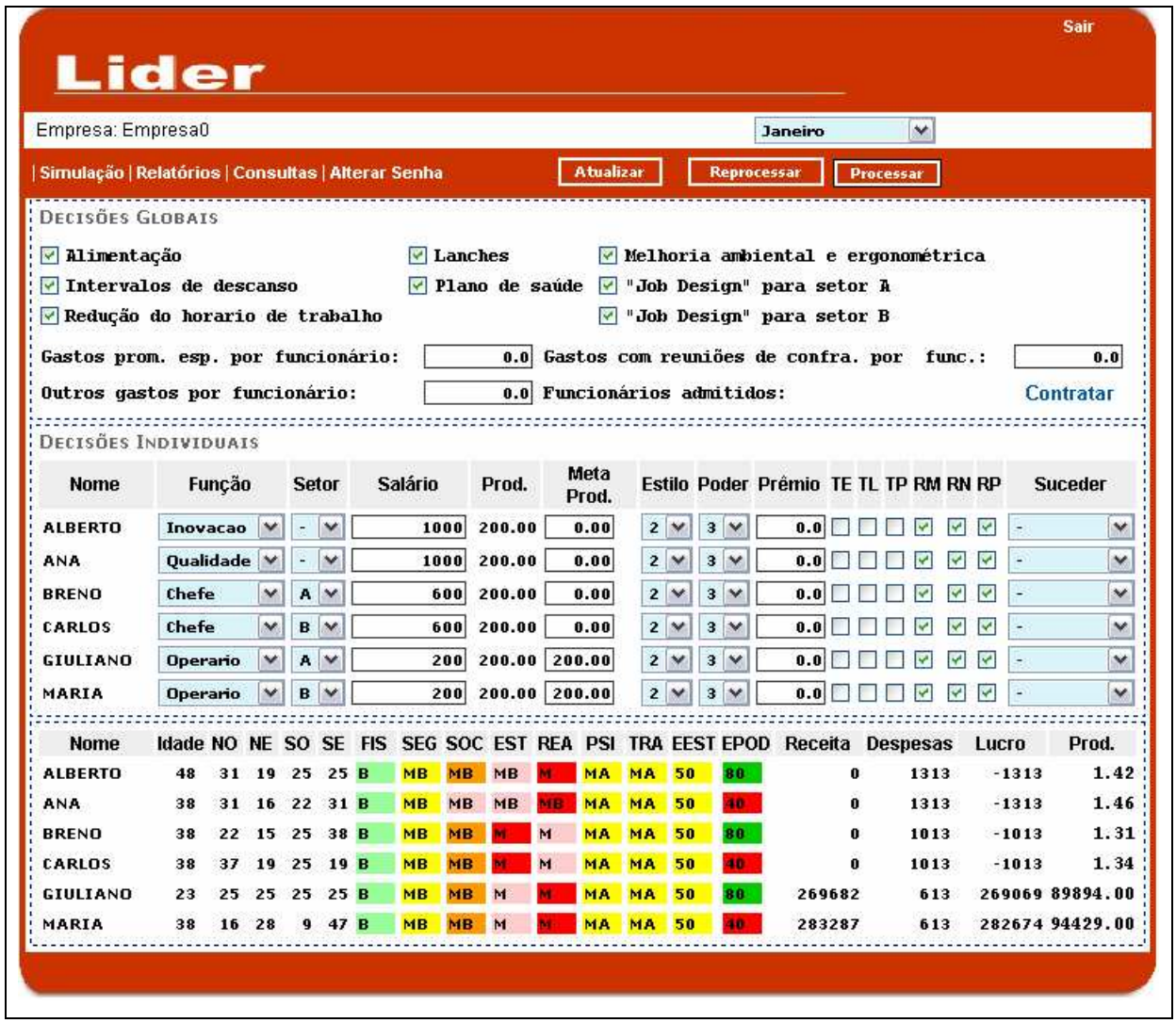
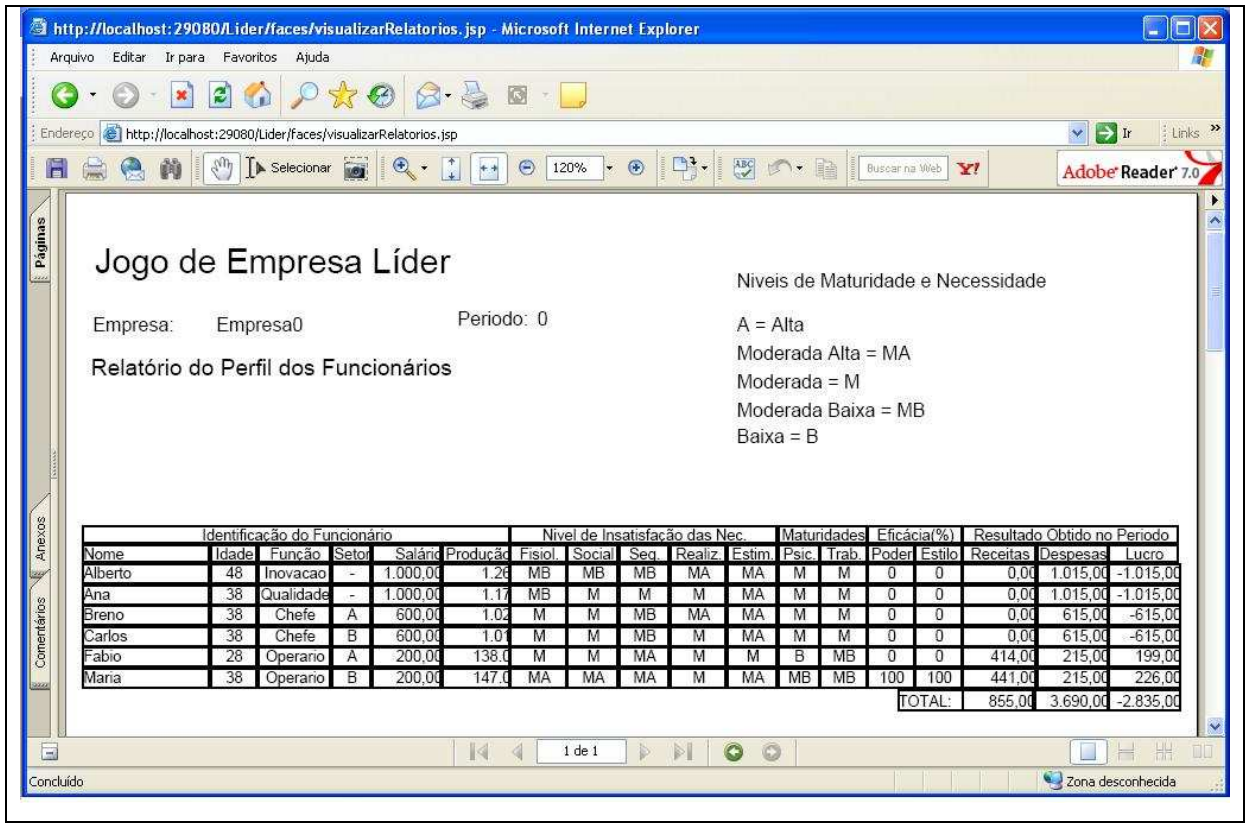


Figura 7 – Tela de simulação do Jogo de Empresas Líder 9

A opção relatórios permite que o Jogador imprima vários relatórios sendo um deles o

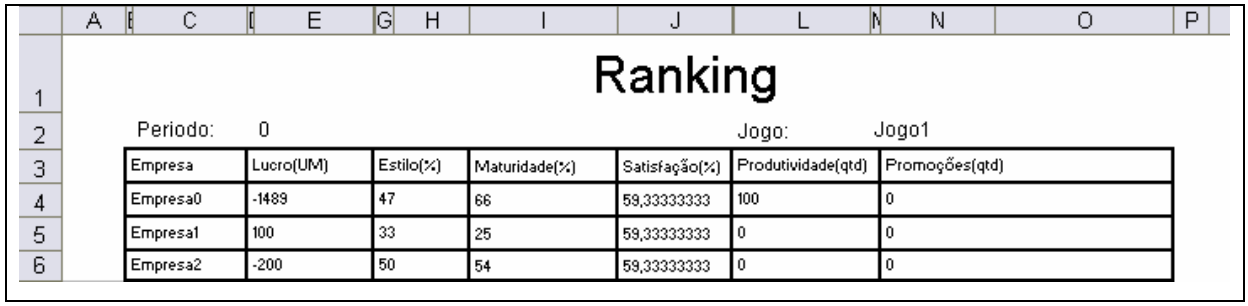
relatório do perfil dos funcionários ilustrado na figura 8. Este relatório é gerado em formato PDF.



Identificação do Funcionário						Nivel de Insatisfação das Nec.					Maturidades		Eficácia(%)		Resultado Obtido no Periodo		
Nome	Idade	Função	Setor	Salário	Produção	Fisiol.	Social	Seg.	Realiz	Estim.	Psic.	Trab.	Poder	Estilo	Receitas	Despesas	Lucro
Alberto	48	Inovacao	-	1.000,00	1.26	MB	MB	MB	MA	MA	M	M	0	0	0,00	1.015,00	-1.015,00
Ana	38	Qualidade	-	1.000,00	1.17	MB	M	M	M	MA	M	M	0	0	0,00	1.015,00	-1.015,00
Breno	38	Chefe	A	600,00	1.02	M	M	MB	MA	MA	M	M	0	0	0,00	615,00	-615,00
Carlos	38	Chefe	B	600,00	1.01	M	M	MB	M	MA	M	M	0	0	0,00	615,00	-615,00
Fabio	28	Operario	A	200,00	138.0	M	M	MA	M	M	B	MB	0	0	414,00	215,00	199,00
Maria	38	Operario	B	200,00	147.0	MA	MA	MA	M	MA	MB	MB	100	100	441,00	215,00	226,00
														TOTAL:	855,00	3.690,00	-2.835,00

Figura 8 – Relatório de Perfil do Jogo de Empresas Líder 9

Após todas as empresas já terem efetuado o processamento de suas jogadas é possível que o administrador imprima um relatório com o ranking do jogo através da tela de relatórios do sistema. Este relatório é gerado em formato XLS. A figura 9 mostra o relatório de ranking gerado.



Ranking

Periodo: 0 Jogo: Jogo1

Empresa	Lucro(UM)	Estilo(%)	Maturidade(%)	Satisfação(%)	Produtividade(qtd)	Promoções(qtd)
Empresa0	-1489	47	66	59,33333333	100	0
Empresa1	100	33	25	59,33333333	0	0
Empresa2	-200	50	54	59,33333333	0	0

Figura 9 – Relatório de Ranking do Jogo de Empresas Líder 9

A versão do jogo de empresas LÍDER 9, foi escrita em Java e roda em ambiente web, possibilitando que o sistema seja executado em qualquer sistema operacional. Porém esta versão ainda não foi aplicada como ferramenta de ensino, não permitindo ainda ter um *feedback* sobre a aceitação do sistema, por parte dos participantes do jogo.

2.3 RECRUTAMENTO E SELEÇÃO DE PESSOAL

A disputa por mão de obra qualificada vem crescendo e sendo estudada cada vez mais nos últimos tempos.

Pontes (2004), embasado na afirmativa de Gary Becker, professor na Universidade de Chicago, de que o sucesso de países como Estados Unidos, Japão e Alemanha, estão diretamente ligados ao nível de seu capital intelectual, diz que a afirmativa também é válida para uma organização. Ele ainda afirma que na era do capital intelectual, as pessoas, que já tinham papel importante, passam a ser o fator vital para a competitividade das empresas, levando em conta principalmente os rápidos avanços tecnológicos e do comércio globalizado. Assim, vem surgindo a necessidade das organizações cada vez mais buscarem e reterem pessoas qualificadas, dando início ao que Almeida (2004) citou como guerra pelos talentos.

Segundo Almeida (2004), o mercado de trabalho brasileiro, a exemplo de outros países, passou por mudanças significativas, muitas delas relacionadas às políticas econômicas e sociais. O mercado passou a ser mais seletivo, principalmente quanto à formação e experiência, sem falar nas competências comportamentais, cada vez mais valorizadas.

A fim de tornar os processos seletivos mais eficazes, Adler (2003 apud ALMEIDA, 2004) divide o processo de recrutamento e seleção em quatro processos que funcionam como um filtro, objetivando selecionar os profissionais mais talentosos, sendo que cada processo deve estar integrado com os demais, possuindo cada um seus próprios objetivos:

a) atração: qualquer tipo de ação para atrair candidatos;
b) triagem: ações voltadas para a eliminação de candidatos não qualificados;
c) avaliação: ações voltadas para avaliar as qualificações dos candidatos;
d) decisão: ações para decidir entre os candidatos finais;

Segundo Pontes (2004), para atrair e reter talentos é fundamental que, além de programas de gestão de pessoal modernos, flexíveis e ajustados à nova realidade, a empresa mantenha um clima de trabalho sadio, motivador, voltado ao contínuo desenvolvimento das pessoas e que reconheça as que fazem diferença. Entre tantas recomendações para a política de pessoal, quatro questões são as mais marcantes:

a) oportunidade de aprendizado: talentos valorizam as oportunidades que têm de aprender;
b) feedback: a avaliação e a orientação dada por profissionais experientes são fundamentais para novos talentos;

c) chefes: muitas vezes profissionais de alto desempenho se desligam da empresa por motivos de conflitos com seus chefes;
d) salário: remuneração competitiva é um item básico na hora de atrair e reter talentos;

Pontes (2004) ainda afirma que, a agilidade requerida das empresas leva a necessidade de pessoas certas nos lugares certos, por isso o processo de seleção não pode ser errado. A seleção de pessoas erradas pode significar montanhas de prejuízos para a empresa em perda de oportunidades ou verdadeiros desastres econômicos produzidos. Portanto, o cuidado deve ser redobrado em todo o processo de seleção.

Devido a esta grande importância que a área de recrutamento e seleção exerce perante as organizações, identificou-se a oportunidade de ampliar a área de atuação do jogo de empresas LÍDER. Possibilitando ao jogador, vivenciar os desafios encontrados no momento de contratar um colaborador, ou quando este se desliga da organização.

2.4 TECNOLOGIAS

Este capítulo apresenta algumas das tecnologias utilizadas para o desenvolvimento do jogo de empresas LÍDER.

2.4.1 Java Enterprise Edition

JEE é a especificação de uma plataforma que roda num servidor. Esta plataforma fornece uma infra-estrutura robusta que permite a criação de sistemas corporativos, e esses sistemas podem ser distribuídos e acessados por várias pessoas simultaneamente num ambiente web. O padrão JEE proporciona que a aplicação desenvolvida seja independente de plataforma (BOND et al., 2004).

As principais características da plataforma JEE são: segurança, escalabilidade, independência de sistema operacional e arquitetura distribuída (BOND et al., 2004).

A JEE fornece uma estrutura de múltiplas camadas físicas e lógicas e seu modelo proporciona que sejam construídas soluções baseadas em componentes que permitem a reutilização dos mesmos. Os componentes lógicos podem ser divididos em componentes web

e componentes EJB (Enterprise JavaBeans). Os componentes da camada do cliente são executados na máquina do cliente, enquanto que os componentes da camada web e de negócios são executados no servidor JEE. Os componentes web podem ser classes *servlets* ou páginas JSP (JavaServer Pages). Já os componentes EJBs são componentes de negócios que são executados no servidor (BODOFF et al., 2002).

Os componentes web são processados no contêiner web e os componentes EJB são processados no contêiner EJB no lado do servidor, enquanto que a interface é apresentada no lado do cliente.

O contêiner web manipula as informações apresentadas ao cliente, processando *servlets* ou páginas JSPs, e gerenciando o seu ciclo de vida, enquanto que o contêiner EJB processa e gerencia a execução de *enterprise javaBeans* (BOND et al., 2004).

2.4.2 JavaServer Faces

O JavaServer Faces é um *framework* para criação de interface gráfica para aplicações web, desenvolvido através do Java Community Process com o objetivo de simplificar o desenvolvimento de aplicações web. O *framework* é composto por um conjunto de APIs para representação de componentes de interface com o usuário, gerenciamento do estado dos componentes, tratamentos de evento de interface gráfica, validação dos dados de entrada, e navegação, que podem ser utilizados em páginas JSP através de uma biblioteca de tags (BELLIA; SENGER, 2007, p. 30-39).

Ainda segundo Bellia e Senger (2007, p. 30-39), as tags JSF podem ser associadas a componentes de interface de usuário no lado do servidor, que por sua vez fabricam código HTML. Por exemplo, ao processar uma página contendo uma tag `<h:inputText>`, o JSF criará elementos no servidor web e no *browser* do usuário para controlar a entrada de dados via uma caixa de texto, sem exigir esforço do programador.

Kurniawan (2004) divide o cliclo de vida de um página JSF em seis fases, executadas na seguinte ordem:

a) reconstruir árvore de componentes: uma página JSP em um aplicativo JSF é representada por uma árvore de componentes. Esta fase inicia o processamento da requisição *Lifecycle* por meio da construção desta árvore. Cada árvore de componentes tem um identificador que é único durante todo o aplicativo. Para um requisição com a URI /faces/index.jsp, por exemplo, o

identificador árvore é /index.jsp. A árvore de componentes construída é então salva no objeto FacesContext para processamento das fases seguintes de processamento da requisição.

b) aplicar valores da requisição: nesta fase, o valor local de cada componente na árvore de componentes é atualizado pela requisição corrente. Um valor pode vir de um parâmetro de requisição, um cabeçalho, um *cookie*, e assim por diante. Durante esta fase, um componente pode enfileirar eventos. Estes eventos serão processados durantes as etapas de evento do processo, no ciclo de vida de processamento da requisição.

c) processar validações: depois do valor de cada componente ser atualizado na árvore de componentes, na fase processar validações, o objeto *Lifecycle* validará aqueles valores, se necessário. Um componente que requer validação deve fornecer implementação da lógica de validação. Alternativamente, um programador JSF pode registrar zero ou mais validadores com o componente. Se um ou mais validadores externos são encontrados, o valor local de cada componente será validado usando a lógica de validação dentro destes validadores externos.

d) atualizar valores de modelos: esta fase pode ser alcançada somente se os valores locais de todos os componentes na árvore forem válidos. Nesta fase, o objeto *Lifecycle* atualiza os dados de modelo do aplicativo. Durante esta fase um componente pode enfileirar eventos novamente.

e) chamar o aplicativo: durante esta fase, a implementação JSF manipula quaisquer eventos em nível do aplicativo, tal como submeter um formulário ou um link para uma outra página.

