UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

SISTEMA DE VIGILÂNCIA REMOTA ATRAVÉS DE UMA INTERFACE WEB/WAP BASEADO EM VÍDEO GERADO POR WEBCAM

Autor: _____
Victor Pantoja

Orientador: _____
Prof. Antônio Cláudio Gómez de Souza, M.Sc.

Examinador: _____
Prof. Aloysio de Castro Pinto Pedroza, D.Sc

Examinador: _____
Prof. Marcelo Luiz Drumond Lanza, M.Sc.

DEL

Novembro de 2006

Dedicatória

Dedico este trabalho de conclusão ao meu pai e a minha mãe (*in memorian*), pois tudo que eu sou eu devo a eles. E a minha namorada, por ter estado sempre ao meu lado, me incentivando, sofrendo e comemorando comigo. Esse projeto final também é uma conquista sua.

Agradecimento

Agradeço primeiramente a pessoa que mais me incentivou e me compreendeu nesses últimos meses, minha namorada.

Aos professores e funcionários do DEL, pelo trabalho de ensino realizado.

Ao GTA, em especial ao Danilo, pelo suporte e infra-estrutura fornecidos para a apresentação final.

Ao Felipe Grael, por ter emprestado o chip da TIM e me cedido alguns créditos para a apresentação final.

Aos meus amigos de longa data que sempre me incentivaram e compreenderam a necessidade de me ausentar em inúmeras ocasiões.

Ao meu amigo e colega de trabalho Ricardo, pelas inúmeras dicas e tutoriais fornecidos. E a todos da minha equipe, por terem me dado o suporte técnico necessário e a motivação para terminar este projeto.

Resumo

No cenário atual, é muito importante considerar a segurança de computadores que participam de uma rede local (intranet) e dos dados armazenados por esses computadores. Tais máquinas podem estar trabalhando como servidores de uma empresa e, sendo assim, há a necessidade de manter esses computadores não apenas em funcionamento, mas também protegê-los de certos danos. Empresas de grande porte podem adquirir sistemas de segurança extremamente sofisticados e de alto custo de aquisição e manutenção. Este trabalho tem como objetivo introduzir os conceitos, protocolos e mecanismos necessários a uma solução de segurança para empresas com orçamento restrito. A solução apresentada envolve *streaming* de vídeo e transmissão de sinal de vídeo gerado por uma *webcam* em tempo real, além de um repositório de vídeos.

Palavras-chaves

Webcam, *streaming*, segurança, programação, Java.

Índice

Índice de Figuras

Índice de Tabelas

Siglas

GPRS	*General Packet Radio Service*
HTTP	*Hyper Text Transfer Protocol*
IDE	*Integrated Development Environment*
J2SE	*Java 2 Platform Standard Edition*
JSP	*Java Servlet Pages*
RTP	*Real-Time Transport Protocol*
RTCP	*RTP Control Protocol*
RTSP	*Real-Time Streaming Protocol*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*
JTA	*Java Transaction API*
JDBC	*Java Database Connectivity*
JVM	*Java Virtual Machine*
RAM	*Read Access Only*
ERS	Especificações de Requisitos de *Software*

1 Introdução

A crescente demanda por *software*s que possibilitem a monitoração e o controle de tudo o que trafega através de uma rede de computadores é uma resposta às crescentes tentativas de invasão feita por *hackers*. Nascido na década de 90, o fenômeno hacker multiplicou-se de forma surpreendente no Brasil.

"A cada dia surgem formas diferentes de driblar a proteção. De posse do mapa com alguns pontos fracos da rede, os *hackers* passam à ação. Primeiramente, identificam os computadores conectados naquele momento que estão vulneráveis. A máquina de um navegante desavisado pode servir para atacar outros sites, sem que o incauto proprietário se dê conta. O próximo passo é achar grandes alvos, como páginas de empresas e governos" [1].

Programas disponíveis gratuitamente na própria Internet que realizam automaticamente uma "ronda" em milhões de páginas na Internet e acabam detectando alguma falha tem facilitado em muito a invasão de computadores. Dessa forma, é possível conseguir acesso externo a um computador vulnerável.

Mas e se algum intruso conseguir acesso não-autorizado a uma rede simplesmente sentando em um terminal e digitando um login e uma senha válida roubados de um usuário autorizado? É neste contexto que se encaixa este projeto final.

