UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

SISTEMA DE *HELP DESK* E CONTROLE DE CHAMADOS BASEADO EM *WORKFLOW*

CRISTIAN PAULO PRIGOL

BLUMENAU
2007

2007/1-08

[w1] Comentário: INFORMAR O SEU NÚMERO DE SEQÜÊNCIA DO RELATÓRIO QUINZENAL

CRISTIAN PAULO PRIGOL

SISTEMA DE *HELP DESK* E CONTROLE DE CHAMADOS BASEADO EM *WORKFLOW*

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. Marcel Hugo, M. Eng. - Orientador

BLUMENAU
2007

2007/1-08

[w2] Comentário: INFORMAR O SEU NÚMERO DE SEQÜÊNCIA DO RELATÓRIO QUINZENAL

SISTEMA DE *HELP DESK* E CONTROLE DE CHAMADOS

BASEADO EM *WORKFLOW*

Por

CRISTIAN PAULO PRIGOL

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

	_____
Presidente:	Prof. Marcel Hugo, M. Eng – Orientador, FURB
	_____
Membro:	Prof. Everaldo Artur, Grahl, M. Eng – FURB
	_____
Membro:	Prof. Evaristo Baptista, M. Eng – FURB

Blumenau, 04 de julho de 2007

[w3] Comentário: DATA DA APRESENTAÇÃO

Dedico este trabalho a minha família, que sempre me apoiou e ajudou-me para que eu conseguisse concluir o curso de Sistemas de Informação.

AGRADECIMENTOS

À minha família, que mesmo longe, sempre esteve presente me apoiando.

À minha namorada, por ter me compreendido nos momentos que eu estava estressado, ou sem tempo pra ela. Sem o seu apoio eu não teria concluído este trabalho.

Ao meu orientador, Marcel Hugo, por ter acreditado, e me ajudado tanto para a conclusão deste trabalho.

O homem que não lê bons livros não tem nenhuma vantagem sobre o homem que não sabe ler. (Mark Twain)

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de *Help Desk* aplicado a Seção de Apoio ao Usuário da FURB. A proposta deste sistema foi baseado na metodologia de controle de processos workflow utilizando como linguagem o Java padrão Java 2 Enterprise Edition (J2EE). A sua implementação utilizou o framework de workflow Java Business Process Management (JBPM), descrevendo sua utilização e seus principais conceitos. O trabalho demonstra também o uso do framework Hibernate, apresentando algumas de suas definições e a sua utilização para fazer persistência dos dados no banco de dados Oracle.

Palavras-chave: *workflow* , JBPM. *Help Desk.*

ABSTRACT

This paper presents the development of a *Help Desk*'s system applied to Seção de Apoio ao Usuário of FURB. The proposal of this system was based on the methodology of process´ control workflow using as language the Java pattern Java 2 Enterprise Edition (J2EE). Its implementation used the framework of workflow Java Business Process Management (JBPM), describing its utilization and its main concepts. This paper also demonstrates the use of framework Hibernate, presenting some of its definitions and its utilization to make persistence of data on the database Oracle.

Key-words: *workflow* , JBPM, *Help Desk*.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

"A cada dia, as organizações buscam melhorar seus procedimentos de negócio, visando maior qualidade dos seus produtos e satisfação de seu público-alvo" (ZSCHORNACK, 2003, p. 1). No ambiente atual das organizações uma das principais ferramentas de produção é o computador e, portanto é primordial que ele esteja sempre em perfeito funcionamento (CAVALARI; COSTA, 2005, p. 1). Diante disso, as empresas possuem equipes de suporte técnico para auxiliar seus usuários na resolução de problemas e manter o funcionamento de seus computadores.

Este ambiente pode demandar muitos controles no gerenciamento das solicitações existentes, nas etapas da resolução de problemas, no gerenciamento de equipamentos e na satisfação do usuário com acompanhamento dos resultados. Estes controles acabam gerando vários processos dentro da equipe de suporte.

Sendo assim, as empresas utilizam sistemas denominados *Help Desk.* Este tipo de software tem a função de auxiliar a equipe de suporte para coordenar e solucionar os incidentes que ocorrem com os usuários, assegurando que os chamados gerados por estes incidentes não sejam perdidos, esquecidos ou negligenciados. Este tipo de sistema constitui em um mecanismo computacional facilitador da informação (CAVALARI; COSTA, 2005, p. 1).

Conforme BC Associates (2006) os softwares de *Help Desk* podem ser de dois tipos: o primeiro serve apenas para registrar chamados, e escalá-los para que técnicos mais experimentes os resolvam. O segundo tipo fornece um elevado grau de informações para o atendente, auxiliando-o para que o incidente seja resolvido imediatamente.

O ambiente de *Help Desk* gera vários processos para a resolução dos incidentes que são reportados a ele. Gerenciar estes processos cada vez mais complexos torna-se um desafio hoje nas organizações.

Segundo Telecken (2004, p. 11), "a crescente necessidade de padronização, eficácia e gerenciamento de processos tem aumentado o interesse das mais diferentes organizações pela tecnologia de *workflow*". Um *workflow* pode ser definido como a automatização total ou parcial de um processo de negócio, durante a qual são passados documentos, informações ou tarefas de um participante para outro, de acordo com um conjunto de regras processuais (Hollingsworth, 1995, p. 06).

Conforme França (2004, p. 13), “sistemas de *workflow* podem ser utilizados em inúmeros tipos de organizações e, mais do que isso, podem ser utilizados no relacionamento entre organizações.”

O presente trabalho utiliza como base no seu desenvolvimento o *framework* de *workflow* chamado *Java Business Process Management* (JBPM). É uma ferramenta que permite a criação de processos de negócio para coordenar grupos de pessoas, aplicações ou serviços. O JBPM é uma ferramenta de código aberto proposta pela JBOSS. Ele foi criado a partir de padrões de *workflow* desenvolvidos em pesquisas acadêmicas na universidade de tecnologia de Eindhoven na Holanda (RED HAT MIDDLEWARE, 2007).

“A utilização de um sistema baseado em *workflow* em um *Help Desk* proporciona a definição detalhada de todas as atividades do processo, bem como de tarefas específicas de cada uma destas etapas” (CRUZ, 1998, p. 149).

A Seção de Apoio ao Usuário (APUS) da Fundação Universidade Regional de Blumenau, (FURB) como equipe de suporte da universidade está inserida neste contexto. Ela contém vários processos e estes não estão mapeados de uma maneira formal, e na maioria dos casos não são conduzidos dentro de alguma metodologia de gerenciamento de processos. O sistema utilizado neste ambiente de trabalho não proporciona o acompanhamento dos processos do setor. Por isto este trabalho de conclusão de curso visa desenvolver um sistema de *Help Desk* baseado em *workflow* que auxilie o APUS no gerenciamento da sua equipe de suporte e dos processos inerentes a esta equipe.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um sistema de *Help Desk* para o APUS da FURB utilizando a tecnologia de *workflow* .

Os objetivos específicos deste trabalho são:

a) possibilitar que o sistema gerencie o andamento dos chamados;
b) disponibilizar ao gestor de tecnologia da informação (TI) um acompanhamento detalhado dos chamados e do desempenho de seus técnicos;
c) desenvolver um sistema de *Help Desk* utilizando o *framework* JBPM.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos.

A seguir será descrito brevemente cada capítulo do referido trabalho.

No primeiro capítulo tem-se a introdução, a justificativa do trabalho, o objetivo geral e objetivos específicos.

No segundo capítulo apresenta-se a fundamentação teórica das áreas envolvidas. São elas: DTI, *Help Desk*, *wokflow*, o *framework* JBPM, o *framework* Hibernate. Também são apresentados os trabalhos correlatos.

No terceiro capitulo é apresentado o desenvolvimento do trabalho, comentando as técnicas e as ferramentas envolvidas na diagramação, desenvolvimento do sistema e banco de dados utilizado, como foi implementado o sistema, enfim, tudo o que diz respeito ao desenvolvimento do sistema está explicado nesse capítulo.

Por fim, no quarto capítulo são apresentadas as principais conclusões e sugestões para futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo aborda conceitos e técnicas necessárias para o desenvolvimento do trabalho.

2.1 A DIVISÃO DE TECNOLOGIA DA INFORMAÇÃO

A Divisão de Tecnologia da Informação (DTI) da Universidade Regional de Blumenau é o setor responsável pela gestão de TI da universidade. Ela está subdividida em três seções: Seção de Desenvolvimento de Sistemas, Seção de Suporte Técnico e Seção de Apoio ao Usuário. Esta última é responsável pela manutenção de computadores e *softwares* da administração da universidade. Conta atualmente com 15 funcionários para efetuar os atendimentos, planejar e executar projetos de microinformática. Esta seção conta com um atendimento de *Help Desk*, através de telefone e o atendimento presencial ou remoto para resolver problemas em computadores.

2.2 *HELP DESK*

Atualmente o gerenciamento do setor de TI das empresas vem tornando-se complexo, sendo um desafio para seus gestores.

> Os clientes hoje, particularmente com o surgimento da internet, esperam e exigem um serviço de qualidade e dentro do prazo. As empresas que estão olhando para o futuro, estão investindo em operações de serviços ao cliente, para atingirem uma vantagem competitiva (Unipress, 2001, p. 4).

Para melhorar o nível do serviço de atendimento e manter a qualidade e a eficiência do mesmo, torna-se obrigatório uso de um sistema de *Help Desk*. Este tipo de sistema está no centro das atividades de TI. É a primeira linha de defesa para resolver os problemas relacionados a seus clientes. Em muitas empresas ele vem sendo utilizado para resolver as questões comerciais, solicitações de clientes, gerenciar produtos e liberar serviços. Segundo Coêlho (2003, p.2), sistemas de *Help Desk* compõem cada vez mais o crescente mercado de

tecnologias computacionais, assim como *Data Warehouse*, *Groupware*, *Data Mining* etc. Conforme Silva (2004, P.12) “uma ferramenta de *Help Desk*, sem dúvida proporciona um diferencial que além da documentação e sistematização oferece possibilidade de análise dos processos e fluxo de informações”.

Conforme Unipress (2001, p.4) os desafios de um sistema de *Help Desk* corporativo atualizado devem incluir:

a) o volume de chamados, que vem aumentando devido a complexidade e expansão dos negócios;
b) os métodos de comunicação: telefone, fax, e-mail e computação móvel;
c) as iniciativas da empresa para fazer o *Help Desk* uma parte integral das estratégias do *eBusiness.* A demanda e o volume de trabalho das equipes de suporte estão aumentando, fazendo com que ela se torne um componente crítico.

“Para ser capaz de fornecer os serviços de TI de acordo com a funcionalidade, níveis de serviço e custo acordados, a forma de trabalho orientada a processos é a melhor forma de estruturação da área de TI” (MAGALHÃES; PINHEIRO, 2007, p.23).

2.3 *WORKFLOW*

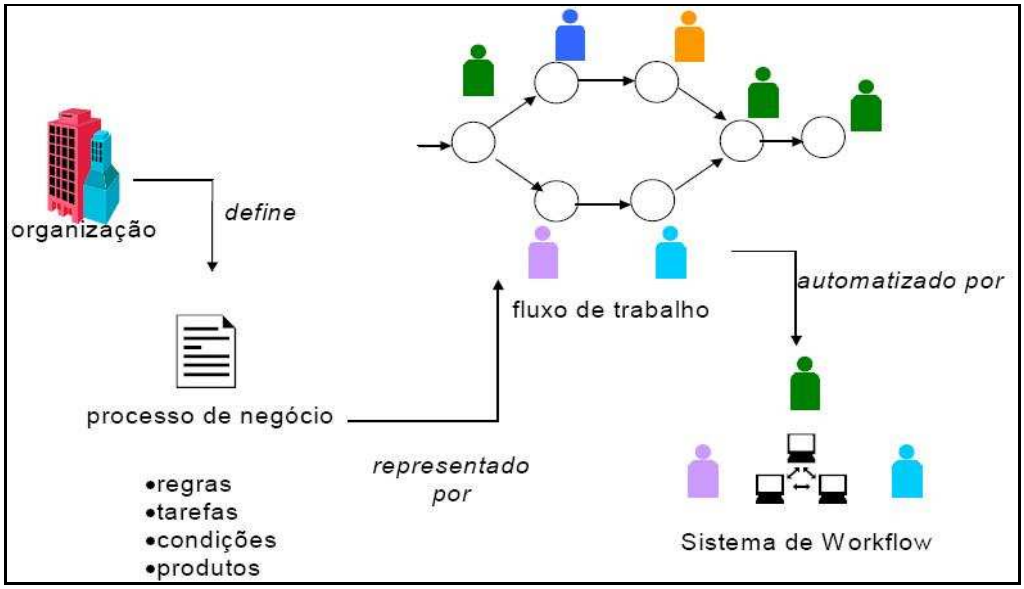
“Um *workflow* pode ser definido como a automatização total ou parcial de um processo de negócio, durante a qual são passados documentos, informações ou tarefas de um participante para outro, de acordo com um conjunto de regras processuais” (Hollingsworth, 1995, p. 06). A definição de Cruz (1998, p.10) define que “*workflow* é um conjunto de ferramentas que possibilita a análise pró-ativa, compreensão e automação de atividades e tarefas baseadas em informação”. Outra definição dada por Villas (2002, apud GOETZ, 2003, p. 11) diz que: “*workflow* é o processo pelo qual tarefas individuais convergem para completar uma transação – um processo de negócio bem definido – dentro de uma empresa”.

Sistemas de *workflow* têm suas origens a partir de pesquisas em automação de escritórios da década de 70. O principal foco destas pesquisas era o de oferecer soluções sobre como gerar, armazenar, compartilhar e rotear documentos nas organizações visando à diminuição da manipulação física de documentos de papel (ARAUJO; BORGES, 2001, p.3).

Hoje em dia, os sistemas de *workflow* não se limitam a rotear documentos, eles a partir da década de 80 se tornaram sistemas para a coordenação de trabalhos em equipes.

Conforme Villas (2002, apud GOETZ, 2003, p. 11) o principal objetivo do software de *workflow* é aumentar a eficiência de processos de negócio e a efetividade das pessoas que trabalham em conjunto para executá-los. Ainda segundo CRUZ (2000, apud ARAUJO; BORGES, 2001, p.5) "um processo de negócio é um procedimento onde documentos, informações e tarefas são passadas entre participantes de um acordo com um conjunto de regras definidas a serem alcançadas".

Representa-se processo de negócio através de fluxos de trabalho, ou seja, representá-lo através de um diagrama, envolvendo documentos, atividades a serem executadas, a ordem delas e as ferramentas a serem utilizadas. Para fazer estas representações, podemos utilizar sistemas baseados em *workflow* . Este tipo de sistema pode automatizar os fluxos de trabalho em um processo de negócio, conforme representação da Figura 1:



Fonte: (ARAUJO; BORGES, 2001, p.6)

Figura 1 - Representação de processos em *workflow*

Os sistemas de *workflow* ainda podem ser divididos em várias categorias. Um processo de transações financeiras, um processo de documentos, um processo administrativo, tem em sua plenitude necessidades diferenciadas. A escolha de um modelo incorreto pode fazer com que a modelagem do diagrama fique complexa e muitas vezes inadequada para a

automatização do processo de negócio. A seguir conforme Villas e Fleury (1998, p.3) serão apresentadas as quatro categorias que o *workflow* pode se classificar:

a) *Ad Hoc*: pode ser utilizado quando o processo é executado uma única vez. São utilizados onde não há um padrão pré-determinado de movimentação das informações. Neste tipo de *workflow* as pessoas atuam sobre o fluxo do mesmo, onde cada uma é responsável por passar a execução do processo para o próximo agente interventor. Este tipo de *workflow* é tipicamente baseado em e-mail;
b) Coordenação: Foi desenvolvido para facilitar ações de coordenação de processos continuamente desenvolvidos na empresa. Nestas pessoas ou grupos trabalham em colaboração para atingir determinado resultado, permitindo ao gerente determinar seu fluxo, regras de rejeição ou aprovação, e caminhos alternativos para a informação;
c) Administrativo: Este tipo de *workflow* envolve problemas repetitivos, envolvendo regras de coordenação de tarefas simples. Foi criado para fazer um roteamento inteligente de formulários, através da organização dos mesmos. Envolve a criação de campos para preenchimento e conforme o conteúdo preenchido, a rota do processo é alterada;
d) Produção: Este tipo de *workflow* envolve a automatização do fluxo de papéis na organização, transformado os mesmos em "imagens" digitais. Envolve um processamento de informações complexas, envolvendo acesso a múltiplos sistemas de informações.