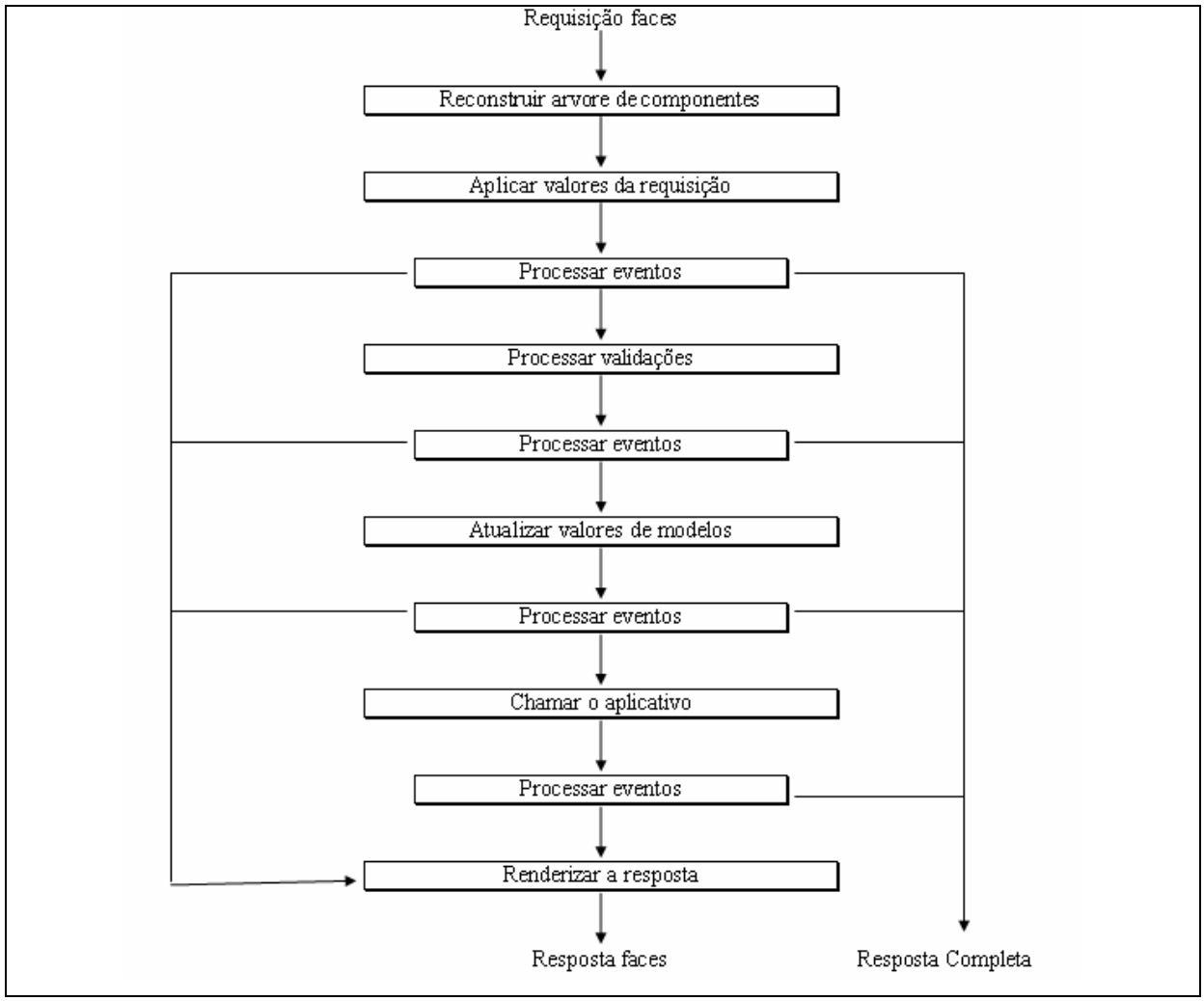
f) renderizar a resposta: nesta fase, a implementação JSF renderiza a resposta ao cliente.

As fases aplicar valores de requisição, processar validações, atualizar valores de modelo e chamar aplicativo no ciclo de vida de processamento da requisição podem enfileirar eventos na instância FacesContext associada com uma requisição corrente. Portanto, a implementação JSF deve tratar estes eventos antes destas três fases.

Entre duas fases, o objeto *Lifecycle* checa os eventos que precisam ser chamados. Ao escrever um evento, o programador pode escolher depois de qual fase o mesmo deve ser executado. Alternativamente, um evento pode ser chamado após várias fases.

A figura 10 ilustra o processamento de uma requisição JSF através destas fases. As

caixas menores que são rotuladas “Processar eventos” indicam as etapas que o objeto *Lifecycle* leva para executar os ouvidores de eventos.



Fonte: Adaptado de Kurniawan (2004).

Figura 10 – Ciclo de Vida de uma página JSF

2.4.3 AJAX

Segundo Senger e Villela (2007, p. 44-51) AJAX é uma combinação de várias tecnologias que oferecem ao usuário final uma aplicação web mais interativa.

A apresentação final para o usuário é baseada em padrões da web, combinando o uso de HTML e CSS. O CSS define estilos padronizados para formatação dos documentos HTML e permite que esses estilos sejam reutilizados.

Os dados são manipulados dinamicamente usando os recursos do modelo de objetos padrão DOM (Document Object Model). Este modelo permite a manipulação e criação de

elementos HTML dinamicamente.

A recuperação de dados é feita de forma assíncrona usando o objeto XMLHttpRequest. Os dados são solicitados e recuperados através de uma thread, ou seja, a aplicação continua sendo executada independente do status da requisição de dados feita com AJAX. A grande vantagem nesse tipo de recuperação é que a interface gráfica do usuário não fica bloqueada até os dados serem recuperados.

O JavaScript é utilizado para fazer a união de todas estas tecnologias.

Para uma melhor integração entre o AJAX o *framework* JSF, o JBoss Ajax4Jsf foi desenvolvido para ser uma extensão da implementação padrão deste *framework*. Ele permite que sejam adicionadas capacidades AJAX em páginas JSF sem a necessidade de escrever código JavaScript. Seu ciclo de vida é integrado com o ciclo de vida do JSF. Isso permite que quando uma requisição é gerada ao servidor, os dados da página sejam sincronizados com a árvore de componentes JSF, permitindo assim validações, conversões e todas as facilidades proporcionadas pelo ciclo de vida do JSF. O JBoss Ajax4Jsf permite que seja adicionado facilmente suporte a eventos e ações que utilizam AJAX em componentes da implementação padrão do JSF, sem que o componente tenha que ser customizado. Assim, ele permite que sejam disparadas requisições ao servidor e tratadas na própria classe de controle da página JSF, de forma que o desenvolvedor não precise alterar a estrutura do programa. Após a requisição ser tratada, uma resposta é gerada à página onde novamente o JBoss Ajax4Jsf se encarrega de aplicar as alterações requisitadas (JBOSS AJAX4JSF, 2007).

2.4.4 Java Persistence API

Visto que a tecnologia de banco de dados relacionais existe há décadas e hoje os SGBDs são robustos e confiáveis, a utilização destes sistemas se torna uma regra. Porém, esta utilização traz alguns inconvenientes para os programadores de linguagens orientadas a objetos, pois estas tecnologias são bem distintas. O desafio atual dos desenvolvedores é unir dois mundos completamente distintos, utilizando a tecnologia relacional para armazenar objetos (ROCHA; KUBOTA, 2006, p. 28-29).

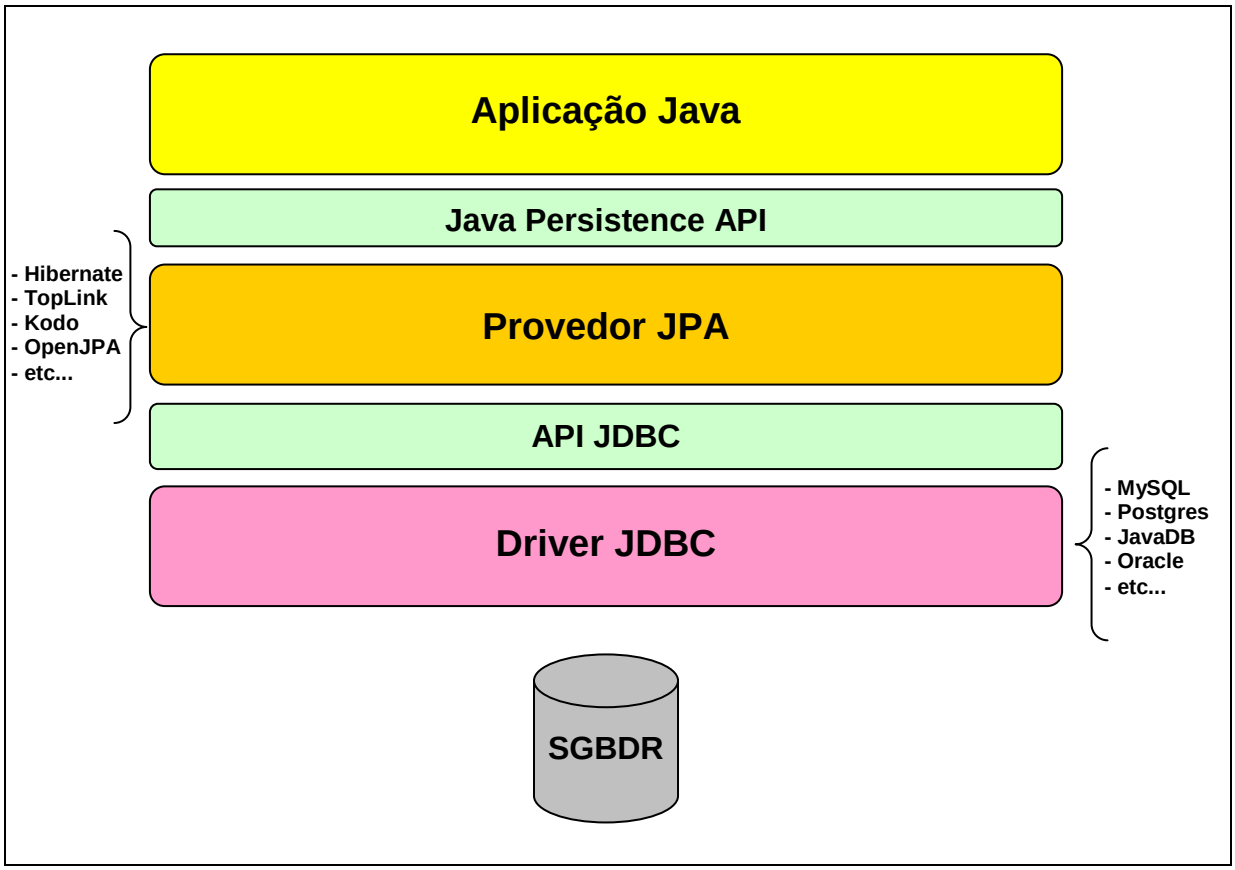
Segundo Rocha e Kubota (2006, p. 28-29), desde as primeiras versões da plataforma Java, ela oferece acesso a banco de dados através da API JDBC, que trabalha no mesmo nível do banco, sendo que o acesso às informações armazenadas é feito através de comandos SQL. Mas a utilização desta API oferece uma abstração OO bastante limitada, pois trabalha-se com

tabelas, registros e *resultsets*, ao invés de objetos.

Para utilizar os recursos dos bancos de dados relacionais em Java e ainda assim aproveitar o conceito do paradigma OO é necessário fazer o que se conhece como mapeamento objeto-relacional. No mapeamento objeto-relacional as classes e os atributos do sistema são mapeados para tabelas e campos, e a persistência é feita de forma transparente para a aplicação. Mas diversas questões surgem durante o mapeamento: dúvidas sobre como mapear uma herança, uma agregação, se toda classe se transformará em uma tabela, e como aproveitar os recursos do banco sem perder a abstração dos objetos. Para suprir estas necessidades, surgiram vários *frameworks* e tecnologias de persistência, a exemplo do Hibernate e iBatis. Apesar destas ferramentas possuírem formas distintas de fazer o mapeamento objeto-relacional, os conceitos são semelhante e relativamente simples (ROCHA; KUBOTA, 2006, p. 28-29).

A Java Persistence API (JPA) padroniza as operações de persistência sobre as entidades Java, definindo uma especificação para o mapeamento objeto-relacional, e apesar de descrita na especificação do EJB 3, a JPA não depende de um conteiner para funcionar.

Quando se utiliza JDBC, é necessário escolher um driver JDBC. Da mesma forma, ao utilizar JPA deve-se escolher um provedor JPA. Assim como JDBC é uma especificação para conectar com bancos de dados relacionais, JPA é uma especificação para frameworks de persistência. A Figura 11 mostra como uma aplicação baseada em JPA é construída.



Fonte: Adaptado de Bellia (2007, p. 28).

Figura 11 – Arquitetura da JPA

2.4.4.1 Hibernate

Hibernate é uma ferramenta que implementa uma solução para a camada de persistência em sistemas escritos na plataforma Java. Ele implementa o mapeamento objeto-relacional do banco de dados e deixa o software independente da maioria dos bancos de dados do mercado, sendo que o mesmo possui mecanismos que permitem que objetos sejam persistidos no banco de dados sem a necessidade de escrever comandos *Structured Query Language* (SQL). Ao contrário de outras soluções para a camada de persistência, o Hibernate não esconde o SQL do desenvolvedor, e garante que seu investimento e conhecimento em tecnologia relacional sejam sempre válidos (HIBERNATE, 2007).

2.5 TRABALHOS CORRELATOS

Niveiros (1998) propõe uma reestruturação da dinâmica das maturidades dos funcionários do jogo de empresas LÍDER, identificando e analisando mecanismos capazes de lhe propiciar uma interação cada vez maior com a realidade.

Valdameri (2001), além de implementar as alterações propostas por Niveiros (1998), incorpora recursos de banco de dados ao simulador e efetuou um primeiro estudo da utilização do simulador em um ambiente Web.

Steinbach (2002) desenvolveu um protótipo de um sistema especialista na área da hierarquia das necessidades de Maslow, utilizando a ferramenta *Shell* Expert Sinta, tornando possível assim um melhor acompanhamento das necessidades dos funcionários do jogo de empresas LÍDER.

Micheluzzi (2006) migrou o jogo de empresas LÍDER 8 da plataforma Delphi para Java utilizando o *framework* JEE JSF e a tecnologia AJAX, e também reviu o modelo de regras de negócio, permitindo a simulação e planejamento de cenários.

Wilhelm (1997) descreve o Virtual, um jogo de empresas onde atuam quatro diretores, sendo eles: Diretor Geral, Diretor Financeiro, Diretor de Produção e Diretor de Mercado. Estes diretores devem criar estratégias, planejar seu processo decisório. O jogo permite a simulação de cenários que auxiliam os diretores, servindo de apoio à tomada de decisão.

3 DESENVOLVIMENTO DO TRABALHO

Este capítulo descreve o desenvolvimento do trabalho, apresentando a análise dos requisitos, a especificação do sistema e a sua implementação.

3.1 REQUISITOS DO PROBLEMA A SER TRABALHADO

O Quadro 1 apresenta os requisitos funcionais previstos para o sistema e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir ao administrador gerenciar o cadastro de candidatos. Definindo se este será vinculado diretamente a uma empresa, ou se ficará disponível para futuras contratações.	UC01
RF02: O sistema deverá permitir ao administrador, a criação de jogos com uma ou mais empresas.	UC02
RF03: O sistema deverá permitir aos jogadores contratar funcionários a partir de um banco de currículos, podendo avaliar o perfil dos candidatos a emprego, quanto à experiência, aspectos motivacionais, faixa salarial desejada, entre outros. Este candidato pode ou não já estar vinculado a uma empresa.	UC03

Quadro 1 – Requisitos Funcionais

O Quadro 2 lista os requisitos não funcionais previstos para o sistema.

Requisitos Não Funcionais
RNF01: O sistema deverá possuir documentação do novo módulo implementado
RNF02: O sistema deverá ser desenvolvido em Java com *framework* JSF
RNF03: O sistema deverá utilizar AJAX para melhor interatividade
RNF04: O sistema deverá rodar em ambiente web
RNF05: O sistema deverá ser independente de plataforma
RNF06: O sistema usará o banco de dados MySQL, sendo que todas as equipes compartilharão uma mesma base de dados

Quadro 2 – Requisitos não Funcionais

3.2 ESPECIFICAÇÃO

A especificação do projeto foi feita utilizando diagramas UML e a ferramenta *Enterprise Architect* para gerar os mesmos.

3.2.1 Diagramas de Caso de Uso

Os diagramas de caso de uso são utilizados para ilustrar as principais interações de um usuário ou dispositivo, com o sistema. Os principais elementos deste diagrama são: o ator, ilustrado por um boneco; e uma elipse, que representa o caso de uso, possuindo também a sua descrição.

A seguir serão apresentados somente os casos de uso referentes a este projeto, ou seja, casos de uso adicionados ou alterados, relacionado ao projeto original do Líder 9.

A figura 12 ilustra o diagrama de caso de uso em um cenário em que o administrador pode cadastrar um candidato e criar um jogo.

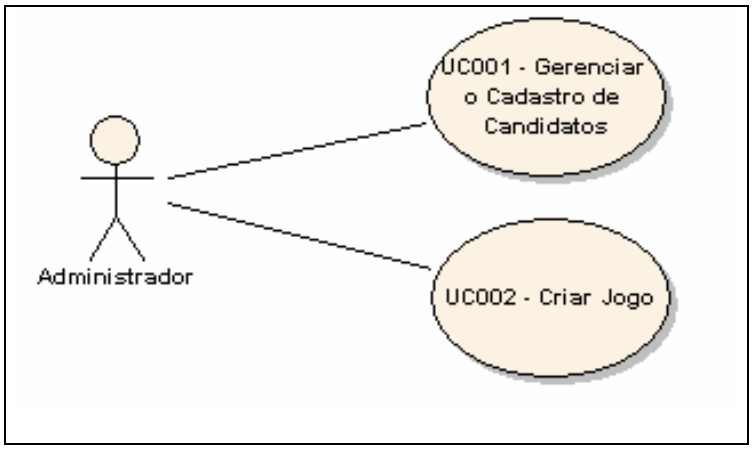


Figura 12 – Diagrama de caso de uso UC001 e UC002

A alteração feita no cadastro do candidato foi a inclusão do código da empresa entre os dados a serem cadastrados. Esta empresa será aquela na qual o candidato será vinculado no momento da criação do jogo. Nas versões anteriores o administrador selecionava os candidatos que fariam parte do jogo e, desta forma todas as empresas possuíam os mesmo colaboradores. Na nova versão do sistema, não será possível trabalhar desta forma devido a centralização do banco de currículos, fazendo com que os candidatos sejam únicos para todas as empresas do o jogo.

Nos quadros 3 e 4 são apresentados os detalhamentos destes casos de uso, sendo que no quadro 3 está destacada a alteração realizada, e no quadro 4 foi modificado todo o fluxo deste cenário.

UC001 – Gerenciar o Cadastro de Candidatos

Objetivo: Fazer e alterar cadastros de candidatos, para que estes possam, futuramente, ser contratados pelas empresas do jogo.

Pré-condição: Estar logado no sistema com o perfil de administrador.

Atores: Administrador

Fluxo Principal:

1. O Sistema mostra a tela de cadastro de Candidatos.
2. O Administrador informa um nome para o Candidato.
3. O Administrador informa uma idade para o Candidato.
4. O Administrador informa um salário para o Candidato.
5. O Administrador informa se o Candidato estará vinculado a uma empresa, e qual será esta empresa.
6. O Administrador informa qual será o comportamento do Candidato.
7. O Administrador cadastra as informações de Produção do Candidato.
8. O Administrador cadastra as informações de Maturidade do Trabalho do Candidato.
9. O Administrador cadastra as informações de Maturidade Psicológica do Candidato.
10. O Administrador informa os Fatores Perturbadores do Candidato.
11. O Sistema cria o candidato informado para o jogo.

Pós-condições: Candidato criado.

Quadro 3 – Detalhamento do caso de uso UC001 - Gerenciar o Cadastro de Candidatos

UC002 – Criar jogo

Objetivo: Criar um jogo composto por uma ou várias Empresas que serão disponibilizadas aos seus respectivos Jogadores.

Pré-condição: Possuir Candidatos cadastrados na base de dados.
Estar logado no sistema com o perfil de administrador.

Atores: Administrador

Fluxo Principal:

1. O Sistema apresenta a página para criação do jogo.
2. O Administrador informa o número de empresas a serem criadas.
3. O Administrador informa um nome para o jogo.
4. O Administrador informa se o jogo está integrado ao Virtual.
5. O Administrador confirma o cadastro do jogo.
6. O Sistema cria as empresa(s) informada(s) para o jogo, vincula e apresenta os candidatos pré-cadastrados à sua(s) respectiva(s) empresa(s).

Pós-condições: Candidatos associados às empresas.