A solução apresentada não visa combater ataques de *hackers* ou encontrar *backdoors* em algum programa instalado em qualquer máquina da rede. Ela visa, sobretudo, a evitar e coibir o mau uso dos equipamentos, o que pode causar danos a esses equipamentos e, principalmente, aos dados que ele guarda ou simplesmente processa.

O sistema se baseia em um computador central dotado com uma *webcam*, ativa 24h por dia, gerando arquivos de vídeo com uma hora de duração, e de *workstations*, igualmente dotadas de *webcam*, mas cujas imagens são acessadas em tempo real. Esses acessos podem ser feitos de qualquer computador, através do módulo de gerenciamento central. Para aumentar a segurança do sistema, foi desenvolvido um sistema de cadastro de usuários de computadores. Existe um usuário administrador que pode definir que um usuário terá permissão sobre uma ou várias máquinas.

Para a transmissão em tempo real será utilizado *streaming* de vídeo. Para tal, é necessário um servidor de *streaming*. No caso deste projeto, foi escolhido o *Darwin Streaming Server* [2].

Em resumo, a solução proposta consiste em um sistema de gerenciamento remoto, que pode ser acessado tanto por celular quanto por um navegador web, ao qual se conectam outros módulos responsáveis pela transmissão e gravação dos vídeos de segurança. Esses módulos estão em cada um dos computadores da rede que possuam uma *webcam* instalada e que servirá como fonte de dados.

A arquitetura do sistema é mostrada na figura a seguir (será detalhada na seção 4 Detalhamento da Solução):