Implantar sistemas de *wokflow* nem sempre é uma tarefa fácil. Manzoni (2001, apud Oliveira, 2001?, p.4) aponta algumas dificuldades em implantar sistemas de *workflow*, que podem ter duas origens: problemas técnicos e problemas informais. No aspecto técnico, são encontrados problemas como:

a) inflexibilidade;
b) complexidade nas atualizações;
c) tratamento de exceção;
d) interoperabilidade;
e) atividades não formalizadas.

Já no ponto de vista informal, são identificados como principais problemas:

a) cultura organizacional e nacional;
b) interferência nas negociações durante o processo;

c) representação do trabalho;
d) comunicação entre usuários e desenvolvedores;
e) comunicação informal entre os trabalhadores;
f) resistência a mudança;
g) mudança nos relacionamentos pessoais.

O uso de *workflow* pode trazer inúmeras vantagens na gerência dos processos de negócio. Conforme Oliveira(2001?, p.3) "Com a aplicação do *workflow* , a produtividade experimenta um inacreditável aumento. E o tempo do ciclo dos negócios diminui de forma absolutamente extraordinária". Ainda conforme *Worflow Management Coalition* (apud Oliveira 2001?, p.3*)* o uso de *workflow* traz as seguintes vantagens: Flexibilidade para mudar o modelo de processos de negócio, integração com outras aplicações, reuso das atividades e modelos de negócio, escalabilidade no desenvolvimento e execução de aplicações.

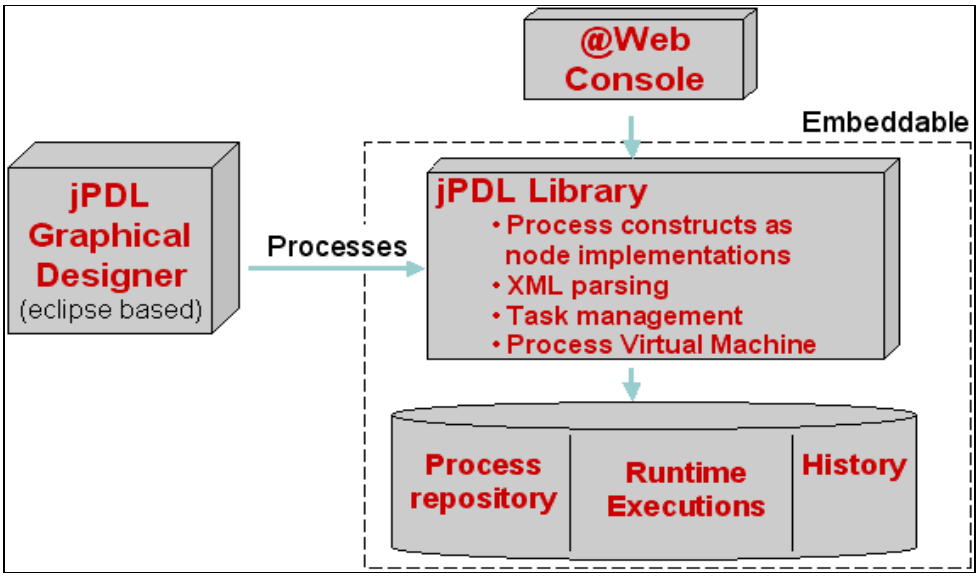
2.4 O *FRAMEWORK* JBPM

O JBPM é um *framework* desenvolvido pela JBOSS, que permite a modelagem de processos de negócio para coordenar fluxos de trabalhos entre pessoas, serviços e aplicações. Ele é um *engine* de *workflow* e *Business Process Management* (BPM) que utiliza vários *workflow patterns*. Estes padrões nasceram de uma pesquisa acadêmica feita pelo Prof. Wil van der Aalst e sua equipe na Universidade de Tecnologia de Eindhoven na Holanda. Este professor é autor de vários livros sobre a tecnologia de *workflow* e atua no Departamento de Sistemas de Informação desta universidade (WORKFLOW PATTERNS INITIATIVE, 2007).

As pesquisas do Prof. Aalst iniciaram no ano de 1999 e foram publicadas no ano de 2000 em Israel na *International Conference on Cooperative Information Systems* (IFCIS). Desde a publicação sua pesquisa vem ganhando notoriedade e é utilizada por vários sistemas de *workflow* dentre eles o JBPM, OpenWFE, YAWL entre outros (WORKFLOW PATTERNS INITIATIVE, 2007).

Os principais componentes do *engine* consistem em classes Java para gerenciamento de processos e uma interface gráfica para a modelagem do processo de negócio, conforme demonstrado na

Figura 2:



Fonte: (RED HAT MIDDLEWARE, 2007)

Figura 2 - Arquitetura do JBPM

O processo modelado na interface *jPDL Graphical Designer* é representado por um grafo. Este possui um início e pode ter vários finais. Possui também vários nodos ao longo da execução que representam ações ou estados do processo conforme visto na Figura 3:

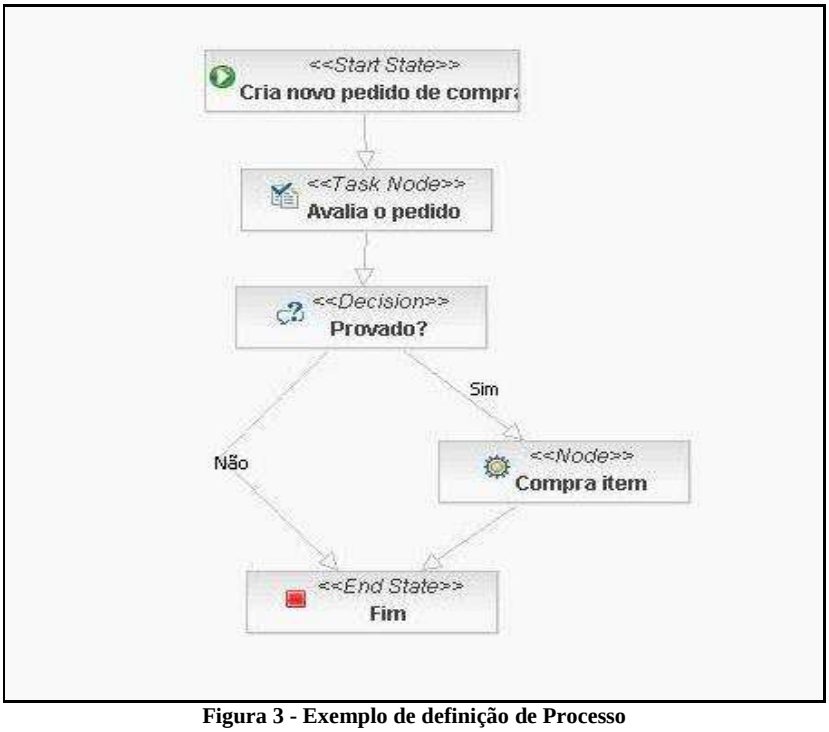


Figura 3 - Exemplo de definição de Processo

A modelagem do processo é feita com um *plugin* instalado no ambiente de desenvolvimento Eclipse. Ele gera um arquivo do tipo JPDL com o código fonte do processo. Este arquivo é carregado pelo *engine* do JBPM e cria os nodos e transições para o processo conforme a Figura 4.

```
<?xml version="1.0"?>

<process-definition name="compra"
                    xmlns="urn:jbpm.org:jpdl-3.2">

  <!-- NODES -->

  <start-state name="Cria novo pedido de compra">
    <task swimlane="comprador" />
    <transition to="Avalia o pedido" />
  </start-state>

  <task-node name="Avalia o pedido">
    <timer duedate="20 seconds" repeat="10 seconds">
        <action class="org.jbpm.websale.RemindActor">
          </action>
      </timer>
    </task>
      <transition name="" to="Provado?"></transition>
  </task-node>

  <node name="Compra item">
    <action class="org.jbpm.websale.CompraItem">

        </action>
      <transition name="" to="Fim"></transition>
  </node>

  <end-state name="Fim" />
   <decision name="Provado?">
      <transition name="Sim" to="Compra item"></transition>
      <transition name="Não" to="Fim"></transition>
   </decision>

</process-definition>
```

Figura 4 - Código fonte do exemplo de definição de processo

Cada nodo do processo tem um tipo específico (*state, decision, fork, join*...) com uma ou mais transições de saída para o próximo nodo. A execução do processo segue um ou mais caminhos, conhecidos como *tokens*. Quando uma instância do processo é criada, junto com ela, um *token* denominado *root token* é criado, e posicionado no estado inicial do processo. Quando ele aponta para um nodo, o mesmo é executado. Depois da sua execução o *token* vai para a transição que leva ao próximo nodo. A execução do processo termina quando o *token* for posicionado no estado final do processo. A seguir, conforme Red Hat Middleware (2007)

é apresentada uma breve explanação sobre os tipos de nodos do processo:

a) *Task*: Refere-se a um nodo que apresenta uma ou mais tarefas a serem executadas por indivíduos ou um aplicativo. Estas tarefas são descritas estaticamente no arquivo de definição do processo. Quando o *token* aponta, entra, ou sai do nodo, automaticamente as tarefas são executadas. É o tipo mais comum de nodo utilizado na modelagem do processo. Para representar os nodos do processo do tipo *task* é usada a classe *TaskInstance*. Esta classe inclui vários métodos entre eles: *actorId*, para definir o ator que está executando a tarefa, *start* para iniciar a tarefa e *end* para finalizar a tarefa;
b) *State*: Representa um estado de espera. É utilizado quando o sistema aguarda a resposta de algum aplicativo externo. Deste modo, entrando em estado de espera;
c) *Decision*: Neste tipo de nodo são implementadas condições, as quais são utilizadas para a escolha da transição de saída do nodo;
d) Node: Este tipo de nodo permite a criação de um elemento *action*. Este elemento é representado por uma classe que implementa os métodos da classe *ActionHandler*. Esta classe pode executar lógica de programação definida na criação do processo. Para isto, deve-se reescrever o método *execute* da classe a ser criada.

Para fazer a representação de um processo, o JBPM usa a classe *ProcessInstance.* Esta classe representa a instância de um processo em andamento. Para criar uma nova instância de processo, passa-se como parâmetro no construtor da classe um objeto do tipo *ProcessDefinition*. Este objeto tem como atributo a definição de processo elaborada no arquivo JPDL.

Para fazer a persistência das informações, o JBPM utiliza o *framework* Hibernate. Isso traz inúmeras vantagens ao produto, entre elas, a compatibilidade da aplicação com os principais bancos de dados disponíveis no mercado.

2.5 O *Framework* Hibernate

O Hibernate é um *framework* de código aberto responsável por fazer o mapeamento objeto-relacional, ou seja, fazer o mapeamento de objetos Java para tabelas de um banco de

dados relacional. Ele facilita o trabalho do programador, diminuindo a necessidade de escrever códigos repetitivos para acessar registros no banco de dados. O Hibernate suporta vários tipos de objetos Java, com herança, polimorfismo, coleções etc. Ele utiliza como parâmetro para fazer o mapeamento dos objetos no banco de dados, arquivos do tipo *Extensible Markup Language* (XML). Estes arquivos indicam quais são os atributos de um determinado objeto e em quais tabelas e campos eles serão mapeados.

O Hibernate utiliza sua própria linguagem para fazer buscas no banco de dados. Esta linguagem é chamada HQL.

A utilização do Hibernate é bastante simples, bastando apenas de cinco passos:

a) cria-se uma tabela no banco de dados onde o objeto vai ser persistido;
b) cria-se a classe, cujos atributos serão persistidos;
c) cria-se também um arquivo do tipo XML, para relacionar os atributos da classe aos campos na tabela criada no banco de dados;
d) cria-se uma classe contendo os métodos necessários para persistir o objeto, utilizando-se das classes do Hibernate que abstraem os detalhes desta operação. Geralmente usa-se o mesmo nome da classe a ser persistida adicionando-se depois do nome a palavra DAO;
e) cria-se um arquivo contendo os parâmetros de conexão do banco de dados para que o Hibernate possa conectar-se a ele.

Depois disso instanciam-se as classes criadas e utilizam-se os métodos da classe DAO para gravar ou buscar objetos no banco de dados.

3 DESENVOLVIMENTO DO TRABALHO

Foi desenvolvido um sistema de *Help Desk* baseado em *workflow* para auxiliar no controle de chamados da Seção de Apoio ao Usuário da FURB

3.1 SISTEMA ATUAL

O APUS, onde o sistema será implantado, é a seção da FURB responsável pelo suporte técnico de hardware e software das estações de trabalho ligadas à área administrativa da universidade, bem como o esclarecimento de dúvidas dos usuários relacionadas a softwares instalados nestes equipamentos.

Atualmente o sistema de *Help Desk* utilizado chama-se Relativa Manager (RM). Este software é desenvolvido pela empresa Relativa Soluções em Informática, contemplando o cadastro de chamados, o cadastro de hardware e software e a geração de relatórios.

O fluxo de trabalho funciona da seguinte maneira: os usuários ligam para uma central de atendimento, a qual abre um chamado no sistema que passa por uma série de tarefas até ser resolvido. Este fluxo está representado na Figura 5 através do diagrama de atividades.

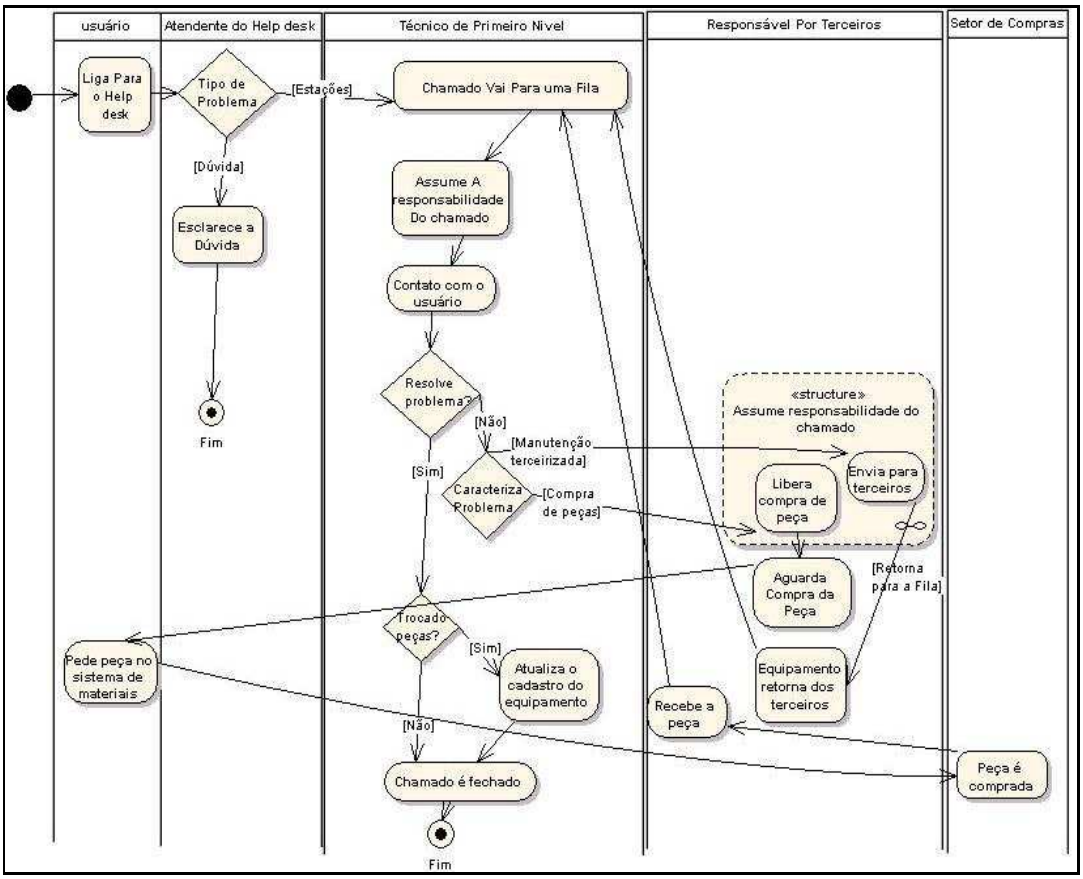


Figura 5 - Fluxo do Sistema Atual

Este fluxo de trabalho apresenta várias deficiências que serão apresentadas a seguir.

A primeira deficiência está relacionada ao atendente de *Help Desk* , nem todas as solicitações do usuário são registradas, somente é registrado um chamado se foi necessário passar o atendimento a um técnico.

A segunda deficiência apresentada está relacionada aos técnicos especialistas, que são os técnicos que resolvem os problemas dos servidores da rede da universidade: se o problema for passado para um destes técnicos isto não é registrado no sistema e com isso não será contabilizado nas estatísticas de trabalho no setor.