Quadro 4 – Detalhamento do caso de uso UC002 – Criar jogo

A figura 13 ilustra o diagrama de caso de uso em um cenário em que o jogador

contrata um funcionário.

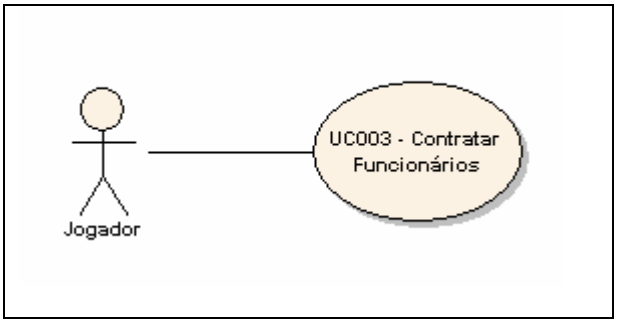


Figura 13 – Diagrama de caso de uso UC003

No quadro 5 pode-se ver o detalhamento desse caso de uso que foi totalmente alterado em relação a versão anterior do sistema. A alteração feita na contratação do funcionário é o principal objetivo deste projeto. Na versão anterior do sistema, o jogador poderia contratar qualquer candidato que ainda não estivesse vinculado à sua empresa, pois o seu banco de currículos era individual. Com a centralização deste banco, o sistema permite que uma empresa contrate um candidato já vinculado à outra empresa, fazendo, para isso, sua demissão na empresa anterior.

UC003 – Contratar Funcionários

Objetivo: Fazer a contratação de um candidato.

Pré-condição: Possuir uma empresa cadastrada pelo Administrador
Estar logado no sistema com o perfil de Jogador.
Atores: Jogador

Fluxo Principal:
1. O Sistema mostra a tela de Contratação.
2. O Jogador seleciona o(s) Candidato(s) que deseja contratar.
3. O Sistema vincula o candidato à empresa.

Fluxo Alternativo:
1. O Sistema mostra a tela de Contratação.
2. O Jogador seleciona o(s) Candidato(s) que deseja contratar.
3. O Sistema identifica que o Candidato já está vinculado a uma outra empresa, e disponibiliza uma caixa de texto para que o jogador possa fazer sua proposta.
4. O Jogador faz a proposta e confirma a contratação.
5. O Sistema processa as informações, e informa se a proposta será aceita ou não.

Pós-condições: O Sistema aloca este candidato no quadro de colaboradores no próximo período. No caso do fluxo alternativo, informa à empresa original que determinado funcionário recebeu proposta de outra empresa e se ele aceitou ou não.

Quadro 5 – Detalhamento do caso de uso UC003 – Contratar Funcionários

3.2.2 Diagramas de Atividades

O diagrama de atividades ilustra a seqüência de atividades, descrevendo os comportamentos condicionais ou de execução em paralelo. Este diagrama pode estar vinculado a um caso de uso, fazendo a demonstração visual do fluxo das atividades relacionadas às interações deste caso de uso. O diagrama de atividades também serve para fazer uma análise mais aprofundada em um determinado processo, sendo possível descobrir outros requisitos relacionados ao mesmo. A figura 14 ilustra as atividades de cadastrar um funcionário e a figura 15 a atividade de criar um jogo. A alteração feita neste processo foi a extinção da necessidade de selecionar os funcionários que irão fazer parte do quadro de colaboradores inicial da empresa.

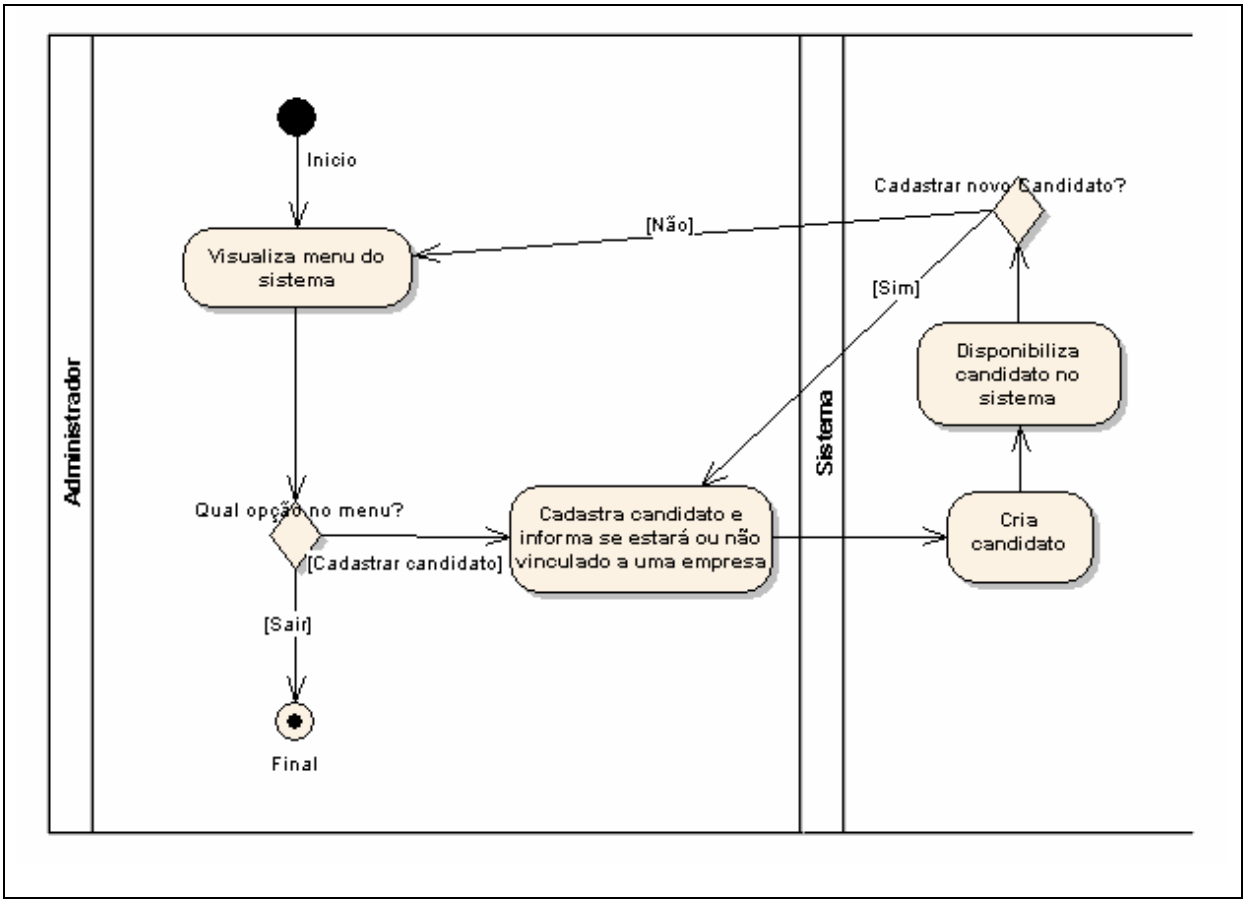


Figura 14 – Diagrama de atividade relacionado à atividade cadastrar Candidato

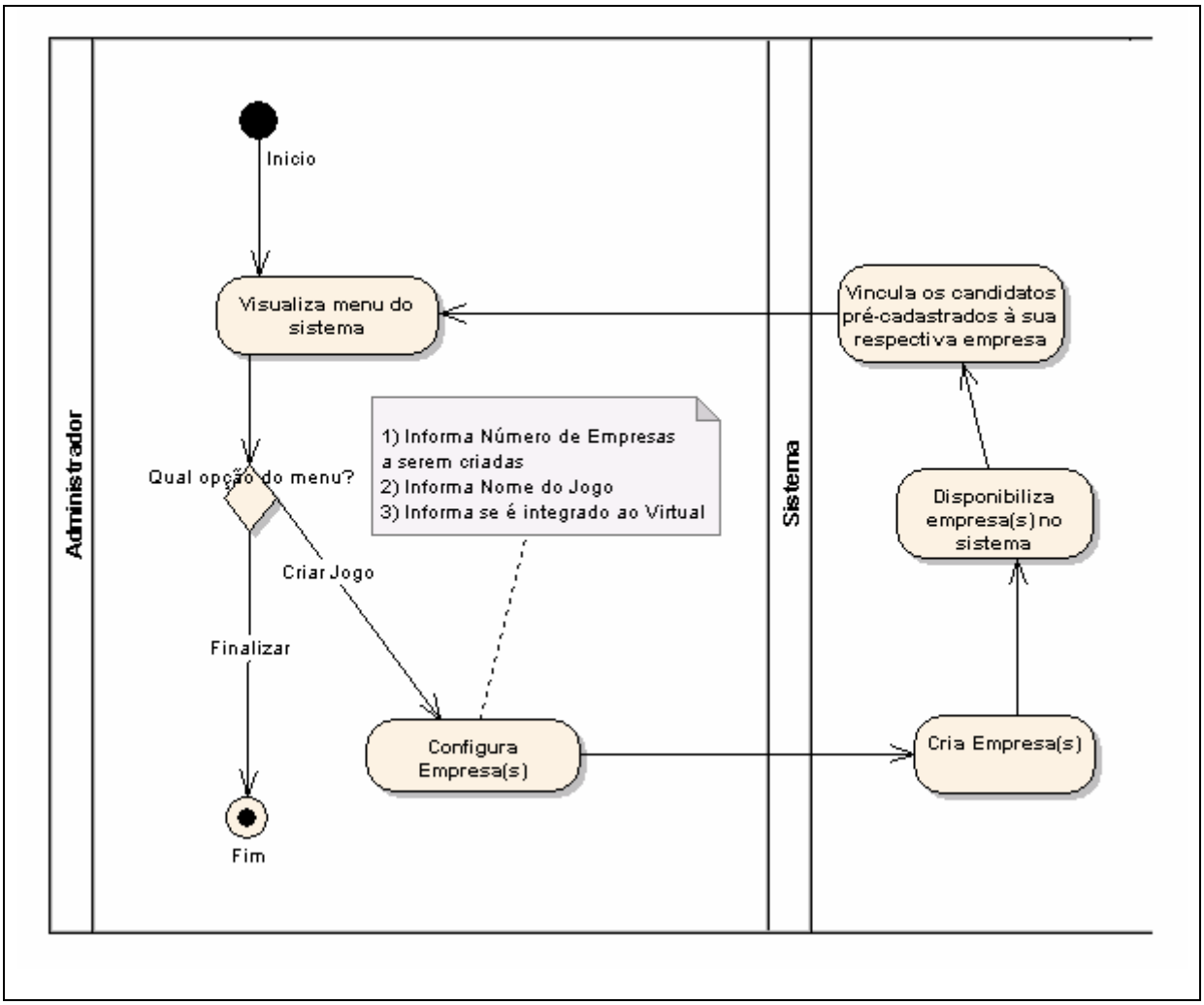


Figura 15 – Diagrama de atividade relacionado à atividade criar jogo

Na figura 16 é ilustrada a atividade de contratar um funcionário, baseada no UC003. Neste diagrama é possível visualizar qual o fluxo seguido pelo sistema quando o candidato a ser contratado, está ou não vinculado a uma empresa.

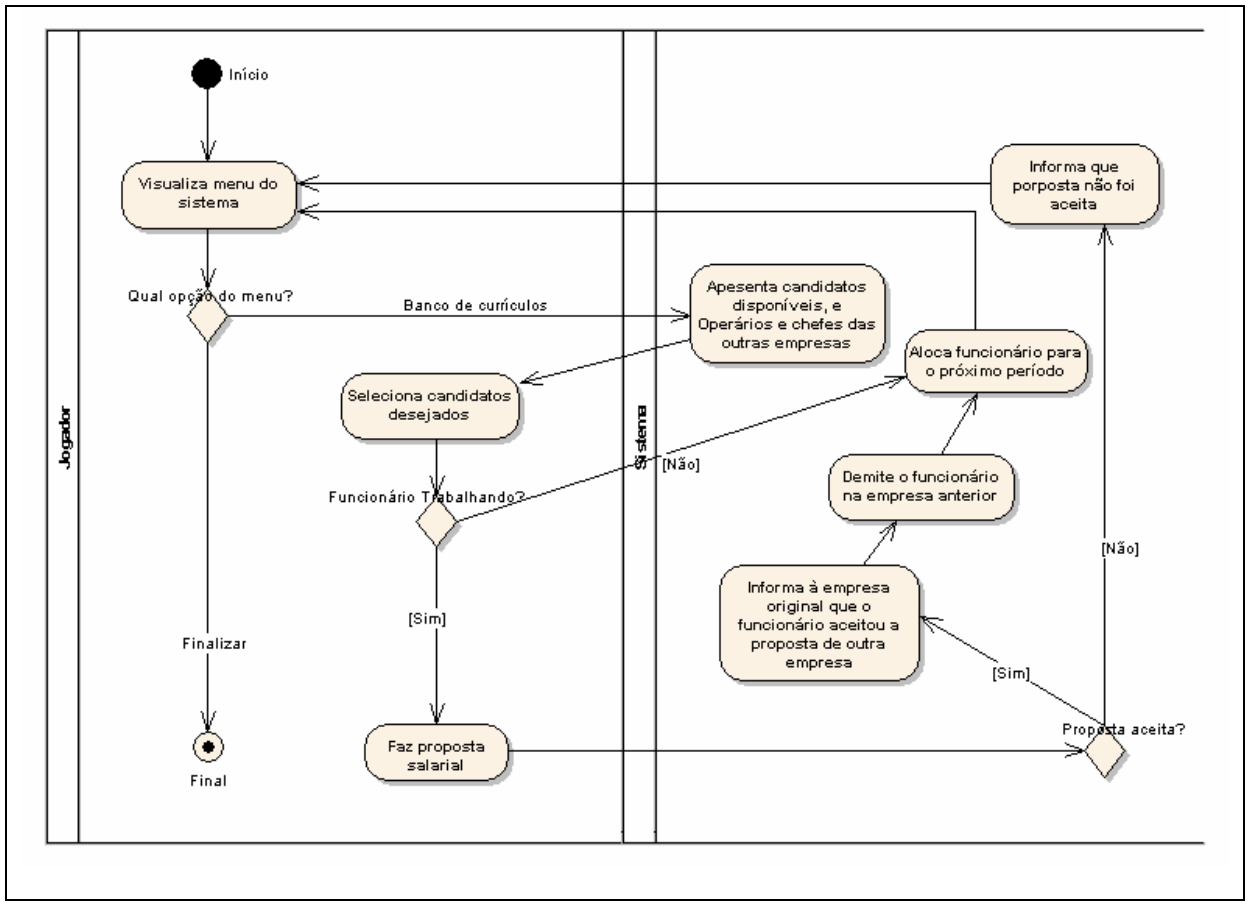


Figura 16 – Diagrama de atividade relacionado à atividade contratar funcionário

3.2.3 Diagramas de Seqüência

O diagrama de seqüência ilustra as mensagens trocadas entre o ator e o sistema, sendo que o sistema pode ser representado por dois artefatos: um que representa a interface na qual o ator irá executar as ações e outro que representa o controlador de sistema. A figura 17 ilustra o diagrama de seqüência relacionado à atividade criar jogo.

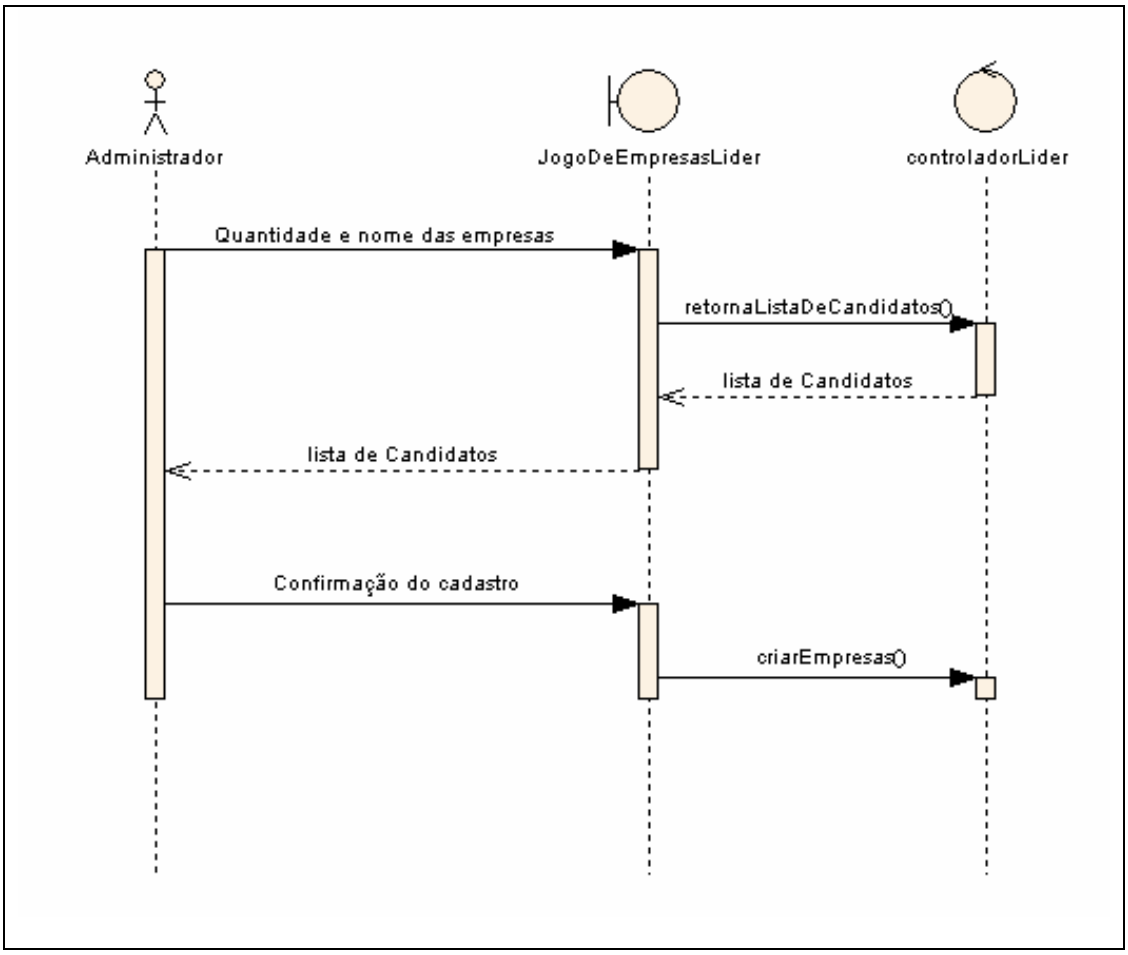


Figura 17 – Diagrama de seqüência relacionado à atividade criar jogo

A seqüência desta atividade foi bastante alterada, pois na versão anterior do jogo, primeiramente o sistema apresentava a lista de candidatos para o usuário selecionar, e agora não existe mais esta necessidade, devido ao vínculo entre empresa e candidato já ser feito no momento do cadastramento deste candidato.

A figura 18 ilustra o diagrama de seqüência relacionado à atividade contratar funcionário.