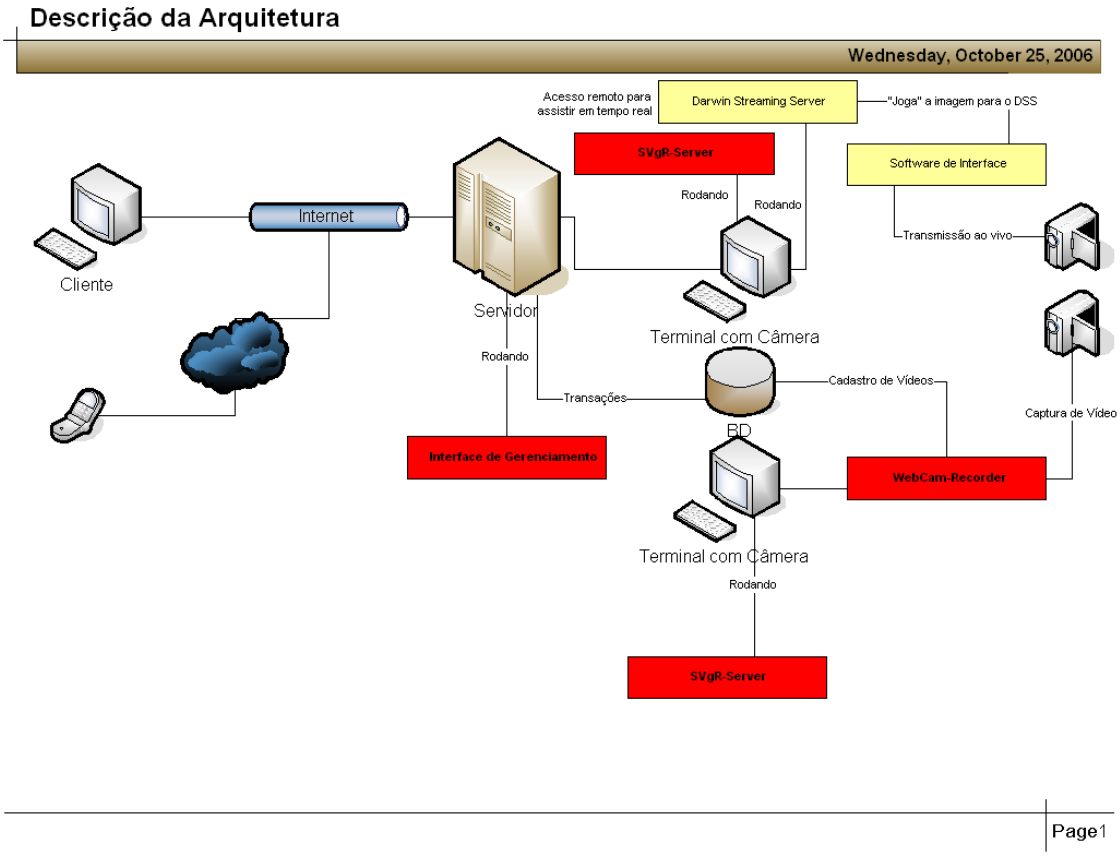

Figura 1: Arquitetura da Solução

É importante ressaltar, como premissa do projeto, que a comunicação com o servidor de *streaming* de vídeo e o transporte do conteúdo, em si, deve utilizar protocolos padronizados e estabelecidos no mercado. Os vídeos utilizados também devem ser codificados através de codificadores padronizados.

Outra premissa do projeto é a orientação para o uso de *software* livre. Tudo o que foi desenvolvido, as ferramentas utilizadas e até mesmo a aplicação servidora utilizam ao máximo *software* livre. Somente quando não houve alternativas foi usado, no pior caso, um *software* apenas gratuito. Sendo assim, o custo da solução em termos de *software* é zero.

Para apresentar o que foi feito, este documento está dividido em capítulos da seguinte forma: no capítulo 2, é dada uma visão geral da solução, mostrando os principais componentes do sistema, explicando sucintamente o papel de cada um, e descrevendo todo o processo de desenvolvimento de cada um dos módulos que compõem esse projeto. Ainda neste capítulo, serão introduzidas as principais tecnologias usadas, sempre se tentando fazer um paralelo com o projeto em si, ou seja, buscando sempre exemplificar o que está sendo dito utilizando códigos do projeto.

No capítulo 3 são explicados os conceitos de base para o entendimento de um servidor de *streaming* de vídeo e para o desenvolvimento de um aplicativo utilizando J2EE e algumas tecnologias inerentes a essa filosofia de programação Java, como o Hibernate e o *Struts*.

No capítulo 4 são apresentadas com mais detalhes as soluções da transmissão e aquisição de vídeo. Neste capítulo, será feito um detalhamento de cada um dos aplicativos que fazem parte deste projeto: o módulo de gerenciamento, o SVgR-*Server* e o Webcam-Recorder.

No capítulo 5, são apresentadas as conclusões deste trabalho e são feitas algumas propostas de trabalhos futuros em cima deste projeto final.

Há ainda um apêndice, que compõe o anexo deste projeto. Ele é composto pelas Especificações de Requisitos de *Software* (ERS), baseada na norma ANSI/IEEE 830 simplificada [3]. Decidiu-se desenvolver essa análise utilizando a análise essencial, pois a documentação do projeto final anterior a este apêndice já apresentou as funcionalidades do *software*. Assim, a visão *bottom-up* torna-se mais prática. A ERS está dividida em 3 seções e representa um maior detalhamento da solução proposta. Na primeira seção é feita uma breve introdução sobre o *software* desenvolvido, sendo apresentada sua finalidade e escopo. Na seção 2 é feita a descrição geral do produto, sendo uma complementação às descrições feitas nos capítulos da documentação anteriores ao apêndice. Na última seção, são detalhados os requisitos de *software*, mostrando-se o diagrama de entidades e relacionamentos (e seu dicionário de dados) e a análise através de Diagramas de Fluxo de Dados.

2 Desenvolvimento

Neste capítulo será apresentado todo o trabalho realizado durante o projeto. Será apresentada a solução de segurança dividida em várias fases. Em cada uma das fases serão apresentadas as questões relevantes que envolveram pesquisa, decisão ou desenvolvimento durante o andamento do projeto.

Primeiramente, como ambiente de desenvolvimento foi escolhida a ferramenta Eclipse. Trata-se de uma IDE *(Integrated Development Environmen*t - Ambiente Integrado de Desenvolvimento) de código aberto e de maior popularidade para o desenvolvimento de aplicações Java. Foram adicionados dois *plugins* ao Eclipse:

- *plugin* do apache-tomcat, que disponibiliza meios de inicializar e parar o tomcat pelo Eclipse além de possibilitar o modo *debug*;
- *plugin* WTP, para gerar templates de JSP's, ou seja, esse plugin disponibiliza uma funcionalidade que gera jsp's padronizados.