Outra deficiência atualmente presente no sistema está no controle do chamado, ele passa de pessoa para pessoa. Para fins de estatística no sistema atual, a pessoa que resolveu o chamado foi a última a trabalhar nele, não existe registro dos outros técnicos que passaram pelo processo. Fica parecendo que a única pessoa que trabalhou no chamado foi a última a assumí-lo. Neste fluxo também as pessoas tem que passar os chamados umas para as outras de

maneira manual, ou seja, o técnico muda no campo “responsável pelo chamado” e altera este campo escolhendo outra pessoa como responsável.

Segundo Cruz (1998, p. 70), este tipo de abordagem, onde as atividades ficam esperando para serem executadas, ou seja, os chamados ficam disponíveis no sistema esperando para que os atendentes os executem, denomina-se processo passivo. Neste tipo de abordagem as atividades são puxadas e não puxam o fluxo de trabalho.

Na Figura 5 também se pode-se observar o fluxo referente à substituição de peças e conserto de equipamentos por terceiros. O problema desta atividade é que ela é feita de maneira manual: as peças são compradas, mas não ficam vinculadas ao equipamento onde elas são instaladas. Por isso não existe rastreabilidade das mesmas.

Todos os equipamentos de informática da universidade como impressoras, computadores monitores e scanners. estão cadastrados no sistema de *Help Desk* com seus identificadores patrimoniais e seus dados técnicos. Hoje, este cadastro é mantido de maneira manual pelo técnico responsável pela troca da peças ou por uma pessoa que passa de setor em setor colhendo os dados e atualizando no sistema. Este modo não é muito eficiente, pois quando muda algum detalhe no computador pode levar meses até que o cadastro seja atualizado.

3.2 SISTEMA IMPLEMENTADO

O sistema implementado neste trabalho visa melhorar o fluxo de trabalho do APUS usando a tecnologia de gerenciamento de processos do *workflow*. Foram feitas mudanças no fluxo de trabalho que podem ser observadas através do diagrama de atividades na Figura 6.

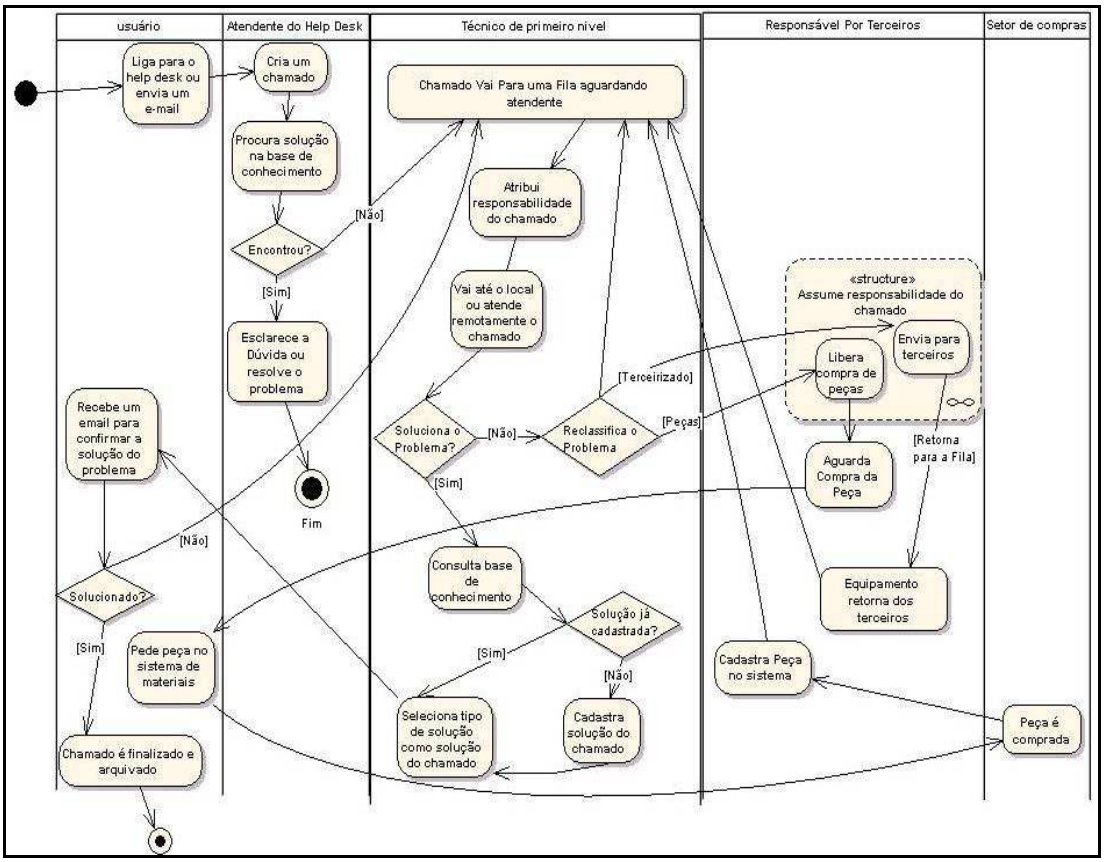


Figura 6 - Novo Fluxo do sistema

No novo sistema implementado quando um usuário liga para o APUS a primeira atividade do atendente de *Help Desk* é criar um novo chamado no sistema, com isso todo o atendimento é registrado.

Como o sistema era desenvolvido anteriormente, o chamado era passado de pessoa para pessoa conforme os acontecimentos iam evoluindo. Desta maneira fica muito difícil para mapear as atividades. No sistema implementado, o chamado é classificado como processo, ele é dividido em várias tarefas e estas tarefas estão vinculadas aos usuários do sistema.

Cruz (1998 p.38) define que todo processo é composto por várias atividades e essas formam um conjunto de procedimentos que quando executados com sucesso devem produzir um determinado resultado. Segundo Usirono (2003, p. 41),

> o modelo de *workflow* é centrado no processo. Ele não atua de forma passiva onde as informações são armazenadas e os usuários acessam estas informações como no modelo *workgroup.* Ele atua ativamente entre os usuários de forma que estes cumpram o fluxo de trabalho que está determinado.

Com isto as tarefas são automaticamente vinculadas às pessoas, ou seja, o técnico ao terminar de resolver uma tarefa, automaticamente o sistema vinculará outra tarefa a ele, conforme o tipo de tarefa e a especialidade de atendimento.

Cruz (1998, p. 108) define rotas como o "caminho lógico que, definido sob regras específicas, tem a função de transferir a informação dentro do processo, ligando as atividades associadas ao fluxo de trabalho." No novo sistema o fluxo de chamados entre os técnicos é feito de maneira automatizada através de rotas, ou seja, o técnico registra no sistema que não conseguiu resolver o problema, registra a sua suspeita de qual área o problema está relacionado e automaticamente o sistema cria uma nova tarefa enviando um e-mail para outro técnico para que ele tente resolver o problema em questão.

A base de usuários do sistema, diferentemente do que acontecia antigamente é integrada com a base de usuários da FURB, sendo apenas cadastrados os privilégios dos usuários dentro do novo sistema.

Assim como acontecia antigamente, foi implementado um cadastro de hardwares (computadores, monitores, scanners e impressoras). O sistema proposto vincula as peças ao cadastro do micro quando uma nova peça for instalada nele.

Dentre as principais melhorias implantadas no sistema podemos resumir:

a) Chamado classificado como processo e dividido em tarefas;
b) Tarefas vinculadas automaticamente às pessoas através de e-mail;
c) Fluxo segue rotas do processo;
d) Base de usuários vinculada à base principal da FURB;
e) Implementado cadastro de hardwares e peças.

3.3 REQUISITOS DO SISTEMA

O Quadro 2 apresenta os requisitos funcionais definidos para o sistema e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Requisitos Funcionais	**Caso de Uso**
RF1. O sistema deverá permitir ao administrador manter cadastro de privilégios dos usuários.	UC01

RF2. O sistema deverá possibilitar cadastro de chamados pelo atendente.	UC02
RF3. O sistema deverá enviar um e-mail atribuindo o próximo chamado da fila ao técnico que estiver disponível	UC02
RF4. Quando o atendente der por concluído o chamado o sistema deve enviar um e-mail com um link para que o usuário confirme a resolução do chamado. E um e-mail para o atendente que abriu o chamado.	UC03
RF5. O sistema deverá enviar um e-mail notificando o gestor sempre que um chamado extrapolar um prazo determinado pelo mesmo.	UC09
RF6. O sistema deve fechar o chamado e arquivá-lo quando o técnico encerre a última tarefa do chamado	UC03
RF7. O sistema deve permitir que o usuários e técnicos consultem o status dos seus chamado.	UC04
RF8. O sistema deverá emitir um relatório ao gestor contendo os chamados em aberto.	UC05
RF9. Quando o técnico solicitar uma peça o sistema deverá enviar um e-mail e atribuir uma tarefa ao responsável pelas peças.	UC06
RF10. O sistema deverá permitir ao técnico manter um cadastro de hardware (computadores, monitores, scanners e impressoras etc).	UC07
RF11. O sistema deverá permitir ao responsável manter um cadastro de peças de computador disponíveis no estoque.	UC08
RF12. O sistema deverá vincular as peças de estoque ao equipamento onde foram instaladas.	UC06
RF13. O sistema deve permitir que após concluir o chamado, o técnico consulte a base de conhecimentos e caso não exista a solução ele a cadastra para posterior consulta dos atendentes.	UC03
RF14. O sistema deve emitir um relatório de atendimentos separado por hardwares.	UC05
RF15. O sistema deve possibilitar ao atendente classificar o chamado como problema de hardware, software, equipamento terceirizado, servidores. E conforme o tipo criar uma tarefa para o técnico responsável pelo tipo de atendimento.	UC02

Quadro 1: Requisitos funcionais

O Quadro 3 lista os requisitos não funcionais definidos para o sistema.

Requisitos Não Funcionais.
RNF01. O sistema deverá ser implementado na linguagem Java padrão J2EE.
RNF02. O sistema deverá utilizar a base de usuários já existente nos sistemas da FURB.
RNF03. O sistema deverá utilizar o *workflow Engine JBPM.*
RNF04. O sistema deverá utilizar o banco de dados *Oracle.*

Quadro 2: Requisitos não funcionais

3.4 DIAGRAMA DE CASOS DE USO

A seguir na Figura 7 apresenta-se o modelo de casos de uso que foi implementado no sistema. Este modelo foi criado com a ajuda da ferramenta *Enterprise Architect*.

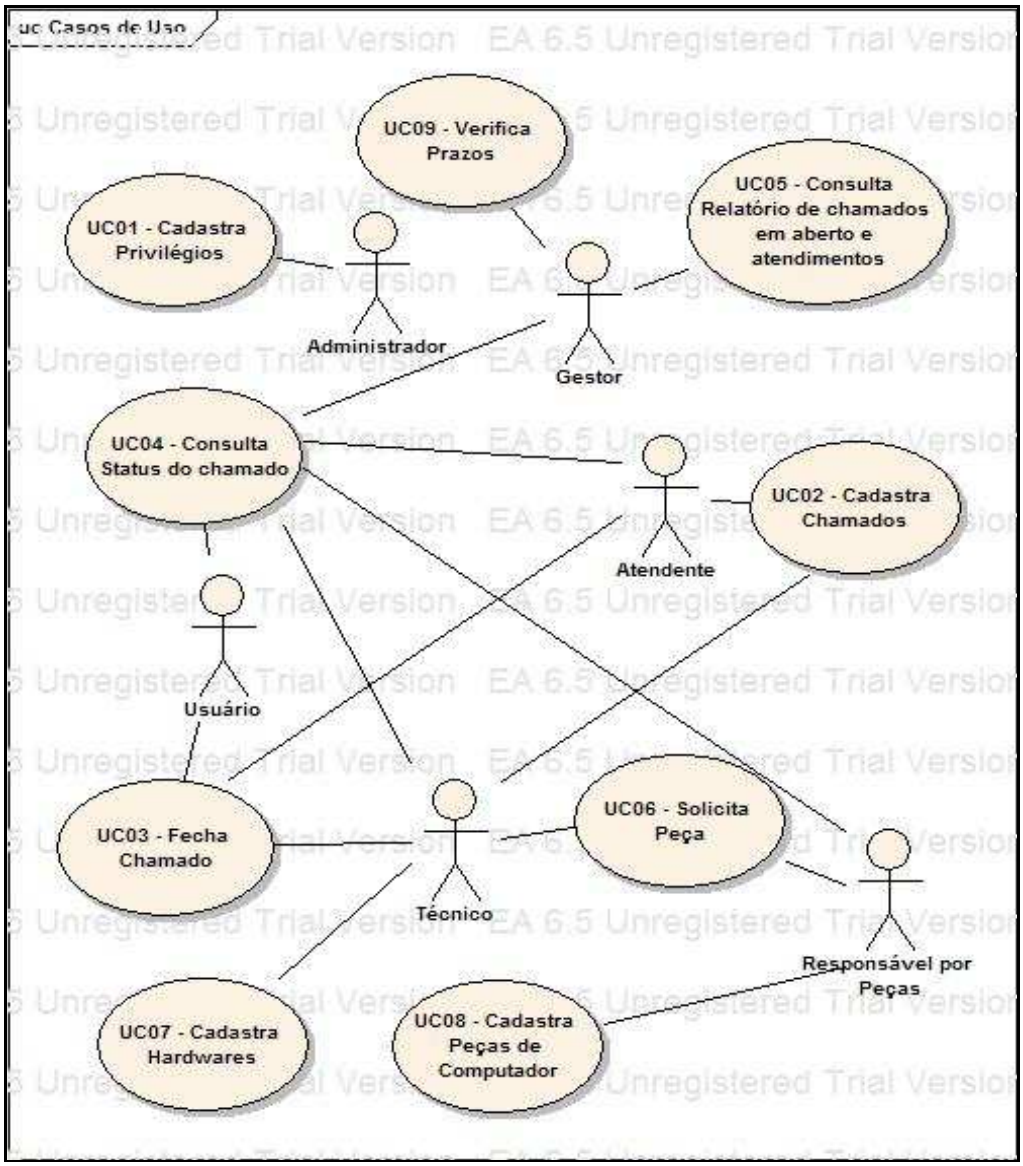


Figura 7 - Diagrama de Casos de Uso

Para representar os usuários que farão interações no sistema foram definidos seis atores nos os casos de uso, que são eles:

a) Gestor: Chefe da seção de apoio ao usuário;

b) Administrador: Pessoa responsável pela administração do sistema;

c) Atendente: Pessoa responsável por atender telefone do *Help Desk* ;

d) Usuário: Pessoa que solicita um atendimento;

e) Técnico: Pessoa responsável por atender os chamados;

f) Responsável por peças: Pessoa responsável pelo controle de peças e

manutenções terceirizadas.

A seguir apresenta-se uma breve explanação sobre cada um dos casos de uso e onde estes atores farão interações:

a) UC01 – Cadastra privilégios: Deve ser definido no sistema os privilégios de segurança de acordo com o tipo de usuário. Este usuário é previamente cadastrado na base de dados pelo sistema de RH. Nesta tela o administrador deve informar qual vai ser o nível de privilégios do usuário podendo ser: usuário comum, técnico, gestor, ou administrador;
b) UC02 – Cadastra chamados: Deve ser cadastrado um novo chamado no sistema, informando o equipamento a ser atendido, o setor de localização do mesmo, o tipo de atendimento e as observações quanto ao problema;
c) UC03 – Fecha chamado: Deve ser finalizada a tarefa que o técnico estava executando. O técnico informa a solução da tarefa e se o chamado foi finalizado com sucesso ou não.

   Se o técnico informar que o chamado foi finalizado com sucesso, o sistema envia um e-mail para o usuário que solicitou o chamado, solicitando a confirmação de que o problema foi resolvido, e com isso o chamado é finalizado. Se o técnico informar que o chamado não foi finalizado com sucesso, ele encaminha o chamado para outro técnico informando qual é a área da suspeita do problema, com isso o sistema automaticamente cria uma tarefa para outro técnico da área;
d) UC04 – consulta status do chamado: Deve ser consultado o status do chamado em aberto: no caso do usuário que solicitou o chamado, ao entrar no sistema aparece automaticamente o status dos chamados em aberto pra ele, no caso dos outros usuários eles consultam o status dos chamados em aberto através da opção no menu do sistema;
e) UC05 – Consulta relatório de chamados abertos e atendimentos: Neste caso é tirado um relatório no sistema de todos os chamados em aberto ou em andamento com algum dos técnicos;
f) UC06 – Solicita peça: Deve ser solicitada uma peça pelo técnico quando necessário para resolver um chamado. Neste caso o técnico solicita a peça e o sistema envia um e-mail para o responsável por peças atribuindo a tarefa a ele;
g) UC07 – Cadastra hardware: Deve ser cadastrado um equipamento (impressora, scanner, microcomputador, monitor). O técnico informa os dados do hardware

e cadastra no sistema;

h) UC08 – Cadastra peças de computador: Deve ser cadastrada pelo responsável uma nova peça no sistema quando ela for comprada pelo departamento de compras;
i) UC09 – Verifica prazos: O sistema envia um aviso ao gestor indicando que o prazo de atendimento do chamado que foi pré-cadastrado expirou.