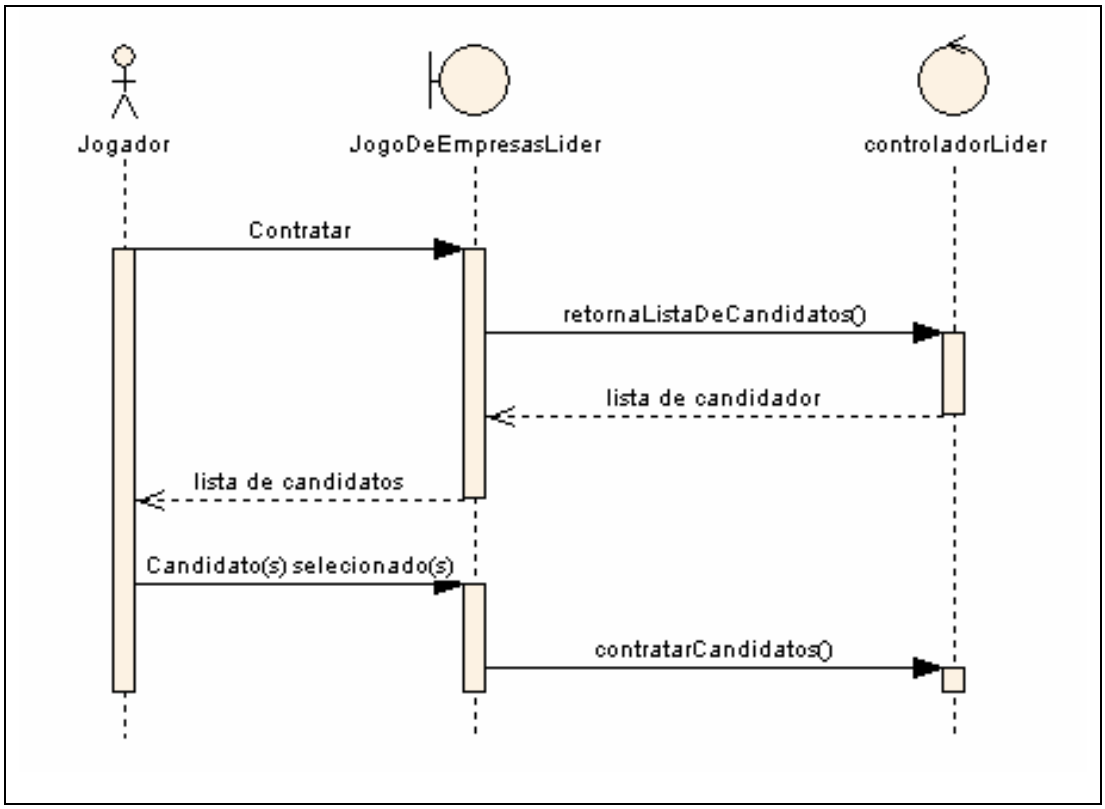


Figura 18 – Diagrama de seqüência relacionado à atividade contratar funcionário

A seqüência desta atividade não foi alterada, mas é importante apresentá-la, visto que a atividade é o foco principal do projeto.

3.2.4 Diagramas de Pacotes e de Classes

O diagrama de pacotes do jogo de empresas LÍDER 9 não sofreu alterações. No anexo A pode ser visualizado o diagrama de pacotes do sistema e sua descrição.

Na figura 19 é apresentado o diagrama de classes do pacote `controle`, que é encarregado de administrar as ações e eventos disparados na camada de visão do sistema, que provê a interface com o usuário. Neste projeto foi adicionada a classe `BancoCurriculo` a este pacote, que controla as requisições feitas à página JSF `BancoCurriculo.jsp`. A descrição das demais classes encontra-se no trabalho de Micheluzzi (2006).

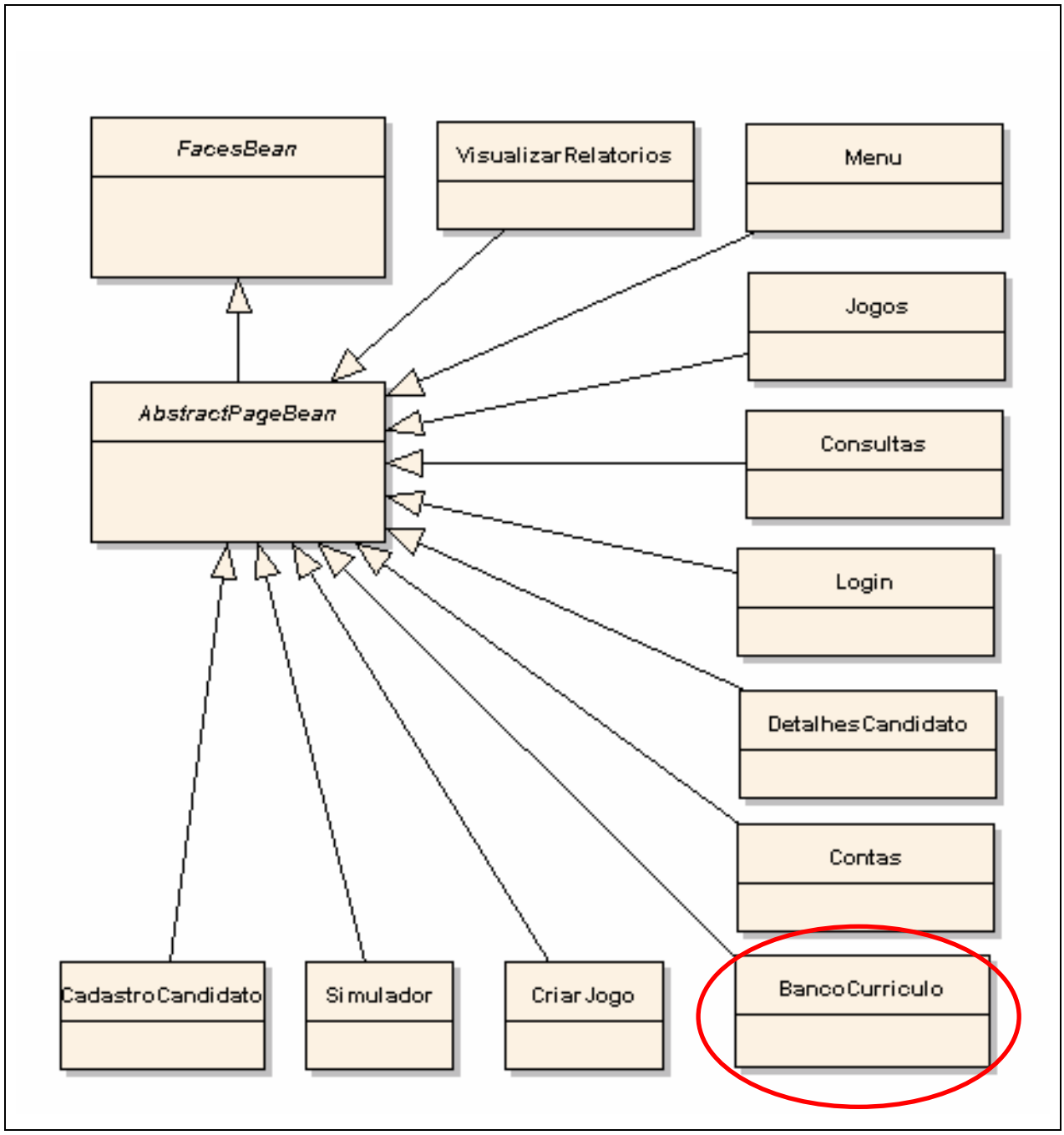


Figura 19 – Diagrama de classes das classes que compõem o pacote `controle` na nova versão do Jogo de Empresas Líder

Na figura 20 pode ser visto o diagrama de classes do pacote `bean`, Estas classes representam uma entidade do banco de dados. As classes desse pacote utilizam o padrão *Java Beans* para apenas encapsular informações relacionadas aos seus objetos. Elas são utilizadas pelo Hibernate para carregar os dados providos de um banco de dados e também são utilizadas em todo o modelo do sistema no qual manipula as informações encapsuladas nestes *beans* (MICHELUZZI, 2006). No desenvolvimento deste projeto foi adicionada a classe `Contratacao` a este pacote, esta classe encapsula os atributos relacionados às contratações realizadas em um período. A classe é utilizada para mapear as colunas da tabela `CONTRATACAO`

do banco de dados do sistema.. A descrição das classes do pacote `bean` podem ser visualizadas no trabalho de Micheluzzi (2006).

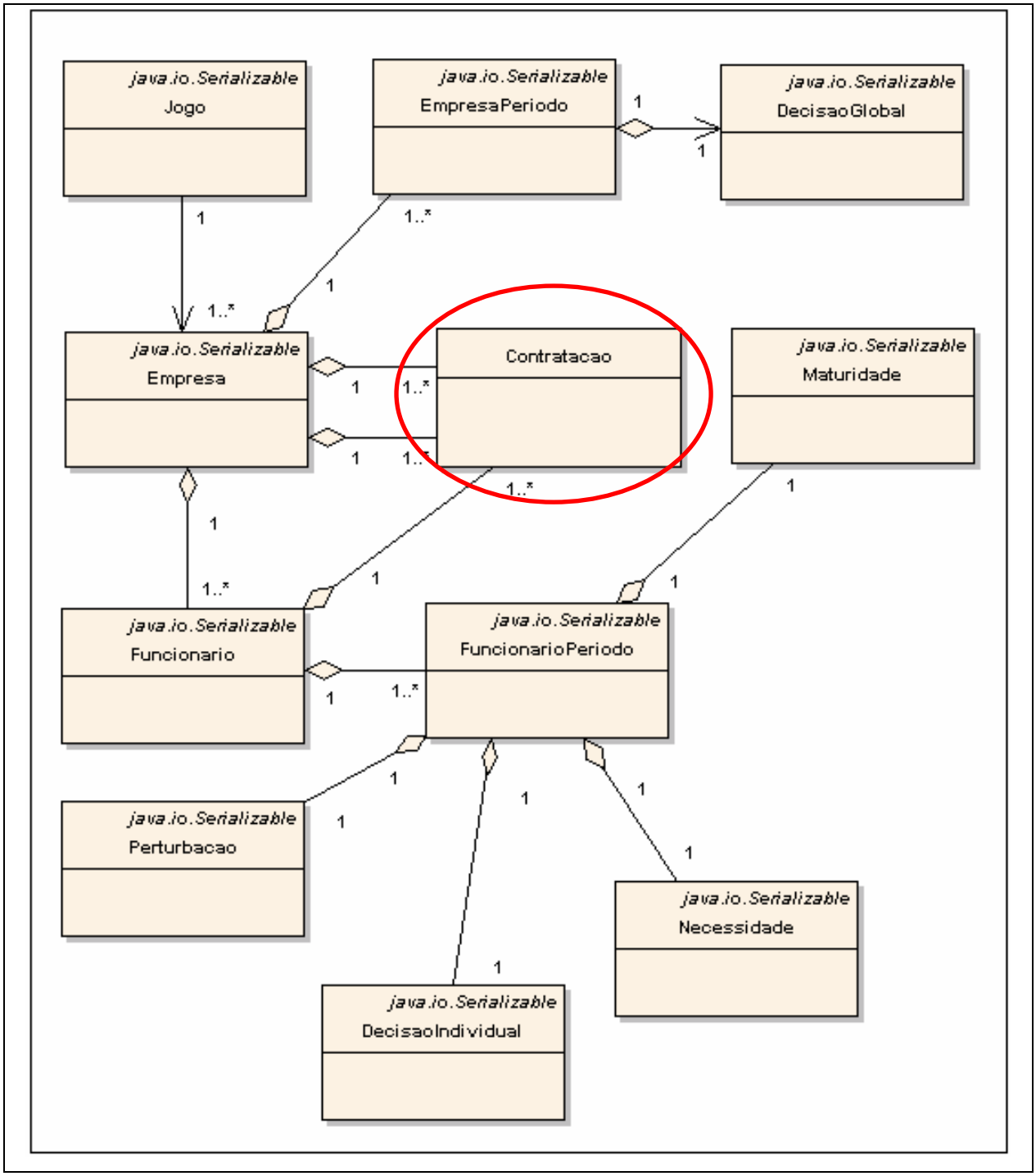


Figura 20 – Diagrama de classes das classes que compõem o pacote `bean` na nova versão do Jogo de Empresas Líder

As classes do pacote `business` foram desenvolvidas utilizando o padrão de projeto *Façade*, permitindo a redução do acoplamento entre as camadas de controle e de modelo. Sendo assim estas classes têm a função de fazer a ligação dos controladores com o modelo (MICHELUZZI, 2006). Neste projeto foi adicionada a classe `ContratacaoFacade`, que

possui os métodos relacionados com a manipulação dos objetos da classe `Contratacao` no modelo. O diagrama e a descrição do que cada classe do pacote `business` realiza encontra-se no trabalho de Micheluzzi (2006).

As classes que compõem o pacote `persistencia` possuem a responsabilidade de disponibilizar todas as funcionalidades de *create, read, update e delete* com o banco de dados. Elas foram projetadas utilizando o padrão de projeto DAO que abstrai a lógica de acesso ao banco de dados do resto da aplicação (MICHELUZZI, 2006). A alteração feita neste pacote, foi a inclusão da classe `ContratacaoDAO`, que é responsável por executar toda a lógica de acesso ao banco de dados para persistir objetos da classe `Contratacao` (pacote `bean`) na tabela `CONTRATACAO`. A figura 21 ilustra o diagrama de classes das classes que compõem o pacote `persistencia`, e a descrição destas classes encontra-se no trabalho de Micheluzzi (2006).

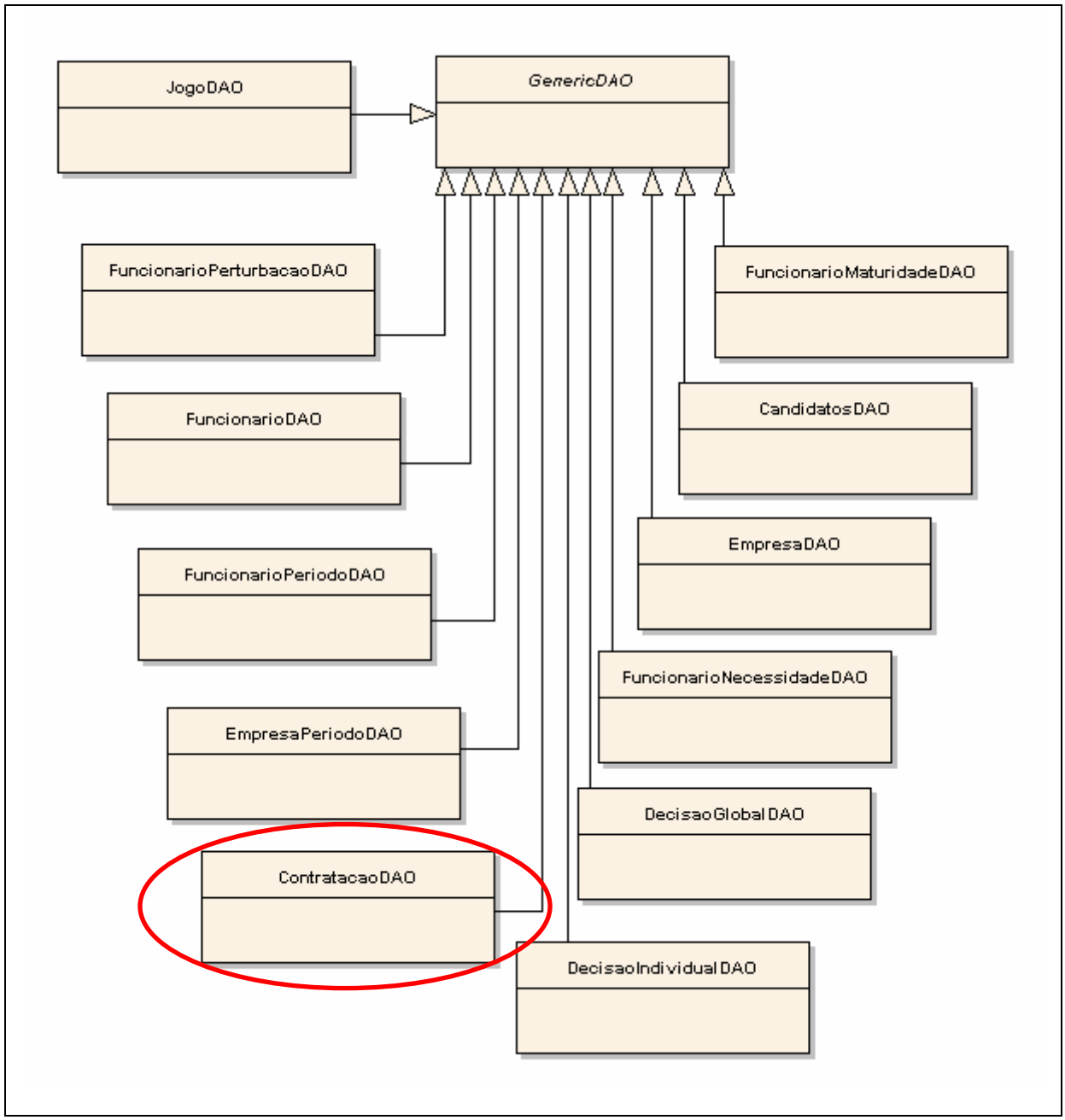


Figura 21 – Diagrama de classes das classes que compõem o pacote persistencia na nova versão do Jogo de Empresas Líder

3.2.5 Modelo de Entidade/Relacionamento (MER)

A figura 22 ilustra o MER do jogo de empresas Líder.

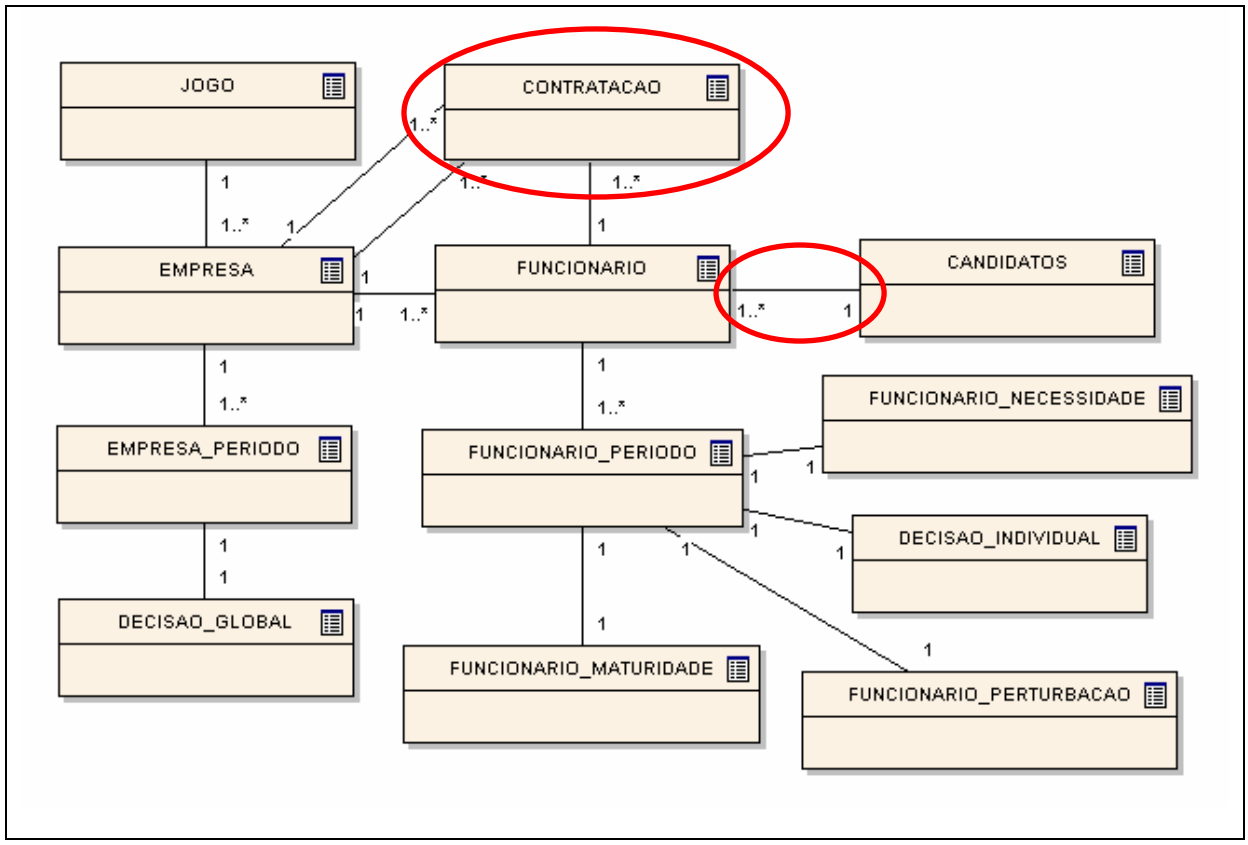


Figura 22 – MER definido para a nova versão do Jogo de Empresas Líder

Para o desenvolvimento deste projeto, algumas alterações na estrutura do banco de dados foram necessárias, como a inclusão do campo CODIGO_CANDIDATO na tabela FUNCIONARIO, fazendo assim o relacionamento entre as tabelas CANDIDATOS e FUNCIONARIOS que anteriormente não existia. Outra alteração foi a criação do campo CD_EMPRESA na tabela CANDIDATOS. Este campo nada tem a ver com a tabela EMPRESA, servindo apenas para indicar quais serão os funcionários iniciais de cada empresa no momento da criação de um novo jogo.