O desenvolvimento inicial se deu em cima do módulo principal: o módulo de gerenciamento, cujo *front-end* foi desenvolvido totalmente usando JSP's. Como *back-end*, desenvolveram-se várias classes JAVA, seguindo o conceito de arquitetura MVC, que será explicado mais adiante. Seguindo este conceito, foi necessária a utilização do *Framework Struts* (ver 3.4.2 Struts e Actions), sobre o qual foi necessário um estudo profundo sobre aplicabilidade e configurações específicas.

Assim, os primeiros passos deste projeto foram no sentido de se criar as configurações iniciais de modo que ele pudesse funcionar no *application Server* escolhido: o apache-tomcat. Essa configuração é feita através dos seguintes arquivos xml:

- context.xml: define todo o contexto no qual o projeto, incluindo o path da aplicação relativo ao tomcat (/projetofinal)
- web.xml: faz o mapeamento do arquivo de configuração do *Struts*, das taglibs utilizadas e define algumas características como extensão das *Actions* (.ssp, para o caso deste projeto) e o arquivo de "boas-vindas" (index.jsp)
- *Struts*-config.xml: faz o mapeamento de todas as *Actions* envolvidas no projeto. Ele mapeia o path relativo (`/svgr/userAutenticationAction`, por exemplo) em uma classe (`br.ufrj.projetofinal.web.UserAutenticationAction`, para o exemplo)

Na verdade, o exemplo de mapeamento dado acima foi justamente o primeiro realizado, feito inclusive como base para testes. A referida *action* é chamada pela página inicial do projeto, a index.jsp, acessada através da url *http://<host>:8080/projetofinal*.

Um trecho do arquivo *Struts*-config.xml é mostrado a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.1//EN" "http://jakarta.apache.org/Struts/dtds/Struts-config_1_1.dtd">

<struts-config>
<!--  Action Mapping Definitions-->
<action-mappings>
        <action path="/svgr/userAutenticationAction"

        type="br.ufrj.projetofinal.web.UserAutenticationAction"
        scope="request"
        input="index.jsp"
        validate="true">
        <forward name="erro" path="/svgr/erro.ssp"/>
        <forward name="showMenu"  path="/svgr/frame.ssp"/>
        <forward name="loginAccept"  path="/svgr/loginAccept.ssp"/>
        <forward name="showMobileMenu" path="/svgr/mobileMenu.ssp"/>
        <forward name="mobileResult" path="/svgr/mobileResult.ssp"/>
        </action>

</action-mappings>
</struts-config>
```

A estrutura desse arquivo será explicada na seção 3.4.2 Struts e Actions.

Também foi necessária a configuração da conexão com um banco de dados. Escolheu-se trabalhar com o hibernate (3.4.4 Hibernate), pois ele possibilita que se possa trabalhar com inúmeros bancos de dados. A configuração do hibernate se mostrou bastante simples, de um modo geral. Já a configuração do *Struts* foi um pouco mais complicada, sobretudo pelas diversas maneiras de se fazê-lo. Foi necessário basear-se em uma configuração de um projeto já implantado, pois só assim foi possível realmente entender seu funcionamento.

A configuração do arquivo de log também foi necessária. O mecanismo escolhido para tal foi o uso do log4j. O Log4j é um projeto *open source* baseado no trabalho de diversos autores [4]. Ele permite ao desenvolvedor controlar, de maneira flexível, cada saída de log. Com ele, pode-se ativar o log em tempo de execução sem modificar os binários da aplicação, sendo necessário apenas editar um arquivo de configuração.

Para poder usar o log4j, criou-se um arquivo chamado log4j.properties, como mostrado abaixo, além de ter-se adicionado o jar referente ao log4j ao projeto.