3.5 DIAGRAMA DE CLASSES

Na Figura 8 é apresentado o diagrama de classes utilizado na implementação do sistema. Este diagrama foi construído com a ajuda da ferramenta *Enterprise Architect.*

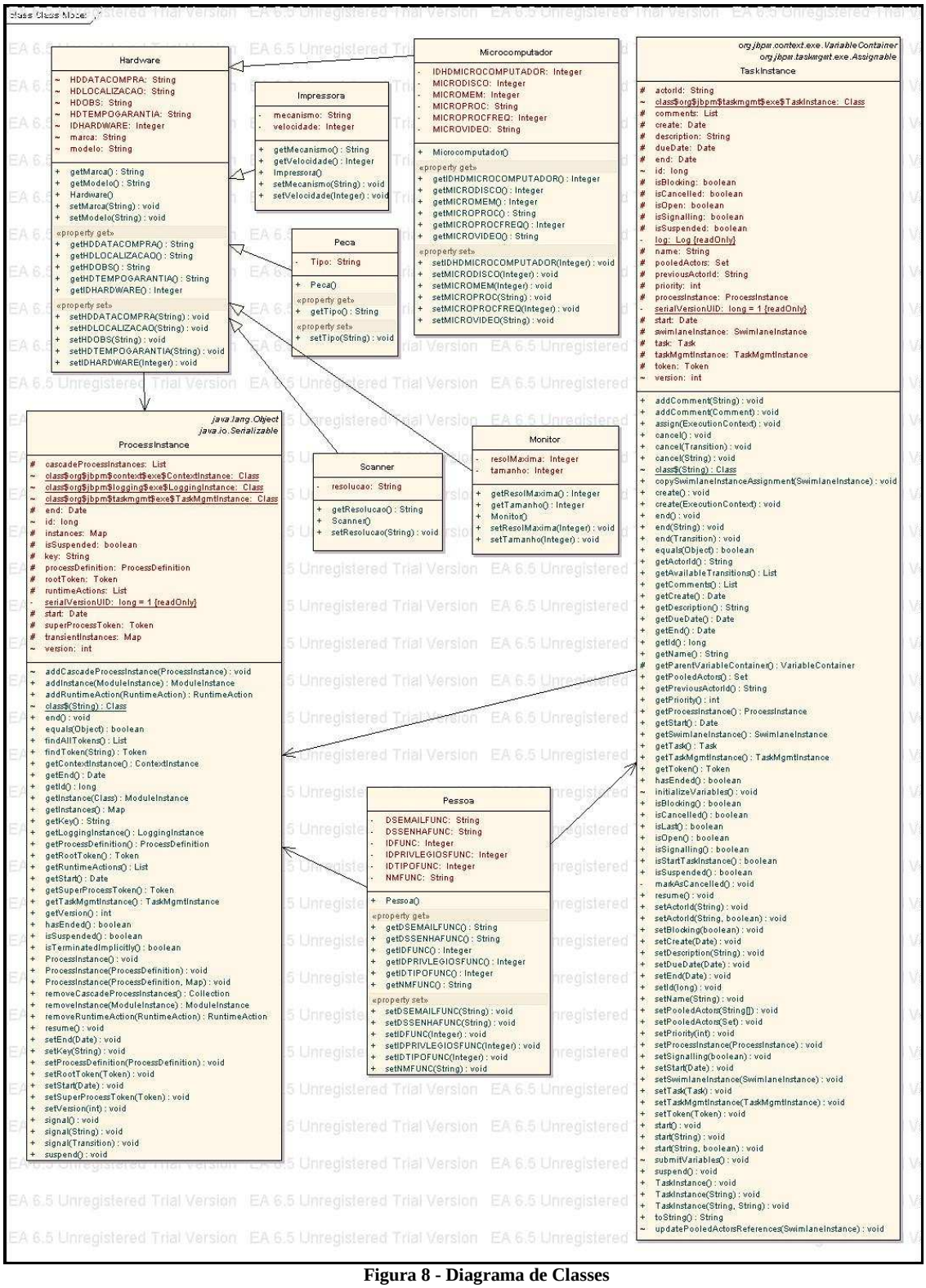


Figura 8 - Diagrama de Classes

A classe Hardware representa um equipamento cadastrado no sistema. Possui os atributos comuns das classes que fazem herança desta classe. As classes que herdam a classe hardware são: Micromputador, Scanner, Monitor, Peca e Impressora. Todas estas classes são utilizadas para representar equipamentos cadastrados no sistema.

A classe pessoa representa um funcionário da FURB cadastrado no sistema de RH. Ela é utilizada para selecionar a pessoa que abriu o chamado, na validação do *login* do sistema, e no cadastro de técnicos.

A classe ProcessInstance serve para representar um processo, ou seja um chamado aberto no sistema. Ela é fornecida pelo JBPM, e possui vários métodos para tratar do processo.

A classe TaskInstance representa uma tarefa que está sendo executada no sistema. Ela também é fornecida pelo JBPM. Esta tarefa é executada pelos técnicos cadastrados no sistema.

3.6 MODELO ENTIDADE RELACIONAMENTO

A Figura 9 apresenta o modelo entidade relacionamento do sistema. As tabelas em que o nome inicia com JBPM_ são fornecidas pelo próprio *framework.* As demais tabelas foram criadas para atender as necessidades.

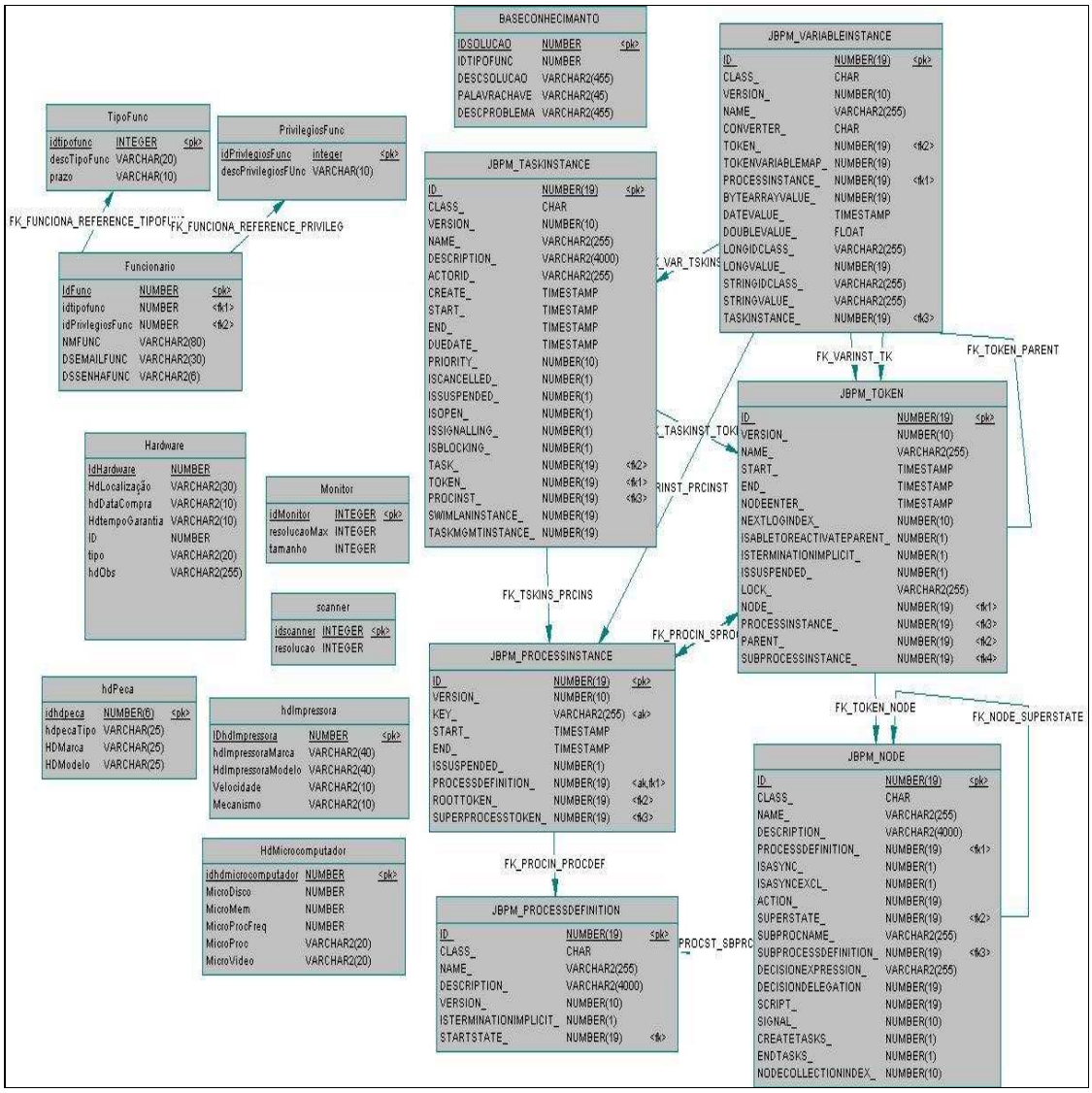


Figura 9 - Modelo entidade relacionamento

A seguir será descrito a utilidade de cada tabela no modelo entidade relacionamento:

a) Hardware, Monitor, hdImpressora, hdMicrocomputador, hdPeca, scanner: Estas tabelas são utilizadas para o cadastramento dos equipamentos atendidos pelo APUS. A tabela Hardware armazena os atributos comuns de todos os tipos de hardware, ou seja, os atributos da classe hardware. As demais tabelas armazenam os atributos das classes herdadas da classe hardware. No modelo demonstrado na Figura 9, não existe chave estrangeira ligando estas tabelas, isto ocorre porque o armazenamento desta herança é controlado pelo

Hibernate. O Hibernate utiliza o mesmo valor de chave primária para a tabela pai e para a tabela herdada, conforme demonstrado mais a frente na Figura 14;

b) Funcionário: Armazena dados sobre todos os funcionários da FURB cadastrados pelo sistema de RH. Esta tabela armazena tambem uma cópia das colunas DSEMAILFUNC e DSSENHAFUNC que é copia de outra tabela no banco de dados principal da FURB;
c) BASECONHECIMENTO: Nesta tabela são armazenados os dados referentes a base de conhecimentos do sistema, ou seja, os problemas e soluções cadastrados pelos técnicos para posterior consulta;
d) JBPM_TASKINSTANCE: Armazena as instâncias das tarefas que são criadas no sistema;
e) JBPM_PROCESSINSTAMCE: Armazena as instâncias dos processos que estão sendo executados ou foram encerrados no sistema, ou seja, os chamados;
f) JBPM_PROCESSDEFINITION: Esta tarefa é responsável por armazenar as definições de processo criadas no Eclipse;
g) JBPM_VARIABLEINSTANCE: Esta tabela armazena as instâncias das variáveis que podem ser armazenadas na instância de um processo;
h) JBPM_TOKEN: É responsável por armazenar os *tokens* de execução do processo;
i) JBPM_NODE: Esta tabela armazena as instâncias dos nodos do processo.

3.7 IMPLEMENTAÇÃO

Nos próximos tópicos serão demonstradas as ferramentas e tecnologias abordadas no desenvolvimento de trabalho.

3.7.1 Técnicas e ferramentas utilizadas

Este trabalho foi desenvolvido utilizando a linguagem Java padrão J2EE, o banco de dados Oracle para persistência dos dados, o ambiente de desenvolvimento Eclipse, o *framework* JBPM para desenvolvimento do *workflow*, o *framework* Hibernate para

persistência das classes Java, a ferramenta Enterprise Architect para construção dos diagramas da UML e a ferramenta de modelagem de banco de dados Power Designer.

3.7.1.1 Utilizando o JBPM

O JBPM é um pacote fornecido pela JBOSS, que pode ser baixado em: (http://labs.jboss.com/jbossjbpm/downloads). O pacote JBPM Suíte traz a maior parte das ferramentas necessárias para desenvolver um sistema usando a plataforma.

O primeiro passo para criação do sistema é criar o arquivo de definição de processo. Este arquivo será responsável por mapear o processo de negócio para o Jbpm. O *plugin* para o Eclipse disponível juntamente com o JBPM fornece o suporte a criação deste arquivo. Ele fornece uma interface gráfica para a modelagem dos componentes do processo. Quando um novo arquivo é criado, automaticamente o eclipse cria um arquivo do tipo XML para armazenar a definição de processo e outro arquivo do tipo JPG para visualização da definição do processo. A seguir na Figura 10 apresenta-se um trecho do arquivo de definição de processo utilizado no projeto deste TCC. A segunda linha do arquivo define o nome do processo, as demais linhas definem os nodos, as transições e regras do processo.

```
<?xml version="1.0" ?>
- <process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="Chamado">
    <!-- SWIMLANES (= process roles) -->
    <swimlane name="Atendente" />
  - <swimlane name="Tecnico">
      <assignment actor-id="#{ contextInstance.variables['atendente']}" />
    </swimlane>
  - <swimlane name="Sistema">
      <assignment actor-id="Sistema" />
    </swimlane>
    <!-- NODES -->
  - <start-state name="Cria novo chamado">
      <task name="Cria novo Chamado" swimlane="Atendente" />
      <transition name="" to="Procura Solucao" />
    </start-state>
    <end-state name="Fecha Chamado" />
  - <task-node name="Atende">
      <task name="AtendeChamado" swimlane="Tecnico" />
      <transition name="Solucionou" to="Confirmaçao do Usuario" />
      <transition name="Nao Solucionou" to="Problema de Peças?" />
    </task-node>
  - <task-node name="Procura Solucao">
      <task name="Procura Solucao" swimlane="Atendente" />
      <transition name="Sim" to="Fecha Chamado" />
      <transition name="Nao" to="Atribui Chamado?" />
    </task-node>
  - <task-node name="Aguardando">
    - <task name="Aguardando" swimlane="Sistema">
        <assignment actor-id="Sistema" />
      </task>
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.CreateTimeActionHandler" />
      </event>
      <transition name="Atender" to="Atende" />
    </task-node>
  - <task-node name="Confirmaçao do Usuario">
    - <task name="Aguardando Usuario">
        <assignment actor-id="#{contextInstance.variables
          ['idSolicitante']}" />
      </task>
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.EnviaEmaiActonHandler" />
      </event>
      <transition name="Resolvido" to="Fecha Chamado" />
      <transition name="Nao Resolvido" to="Aguardando" />
    </task-node>
  - <decision name="Atribui Chamado?">
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.AtribuiResponsabilidadeHandler" />
      </event>
    - <transition name="Nao Atribuido" to="Aguardando">
        <condition>#{contextInstance.variables['atendente'] == 'Sistema'}
          </condition>
```

Figura 10 - Arquivo de Definição do Processo Chamado

Depois de definido o processo é preciso armazená-lo no banco de dados. Agora para criar uma nova instância do processo será somente necessário, passar como parâmetro o nome do processo.

A classe de exemplo da Figura 11 é responsável por criar um chamado no sistema, ou seja, ela cria uma nova instância da definição de processo armazenada no banco de dados.

```
package helpdesk.web;

import java.io.IOException;

/**
 * @web.servlet name="CriaChamado" display-name="Criar Um Chamado"
 *              description="Cria Chamado"
 *
 * @web.servlet-mapping url-pattern="/CriaChamado"
 *
 */

public class CriaChamadoServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
        response.setContentType("text/html");
        JbpmConfiguration jbpmConfiguration = JbpmConfiguration.getInstance();
        JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
        HttpSession session = request.getSession();
        /*
         * seta o ator que está abrindo o chamado atraves de um atributo de
         * sessao
         */
        jbpmContext.setActorId((String) session.getAttribute("idusuario"));
        GraphSession gs = jbpmContext.getGraphSession();
        /*
         * cria uma instancia de ProcessDefinition com a definição de processo
         * "chamado"
         */
        ProcessDefinition pd = gs.findLatestProcessDefinition("Chamado");
        ProcessInstance pi = new ProcessInstance(pd);

        TaskInstance taskInstance = pi.getTaskMgmtInstance()
                .createStartTaskInstance();
        /*
         * Cria um map com as variaveis que vão fazer parte do processo As
         * variaveis estão na sessão do usuário
         */
        Map taskVariables = new HashMap();
        taskVariables.put("idSolicitante", request.getParameter("idPessoa"));
        taskVariables.put("idSetor", request.getParameter("idSetor"));
        taskVariables.put("idEquipamento", request
                .getParameter("idEquipamento"));
        taskVariables.put("idChtipo", request.getParameter("idChtipo"));
        taskVariables.put("Obs", request.getParameter("Obs"));
        taskVariables.put("atendente", request.getParameter(""));
        taskInstance.addVariables(taskVariables);
        /*
         * O metodo close do jbpmContext presiste no banco de dados as
         * informações
         */
        taskInstance.end();
        jbpmContext.save(taskInstance);
        jbpmContext.close();
        /*
         * Redireciona para a página para escolher a solução do problema
         */
        long idProcessoTemp = pi.getId();
        Long idProcesso = new Long(idProcessoTemp);
        session.setAttribute("idProcesso", idProcesso);
        response.sendRedirect("escolheSolucao.jsp");
    }

}
```

Figura 11 - Classe CriaChamadoServlet

Esta classe é estendida de HttpServlet. O método post do *servlet* é responsável por criar no novo chamado e no final redirecionar para o *servlet* que seleciona a solução da tarefa.