Também foi criada a tabela CONTRATACAO, que armazena as contratações feitas em um determinado período, indicando a empresa que o funcionário trabalhava anteriormente, a empresa que o contratou, o código deste candidato, o período que ocorreu a contratação e o salário oferecido a ele.

3.3 IMPLEMENTAÇÃO

Nesta seção é apresentada a fase de implementação do projeto assim como as ferramentas e técnicas utilizadas.

3.3.1 Técnicas e ferramentas utilizadas

Este projeto foi desenvolvido para a plataforma Java. O *framework* utilizado para o desenvolvimento do sistema foi o JSF, juntamente com a IDE Sun Java Studio Creator que permite um rápido desenvolvimento de aplicações JSF. O servidor de aplicações utilizado no desenvolvimento foi o Sun Java *System Application Server* 8.2 que já vem junto ao IDE por padrão. A camada de persistência foi desenvolvida com o *framework* Hibernate para fazer os mapeamentos objeto-relacional. Para a implantação do AJAX no sistema utilizou-se o *framework* Ajax4Jsf. O banco de dados escolhido para o desenvolvimento do sistema foi o MySQL.

A definição destas ferramentas foi feita por Micheluzzi (2006), que fez o trabalho de conversão do jogo, da plataforma Delphi para Java. E no projeto atual foram utilizadas as mesmas técnicas e ferramentas.

3.3.2 Implementação do sistema

A implementação do sistema foi feita conforme os seguintes passos:

a) especificação do sistema a partir de informações coletadas do software anterior;
b) alterações no banco de dados;
c) alteração da rotina de criação do jogo;
d) criação da possibilidade de uma empresa contratar o funcionário de outra;
e) criação de fatores condicionais para permitir ou não uma contratação.

3.3.2.1 Especificação do sistema a partir do software anterior

Nesta primeira etapa foi feito um estudo aprofundado de como havia sido implementado o software Líder 9 e quais os padrões utilizados. O software estava bem documentado e estruturado, o que facilitou este estudo mesmo sendo um projeto complexo e extenso.

3.3.2.2 Alterações na camada de persistência

Com a criação da tabela CONTRATACAO, foi necessário o tratamento dos dados relacionados a esta tabela na camada de persistência. A camada de persistência do sistema foi elaborada utilizando o *framework* Hibernate e o padrão DAO. Este padrão tem por objetivo separar a lógica de acesso ao banco de dados do resto da aplicação, ou seja, são criados uma ou várias classes DAOs que irão fazer todo o trabalho de consultar, alterar, remover ou inserir dados no banco (MICHELUZZI, 2006).

O primeiro passo foi o mapeamento objeto-relacional, feito na classe *bean* utilizando as *annotations*[2] disponibilizadas no pacote `javax.persistence`, conforme pode ser visualizado no quadro 6.

```
package lider.model.bean;
[...]
//Define que o bean é uma entidade persistente
@Entity
//mapeia a tabela que o bean está relacionado
@Table(name = "contratacao", catalog = "lider", uniqueConstraints = {})
public class Contratacao implements java.io.Serializable {
   private Integer idContratacao;
   private Integer empresaAnterior;
   private Integer empresaNova;
   private Integer codigoCandidato;
   private Integer codigoEmpresa;
   private Double codigoCandidato;
   private String periodoContratacao;
   private boolean salarioNovo;
   [...]
   // mapeia a coluna que possui a chave primária da tabela
   @Id @GeneratedValue(strategy=GenerationType.AUTO )
   @Column(name=" ID_CONTRATACAO", unique=true, nullable=false, insertable=true
updatable=true)
   public Integer getIdContratacao() {
      return this. idContratacao;
   }
   [...]
   // mapeia a coluna EMPRESA_ANTERIOR
   @Column(name="EMPRESA_ANTERIOR", unique=false, nullable=true, insertable=true,
updatable=true)
   public Integer getEmpresaAnterior() {
     return this. empresaAnterior;
   }
   [...]
   // mapeia a coluna EMPRESA_NOVA
   @Column(name="EMPRESA_NOVA", unique=false, nullable=true, insertable=true,
updatable=true)
   public Integer getEmpresaNova() {
      return this. empresaNova;
   }
   [...]
}
```

Quadro 6 – Código fonte da classe `Contratacao`

[2] As *annotations* são uma alternativa ao XML e podem ser amplamente utilizadas em todo o sistema com finalidades diferentes (ZUKOWSKI, 2006).

Em seguida foi necessário a inclusão da classe `Contratacao` no arquivo hibernate.cfg.xml, informando que esta classe está mapeada e deve ser persistida no banco de dados, conforme quadro 7.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Configura a conexão com banco de dados assim como o dialeto que o
         Hibernate irá utilizar-->
        <property name="hibernate.bytecode.use_reflection_optimizer">
            false</property>
        [...]

        <!-- Configurações do hibernate -->
        <property name="hibernate.format_sql">true</property>
        [...]

        <!-- Configura o pool de conexões interno -->
        <property name="hibernate.c3p0.max_size">10</property>
             <property name="hibernate.c3p0.min_size">2</property>
             [...]

        <!--Classes mapeadas -->
        <mapping class="lider.modelo.bean.Maturidade" />
        [...]
        <mapping class="lider.modelo.bean.Contratacao"/>
    </session-factory>
</hibernate-configuration>
```

Quadro 7 – arquivo `hibernate.cfg.xml`

O próximo passo foi a criação da classe `ContratacaoDAO`, que possui os comandos de manipulação do banco de dados de sua classe ancestral `GenericDAO`.

3.3.2.3 Alteração da rotina de criação do jogo

Na versão anterior do software, para a criação do jogo o administrador, além das empresas que iriam participar do jogo, também selecionava os candidatos que iriam fazer parte do quadro de funcionários de cada empresa. Na nova versão, os candidatos já estão pré-

vinculados às empresas, e a primeira alteração no software foi na camada de visão da criação do jogo, utilizando a tabela de candidatos, mas somente para visualização dos candidatos que serão vinculados à empresa.

No quadro 8 pode-se observar a implementação do AJAX4JSF na página JSP, utilizada para popular uma tabela com os candidatos que serão vinculados à empresa, no momento em que o botão a frente do nome desta empresa for clicado.

```
<h:dataTable binding="#{controle$criaJogo.dataTable1}" headerClass="tabela-header"
    id="dataTable1" rowClasses="list-row-even,list-row-odd"
    value="#{controle$criaJogo.tabelaNomeEmpresas}" var="currentRow">
    <h:column binding="#{controle$criaJogo.column14}" id="column14">
        <a4j:commandButton action="#{controle$criaJogo.buscarCandidatosEmpresa}"
            id="empresaButton" reRender="dataTable2"
            value="#{currentRow['cdEmpresa']}"/>
        <f:facet name="header">
            <h:outputText binding="#{controle$criaJogo.outputText28}"
                id="outputText28" value="Cand."/>
        </f:facet>
    </h:column>
    <h:column binding="#{controle$criaJogo.column1}" id="column1">
        <h:inputText binding="#{controle$criaJogo.textField1}" id="textField1"
            value="#{currentRow['empresa']}"/>
        <f:facet name="header">
           <h:outputText binding="#{controle$criaJogo.outputText3}"
                id="outputText3" value="Nome Empresa"/>
        </f:facet>
    </h:column>
</h:dataTable>
```

Quadro 8 – Implementação do AJAX4JSF utilizada para popular uma tabela com os candidatos que serão vinculados a empresa

No quadro 9 é apresentado o método da classe de controle `CriaJogo`, que popula a tabela de candidatos.

```
public void buscarCandidatosEmpresa(){
    CandidatoFacade cb = new CandidatoFacade();
    HashMap linhaAtual = new HashMap();
    Integer cdEmpresa = null;
    if(this.tabelaNomeEmpresas.getRowData() != null  ){
        linhaAtual = (HashMap) this.tabelaNomeEmpresas.getRowData();
        cdEmpresa = (Integer) linhaAtual.get("cdEmpresa");

        List candidatosEmpresa = cb.retornaCandidatosEmpresa(cdEmpresa);
        List tabela= new ArrayList();
        DecimalFormat dfProdicao = new DecimalFormat();
        dfProdicao.applyPattern("#####0.00");
        for ( int i = 0 ; i < candidatosEmpresa.size() ; i++ ){
            Candidatos c = (Candidatos) candidatosEmpresa.get(i);
            HashMap linha = new HashMap();
            linha.put("nome", c.getNome() );
            linha.put("idade", c.getIdade() );
            linha.put("empresa", c.getEmpresa() );
            linha.put("cargo", c.getCargo() );
            linha.put("setor", c.getSetor() );
            linha.put("producao", dfProdicao.format(c.getProducaoReal()));
            linha.put("no", Integer.valueOf(c.getNo().intValue()));
            linha.put("ne", Integer.valueOf(c.getNe().intValue()));
            linha.put("so", Integer.valueOf(c.getSo().intValue()));
            linha.put("se", Integer.valueOf(c.getSe().intValue()));
            linha.put("detalhe", c.getCodigoCandidato());
            linha.put("candidato",c);
            tabela.add(linha);
        }
        this.tabelaCandidatos.setWrappedData(tabela);
    }
}
```

Quadro 9 – Método que popula a tabela de candidatos

No quadro 10 é apresentado o método para fazer a chamada de uma classe DAO que faz a busca dos candidatos. Este método pertence a uma classe que implementa o padrão *Façade* que faz a conexão entre a camada de controle e o modelo.

```
public List retornaCandidatosEmpresa(Integer empresa)throws LiderExceptions{
    try {
        CandidatosDAO candidatosDAO = new CandidatosDAO();
        List lista = candidatosDAO.retornaCandidatosEmpresa(empresa);
        HibernateUtil.comitarTransacao();
        return lista;
    } catch (PersistenciaExceptions e){
        HibernateUtil.cancelarTransacao();
        throw new LiderExceptions(e.getMessage());
    } catch (LiderExceptions e){
        HibernateUtil.cancelarTransacao();
        throw e;
    } catch (Exception e){
        HibernateUtil.cancelarTransacao();
        throw new LiderExceptions(e.getMessage());
    }
}
```

Quadro 10 – Método para fazer a chamada de uma classe DAO que faz a busca dos candidatos

O quadro 11 apresenta o método da classe DAO que faz a busca dos candidatos.

```
public List retornaCandidatosEmpresa(Integer empresa) throws PersistenciaExceptions
{
    List lista = new ArrayList();
    try{
        Query select = HibernateUtil.getSession().createQuery(" from Candidatos c
                                where c.empresa = :cd_empresa order by c.nome ");
        select.setInteger("cd_empresa", empresa);
        lista = select.list();
    } catch ( HibernateException h ){
        HibernateUtil.cancelarTransacao();
        throw new PersistenciaExceptions("Erro ao buscar os candidatos da
            empresa.",h);
    } catch ( Exception e){
        HibernateUtil.cancelarTransacao();
        throw new PersistenciaExceptions("Erro ao buscar os candidatos.",e);
    }
    return lista;
}
```

Quadro 11 – Método da classe DAO que faz a busca dos candidatos

Para este projeto também foram necessárias algumas alterações no método `salvarJogoButton1_action()`, como a inclusão do código da empresa como parâmetro, no método `retornaListNovosFuncionarios()` que transforma os candidatos em funcionários, como mostra o quadro 12.

```
public String salvarJogoButton1_action() {

    try{
        Integer numEmp = (Integer) this.numEmpresasTF.getValue();

        List listaEmpresa = new ArrayList();

        //TODO Validar se entrou com a mesma empresa e o mesmo jogador....esta
        //situação nao pode ser permitida.
        List listEmpresaJogo;
        if(this.tabelaNomeEmpresas.getWrappedData() != null  ){
            listEmpresaJogo = (List) this.tabelaNomeEmpresas.getWrappedData();
        } else {
            listEmpresaJogo = new ArrayList();
        }
        // cria uma lista com todas as empresas que irão compor o jogo.
        for(int i = 0 ; i < listEmpresaJogo.size(); i++){
            CandidatoFacade cb = new CandidatoFacade();
            List listaCandidatos = cb.retornaCandidatosEmpresa(i);

            [...]
            Empresa empresa = new Empresa();
            HashMap linha = (HashMap) listEmpresaJogo.get(i);
            String nomeEmpresa = (String) linha.get("empresa");
            empresa.setNomeEmpresa(nomeEmpresa);
            empresa.setIntegradoVirtual(this.integradoVirtualCheckbox.isChecked());
            empresaPeriodo.setEmpresa(empresa);
            Set setEmpresaPeriodo = new HashSet();
            setEmpresaPeriodo.add(empresaPeriodo);
            empresa.setEmpresaPeriodo(setEmpresaPeriodo);
            List setFuncionarios;
            setFuncionarios =
                this.retornaListNovosFuncionarios(listaCandidatos,empresa);

            empresa.setFuncionario(setFuncionarios);

            listaEmpresa.add(empresa);
        }

        String nomeJogo = (String) this.nomeJogoTF.getValue();
        String senhaInicial = (String) this.senhaTF.getValue();
        EmpresaFacade empresaBusiness = new EmpresaFacade();
        // salva o jogo
        empresaBusiness.criarJogo(listaEmpresa,senhaInicial,nomeJogo);
```

```
            info("Empresa(s) cadastrada(s) com sucesso!");
            this.nomeJogoTF.setValue("");
            this.senhaTF.setValue("");
            this.numEmpresasTF.setValue("");
            this.tabelaNomeEmpresas.setWrappedData(new ArrayList());
        } catch (LiderExceptions e) {
            error(e.getMessage());
        }
        return "confirmaCadastro";
    }
```

Quadro 12 – Método `salvarJogoButton1_action()`

Após estas implementações foram feitos vários testes de validação e a rotina comportou-se da maneira esperada, permitindo assim que fosse passado para a próxima etapa do projeto.

3.3.2.4 Criação da possibilidade de uma empresa contratar o funcionário de outra

Nesta etapa foi implementado um dos principais objetivos do projeto que é a possibilidade de uma empresa contratar o funcionário de outra, gerando assim um ambiente mais competitivo para o jogo.

Para esta implementação foi criada a página bancoCurriculo.jsp. Seu desenvolvimento foi baseado na página lvContratar.jsp, existente na versão anterior. Esta nova página apresenta, além dos candidatos disponíveis para contratação, também os funcionários das outras empresas participantes do jogo.

Um dos maiores desafios desta implementação, foi garantir que a situação inicial do funcionário contratado fosse à última situação da empresa onde trabalhava anteriormente. A dificuldade era devido às informações dos funcionários estarem distribuídas por várias classes e tabelas diferentes, enquanto as informações de um candidato estão todas centralizadas em um só lugar.

O quadro 13 apresenta o método `init()` da classe de controle `BancoCurriculo`. Este método verifica se o período selecionado já foi ou não processado: caso tenha sido processado o sistema irá listar os candidatos contratados neste período; caso contrário será apresentada uma lista de candidatos para o jogador contratar.

```
public void init() {
    [...]
    // Caso o periodo já foi processado ou está liberado para reprocessamento
    // não é mais possivel contratar ou desistir da contratação de candidatos
    if (this.getSessionBeanLider().
        getEmpresaPeriodo().getProcessado().equalsIgnoreCase("P") ||
        this.getSessionBeanLider().
        getEmpresaPeriodo().getProcessado().equalsIgnoreCase("R") ) {
        // lista os funcionarios contratados no periodo.
        this.listaFuncionariosContratadosNoPeriodo();
        this.mostraTabelaContratar = false;
        this.mostraTabelaFuncionariosContratados = true;
    } else {
        // lista todos os candidatos que ainda não foram contratados pela
        // empresa.
        this.permiteContratar();
        this.mostraTabelaContratar = true;
        this.mostraTabelaFuncionariosContratados = false;
    }
}
```

Quadro 13 – Método init() da classe de controle bancoCurriculo

Considerando que o período ainda não tenha sido processado, o sistema irá chamar o método `permiteContratar()`, listado no quadro 14. Este método busca todos os candidatos do jogo que não estão vinculados à empresa corrente. Tendo esta lista, ele verifica se os candidatos estão vinculados em outra empresa ou não. Em caso afirmativo, os dados sobre salário, experiência e comportamento a ser listados na tabela serão os dados atuais do funcionário na empresa em que ele está trabalhando, caso contrário os dados serão buscados do cadastro do candidato.

```
public void permiteContratar(){
    ConsultasFacade consultasFacade = new ConsultasFacade();
    try{
        // lista que contem todos os nomes do funcionarios já existentes. Serve
        // para passar de parametro para que a consulta não retorne os
        // funcionarios já existentes.
        List listaFuncionariosExistentes = new ArrayList();
        if (this.getSessionBeanLider().getEmpresaPeriodo() != null ){
            List funcionarios = this.getSessionBeanLider().getEmpresaPeriodo().
                getEmpresa().getFuncionario();
            for(int i = 0 ; i < funcionarios.size() ; i++){
                Funcionario f = (Funcionario) funcionarios.get(i);
                listaFuncionariosExistentes.add(f.getNome());
            }
        }
        List listaCandidatos = consultasFacade.
            retornaConsultaCandidatoseFuncDoJogo(this.getSessionBeanLider().
            getJogo(),this.getSessionBeanLider().getCodigoEmpresaJogador(),
            this.getSessionBeanLider().getPeriodo());
        List listaTabelaCandidatos = new ArrayList();
        for(Iterator it = listaCandidatos.iterator(); it.hasNext();){
            Candidatos c = (Candidatos) it.next();
            Map linha = new HashMap();
            linha.put("nome",c.getNome());
            linha.put("idade",c.getIdade());
            Funcionario funcionarioTrabalhando = consultasFacade.
                retornaCandidatoTrabalhando(c.getCodigoCandidato(),
                this.getSessionBeanLider().getPeriodo());
            [...]
            if (funcionarioTrabalhando != null){
                //carrega dados do funcionário na tabela
                [...]
            }
            else {
                //carrega dados do candidato na tabela
                [...]
            }
            [...]
            linha.put("candidato",c);
            listaTabelaCandidatos.add(linha);
        }
        this.getSessionBeanLider().getTabelaCandidatos().
            setWrappedData(listaTabelaCandidatos);
    } catch (LiderExceptions e) {
        error(e.getMessage());
```

```
    }
}
```

Quadro 14 – Método permiteContratar() da classe de controle bancoCurriculo

Para concluir a contratação, o jogador seleciona os candidatos desejados e clica em contratar. Este evento irá chamar o método `retornaListNovosFuncionarios()`, apresentado no quadro 15, o qual traz uma lista do candidatos selecionados e, usando a mesma lógica do método `permiteContratar()`, verifica se cada candidato selecionado está ou não vinculado a uma empresa. Se estiver, o sistema busca as informações do período, como necessidades, maturidade e perturbações do último período em que o funcionário estava na outra empresa, caso contrário, estas informações virão do cadastro do candidato.