```
#### Log to a file

log4j.rootCategory=INFO, R
log4j.debug=false
#log4j.logger=DEBUG

#### Appender writes to a file
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.layout.ConversionPattern='.'yyyy-MM-dd
log4j.appender.R.File=c:/opt/projetofinal.log

log4j.appender.LF5=org.apache.log4j.lf5.LF5Appender
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%-5p  %d{yyyy/MM/dd  HH:mm:ss}
%r - %m%n
```

Após os testes, e a comprovação de que as configurações estavam corretas e os mapeamentos bem feitos, foram criados os primeiros pacotes, seguindo o MVC. Assim, criaram-se classes BO (*Business Object*), VO (*Value Object*) e DAO (*Data Access Object*) para usuário e computador. Feito isso, foi possível desenvolver todo o layout das páginas iniciais, nos quais os layouts seguintes se baseariam.

Na verdade, os DAO's exibiam apenas mensagens de debug, pois ainda não havia a conectividade com um banco de dados. Assim, todas as funcionalidades não faziam de fato o que propunham a fazer. O que estava sendo testado inicialmente era apenas a passagem de parâmetros e o fluxo das *Actions*.

O passo seguinte foi a integração com o BD. Decidiu-se por fazer um sistema independente de um banco de dados específico. Assim, o Hibernate foi a ferramenta que permitiu este feito. Sua configuração também é baseada em arquivos XML. Basicamente, esses arquivos mapeiam as classes VO em uma tabela do banco de dados. Dessa forma, a classe UserVO, por exemplo, foi mapeada na tabela *user*. Esse mapeamento significa que todos os atributos do referido objeto possuem um mapeamento no BD (cada atributo representa uma coluna da tabela). A seção 3.4.4 Hibernate trará mais detalhes sobre essa tecnologia.

Com as classes e os atributos devidamente modelados e mapeados, foi possível implementar os objetos DAO, que são os responsáveis por se conectar ao banco de dados e realizar as transações.

Pelo que foi descrito, foi possível desenvolver quase todo o módulo de administração isoladamente. A conectividade do sistema de gerenciamento com os computadores da rede foi o passo seguinte.

O módulo de comunicação cliente-servidor foi implementado utilizando *sockets* (que será explicado em capítulo posterior), inicialmente de uma maneira isolada, ou seja, desenvolveu-se uma aplicação isolada que serviria como cliente para simular a interface de administração e uma aplicação servidora, responsável por executar as funcionalidades previstas sobre cada máquina.

Ambas as aplicações foram desenvolvidas com o uso do Java Swing, componente para desenvolvimento de aplicativos com interface gráfica. Inicialmente, ambos funcionaram como um comunicador instantâneo: a mensagem era digitada no cliente, por exemplo, e chegava ao servidor. A seguir, digitava-se uma mensagem no servidor e ela chegava ao cliente. O código fonte do cliente foi incorporado ao módulo de gerenciamento (pacote br.ufrj.projetofinal.client) e o *server* deu origem a um novo projeto chamado SVgR – *Server*.

A partir disso, todo o desenvolvimento do módulo de administração e do SVgR-*Server* se deu de forma conjunta. Enviava-se um determinado pacote, contendo as informações necessárias e observava-se o comportamento das máquinas, sempre mostrando na tela do SVgR-*Server* os pacotes que estavam sendo recebidos e enviados (ver Figura 64).

Vale dizer que o método responsável por ligar o computador é independente do SVgR-*Server*, pois na verdade ele apenas envia um pacote broadcast, segundo o protocolo WOL (ver seção 3.2). Assim, ele se constituiu em uma outra fase do projeto para a qual foi desenvolvido um servidor UDP para confirmar que a transmissão de pacotes está sendo feita corretamente. No entanto, o *client-side* foi também incorporado ao pacote br.ufrj.projetofinal.client.

O módulo de vídeo foi o que representou a maior dificuldade de ser implementado, embora possua bem menos linhas de código.

A distribuição de vídeo via *streaming* necessita de um servidor de *streaming* de vídeo. Como já mencionado, o servidor de *streaming* de vídeo escolhido foi o *Darwin Streaming Server* (DSS) da empresa *Apple Computers* [2]. Desenvolvido em C++, o DSS além de possuir código aberto, possui uma excelente documentação para desenvolvedores.

O *Darwin Streaming Server* também disponibiliza uma aplicação web de gerenciamento. Através desta interface, é possível iniciar e parar o serviço de *streaming* de vídeo, e alterar as principais configurações. Esta aplicação é acessada através da porta 1209.