O JBPM tem a opção também de adicionar variáveis à instância do processo criado. Estas variáveis podem armazenar qualquer tipo de objeto que o processo necessitar. As variáveis armazenadas devem ser guardadas em um objeto do tipo *Map.*

Na classe da Figura 11 foi criado um *Map* das variáveis que serão inseridas através do código: *Map taskvariables = new HashMap().* Nas linhas subseqüentes desta parte do código são adicionadas as variáveis que vão fazer parte do processo.

Chegando ao final da classe há a parte do código responsável por finalizar a tarefa de inicialização do processo através do código: *taskInstance.end();* Depois disso o código jbpmContext.close() persiste as informações no banco de dados através do *framework Hibernate*.

Mais detalhes sobre a utilização do JBPM pode ser encontrado no Apêndice A.

3.7.1.2 Utilizando o *Hibernate*

O Hibernate foi utilizado na implementação do sistema tanto pelo *framework* JBPM como pelos cadastros adicionais criados para atender os requisitos. Para utilizar o Hibernate primeiramente deve-se criar o arquivo de configurações(hibernate.cfg.xml). Neste caso foi aproveitado o mesmo arquivo utilizado pelo JBPM, conforme já demonstrado anteriormente na **Erro! Fonte de referência não encontrada.**.

Para persistir uma classe no Hibernate é necessário primeiramente criar a classe a ser persistida com todos os seus métodos *gets* e *sets*. Demonstra-se isso como exemplo na classe hardware, conforme a Figura 12. Esta classe faz parte do sistema implementado neste trabalho.

```
public class Hardware {
	Integer IDHARDWARE;

	String HDLOCALIZACAO;

	String HDDATACOMPRA;

	String HDTEMPOGARANTIA;

	String HDOBS;

	public Hardware() {
		super();
		// TODO Auto-generated constructor stub
	}

	public String getHDDATACOMPRA() {
		return HDDATACOMPRA;
	}

	public void setHDDATACOMPRA(String hddatacompra) {
		HDDATACOMPRA = hddatacompra;
	}

	public String getHDLOCALIZACAO() {
		return HDLOCALIZACAO;
	}

	public void setHDLOCALIZACAO(String hdlocalizacao) {
		HDLOCALIZACAO = hdlocalizacao;
	}

	public String getHDOBS() {
		return HDOBS;
	}

	public void setHDOBS(String hdobs) {
		HDOBS = hdobs;
	}

	public String getHDTEMPOGARANTIA() {
		return HDTEMPOGARANTIA;
	}

	public void setHDTEMPOGARANTIA(String hdtempogarantia) {
		HDTEMPOGARANTIA = hdtempogarantia;
	}

	public Integer getIDHARDWARE() {
		return IDHARDWARE;
	}

	public void setIDHARDWARE(Integer idhardware) {
		IDHARDWARE = idhardware;
	}

}
```

Figura 12 - Classe Hardware

Depois da classe criada é necessário criar ainda uma classe específica para cada uma das classes a serem persistidas. É esta classe que vai fazer a persistência dos objetos no banco de dados. Na Figura 13 demonstra-se a classe utilizada para fazer a persistência da classe

hardware.

```
import java.util.List;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.classic.Session;

public class HardwareDAO {
	private SessionFactory factory;

	public HardwareDAO() {
		super();
		factory = new Configuration().addClass(Hardware.class)
					.buildSessionFactory();
	}

	public void insert(Hardware hardware) throws Exception {
		Session session = factory.openSession();
		session.save(hardware);
		session.flush();
		session.close();
	}

	public java.util.List getList(String condicao) throws Exception {
		Session session = factory.openSession();
		List hardwares = session.find(condicao);
		session.flush();
		session.close();
		return hardwares;
	}

	public Hardware retrieve(Integer pk) throws Exception {
		Session session = factory.openSession();
		Hardware hardware = (Hardware) session.load(Hardware.class, pk);
		session.flush();
		session.close();
		return hardware;
	}

	public void delete(Hardware hardware) throws Exception {
		Session session = factory.openSession();
		session.delete(hardware);
		session.flush();
		session.close();
	}
}
```

Figura 13 - Classe HardwareDAO

Para armazenar uma determinada classe do tipo hardware no banco de dados basta apenas usar os métodos (*insert, delete, getList, retrieve*) da classe HardwareDAO passando como parâmetro o objeto a ser persistido, ou o identificador do objeto a ser excluído, ou no caso de recuperar uma classe, qualquer outro parâmetro da classe a ser recuperada no banco de dados.

Ainda para fazer a persistência das classes no banco de dados é preciso fazer um mapeamento das tabelas onde as classes serão armazenadas e das colunas das tabelas onde os atributos das classes serão armazenados. Para isso é preciso criar um arquivo do tipo XML para fazer este mapeamento. Na Figura 14 demonstra-se um exemplo de arquivo de mapeamento utilizado pelo *framework* H*ibernate* para a classe Hardware na implementação do sistema.

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

<!DOCTYPE hibernate-mapping PUBLIC "
-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
    <class name="helpdesk.util.Hardware" table="Hardware">

        <id name="IDHARDWARE" column="IDHARDWARE" type="integer">
            <generator class="increment" />
        </id>
        <!-- Propriedades comuns -->
        <property name="HDLOCALIZACAO" />
        <property name="HDDATACOMPRA" />
        <property name="HDTEMPOGARANTIA" />