```
private List retornaListNovosFuncionarios(List listaCandidato){
    List setFuncionarios = new ArrayList();
    for(int i = 0 ; i < listaCandidato.size() ; i++){
        Candidatos candidato = (Candidatos) listaCandidato.get(i);
        Funcionario funcionario = new Funcionario();
        //carrega para o funcionário, os dados que não alteram ao passar do tempo
        [...]
        ConsultasFacade consultasFacade = new ConsultasFacade();
        Funcionario funcionarioTrabalhando = consultasFacade.
            retornaCandidatoTrabalhando(candidato.getCodigoCandidato(),
            this.getSessionBeanLider().getPeriodo());
        //se esse candidato estiver em outra empresa, buscar os dados da empresa
        if (funcionarioTrabalhando != null) {
            FuncionarioPeriodo funcionarioPeriodoTrabalhando = (FuncionarioPeriodo)
                funcionarioTrabalhando.getFuncionarioPeriodo();
            funcionarioPeriodoTrabalhando.setPeriodo(this.getSessionBeanLider().
                getPeriodo());
            Set setFuncionarioPeriodo = new HashSet();
            setFuncionarioPeriodo.add(funcionarioPeriodoTrabalhando);
            funcionario.setFuncionarioPeriodo(setFuncionarioPeriodo);
            funcionarioPeriodoTrabalhando.setFuncionario(funcionario);
            setFuncionarios.add(funcionario);
            Necessidade necessidadeTrabalhando = (Necessidade)
                funcionarioPeriodoTrabalhando.getNecessidade();
            necessidadeTrabalhando.setFuncionarioPeriodo(
                funcionarioPeriodoTrabalhando);
            funcionarioPeriodoTrabalhando.setNecessidade(necessidadeTrabalhando);
            Maturidade maturidadeTrabalhando = (Maturidade)
                funcionarioPeriodoTrabalhando.getMaturidade();
            maturidadeTrabalhando.setFuncionarioPeriodo(
                funcionarioPeriodoTrabalhando);
            funcionarioPeriodoTrabalhando.setMaturidade(maturidadeTrabalhando);
            Perturbacao perturbacaoTrabalhando = (Perturbacao)
                funcionarioPeriodoTrabalhando.getPerturbacao();
            perturbacaoTrabalhando.setFuncionarioPeriodo(
                funcionarioPeriodoTrabalhando);
            funcionarioPeriodoTrabalhando.setPerturbacao(perturbacaoTrabalhando);
            DecisaoIndividual decisaoTrabalhando = (DecisaoIndividual)
                funcionarioPeriodoTrabalhando.getDecisaoIndividual();
            decisaoTrabalhando.setFuncionarioPeriodo(
                funcionarioPeriodoTrabalhando);
            funcionarioPeriodoTrabalhando.setDecisaoIndividual(decisaoTrabalhando);
            //Salva as informações da contratação
            Contratacao contratacao = new Contratacao();
```