Depois de iniciado, o servidor passa a escutar na porta 554, referente ao protocolo de aplicação RTSP, e responde ao endereço rtsp://<*hostname*>.

O primeiro teste, então, foi verificar se a aplicação estava corretamente configurada, e se atenderia às expectativas. O teste foi bem simples: o DSS foi iniciado na máquina de desenvolvimento, respondendo pelo endereço rtsp://192.168.1.58 e, em um outro computador, da mesma rede, iniciou-se o QuickTime, usando o menu "Abrir URL" para abrir a url do servidor. Usaram-se os vídeos que o DSS disponibiliza em usa instalação para fins de teste, na url rtsp://192.168.1.58/sample_100kbit.mov.

Comprovado o funcionamento do servidor, o passo natural seguinte foi substituir o *player* pela interface de gerenciamento. A princípio, surgiu a hipótese de se desenvolver um *applet* em JMF para servir como um *player* embutido na página. Mas por questões de tempo, a solução adotada foi usar um *plugin* do quicktime *embebed* na página. Essa página, como todas as outras, seria gerada por um JSP, que receberia o IP (através do *request*) do computador responsável por gravar os vídeos (para fins de teste, este computador já possuia os vídeos).

O módulo de gravação de vídeo foi desenvolvido usando JMF e constitui por si só um outro projeto, o Webcam-Recorder. Este aplicativo é responsável por fazer a aquisição do sinal de vídeo gerado pela *webcam* e salvá-lo em um arquivo **.mov**, no padrão Quicktime. Este padrão foi o escolhido por ser o padrão utilizado pelo *Darwin* e deseja-se no futuro transmitir esses vídeos via *streaming*, assim como é feito com o sinal em tempo real. A princípio, este módulo era operado manualmente, ou seja, os comandos de gravação e a escolha do nome do arquivo a ser gerado deveriam ser feitos manualmente. Mas os requisitos funcionais (gravação de arquivos de 1h de duração, com um nome padrão) forçaram a automatização deste processo. Quando ativado, o Recorder irá gravar automaticamente arquivos de 1h de duração no diretório /jakarta-tomcat-5.5.17/webapps/svgr. O Webcam-Recorder também é responsável por publicar o vídeo, ou seja, por inserir o vídeo recém gravado no banco de dados. Na verdade, ele insere as informações de nome, descrição e *path* dos vídeos.

A próxima fase do desenvolvimento foram os testes com a transmissão em tempo real das imagens da *webcam*. Era necessária uma aplicação que fizesse a interface entre a *webcam* e o DSS. A aplicação encontrada foi o Wirecast [5]. Este *software* adquire as imagens da *webcam* e gera um arquivo .sdp, dentro do diretório /Movies/svgr do DSS.

Para testar esse processo, fez-se uso do Quicktime. Novamente, através do menu "Abrir URL", a URL rtsp://192.168.1.58/svgr/*webcam*.sdp foi aberta em um computador remoto. Tendo o teste sido bem sucedido, a integração com o módulo de gerenciamento foi novamente através do *plugin* do quicktime numa página gerada por um JSP, que receberia o IP do computador do qual se quer obter a imagem de *webcam* e também o nome deste computador, pois o arquivo sdp teve seu nome padronizado: *hostname*.sdp, onde hostname é o nome do computador em questão.

Tudo que foi descrito até então foi implementado para a WEB. A interface para dispositivos móveis (WAP) foi o último passo deste projeto. Para realizar os testes, foi instalado um toolkit da Nokia [6], que simula um aparelho de celular e seu browser. Existem quatro formatos básicos de se entregar uma página para um celular: wml 1.0, wml 1.3, xhtml e xhtmlmp. O WML 1.0 seria o formato mais básico, para celulares mais simples, que não suportam, por exemplo, figura. Já o XHTML MP seria para modelos mais avançados, que suportam não apenas imagens, mas tecnologias mais avançadas, como o CSS.

Por questões de tempo, seria inviável desenvolver uma aplicação que entregasse páginas no formato mais adequado para cada modelo de celular. Assim, decidiu-se por desenvolver tudo em XHTML, um formato "meio termo".

As páginas WAP seriam geradas por JSP's, assim como as páginas WEB. Dessa forma, para que se pudesse aproveitar todo o *back-end* já desenvolvido, foi necessário alterações nas *Actions* para diferenciar uma seção web de uma seção *mobile*. Além disso, o mapeamento do *Struts* foi alterado para que as JSP's responsáveis pelas páginas WAP fosse incluídas nos *forwards*.