        <!--Sub-Classe Micro -->
        <joined-subclass name="helpdesk.util.Microcomputador"
            table="HDMICROCOMPUTADOR">
            <key column="IDHDMICROCOMPUTADOR" />
            <property name="MICRODISCO" />
            <property name="MICROMEM" />
            <property name="MICROPROCFREQ" />
            <property name="MICROPROC" />
            <property name="MICROVIDEO" />
        </joined-subclass>
        <joined-subclass name="helpdesk.util.Monitor"
            table="HDMONITOR">
            <key column="IDHDMONITOR" />
            <property name="MONTAMANHO" />
            <property name="MONRESOLMAXIMA" />
        </joined-subclass>
        <joined-subclass name="helpdesk.util.Impressora"
            table="HDIMPRESSORA">
            <key column="IDHDIMPRESSORA" />
            <property name="IMPVELOCIDADE" />
            <property name="IMPRESOLMAXIMA" />
        </joined-subclass>
        <joined-subclass name="helpdesk.util.Scanner"
            table="HDSCANNER">
            <key column="IDHDSCANNER"/>
            <property name="SCRESOLUCAO"/>
        </joined-subclass>
    </class>
</hibernate-mapping>
```

Figura 14 - Arquivo de mapeamento do *Hibernate*

A Figura 14 demonstra o arquivo hardware.hbm.xml. Este arquivo é responsável por

fazer o mapeamento dos atributos da classe hardware, mas também das classes que estendem a classe hardware.

3.7.2 Operacionalidade da implementação

Nesta etapa serão apresentadas as principais telas do sistema bem como algumas linhas de código, pra melhor explicação do software desenvolvido.

Na Figura 15 está o diagrama de processo que foi implementado no sistema. Este diagrama foi desenvolvido no *plugin* do JBPM. Ele representa o fluxo de atendimento de um chamado.

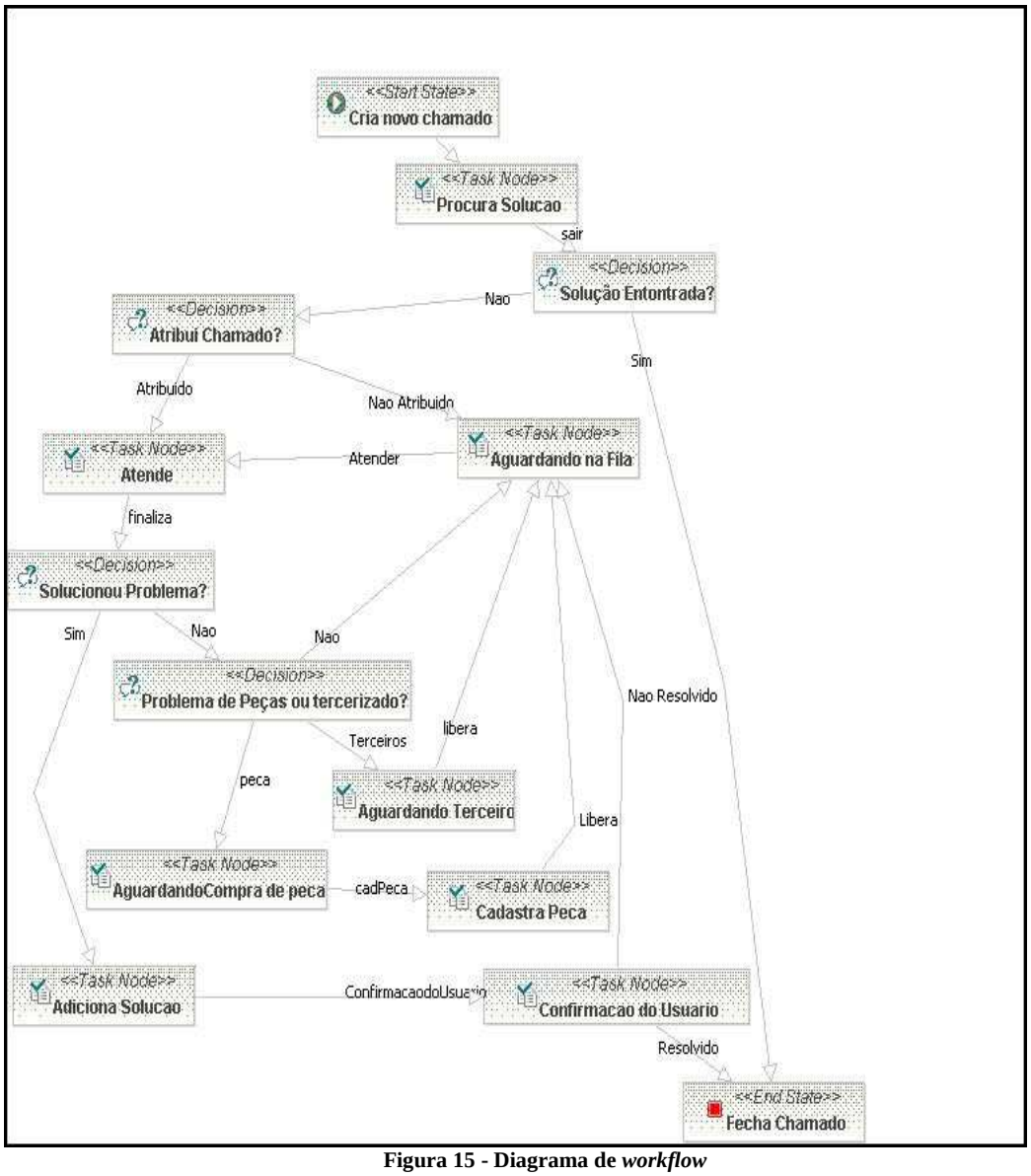


Figura 15 - Diagrama de *workflow*

A seguir será apresentada uma breve descrição das atividades apresentadas no Workflow:

a) Cria novo chamado: Nesta tarefa o processo é iniciado, um novo chamado é criado no sistema;
b) Procura solução: Nesta atividade o atendente procura uma solução no sistema para o problema apresentado pelo usuário;
c) Solução encontrada?: Nesta tarefa o sistema checa se a solução para o problema foi encontrada ele vai para a tarefa “fecha chamado”, se não foi

encontrada vai para a tarefa “Atribui Chamado?”;

d) Atribui chamado?: Caso o chamado for atribuído ele vai para a tarefa “atende chamado” senão vai para “Aguardando na fila”;
e) Aguardando na Fila: Nesta atividade o chamado fica aguardando na fila para que um atendente seja liberado para o atendimento do chamado;
f) Atende: Nesta atividade o técnico atende o chamado, ou seja, vai tentar resolver o problema apresentado pelo usuário;
g) Solucionou Problema?: Nesta atividade o sistema checa se o atendente solucionou o problema apresentado pelo usuário ou não: se o problema foi solucionado o chamado vai para a tarefa “adiciona solução”, senão ele vai para a atividade “Problemas de peças ou terceirizado?”;
h) Adiciona solução: Nesta atividade o técnico adiciona a solução que foi aplicada ao chamado no sistema;
i) Confirmação do usuário: Nesta atividade o sistema aguarda a confirmação do usuário de que o problema foi realmente resolvido: se foi resolvido vai para a tarefa “fecha chamado”, se não foi resolvido vai para a tarefa ”aguardando na fila”;
j) Problema de peças ou terceirizado?: Nesta atividade o sistema checa se o problema do chamado é de peças ou terceirizado, se for, ele vai para a atividade “Aguardando compra ou terceiros”;
k) Peças ou terceiros?: Se o problema for terceirizado o chamado vai para aguardando na fila, se for de peças vai para a tarefa “Cadastra Peça”;
l) Cadastra peça: Nesta tarefa o atendente cadastra uma nova peça no sistema.

A Figura 16 mostra a tela de *login* do sistema:

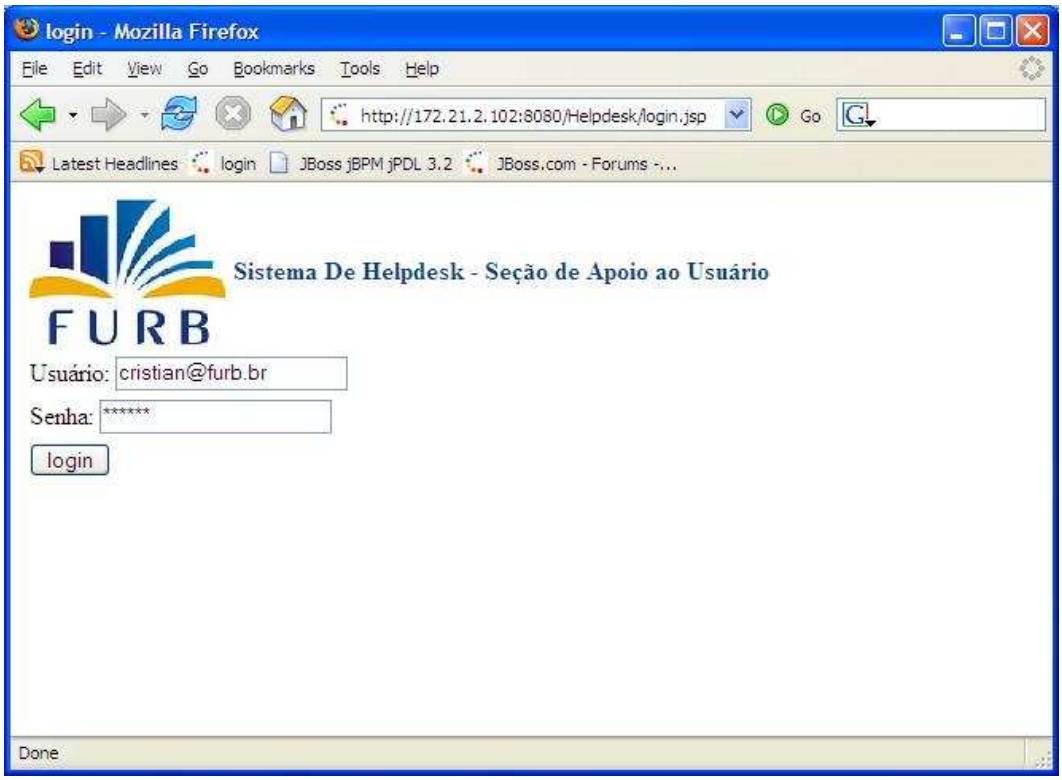


Figura 16 - Tela de *login* do sistema

Nesta tela os usuários informarão seu nome de usuário e senha para poder entrar no sistema. A base de usuários utilizada é a base principal da FURB, mantida pelo sistema de RH (Recursos Humanos) da universidade. Qualquer usuário cadastrado na FURB poderá fazer *login* no sistema. Ao fazer *login*, o sistema checa os privilégios do usuário. Se ele estiver cadastrado no sistema de *Help Desk* ele será redirecionado para a página principal do sistema, senão ele será redirecionado a uma página que mostra os chamados em aberto vinculados a ele.

A Figura 17 mostra a tela principal do sistema com um *menu* de acesso a todas as suas funções e também alguns *links* para as principais funções do sistema.

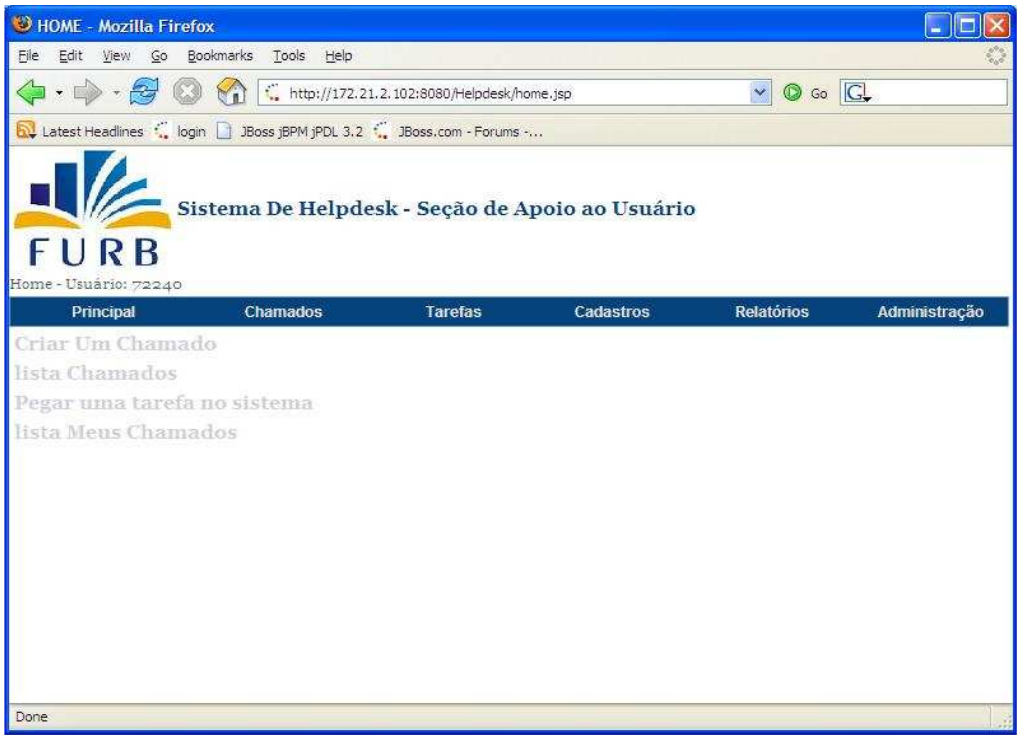


Figura 17 - Tela principal do sistema

A Figura 18 demonstra a tela para criação de um novo chamado. Nesta tela é informado o solicitante do chamado que é previamente cadastrado no sistema de RH da universidade, equipamento e setor de atendimento do chamado, também cadastrados previamente. Além disso, é informado no chamado as observações sobre o problema apresentado pelo equipamento em questão. O atendente também informa o tipo de chamado que ele está criando, para que ele seja direcionado automaticamente ao tipo correspondente de técnico.

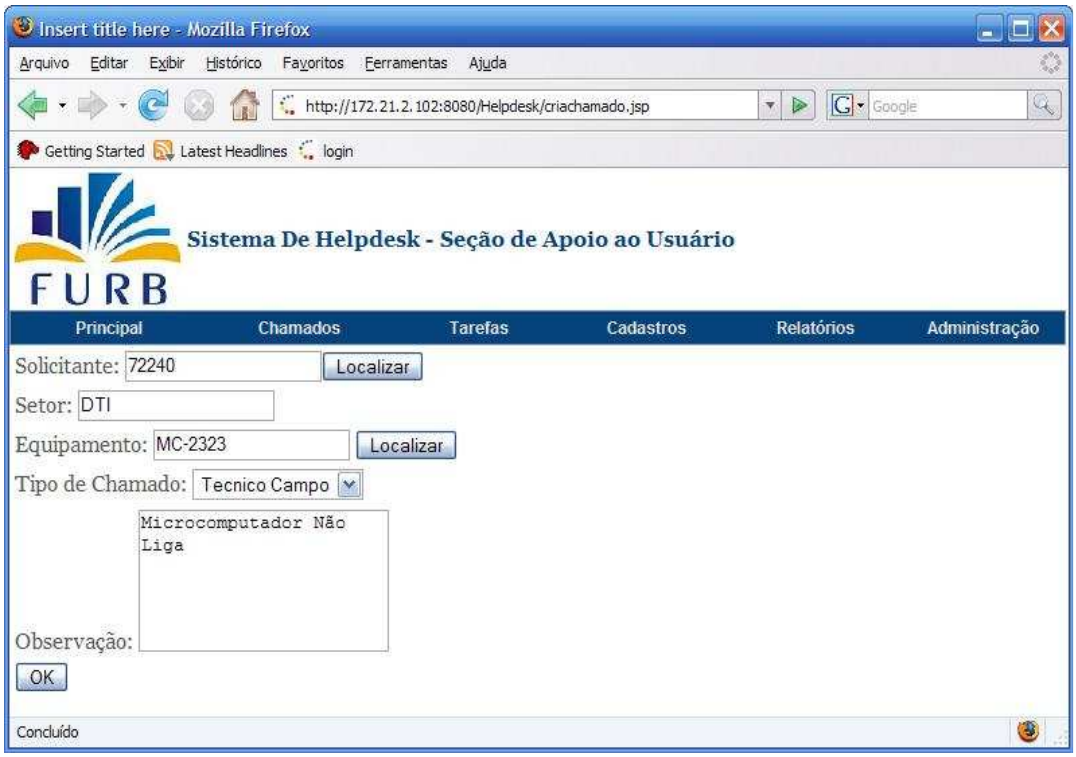


Figura 18 - Criar Chamado

Na Figura 19 está a tela de solução do chamado. Depois de o atendente ter criado o chamado na tela anterior, ele é redirecionado automaticamente para esta tela. Nela, o atendente clica no botão localizar e pode buscar uma solução na base de conhecimentos, conforme a tela na Figura 20. Se o usuário não encontrar nenhuma solução ele pode deixar em branco o campo e clicar no botão “OK”. Com isso o chamado vai para a fila e é encaminhado a um atendente.

Figura 19 - Escolhe solução para o chamado

Na tela da Figura 20, o usuário pode localizar a solução de um problema através de vários critérios.

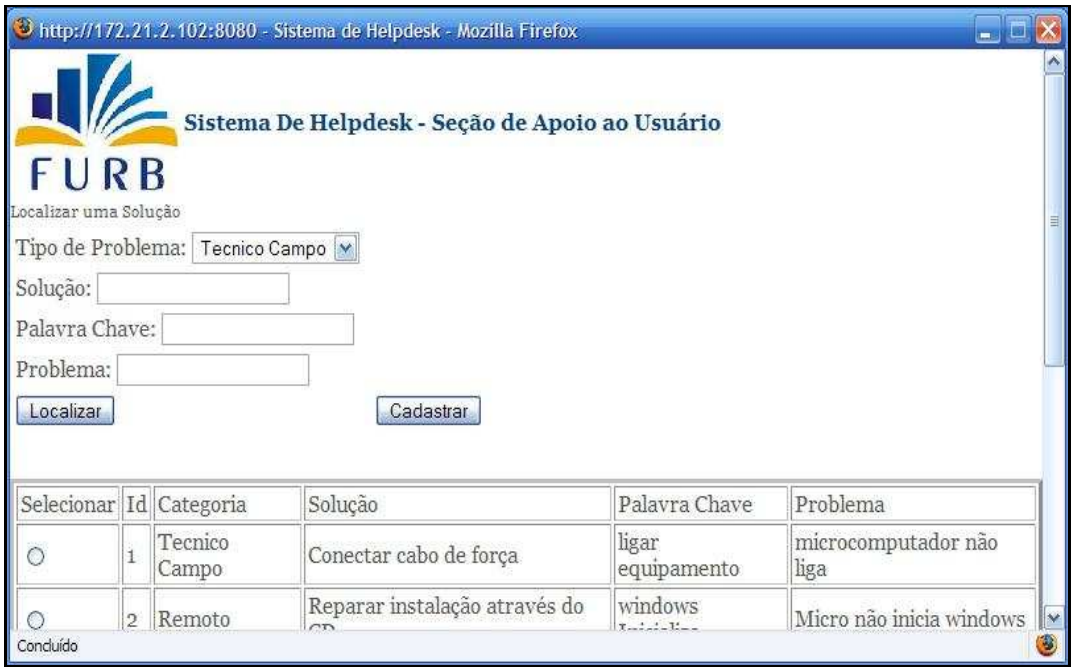


Figura 20 - Base de Conhecimentos

Depois que o chamado é criado, o sistema faz uma busca nos atendentes especializados

na área do problema do chamado, e se algum deles não estiver fazendo atendimento, ele automaticamente cria uma tarefa para este atendente, e encaminha um e-mail notificando-o que o chamado foi atribuído a ele conforme demonstrado na Figura 21 . No sistema também são pré-cadastrados prazos para cada tipo de atendimento. Se os prazos para o início do atendimento forem extrapolados, automaticamente o sistema manda um e-mail para o gerente, para que ele seja alertado na demora dos atendimentos. Isso é feito pela classe PrazoAction cada vez que uma tarefa vai para o nodo aguardando atendimento. A definição de processo cria um uma instância de PrazoAction com o tempo previamente cadastrado no sistema para o tipo de chamado. Para executar estas tarefas, o JBPM fornece um *servlet* chamado JobExecutorServlet, que cria uma *thread* que é responsável por executar o conteúdo desta classe.

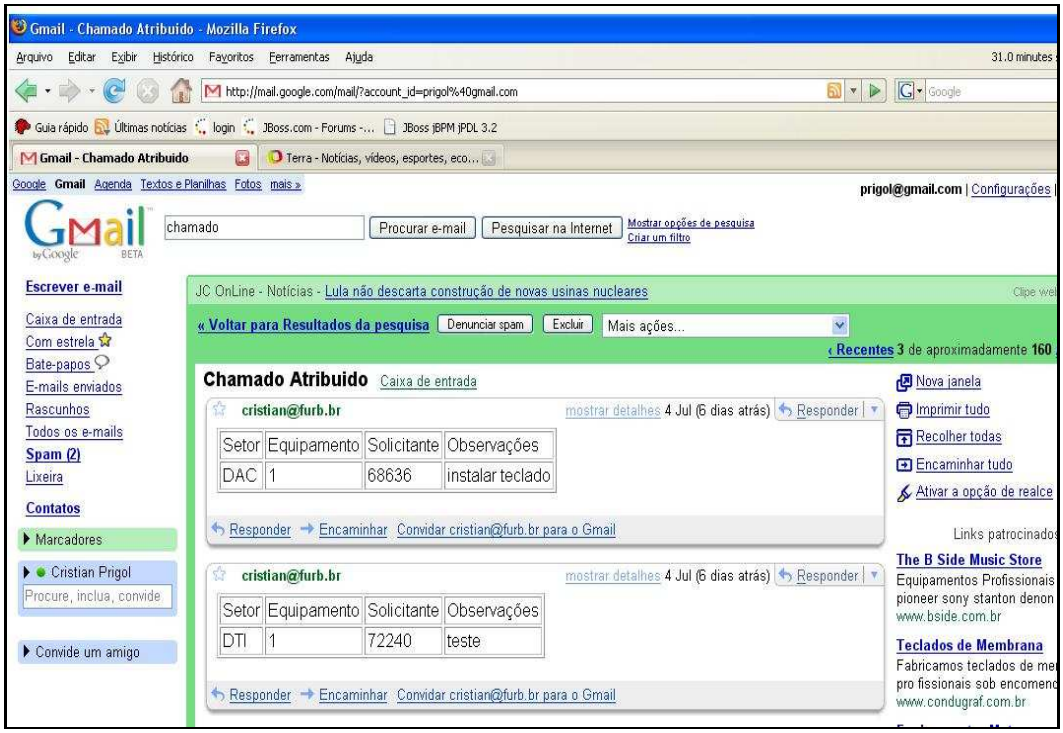


Figura 21- E-mail atribuindo chamado

A Figura 22 exemplifica a tela onde o atendente pode consultar as tarefas de atendimento que estão atribuídas a ele. Nesta tela ele pode finalizar a tarefa através do botão Finalizar.

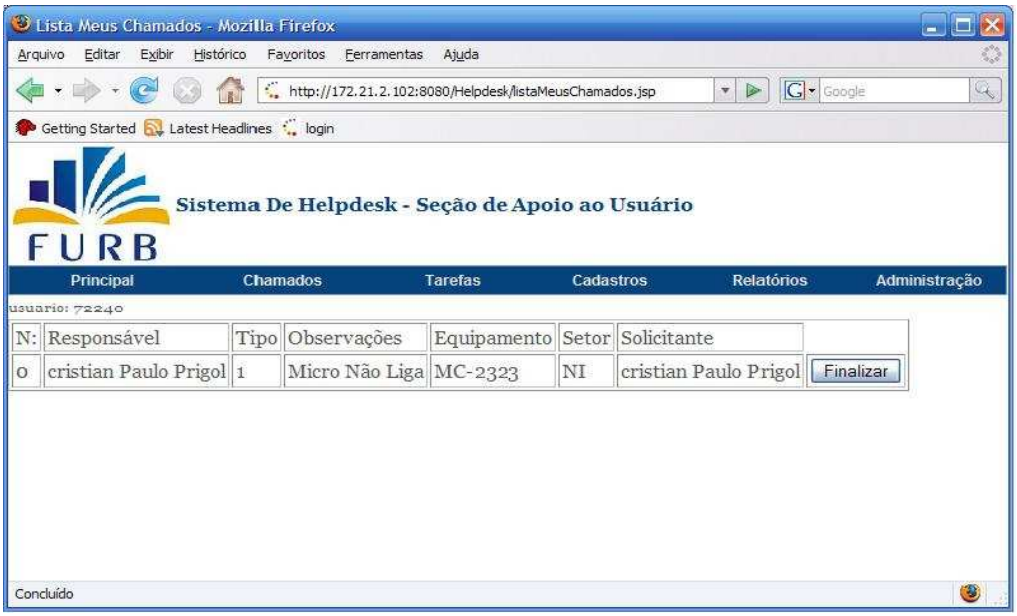


Figura 22 - Lista tarefas em andamento do atendente

Quando ele clica no botão Finalizar ele é redirecionado para a tela da Figura 23.

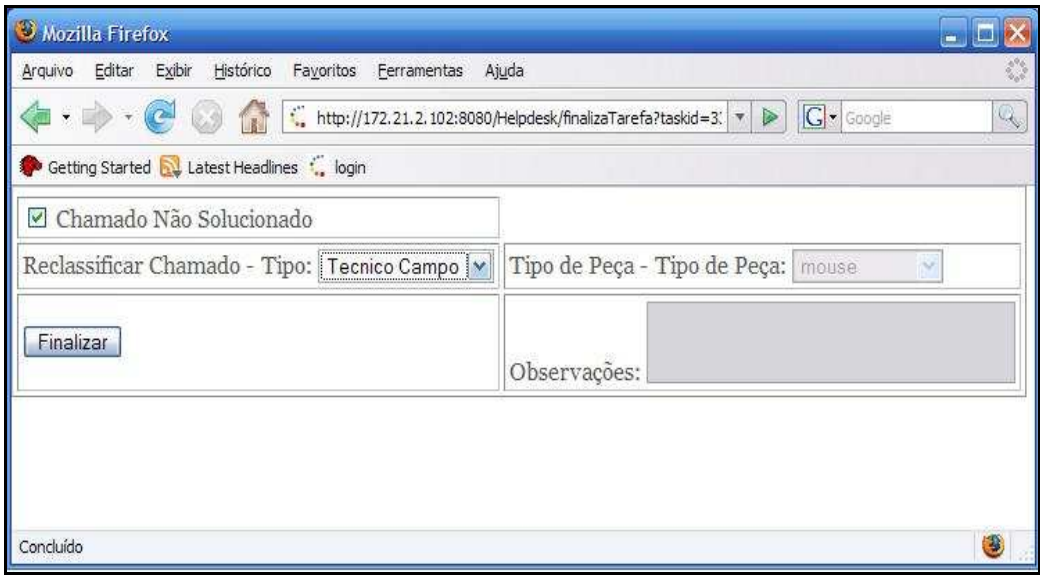


Figura 23 - Finalização de tarefa

Figura 24 - Solução da tarefa

Nesta tela o técnico deve selecionar uma solução para a tarefa, e se necessário adicionar uma observação para ela. Por padrão quando o técnico clica em Finalizar, o sistema fecha o chamado, e envia um e-mail para o solicitante pedindo a confirmação da resolução do problema. Ele pode selecionar também a opção Chamado Não Solucionado, com isso habilita a opção Reclassificar Chamado. Desta maneira o sistema criará uma outra tarefa na fila de atendimentos e a atribuirá para um outro técnico de uma outra área de atuação.

O sistema contempla também o cadastro de hardwares (scanner, impressora, microcomputador). Na Figura 25 um exemplo de tela para cadastro de microcomputador.

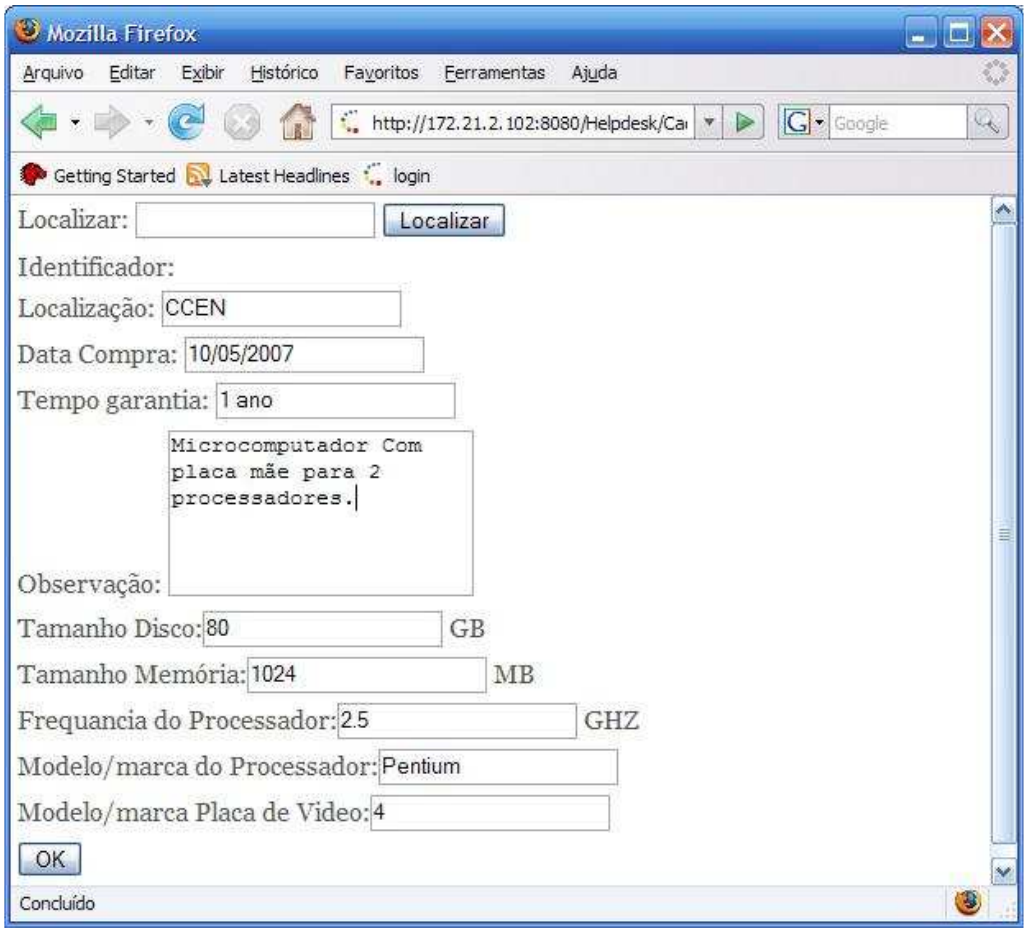


Figura 25- Cadastro de Microcomputador

Para fazer o gerenciamento dos chamados o sistema apresenta alguns relatórios.

Na Figura 26 é apresentado o gráfico de tarefas finalizadas agrupado por tecnico, este gráfico auxilia o gestor na gerencia de seus técnicos.

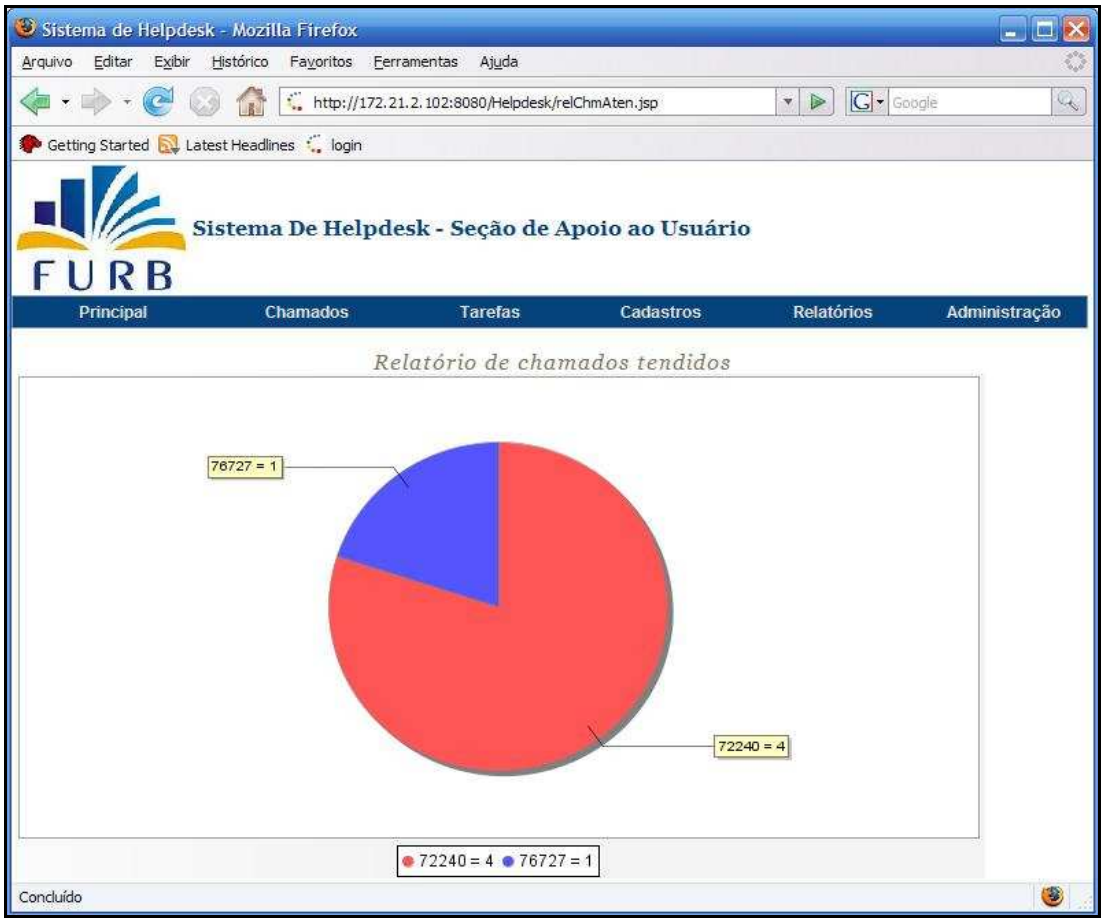


Figura 26 - Relatório das tarefas atendidas

Na Figura 27 é apresentado o relatório de chamados atendidos agrupados por hardware. Neste relatório é possível acompanhar a manutenção dos equipamentos da universidade.

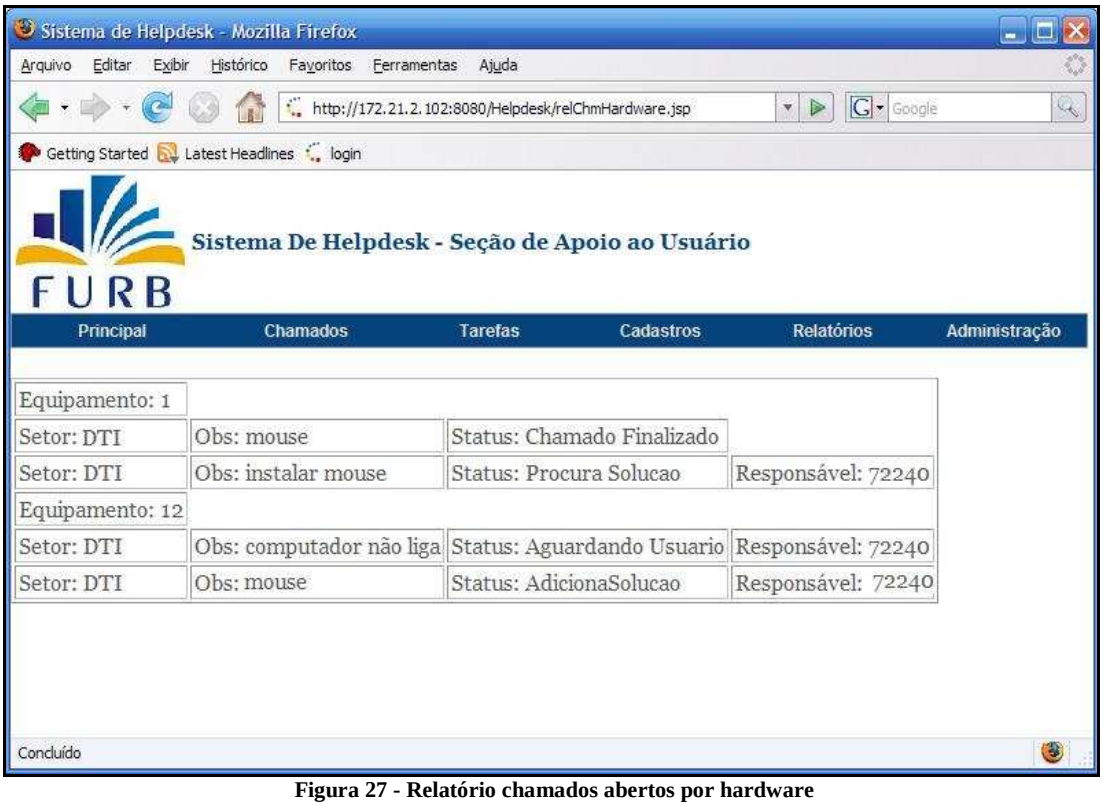


Figura 27 - Relatório chamados abertos por hardware

Na Figura 28 é apresentado o relatório dos chamados em aberto no momento cadastrados no sistema.

Figura 28 - Relatório de chamados em aberto

3.8 RESULTADOS E DISCUSSÃO

O uso de um sistema de *workflow* aplicado ao ambiente de *Help Desk* mostrou-se uma alternativa bastante eficiente para o gerenciamento de chamados.

Inicialmente na proposta deste TCC a intenção era desenvolver um sistema de *Help Desk* utilizando como *framework* de *workflow* a ferramenta WFMOPEN. Quando iniciado o desenvolvimento do trabalho esta ferramenta mostrou-se uma solução bastante complexa para a sua utilização em um TCC. Sua documentação não demonstrara de maneira clara quais eram os passos para utilizar seus recursos e desenvolver um sistema utilizando suas classes.

A documentação desta ferramenta exigia um bom nível de conhecimento em várias tecnologias Java, que não eram abordados na mesma.

Depois de um mês de trabalho, chegou-se a conclusão que o melhor caminho era utilizar outro *framework* de *workflow*. Depois de uma pesquisa com várias ferramentas, o *framework* escolhido foi o JBPM uma solução proposta pela JBOSS que se mostrou bastante fácil de usar e muito eficiente.

Na prática o sistema não chegou a ser implantado, pois durante o desenvolvimento do trabalho, a seção de apoio ao usuário recebeu a proposta de implantação de uma nova versão do sistema atualmente utilizado. Como a nova versão contempla a maioria das funcionalidades aqui propostas, a necessidade de implantação deste sistema implementado já não era mais válida.

No entanto o estudo proposto neste trabalho serviu de base para a modelagem dos processos utilizados pelo novo sistema implantado no APUS, agilizando assim a implantação da nova versão do sistema.

4 CONCLUSÕES

A necessidade do controle de chamados possibilitou o estudo do *framework* JBPM, suas tecnologias, dificuldades e facilidades e desenvolvimento de um sistema baseado nele. Com isso foi possível ampliar os conhecimentos no desenvolvimento de aplicações utilizando tecnologia Java J2EE. Este trabalho também possibilitou o uso e aprofundamento na ferramenta de banco de dados *Power Designer* e da ferramenta de modelagem *Enterprise Architect.*

A análise preliminar do fluxo de trabalho facilitou bastante o desenvolvimento do *workflow* do chamado, agilizando muito o desenvolvimento do sistema.

O *framework* JBPM mostrou-se uma ferramenta bastante eficiente e fácil de usar. Ele oferece um nível de produtividade muito bom. A utilização de suas classes é bastante simplificada e eficiente, mostrando-se uma solução bastante interessante para desenvolvimento de sistemas baseados em *workflow* .

Considera-se que o objetivo principal do trabalho foi atingido: Desenvolver um sistema baseado em *workflow* para a Seção de Apoio ao Usuário.