```
            contratacao.setEmpresaAnterior(
                funcionarioTrabalhando.getEmpresa().getCodigoEmpresa());
            contratacao.setEmpresaNova(
                funcionario.getEmpresa().getCodigoEmpresa());
            contratacao.setCodigoCandidato(funcionario.getCodigoCandidato());
            contratacao.setPeriodoContratacao(
                this.getSessionBeanLider().getPeriodo());
            contratacao.setSalarioNovo(funcionarioPeriodoTrabalhando.getSalario());
            ContratacaoFacade contratacaoF = new ContratacaoFacade();
            contratacaoF.cadastroContratacao(contratacao);
        }
        else {
            FuncionarioPeriodo funcionarioPeriodo = new FuncionarioPeriodo();
            //carrega os dados vindos do cadastro do canditdato
            [...]
        }
    }
    return setFuncionarios;
}
```

Quadro 15 – Método retornaListNovosFuncionarios() da classe de controle bancoCurriculo

No final deste processo, é cadastrado um registro na tabela CONTRATACAO. Foi implementado para quando o jogador for processar um determinado período, o sistema verificar se algum dos funcionários da empresa atual foi contratado por alguma outra empresa sendo que esta verificação pode ser feita na tabela CONTRATACAO.

3.3.2.5 Criação de fatores condicionais para permitir ou não uma contratação

Visando dar mais realismo ao jogo, foram definidos alguns critérios para indicar se uma empresa poderá ou não contratar um funcionário de outra empresa.

Definidas as regras de negócio, foi determinado que as variáveis que decidirão se uma empresa poderá ou não contratar um determinado funcionário, será o valor do salário oferecido, este por sua vez deverá ser maior que o salário recebido na empresa anterior, e a média da eficácia da liderança da nova empresa deverá ser maior que a eficácia da liderança da empresa anterior sobre este funcionário.

Foi definido também que após um candidato ser contratado, este só poderá ser contratado por uma outra empresa, no próximo período.

3.3.3 Operacionalidade da implementação

A operacionalidade completa do jogo, pode ser visualizada na seção 2.2. A seguir serão apresentadas somente as rotinas onde houve alteração.

A primeira tela alterada foi o cadastro do candidato, ilustrado na figura 23. Agora o administrador pode informar a qual empresa o candidato estará pré-vinculado.

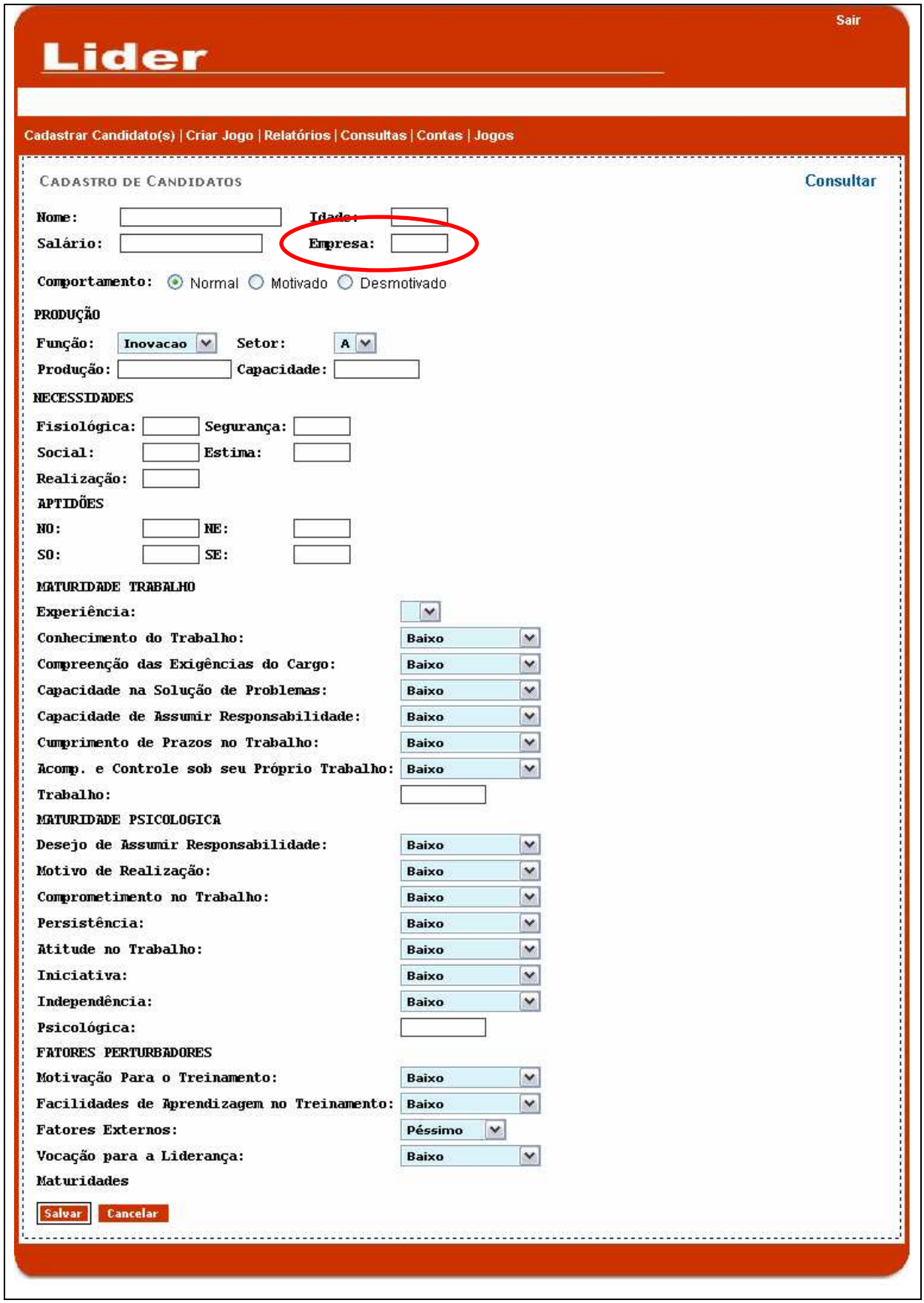


Figura 23 – Tela de cadastro de candidato da nova versão

Outra alteração para o administrador foi na criação de um novo jogo. Agora basta que

ele informe a quantidade de empresas, o nome do jogo e se o jogo será integrado com o Virtual, não havendo mais a necessidade de selecionar os candidatos que irão fazer parte do quadro de funcionário do jogo. A tela em questão pode ser visualizada na figura 24. Na tabela com o nome das empresas foi adicionado a coluna “Func.”. Clicando no botão existente nesta coluna, o sistema apresenta uma tabela com os funcionários que irão fazer parte do quadro de funcionários desta empresa.

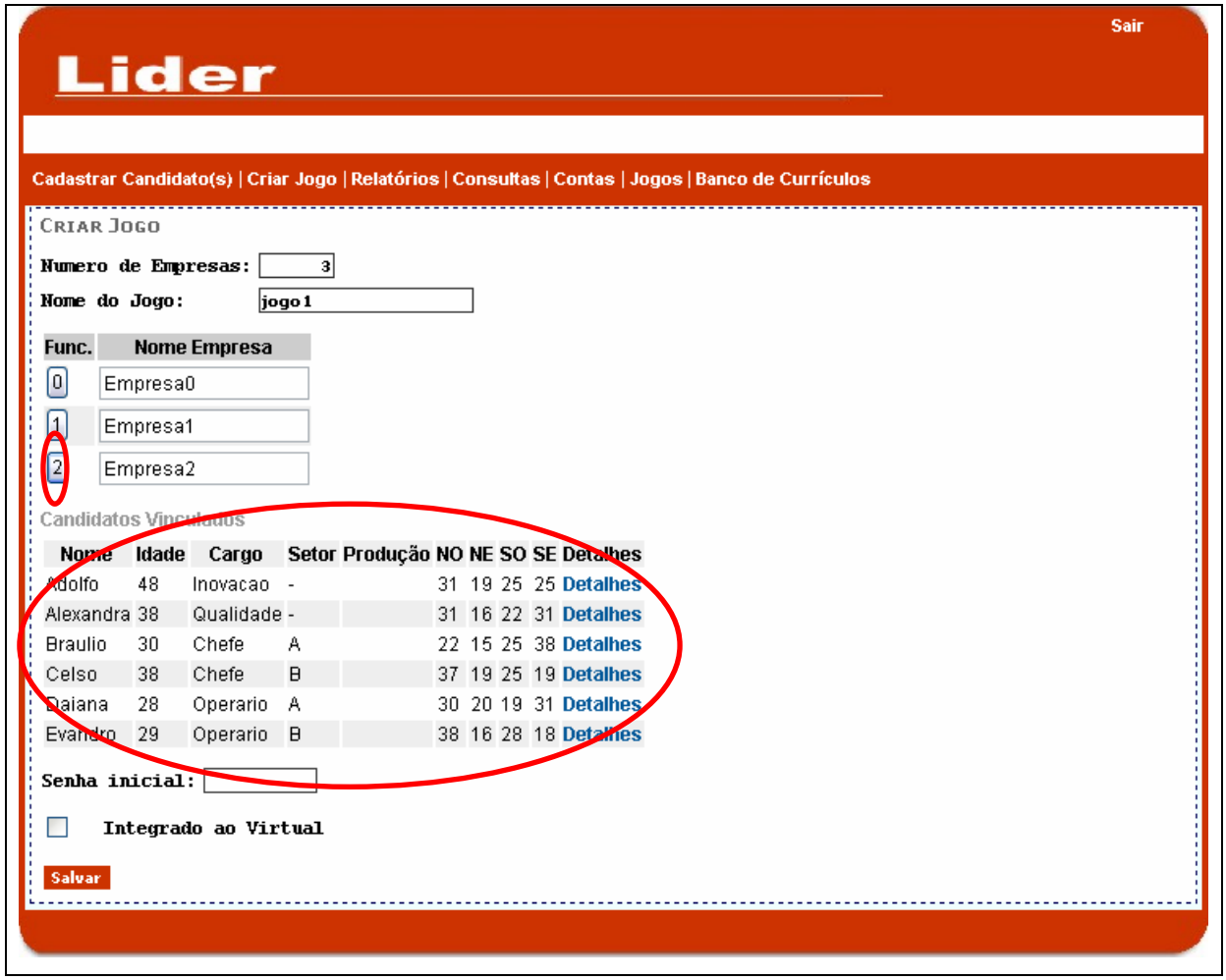


Figura 24 – Tela Criar Jogo da nova versão

Para o jogador, foi adicionado o botão Banco de Currículos, conforme pode ser visualizado na figura 25.

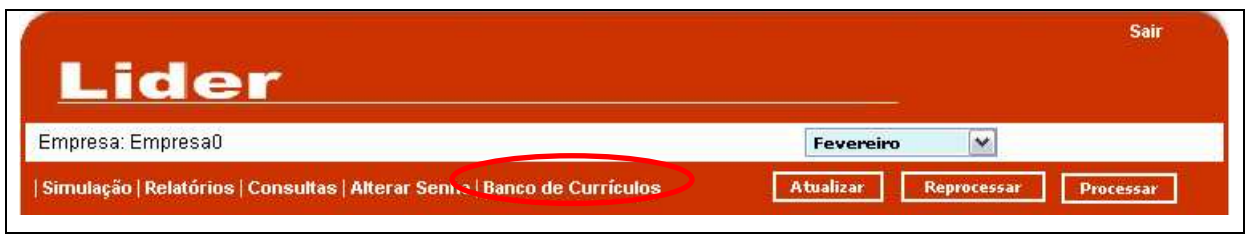


Figura 25 – Menu de jogador da nova versão

Este botão dá acesso à página JSP bancoCurriculo, apresentada na figura 26. Esta página apresenta uma tabela com todos os candidatos pertencentes ao jogo que não estejam

vinculados à empresa atual, informando suas características de comportamento, se está vinculado a uma empresa e qual seu salário atual. O jogador deve selecionar o(s) candidato(s) que deseja contratar, e caso este estiver vinculado a uma outra empresa, deverá fazer uma proposta salarial, preenchendo o campo da coluna “Novo Salário”, pois este valor será utilizado para saber se a proposta será ou não aceita.

Figura 26 – Tela Base de Currículos da nova versão

Quando um candidato que estava trabalhando for contratado, a empresa que perdeu este funcionário deverá ser avisada. Isto acontecerá quando esta empresa tentar fazer o processamento de suas decisões. Neste momento, aparecerá uma mensagem informando que determinado funcionário foi contratado por outra empresa e perguntado se deseja processar assim mesmo ou voltar ao período para fazer novas contratações. Esta mensagem pode ser visualizada na figura 27.

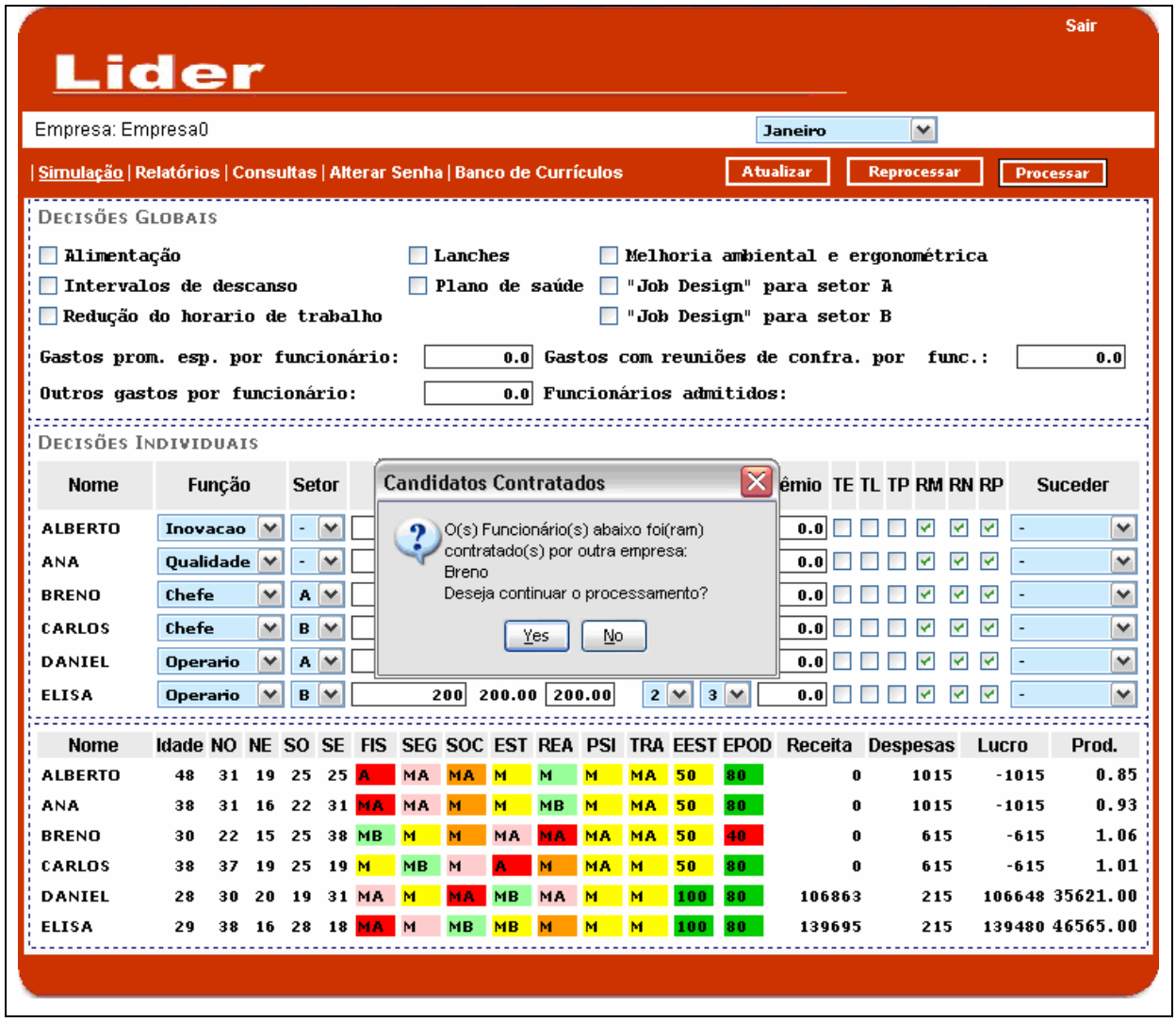


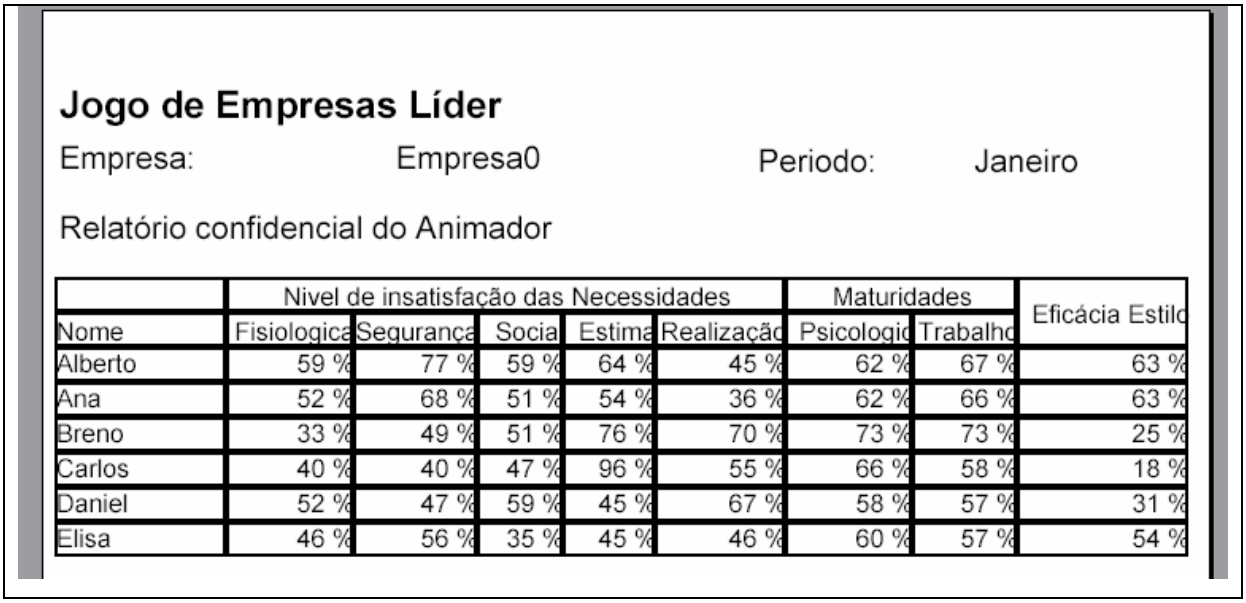
Figura 27 – Mensagem informando que houve funcionários contratados por outra empresa

3.4 RESULTADOS E DISCUSSÃO

Ao término do desenvolvimento, não foi possível aplicar esta nova versão do jogo de empresas LÍDER em ambiente real, porém alguns testes foram feitos em laboratório fazendo a comparação entre a versão anterior e a gerada por este projeto.

Foram montados dois ambientes idênticos, um utilizando a versão do jogo de empresas LÍDER desenvolvida neste projeto e outro na versão anterior de sistema. Estes ambientes são compostos de um jogo com duas empresas e as decisões tomadas foram idênticas, nas duas versões do sistema. A seguir serão apresentados os resultados obtidos por uma determinada empresa.

A figura 28 ilustra o relatório do perfil dos funcionários na versão anterior do sistema, após o processamento do primeiro período do jogo, enquanto a figura 29 ilustra o mesmo relatório, mas da nova versão.



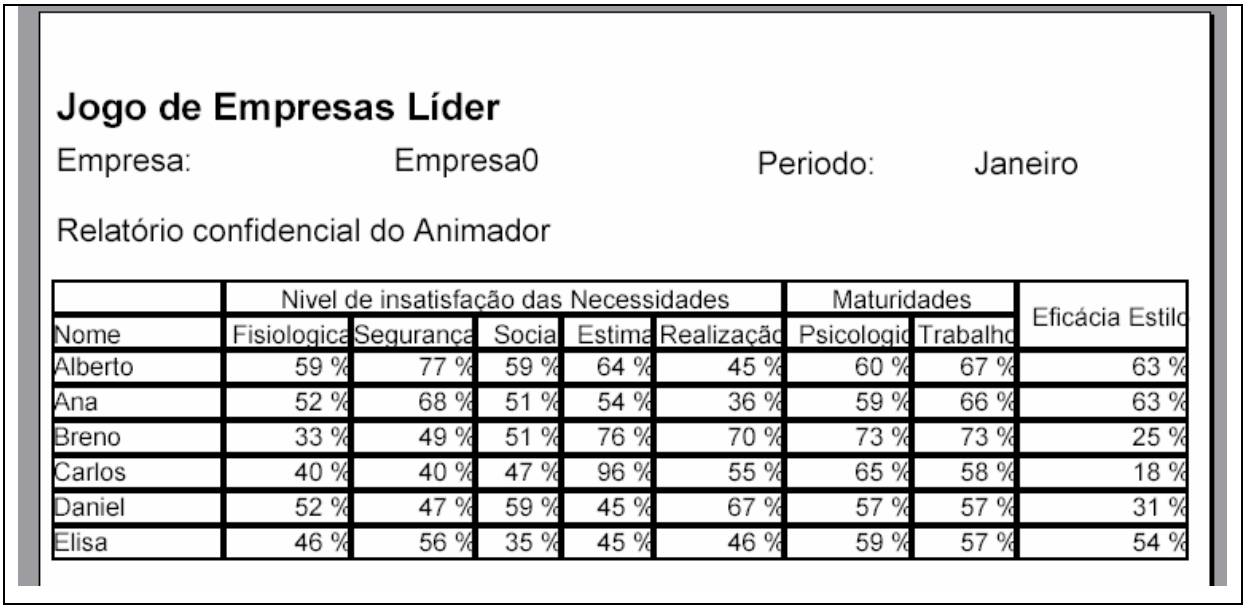
Jogo de Empresas Líder

Empresa: Empresa0 Periodo: Janeiro

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	59 %	77 %	59 %	64 %	45 %	62 %	67 %	63 %
Ana	52 %	68 %	51 %	54 %	36 %	62 %	66 %	63 %
Breno	33 %	49 %	51 %	76 %	70 %	73 %	73 %	25 %
Carlos	40 %	40 %	47 %	96 %	55 %	66 %	58 %	18 %
Daniel	52 %	47 %	59 %	45 %	67 %	58 %	57 %	31 %
Elisa	46 %	56 %	35 %	45 %	46 %	60 %	57 %	54 %

Figura 28 – Relatório de perfil com porcentagens de janeiro da versão anterior do Jogo de Empresas Líder



Jogo de Empresas Líder

Empresa: Empresa0 Periodo: Janeiro

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	59 %	77 %	59 %	64 %	45 %	60 %	67 %	63 %
Ana	52 %	68 %	51 %	54 %	36 %	59 %	66 %	63 %
Breno	33 %	49 %	51 %	76 %	70 %	73 %	73 %	25 %
Carlos	40 %	40 %	47 %	96 %	55 %	65 %	58 %	18 %
Daniel	52 %	47 %	59 %	45 %	67 %	57 %	57 %	31 %
Elisa	46 %	56 %	35 %	45 %	46 %	59 %	57 %	54 %

Figura 29 – Relatório de Perfil com porcentagens de janeiro da nova versão do Jogo de Empresas Líder

Na figura 30 pode-se observar o perfil dos colaboradores após o processamento do segundo período do jogo na versão anterior. Já a figura 31 mostra o mesmo relatório, na nova versão, com a diferença de que neste período a funcionária “Elisa” foi contratada por uma outra empresa integrante do jogo.

Jogo de Empresas Líder

Empresa: Empresa0 Periodo: Fevereiro

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	58 %	71 %	61 %	69 %	51 %	57 %	65 %	56 %
Ana	52 %	63 %	51 %	60 %	42 %	61 %	68 %	56 %
Breno	32 %	48 %	51 %	84 %	73 %	78 %	80 %	37 %
Carlos	40 %	40 %	48 %	108 %	57 %	67 %	66 %	30 %
Daniel	51 %	49 %	60 %	51 %	68 %	59 %	68 %	37 %
Elisa	44 %	59 %	35 %	48 %	50 %	60 %	67 %	50 %

Figura 30 – Relatório de perfil com porcentagens de fevereiro da versão anterior do Jogo de Empresas Líder

Jogo de Empresas Líder

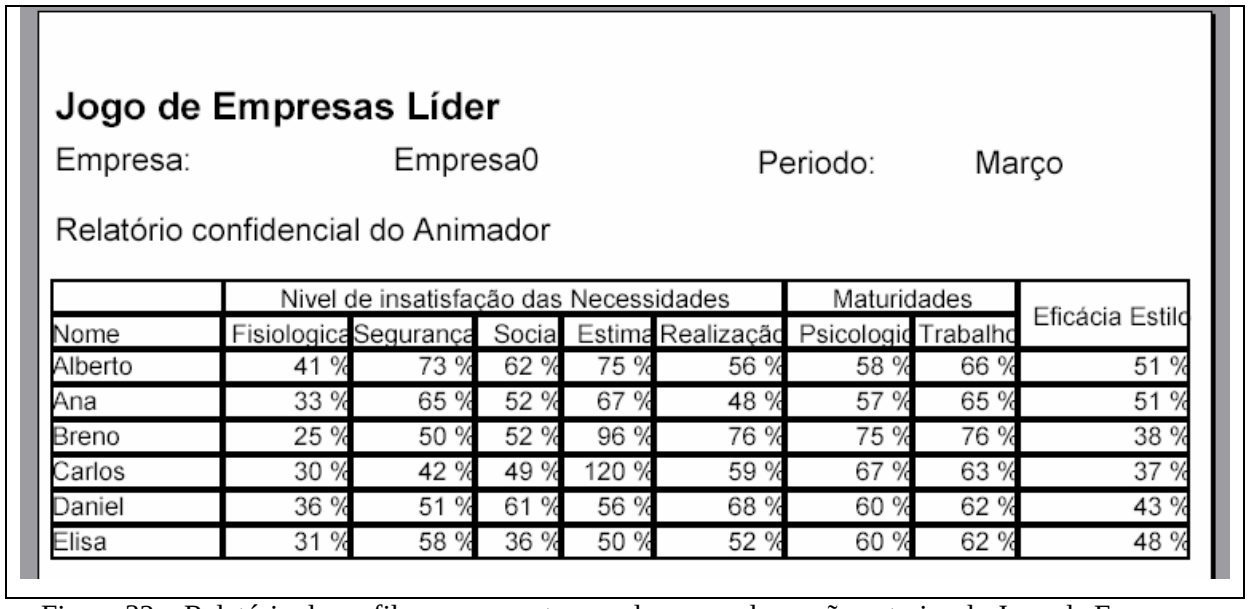
Empresa: Empresa0 Periodo: Fevereiro

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	58 %	71 %	61 %	69 %	51 %	56 %	64 %	56 %
Ana	52 %	63 %	51 %	60 %	42 %	60 %	68 %	56 %
Breno	32 %	48 %	51 %	84 %	73 %	77 %	79 %	37 %
Carlos	40 %	40 %	48 %	108 %	57 %	66 %	65 %	30 %
Daniel	51 %	50 %	60 %	51 %	68 %	59 %	67 %	37 %
Elisa	46 %	56 %	35 %	45 %	46 %	59 %	57 %	54 %

Figura 31 – Relatório de Perfil com porcentagens de fevereiro da nova versão do Jogo de Empresas Líder

As figuras 32 e 33 ilustram o mesmo relatório no terceiro período da antiga e da nova versão, respectivamente.



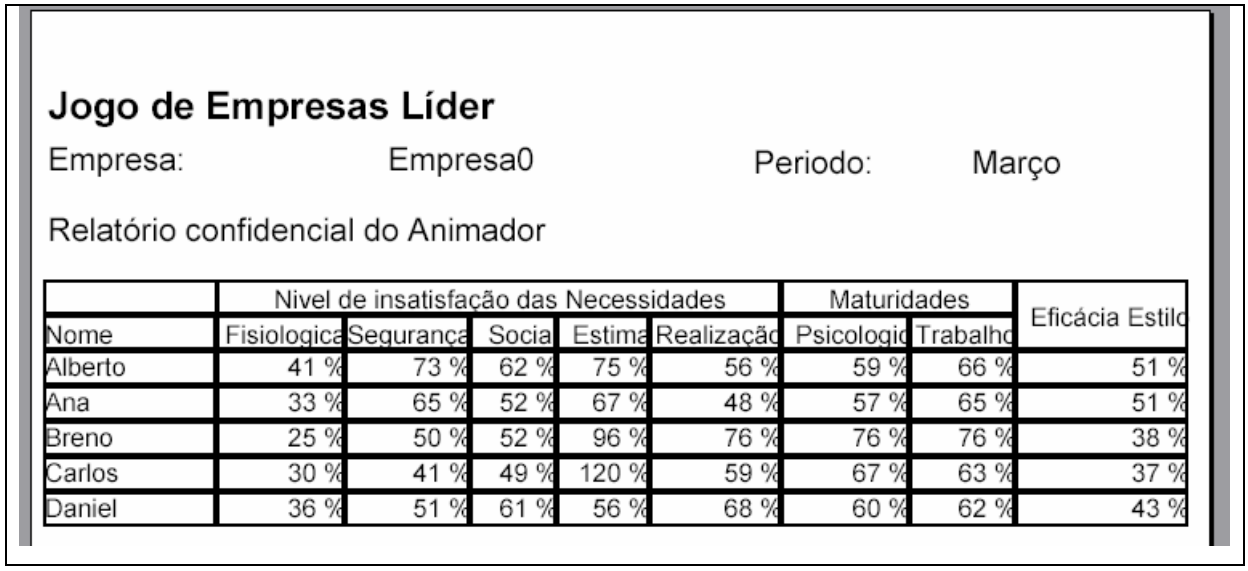
Jogo de Empresas Líder

Empresa: Empresa0 Periodo: Março

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	41 %	73 %	62 %	75 %	56 %	58 %	66 %	51 %
Ana	33 %	65 %	52 %	67 %	48 %	57 %	65 %	51 %
Breno	25 %	50 %	52 %	96 %	76 %	75 %	76 %	38 %
Carlos	30 %	42 %	49 %	120 %	59 %	67 %	63 %	37 %
Daniel	36 %	51 %	61 %	56 %	68 %	60 %	62 %	43 %
Elisa	31 %	58 %	36 %	50 %	52 %	60 %	62 %	48 %

Figura 32 – Relatório de perfil com porcentagens de março da versão anterior do Jogo de Empresas Líder



Jogo de Empresas Líder

Empresa: Empresa0 Periodo: Março

Relatório confidencial do Animador

	Nivel de insatisfação das Necessidades					Maturidades		Eficácia Estilo
Nome	Fisiologica	Segurança	Socia	Estima	Realização	Psicologic	Trabalho	
Alberto	41 %	73 %	62 %	75 %	56 %	59 %	66 %	51 %
Ana	33 %	65 %	52 %	67 %	48 %	57 %	65 %	51 %
Breno	25 %	50 %	52 %	96 %	76 %	76 %	76 %	38 %
Carlos	30 %	41 %	49 %	120 %	59 %	67 %	63 %	37 %
Daniel	36 %	51 %	61 %	56 %	68 %	60 %	62 %	43 %

Figura 33 – Relatório de Perfil com porcentagens de março da nova versão do Jogo de Empresas Líder

Desta forma nota-se que a demissão de um funcionário de uma determinada empresa não está afetando as necessidades dos outros colaboradores, em especial a necessidade de segurança, que deveria ter sido alterada. As poucas diferenças apresentadas são geradas por variáveis randômicas utilizadas para o processamento. Este fato demonstra a necessidade de revisão do modelo do jogo, atividade esta que não era objetivo deste trabalho. Estas alterações poderiam fazer com que, ao perder um funcionário, o modelo faça com que aumente a insegurança dos funcionários em função da maior rotatividade na empresa.

4 CONCLUSÕES

Este trabalho realizou implementações no jogo de empresas LÍDER, possibilitando que o jogo tenha uma maior interatividade entre as equipes participantes, gerando assim um ambiente competitivo e chegando cada vez mais próximo à realidade. Anteriormente o sistema não possibilitava que as decisões tomadas por uma empresa pudessem influenciar as outras, mas com essa nova versão e a centralização da base de candidatos esta situação tornou-se possível.

A base de desenvolvimento deste projeto foi todo o trabalho de Micheluzzi (2006), que fez a migração do software para ambiente Java.

A maior dificuldade no início do projeto foi a total compreensão da estrutura e lógica do sistema gerado por Micheluzzi (2006), mesmo este estando bem documentado, devido a seu tamanho e complexidade.

Por outro lado, as ferramentas utilizadas por Micheluzzi (2006) para desenvolvimento de seu trabalho, também facilitaram as implementações do atual projeto. Mesmo possuindo pouca experiência em desenvolvimento web, foi possível obter os resultados esperados, principalmente em relação à interface, que teve seu desenvolvimento simplificado devido às facilidades geradas pela IDE JSC para desenvolvimento de páginas JSF. Para a implementação do AJAX, foi utilizado o JBoss Ajax4JSF, que permitiu esta implementação sem que fosse necessário escrever sequer uma linha de código JavaScript.

Concluindo este trabalho, é possível dizer que os objetivos iniciais do projeto foram alcançados com sucesso. Foi desenvolvido um banco de currículos centralizado, onde todas as empresas do jogo têm acesso às mesmas informações, o jogador consegue avaliar o perfil do candidato e fazer uma proposta salarial, mesmo que este já esteja vinculado a uma outra empresa, criando assim uma competição direta entre as empresas pelos recursos humanos, visando dar mais realismo ao jogo.

Este trabalho ajuda a evolução de uma importante ferramenta de treinamento, utilizada pela FURB em cursos de graduação, deixando o jogo mais competitivo e dinâmico.

4.1 EXTENSÕES

Algumas sugestões de trabalhos futuros podem ser feitas, tendo como base o trabalho atual, como:

a) melhorar a interface do jogo com um estudo sobre ergonomia, ou analisando as sugestões dos usuários no momento em que o jogo for aplicado em alguma turma.