O sistema possibilita o gerenciamento do andamento dos chamados através dos vários relatórios, para que o gestor tenha um melhor controle sobre eles. Possibilita o acompanhamento do desempenho dos técnicos com a divisão dos chamados em tarefas. Com isto existe a possibilidade de saber o que cada técnico fez no chamado sendo possível ter um controle maior de nível gerencial sobre os chamados.

Os objetivos de aprendizagem do trabalho foram alcançados, pois o desenvolvimento do trabalho permitiu a aplicação dos conceitos estudados durante o curso de Sistemas de Informação, o aprofundamento sobre o estudo de sistemas de informações, o estudo de novas ferramentas e tecnologias de desenvolvimento.

4.1 EXTENSÕES

Como sugestão para trabalhos futuros, sugere-se que seja feito um estudo para adequar este sistema às metodologias e boas práticas de suporte propostas pelo *Information Technology Infrastructure Library* (ITIL).

O ITIL é um modelo de referência para gerenciamento de TI desenvolvido em meados dos anos 80 pelo pelo *Office of Government Commerce*, (OGC) da Inglaterra. Ele descreve os processos necessários para o gerenciamento de TI a fim de garantir o seu bom funcionamento.

Outra sugestão seria adequar o trabalho a norma ISO 10002-2004. Esta norma fornece orientações para implementação de um processo eficiente no atendimento de reclamações visando a satisfação do cliente.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAUJO, R. M.; BORGES, M.R. **Sistemas de workflow.** In: CONGRESSO DA SOCIEDADE BRASILEIRA DA COMPUTAÇÃO, 2001, Fortaleza. Disponível em: <http://chord.nce.ufrj.br/cursos/teesi/textos/apostilaJai2001div.pdf>. Acesso em 16 maio 2007.

BC ASSOCIATES (Org.). **Itil & ITSM world**. Disponível em: <http://www.itil-itsm-world.com/itil-4.htm>. Acesso em: 15 out. 2006.

CAVALARI, Gabriel O. T.; COSTA, Heitor A. X. Modelagem e desenvolvimento de um Sistema Help-Desk para a Prefeitura Municipal de Larvras-MG. **Revista Eletrônica de Sistemas de Informação**, Lavras, Dez. 2005. Disponível em: <www.inf.ufsc.br/resi/edicao06/Artigo52.pdf>. Acesso em: 05 set. 2006.

COÊLHO, A. V. S. ; FERNEDA, E. ; MARTINS, A. de S. ; BARROS, M. A. ; GORGÔNIO, F. L. E. . ***Help Desk*** **inteligente em gestão do conhecimento**: Um tratamento integrador de paradigmas. Inesc Em Revista, Unaí, v. 1, p. 46-51, 2003. Disponível em: < http://www.exercito.gov.br/06OMs/gabcmtex/PEG-EB/artigopdf/help.PDF>. Acesso em 15 abr. 2007.

CRUZ, Tadeu. **Workflow:** a tecnologia que vai revolucionar processos. São Paulo: Atlas, 1998.

FRANÇA, Montgomery Barroso. **Proposta de implementação de uma máquina de workflow para o projeto CEMT.** 2004. 97 f. Dissertação (Mestrado em ciências da computação) –Pós-graduação em computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://sabix.ufrgs.br/ALEPH/>. Acesso em: 10 set. 2006.

GOETZ, Carlos H**. Um ambiente para implementação de modelo de gerência de projetos utilizando técnicas de workflow**. 2003. 91 f. Trabalho de Conclusão de Curso (Faculdade de Informática) – Superior de Tecnologia em Informática, Universidade Luterana do Brasil, Canoas. Disponível em: <http://www.ulbra.tche.br/~tcc-canoas/2003-1/carloshenrique.pdf>. Acesso em: 12 abr. 2007.

HOLLINGSWORTH, David. **workflow management coalition the workflow reference model**. Hampshire: *workflow* Management Coalition, 1995. Disponível em <http://www.wfmc.org/standards/docs/tc003v11.pdf>. Acesso em: 15 out. 2006.

KROTH, Marcelo Lopes. **Mapeamento do modelo temporal usando o TF-ORM para a interface padrão para a definição de processos.** 1998. 1 f. Dissertação (Mestrado em ciências da computação) –Pós-graduação em computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana98/mkroth.html>. Acesso em: 14 out. 2006.

MAGALHÃES, Luizio; PINHEIRO Walfrido B. **Gerenciamento de serviços de TI na prática**: Uma abordagem com base na ITIL. São Paulo:Novatec, 2007.

OLIVEIRA, Alex. **Workflow:** Um diferencial competitivo. [Porto Alegre], [2001?]. Disponível em: <http://www.inf.ufrgs.br/mpi/disciplinas/groupware/alex.pdf>. Acesso em: 16 maio 2007.

RED HAT MIDDLEWARE. **JBOSS JBPM**, [S.I.], 2007. Disponível em: <http://www.jboss.com/products/jbpm>. Acesso em 20 mar. 2007.

SILVA, Jaime J. ***Help Desk* com sistema RBC para as gerências de aplicativos do Banco do Brasi**l. 2004. 45 f. Trabalho de Conclusão de Curso (Curso de Especialização e Desenvolvimento, segurança e Qualidade a Internet) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

TELECKEN, Tiago Lopes. **Um estudo sobre modelos conceituais para ferramentas de definição de processos de workflow .** 2004. 118 f. Dissertação (Mestrado em ciências da computação) – Pós-graduação em computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: < http://sabix.ufrgs.br/ALEPH/>. Acesso em: 03 set. 2006.

UNIPRESS, Software Inc. **O *Help Desk* baseado na web:** O que ele pode fazer por sua empresa e porque você precisa dele. NM Brasil, [São Paulo], n.15, fev, 2001. Disponível em: < http://www.mmbrasil.com.br/images/stories/solucoes/doc/footprints.pdf>. Acesso em: 25 jan. 2007.

USINORO, Carlos Hiroshi. **Tecnologia workflow :** O Impacto de Sua Utilização nos Processos de Negócio. Um Estudo de Casos Múltiplos. 2003. 178 f. Dissertação (Mestrado em administração) – curso de administração, Universidade de São Paulo, São Paulo, 2003. Disponível em: <http://www.teses.usp.br/teses/disponiveis/12/12139/tde-08122003-233842/>. Acesso em: 02 out. 2006.

WORKFLOW PATTERNS INITIATIVE. **Workflow Patterns.** [Eindhoven, Holanda], 2007 Disponível em: < http://www.workflowpatterns.com> Acesso em: 02 jun. 2007.

ZSCHORNACK, Fábio. **Evolução de esquemas de workflow representados em XML**. 2003. 91 f. Dissertação (Mestrado em ciências da computação) – Pós-graduação em computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <www.inf.ufrgs.br/~nina/Dissertacao/FabioZschornack.pdf>. Acesso em: 20 set. 2006.

APENDICE A – Utilizando o JBPM

O JBPM é um pacote fornecido pela JBOSS, que pode ser baixado em: (http://labs.jboss.com/jbossjbpm/downloads). O pacote JBPM Suíte traz a maior parte das ferramentas necessárias para desenvolver um sistema usando a plataforma. Neste pacote estão inclusos:

a) Os scripts para criar a base de dados, personalizados para os bancos de dados mais populares do mercado;
b) O arquivo do tipo .war para o Servidor JBOSS com o JBPM Console que é um sistema *web* para *workflow* , onde basta adicionar o arquivo do tipo xml com a definição de processo;
c) Um arquivo do tipo .bat para configuração automática do *plugin* de desenvolvimento no Eclipse;
d) Uma pasta com documentação sobre criação de processos e utilização do JBPM;
e) Uma pasta com exemplos de utilização das classes do *framework*;
f) Uma pasta com as bibliotecas (classes) necessárias para utilização no desenvolvimento das aplicações;
g) O servidor de aplicações JBOSS já configurado para o JBPM.

Para criar um novo sistema deve ser criado um novo projeto de sistema *web* no Eclipse com as opções padrão. Depois disso deve-se adicionar os arquivos .jar com as classes utilitárias do JBPM.

Depois do projeto criado adiciona-se o arquivo log4j.properties. Neste arquivo ficam as configurações para gravação de log do sistema. O JBPM utiliza o *plugin* log4j da Apache para gravar os seus logs.

O próximo arquivo a ser configurado é o de configuração do Hibernate. Este arquivo chamado hibernate.cfg.xml armazena as configurações do Hibernate, como por exemplo o banco de dados onde ele deve conectar-se, o tipo de bando de dados, o usuário e a senha para fazer a conexão, as classes que deve ser armazenadas no banco de dados. Este arquivo deve ser adicionado à raiz do projeto no Eclipse. Na **Erro! Fonte de referência não encontrada.** é apresentado um trecho do arquivo hibernate.cfg.xml utilizado no projeto deste TCC.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-configuration (View Source for full doctype...)>
<hibernate-configuration>
<session-factory name="java:comp/hibernate/SessionFactory">
    <!-- hibernate dialect -->
    <property name="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <!-- JDBC connection properties (begin) -->
    <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:helpdesk</property
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger</property>
    <!-- JDBC connection properties (end) -->
    <property
      name="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider</proper
  - <!--
       DataSource properties (begin) ===
          <property name="hibernate.connection.datasource">java:/JbpmDS</property>
          ==== DataSource properties (end)
    -->
    <mapping resource="org/jbpm/graph/action/Script.hbm.xml" />
    <!-- identity mappings (begin) -->
    <mapping resource="org/jbpm/identity/User.hbm.xml" />
    <mapping resource="org/jbpm/identity/Group.hbm.xml" />
    <mapping resource="org/jbpm/identity/Membership.hbm.xml" />
    <!-- identity mappings (end) -->
    <!-- following mapping files have a dependendy on -->
    <!-- the JCR API -->
  - <!--
       jcr mappings (begin) ===
          <mapping resource="org/jbpm/context/exe/variableinstance/JcrNodeInstance.hbm.xml"/>
          ==== jcr mappings (end)
    -->
    <!-- ###################### -->
    <!-- # jbpm mapping files # -->
    <!-- ###################### -->
    <!-- hql queries and type defs -->
    <mapping resource="org/jbpm/db/hibernate.queries.hbm.xml" />
    <!-- graph.def mapping files -->
    <mapping resource="org/jbpm/graph/def/ProcessDefinition.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/Node.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/Transition.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/Event.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/Action.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/SuperState.hbm.xml" />
    <mapping resource="org/jbpm/graph/def/ExceptionHandler.hbm.xml" />
    <mapping resource="org/jbpm/instantiation/Delegation.hbm.xml" />
    <!-- graph.node mapping files -->
    <mapping resource="org/jbpm/graph/node/StartState.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/EndState.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/ProcessState.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/Decision.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/Fork.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/Join.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/State.hbm.xml" />
    <mapping resource="org/jbpm/graph/node/TaskNode.hbm.xml" />
    <!-- context.def mapping files -->
    <mapping resource="org/jbpm/context/def/ContextDefinition.hbm.xml" />
    <mapping resource="org/jbpm/context/def/VariableAccess.hbm.xml" />
    <!-- taskmgmt.def mapping files -->
    <mapping resource="org/jbpm/taskmgmt/def/TaskMgmtDefinition.hbm.xml" />
    <mapping resource="org/jbpm/taskmgmt/def/Swimlane.hbm.xml" />
    <mapping resource="org/jbpm/taskmgmt/def/Task.hbm.xml" />
    <mapping resource="org/jbpm/taskmgmt/def/TaskController.hbm.xml" />
    <!-- module.def mapping files -->
    <mapping resource="org/jbpm/module/def/ModuleDefinition.hbm.xml" />
    <!-- bytes mapping files -->
    <mapping resource="org/jbpm/bytes/ByteArray.hbm.xml" />
    <!-- file.def mapping files -->
    <mapping resource="org/jbpm/file/def/FileDefinition.hbm.xml" />
    <!-- scheduler.def mapping files -->
    <mapping resource="org/jbpm/scheduler/def/CreateTimerAction.hbm.xml" />
    <mapping resource="org/jbpm/scheduler/def/CancelTimerAction.hbm.xml" />
    <!-- graph.exe mapping files -->
```

Figura 29 - Arquivo de Configurações do Hibernate

O próximo passo para criação do sistema é criar o arquivo de definição de processo. Este arquivo será responsável por mapear o processo de negócio para o Jbpm. O *plugin* para o Eclipse disponível juntamente com o JBPM fornece o suporte a criação deste arquivo. Ele fornece uma interface gráfica para a modelagem dos componentes do processo. Quando um novo arquivo é criado, automaticamente o eclipse cria um arquivo do tipo XML para armazenar a definição de processo e outro arquivo do tipo JPG para visualização da definição do processo. A seguir na Figura 10 apresenta-se um trecho do arquivo de definição de processo utilizado no projeto deste TCC. A segunda linha do arquivo define o nome do processo, as demais linhas definem os nodos, as transições e regras do processo.

```
<?xml version="1.0" ?>
- <process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="Chamado">
    <!-- SWIMLANES (= process roles) -->
    <swimlane name="Atendente" />
  - <swimlane name="Tecnico">
      <assignment actor-id="#{ contextInstance.variables['atendente']}" />
    </swimlane>
  - <swimlane name="Sistema">
      <assignment actor-id="Sistema" />
    </swimlane>
    <!-- NODES -->
  - <start-state name="Cria novo chamado">
      <task name="Cria novo Chamado" swimlane="Atendente" />
      <transition name="" to="Procura Solucao" />
    </start-state>
    <end-state name="Fecha Chamado" />
  - <task-node name="Atende">
      <task name="AtendeChamado" swimlane="Tecnico" />
      <transition name="Solucionou" to="Confirmaçao do Usuario" />
      <transition name="Nao Solucionou" to="Problema de Peças?" />
    </task-node>
  - <task-node name="Procura Solucao">
      <task name="Procura Solucao" swimlane="Atendente" />
      <transition name="Sim" to="Fecha Chamado" />
      <transition name="Nao" to="Atribui Chamado?" />
    </task-node>
  - <task-node name="Aguardando">
    - <task name="Aguardando" swimlane="Sistema">
        <assignment actor-id="Sistema" />
      </task>
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.CreateTimeActionHandler" />
      </event>
      <transition name="Atender" to="Atende" />
    </task-node>
  - <task-node name="Confirmaçao do Usuario">
    - <task name="Aguardando Usuario">
        <assignment actor-id="#{contextInstance.variables
          ['idSolicitante']}" />
      </task>
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.EnviaEmaiActonHandler" />
      </event>
      <transition name="Resolvido" to="Fecha Chamado" />
      <transition name="Nao Resolvido" to="Aguardando" />
    </task-node>
  - <decision name="Atribui Chamado?">
    - <event type="node-enter">
        <action name="action1"
          class="helpdesk.util.AtribuiResponsabilidadeHandler" />
      </event>
    - <transition name="Nao Atribuido" to="Aguardando">
        <condition>#{contextInstance.variables['atendente'] == 'Sistema'}
          </condition>
```

Figura 30 - Arquivo de Definição do Processo Chamado

Depois de definido o processo é preciso armazená-lo no banco de dados. Para isso são utilizadas as seguintes linhas na aplicação, demonstradas na Figura 31.

```
try {
        JbpmConfiguration jbpmConfiguration = JbpmConfiguration.getInstance();
        JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
        DbPersistenceServiceFactory dbPersistenceServiceFactory = (DbPersistenceServiceFactory)
                jbpmConfiguration.getServiceFactory(Services.SERVICENAME_PERSISTENCE);
        dbPersistenceServiceFactory.createSchema();
        ProcessDefinition processDefinition = ProcessDefinition.parseXmlResource("processdefinition.xml");
        jbpmContext.deployProcessDefinition(processDefinition);

    }catch(Exception ex){
        ex.printStackTrace();
    } finally {
            jbpmContext.close();
        }
```

Figura 31 - Armazenar Definição de Processo

O procedimento descrito na Figura 31 é somente executado depois do processo modelado, ou quando for necessária alguma modificação na sua definição. A linha: dePersistenceServiceFactory.createSchema(); apaga os registros antigos do banco de dados. Enquanto a linha jbpmContext.deployProcessDefinition(processDefinition) armazena no banco de dados a nova definição de processo descrita no arquivo “processdeinition.xml”. Agora para criar uma nova instância do processo será somente necessário, passar como parâmetro o nome do processo.

A classe de exemplo da Figura 11 é responsável por criar um chamado no sistema, ou seja, ela cria uma nova instância da definição de processo armazenada no banco de dados.

```
package helpdesk.web;

import java.io.IOException;

/**
 * @web.servlet name="CriaChamado" display-name="Criar Um Chamado"
 *              description="Cria Chamado"
 *
 * @web.servlet-mapping url-pattern="/CriaChamado"
 *
 */

public class CriaChamadoServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
        response.setContentType("text/html");
        JbpmConfiguration jbpmConfiguration = JbpmConfiguration.getInstance();
        JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
        HttpSession session = request.getSession();
        /*
         * seta o ator que está abrindo o chamado atraves de um atributo de
         * sessao
         */
        jbpmContext.setActorId((String) session.getAttribute("idusuario"));
        GraphSession gs = jbpmContext.getGraphSession();
        /*
         * cria uma instancia de ProcessDefinition com a definição de processo
         * "chamado"
         */
        ProcessDefinition pd = gs.findLatestProcessDefinition("Chamado");
        ProcessInstance pi = new ProcessInstance(pd);

        TaskInstance taskInstance = pi.getTaskMgmtInstance()
                .createStartTaskInstance();
        /*
         * Cria um map com as variaveis que vão fazer parte do processo As
         * variaveis estão na sessão do usuário
         */
        Map taskVariables = new HashMap();
        taskVariables.put("idSolicitante", request.getParameter("idPessoa"));
        taskVariables.put("idSetor", request.getParameter("idSetor"));
        taskVariables.put("idEquipamento", request
                .getParameter("idEquipamento"));
        taskVariables.put("idChtipo", request.getParameter("idChtipo"));
        taskVariables.put("Obs", request.getParameter("Obs"));
        taskVariables.put("atendente", request.getParameter(""));
        taskInstance.addVariables(taskVariables);
        /*
         * O metodo close do jbpmContext presiste no banco de dados as
         * informações
         */
        taskInstance.end();
        jbpmContext.save(taskInstance);
        jbpmContext.close();
        /*
         * Redireciona para a página para escolher a solução do problema
         */
        long idProcessoTemp = pi.getId();
        Long idProcesso = new Long(idProcessoTemp);
        session.setAttribute("idProcesso", idProcesso);
        response.sendRedirect("escolheSolucao.jsp");
    }

}
```

Figura 32 - Classe CriaChamadoServlet

Esta classe é estendida de HttpServlet. O método post do *servlet* é responsável por criar no novo chamado e no final redirecionar para o *servelt* que seleciona a solução da tarefa.

As tarefas criadas no sistema tem um atributo chamado *ActorId*. Este atributo é responsável por armazenar o ator responsável pela execução da tarefa. Na classe da Figura 11 este valor é atribuído pelo método jbpmContext.setActorId((String) session.getAttribute("idusuario")) . No exemplo apresentado, valor é armazenado no objeto jbpmContext e este automaticamente atribui o *ActorId* das tarefas criadas neste contexto.

Neste exemplo também é criado um objeto do tipo *GraphSession* chamado gs. Ele é responsável por executar operações relacionadas a banco de dados. Na classe demonstrada ele é responsável por buscar no banco de dados a definição de processo do chamado através do método *findLatestProcessDefinition*(“Chamado”).

O JBPM tem a opção também de adicionar variáveis à instância do processo criado. Estas variáveis podem armazenar qualquer tipo de objeto que o processo necessitar. As variáveis armazenadas devem ser guardadas em um objeto do tipo *Map.*

Na classe da Figura 11 foi criado um *Map* das variáveis que serão inseridas através do código: *Map taskvariables = new HashMap().* Nas linhas subseqüentes desta parte do código são adicionadas as variáveis que vão fazer parte do processo.

Chegando ao final da classe há a parte do código responsável por finalizar a tarefa de inicialização do processo através do código: *taskInstance.end();* Depois disso o código jbpmContext.close() persiste as informações no banco de dados através do *framework Hibernate*.