b) aprimorar o processo decisório do funcionário para saber se ele deve ou não mudar de empresa, pois atualmente ele está baseado apenas no salário proposto e na eficácia do estilo do chefe.

c) permitir à empresa na qual o funcionário trabalha a possibilidade de fazer uma contraproposta ao funcionário requisitado por outra empresa.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Walnice. **Captação e seleção de talentos:** repensando a teoria e a prática. São Paulo: Atlas, 2004.

BELLIA, Renato. Uma aplicação Java EE completa: Requisitos, modelo de classes e persistência com JPA. **Java Magazine**, São Paulo, n. 44, p. 26-36, mar. 2007.

BELLIA, Renato; SENGER, Yara. Uma aplicação Java EE completa: Implementando a interface web com JSP. **Java Magazine**, São Paulo, n. 45, p. 30-39, abr. 2007.

BODOFF, Stephanie et al. **Tutorial do J2EE**. Tradução Altair Dias Caldas de Moraes, Carlos Augusto Caldas de Moraes. Rio de Janeiro: Campus, 2002.

BOND, Martin et al. **Aprenda J2EE em 21 dias**: com EJB, JSP, servlets, JNDI, JDBC e XML. Tradução João Eduardo Nóbrega Tortello. São Paulo: Pearson Education, 2003.

GRAMIGNA, Maria Rita Miranda. **Jogos de Empresas**. São Paulo: McGraw-Hill, 1993.

HIBERNATE. **Relational persistence for Java and .NET**. [S.l.], [2007]. Disponível em: <http://www.hibernate.org>**.** Acesso em: 14 jul. 2007.

JBOSS AJAX4JSF. **JBoss Ajax4jsf**: open source framework. [S.l.], [2007]. Disponível em: <https://labs.jboss.com/jbossajax4jsf/>**.** Acesso em: 3 jun. 2007.

KURNIAWAN, Budi. **Programando em JavaServer Faces.** Tradução Cláudio Rodrigues Pistilli. Rio de Janeiro: Editora Ciência Moderna Ltda., 2004.

LOPES, Maurício C. **Jogo de empresas Líder**: aperfeiçoamento do modelo e do sistema. 1994. 86 f. Dissertação (Mestrado em Engenharia de Produção) - Departamento de Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

LOPES, Maurício Capobianco; WILHELM, Pedro Paulo Hugo. **Simulador LIDER – 8.0:** manual de informações e orientações. FURB / UFSC, 2006.

LOPES, Maurício Capobianco; RIBEIRO, Fábia Marilia; WILHELM, Pedro Paulo Hugo. Sistema computacional de treinamento em gestão de recursos humanos. In SEMINCO. 1998. Blumenau. **Anais...** Blumenau: FURB, 1998.

MICHELUZZI, Diogo. **Migração do jogo de empresas Líder da plataforma Delphi para Java utilizando o *framework* J2EE JavaServer Faces e Ajax.** 2006. 158 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

NIVEIROS, Sofía Inês. **Estudo e aperfeiçoamento do modelo das maturidades dos funcionários no jogo de empresas Líder.** 1998. 113f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

PONTES, Benedito Rodrigues. **Planejamento, recrutamento e seleção de pessoal.** 4. ed. São Paulo: LTr, 2004.

ROCHA, André D.; KUBOTA, Sérgio O. Persistência no Java EE 5: iniciando com a Java Persistence API. **Java Magazine**, São Paulo, n. 39, p. 28-37, set. 2006.

STEINBACH, José Acácio. **Sistema especialista para auxílio no diagnóstico da hierarquia de necessidades do simulador de empresas Líder**. 2002. 48f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

VALDAMERI, Alexander Roberto. **Novas perspectivas para o jogo de empresas Líder.** 2001. 93f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

WILHELM, Pedro Paulo Hugo. **Uma nova perspectiva de aproveitamento e uso dos jogos de empresas**. 1997. 136f. Tese (Doutorado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

ZUKOWSKI, John. **Introducing annotations and tech tips quis**. [S.l.], 2006. Disponível em: <http://java.sun.com/mailers/techtips/corejava/2006/tt0620.html>. Acesso em: 16 out. 2006.

APÊNDICE A – Relação das tabelas do jogo de empresas Líder.

Os quadros 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 e 28 apresentam a relação das tabelas do jogo de empresas Líder.

Tabela JOGO	
Coluna	**Tipo**
CODIGO_EMPRESA	INTEGER (Foreign Key)
NMJogo	VARCHAR(50)
ID_JOGADOR	INTEGER (Primary Key)
SENHA	VARCHAR(10)
PERFIL	VARCHAR(20)

Quadro 16 – Tabela JOGO da nova versão do sistema Líder

Tabela EMPRESA	
Coluna	**Tipo**
CODIGO_EMPRESA	INTEGER (*Primary Key*)
NOME_EMPRESA	VARCHAR(50)
INTEGRADO_VIRTUAL	BOOLEAN

Quadro 17 – Tabela EMPRESA da nova versão do sistema Líder

Tabela EMPRESA_PERIODO	
Coluna	**Tipo**
CODIGO_EMPRESA	INTEGER (*Primary Key*)
PERIODO	INTEGER
NUMERO_EMPREGADOS	INTEGER
NUMERO_ASSESSORES	INTEGER
NUMERO_DEMITIDOS	INTEGER
NUMERO_PROMOCOES	INTEGER
NUMERO_CONTRATADOS	INTEGER
RECEITA	DOUBLE
DESPESAS	DOUBLE
DESPESAS_DEMISSAO	DOUBLE
REBAIXAMENTO_POSICOES	INTEGER
REBAIXAMENTO_SALARIOS	INTEGER
USO_CORRETO_DELEGAR	DOUBLE
MAIOR_CAP_PRODUTIVA	DOUBLE
PROCESSADO	BOOLEAN
ALEATORIO_REALIZACAO	FLOAT
ALEATORIO_EXTERNO	FLOAT
ALEATORIO_SUCESSO	FLOAT

Quadro 18 – Tabela EMPRESA_PERIODO da nova versão do sistema Líder

Tabela **DECISAO_GLOBAL**	
Coluna	**Tipo**
ID_DECISAO_GLOBAL	INTEGER (*Primary Key*)
CD_EMPRESA_PERIODO	INTEGER (*Foreign Key*)
ALIMENTACAO	BOOLEAN
ERGONOMETRIA	BOOLEAN
GASTO_REUNIOES	DOUBLE
GASTO_JOGOS	DOUBLE
OUTROS_GASTOS	DOUBLE
JOB_DESIGN_MONTAGEM	BOOLEAN
JOB_DESIGN_FABRICACAO	BOOLEAN
LANCHES	BOOLEAN
PLANO_DE_SAUDE	BOOLEAN
REDUCAO_HORARIO	BOOLEAN
INTERVALO_DESCANSO	BOOLEAN

Quadro 19 – Tabela DECISAO_GLOBAL da nova versão do sistema Líder

Tabela **FUNCIONARIO**	
Coluna	**Tipo**
CODIGO_FUNCIONARIO	INTEGER (*Primary Key*)
NOME	VARCHAR(50)
IDADE	INTEGER
CODIGO_EMPRESA	INTEGER (*Foreign Key*)
NO	DOUBLE
NE	DOUBLE
SO	DOUBLE
SE	DOUBLE
DATA_ADMISAO	DATE
CODIGO_CANDIDATO	INTEGER

Quadro 20 – Tabela FUNCIONARIO da nova versão do sistema Líder

Tabela FUNCIONARIO_PERIODO	
Coluna	**Tipo**
ID_FUNCIONARIO_PERIODO	INTEGER (*Primary Key*)
CODIGO_FUNCIONARIO	INTEGER (*Foreign Key*)
PERIODO	INTEGER
CARGO	VARCHAR(50)
SETOR	VARCHAR(50)
EXPERIENCIA	DOUBLE
COMPORTAMENTO	DOUBLE
SALARIO	DOUBLE
PRODUCAO_REAL	DOUBLE
CAPACIDADE_PRODUTIVA	DOUBLE
CAP_PRODUTIVA_INICIAL	DOUBLE
RECEITA	DOUBLE
DESPESA	DOUBLE
NUMERO_TREINAMENTOS_ESPEC	DOUBLE
NUMERO_TREINAMENTOS_PROMO	DOUBLE
NUMERO_TREINAMENTOS_LIDER	DOUBLE
EFICACIA_LIDERANCA	DOUBLE
EFICACIA_ESTILO	DOUBLE
EFICACIA_PODER	DOUBLE
NUMERO_PERIODOS_SUCESSAO	DOUBLE
HIERARQUIA	INTEGER
ESTADO	VARCHAR(1)

Quadro 21 – Tabela FUNCIONARIO_PERIODO da nova versão do sistema Líder

Tabela FUNCIONARIO_PERTURBACAO	
Coluna	**Tipo**
ID_FUNCIONARIO_PERTURBACAO	INTEGER (*Primary Key*)
CD_FUNCIONARIO_PERIODO	INTEGER (*Foreign Key*)
SUPERVALORIZACAO_SEG	DOUBLE
COMP_DEST_ESTIMA	DOUBLE
PARTICIPA_GRUPOS	DOUBLE
DESGASTE_ESTILO	DOUBLE
DESGASTE_PODER	DOUBLE
VEXTERNO	DOUBLE
VMOTIVA	DOUBLE
VFACILIDADE	DOUBLE
VLIDERANCA	DOUBLE

Quadro 22 – Tabela FUNCIONARIO_PERTURBACAO da nova versão do sistema Líder

Tabela **FUNCIONARIO_NECESSIDADE**	
Coluna	**Tipo**
ID_FUNCIONARIO_NECESSIDADE	INTEGER (*Primary Key*)
CD_FUNCIONARIO_PERIODO	INTEGER (*Foreign Key*)
FISIOLOGICA	DOUBLE
SEGURANCA	DOUBLE
SOCIAL	DOUBLE
STATUS	DOUBLE
ESTIMA	DOUBLE
REALIZACAO	DOUBLE
FISIOLOGICA_PRIO	DOUBLE
SEGURANCA_PRIO	DOUBLE
SOCIAL_PRIO	DOUBLE
ESTIMA_PRIO	DOUBLE
REALIZACAO_PRIO	DOUBLE
FISIOLOGICA_INCR	DOUBLE
SEGURANCA_INCR	DOUBLE
SOCIAL_INCR	DOUBLE
ESTIMA_INCR	DOUBLE
REALIZACAO_INCR	DOUBLE
FISIOLOGICA_PURA	DOUBLE
SEGURANCA_PURA	DOUBLE
SOCIAL_PURA	DOUBLE
ESTIMA_PURA	DOUBLE
REALIZACAO_PURA	DOUBLE
FISIOLOGICA_INICIAL	DOUBLE
SEGURANCA_INICIAL	DOUBLE
SOCIAL_INICIAL	DOUBLE
ESTIMA_INICIAL	DOUBLE
REALIZACAO_INICIAL	DOUBLE
FISIOLOGICA_EFET	DOUBLE
SEGURANCA_EFET	DOUBLE
ESTIMA_EFET	DOUBLE
REALIZACAO_EFET	DOUBLE
PREMIO_ESTIMA	DOUBLE

Quadro 23 – Tabela FUNCIONARIO_NECESSIDADE da nova versão do sistema Líder

Tabela **FUNCIONARIO_MATURIDADE**	
Coluna	**Tipo**
ID_FUNCIONARIO_MATURIDADE	INTEGER (*Primary Key*)
CD_FUNCIONARIO_PERIODO	INTEGER (*Foreign Key*)
PSICOLOGICA	DOUBLE
TRABALHO	DOUBLE
COMPR_CARGO	VARCHAR(15)
CAP_SOL_PROBLEMAS	VARCHAR(15)
CAP_AS_RESPONSA	VARCHAR(15)
CUMP_PRAZOS	VARCHAR(15)
PROPRIO_TRAB	VARCHAR(15)
DESEJO_RESPONSA	VARCHAR(15)
MOTIVO_REALIZACAO	VARCHAR(15)
COMPROM_TRAB	VARCHAR(15)
CONHEC_TRAB	VARCHAR(15)
PERSISTENCIA	VARCHAR(15)
ATITUDE_TRABALHO	VARCHAR(15)
INICIATIVA	VARCHAR(15)
INDEPENDENCIA	VARCHAR(15)
TRABALHO_INICIAL	DOUBLE
PSICOLOGICA_INICIAL	DOUBLE

Quadro 24 – Tabela FUNCIONARIO_MATURIDADE da nova versão do sistema Líder

Tabela **DECISAO_INDIVIDUAL**	
Coluna	**Tipo**
ID_DECISAO_INDIVIDUAL	INTEGER (*Primary Key*)
CD_FUNCIONARIO_PERIODO	INTEGER (*Foreign Key*)
NOVO_CARGO	VARCHAR(50)
NOVO_SETOR	VARCHAR(50)
AUMENTO_SALARIO	DOUBLE
META_DE_PRODUCAO	DOUBLE
ESTILO_APLICADO	DOUBLE
PODER_APLICADO	DOUBLE
PREMIO	DOUBLE
TREINAMENTO_LIDERANCA	BOOLEAN
RELATORIO_NECESSIDADES	BOOLEAN
TREINAMENTO_PROMOCAO	BOOLEAN
TREINAMENTO_ESPECIFICO	BOOLEAN
RELATORIO_MATURIDADE	BOOLEAN
RELATORIO_PERTURBADORAS	BOOLEAN
SUCEDER	VARCHAR(10)
HIERARQUIA	INTEGER

Quadro 25 – Tabela DECISAO_INDIVIDUAL da nova versão do sistema Líder

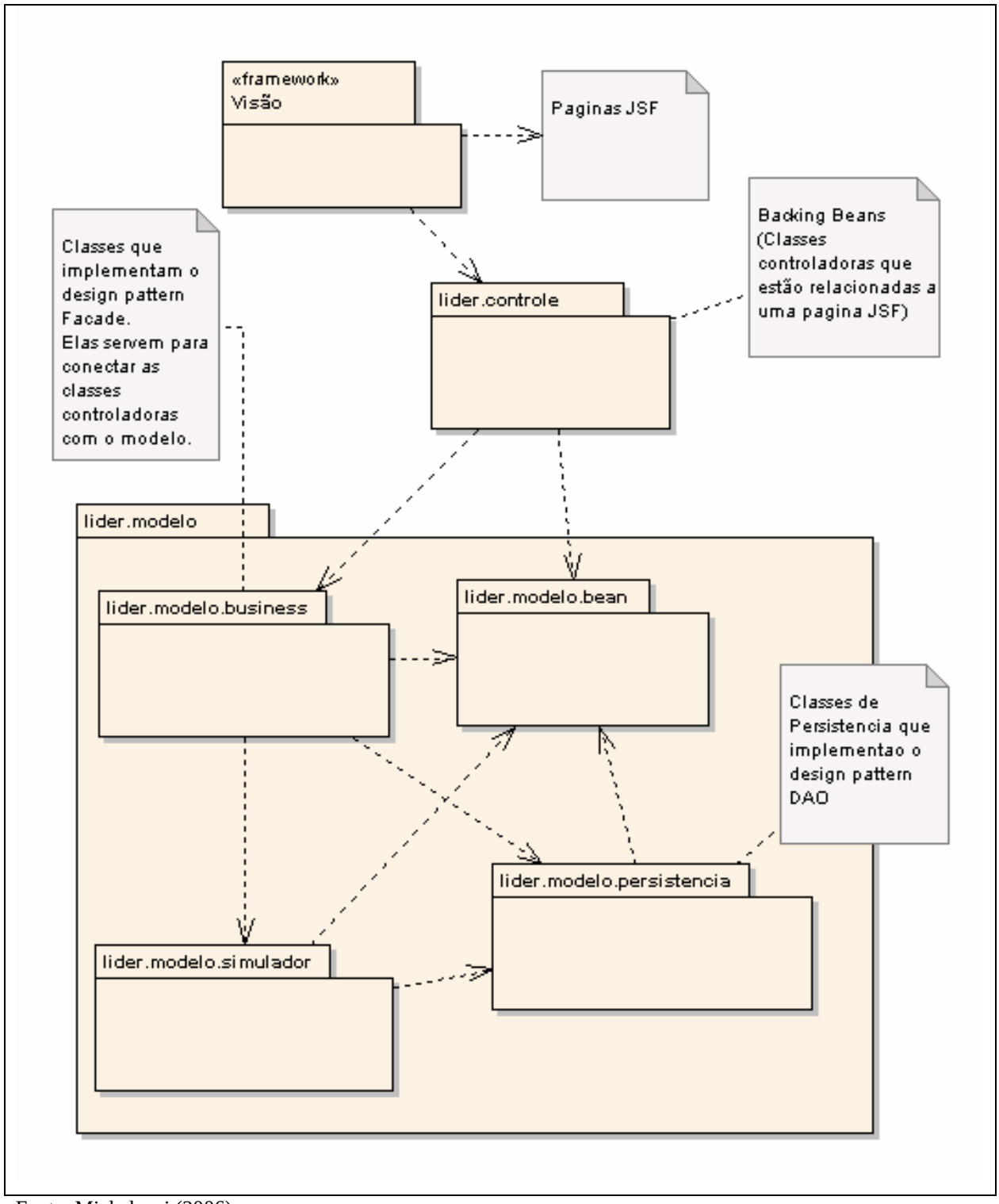
Tabela **CANDIDATO**	
Coluna	**Tipo**
CODIGO_CANDIDATO	INTEGER (*Primary Key*)
NOME	VARCHAR(25)
IDADE	INTEGER
COMPORTAMENTO	DOUBLE
CARGO	VARCHAR(25)
SETOR	VARCHAR(25)
SALARIO	DOUBLE
PRODUCAO_REAL	DOUBLE
CAPACIDADE_PRODUTIVA	DOUBLE
FISIOLOGICA	DOUBLE
SEGURANCA	DOUBLE
SOCIAL	DOUBLE
ESTIMA	DOUBLE
REALIZACAO	DOUBLE
PSICOLOGICA	DOUBLE
TRABALHO	DOUBLE
STATUS	DOUBLE
NE	DOUBLE
NO	DOUBLE
SE	DOUBLE
SO	DOUBLE
EXPERIENCIA	DOUBLE
COMPR_CARGO	VARCHAR(15)
CAP_SOL_PROBLEMAS	VARCHAR(15)
CAP_AS_RESPONSA	VARCHAR(15)
CUMP_PRAZOS	VARCHAR(15)
PROPRIO_TRAB	VARCHAR(15)
CONHEC_TRAB	VARCHAR(15)
DESEJO_RESPONSA	VARCHAR(15)
MOTIVO_REALIZACAO	VARCHAR(15)
COMPROM_TRAB	VARCHAR(15)
PERSISTENCIA	VARCHAR(15)
ATITUDE_TRABALHO	VARCHAR(15)
INICIATIVA	VARCHAR(15)
INDEPENDENCIA	VARCHAR(15)
SUPERVALORIZACAO_SEG	DOUBLE
COMP_DEST_ESTIMA	DOUBLE
PARTICIPA_GRUPOS	DOUBLE
DESGASTE_ESTILO	DOUBLE
DESGASTE_PODER	DOUBLE
VEXTERNO	DOUBLE
VMOTIVA	DOUBLE
VFACILIDADE	DOUBLE
VLIDADERANCA	DOUBLE
CD_EMPRESA	INTEGER

Quadro 26 – Tabela CANDIDATO da nova versão do sistema Líder

Tabela CONTRATACAO	
Coluna	**Tipo**
ID_CONTRATACAO	INTEGER (*Primary Key*)
EMPRESA_ANTERIOR	INTEGER
EMPRESA_NOVA	INTEGER
CODIGO_CANDIDATO	INTEGER
PERIODO_CONTRATACAO	INTEGER
SALARIO_NOVO	DOUBLE

Quadro 27 – Tabela CONTRATACAO da nova versão do sistema Líder

ANEXO A – Diagrama de Pacotes.



Fonte: Micheluzzi (2006).

Figura 34 – Diagrama de pacotes do Jogo de Empresas Líder

Pacote	Descrição
Visão	Este pacote possui todas as páginas JSF do sistema. Ele é responsável em prover a camada de visão do sistema que se utiliza do *framework* JSF para fazer o controle da mesma.
lider.controle	Este pacote engloba todas as classes de controles de interface, ou seja, as classes *backing beans* que estão ligadas a uma página JSF da camada de visão. Este pacote é responsável em disponibilizar a camada de controle do sistema.
lider.modelo	Este pacote é composto por vários outros pacotes que formam a camada de modelo do sistema Líder.
lider.modelo.business	Este pacote contém todas as classes que irão fazer a conexão das classes controladoras com as classes de modelo. As classes deste pacote implementam o padrão Façade para diminuir a complexidade de acesso à camada de modelo assim como o acoplamento entre as camadas de controle e modelo.
lider.modelo.bean	Este pacote possui todas as classes *beans* utilizadas para prover o mapeamento objeto-relacional. Cada classe deste pacote corresponde a uma tabela do banco de dados e serão utilizadas pelo *framework* de persistência que no caso deste projeto é o Hibernate.
lider.modelo.persistencia	Neste pacote se concentra todas as classes que fazem o acesso ao banco de dados. As classes deste pacote foram desenvolvidas utilizando o padrão DAO e estão programadas para fazer a persistência dos dados utilizando o *framework* Hibernate.
lider.modelo.simulador	Este pacote é composto pelas principais classes de negócio do sistema, ou seja, as classes responsáveis por fazer o cálculo da simulação e do processamento de um jogo.

Fonte: Adaptado de Micheluzzi (2006).

Quadro 28 – Descrição dos pacotes do sistema