Os testes iniciais foram feitos no browser disponibilizado pelo toolkit. Nestes testes, foram observados aspectos como layout das páginas e se o mapeamento do *Struts* estava correto. Após esses testes, era necessário testar-se em um celular de verdade. Para tal, o módulo de gerenciamento foi instalado em uma máquina com IP válido é que tivesse acesso liberado para as operadoras de telefonia móvel. O teste foi realizado com um Motorola V3, da operadora Oi, por ser um celular avançado, que suporta tanto XHTML quando XHTML MP.

Apesar da maioria dos testes terem sido bem sucedidos, alguns deles colocaram em evidência um problema freqüente em celulares, que até então não havia sido pensado: alguns celulares suportam apenas páginas WAP com certo limite de tamanho em kbytes. Assim, foi necessário refazer o *layout* de algumas páginas para que elas não ultrapassassem esse limite e implementar uma lógica de paginação para a listagem de usuários, computadores e permissões.

3 Fundamentação Teórica

Neste capítulo, serão apresentadas as principais tecnologias, tanto de protocolos de comunicação, quanto de desenvolvimento, utilizadas no projeto.

3.1 Socket

Um *socket* pode ser definido como um programa, ou rotinas de programas que permitem a comunicação, interligação, e troca de dados entre dois ou mais processos distintos, que podem estar em execução na mesma máquina ou em máquinas diferentes ligadas via rede. Uma vez estabelecida a ligação entre dois processos através dos *socket*s, podem ser enviados dados em ambos os sentidos até que um dos pontos encerre a conexão. *Socket*s são a base para a comunicação com outras máquinas. Programas como o telnet, ftp, talk, entre outros utilizam *socket*s.

Ao criarmos um *socket*, é necessário especificar o tipo de comunicação a ser utilizado e o protocolo a ser usado para implementar a comunicação. O tipo de comunicação de um *socket* representa a semântica da transmissão e recepção de dados no *socket*. A escolha do tipo de comunicação deve satisfazer aos seguintes requisitos:

- o tamanho dos dados a serem transmitidos, que pode ser alguns bytes ou pacotes de dados;
- a relevância de perda de dados, já que para algumas aplicações é necessária a existência de uma garantia da entrega de dados, enquanto para outras aplicações perdas de dados podem ser tolerados;
- a maneira que a comunicação deve ser implementada, ou seja, há casos em que a comunicação deve ser como uma ligação telefônica e em outros casos a comunicação deve ser estabelecida para se transmitir uma única mensagem e deve-se especificar um endereço de destino.

Além disso, também dever ser escolhido um domínio para o *socket*, já que o tipo de dado usado em um *socket* depende do domínio. Cada domínio possui um nome simbólico começado por “PF_”. Um nome simbólico correspondente começado por “AF_” designa o formato do endereço para o domínio.

Finalmente, deve ser escolhido o protocolo para a comunicação. O protocolo irá determinar o mecanismo a ser usado para transmitir e receber dados. Cada protocolo é válido para um domínio e um tipo de comunicação particular e, por isso, o domínio também é chamado de família de protocolos (*protocols family*) e daí vem o fato dos domínios terem seus nomes começados por "PF_".

As regras de um protocolo devem ser aplicadas à transmissão de dados entre dois programas, possivelmente entre duas máquinas diferentes. A maioria dessas regras deve ser suportada pelo sistema operacional e, por isso, não há necessidade de se aprofundar nessa parte. Deve-se ressaltar que, para existir comunicação entre dois *sockets*, o protocolo escolhido para ambos os *sockets* deve ser o mesmo.

3.2 WOL (WAKE-ON-LAN)

O Wake-on-LAN é uma tecnologia que permite o envio de pacotes, com o objetivo de ligar máquinas que se encontram inativas. Com isso, é possível ligar um computador e executar, por exemplo, tarefas de manutenção, sem a necessidade de haver uma pessoa operando a máquina fisicamente.

Essa tecnologia é baseada na transmissão de um pacote de excitação a uma máquina cliente a partir de um servidor que apresente o *software* de gerência instalado. O cliente deve possuir um adaptador de rede Wake-on-LAN, uma placa-mãe com Wake-on-LAN habilitado e um *software* de gerência remota instalado. O adaptador de rede Wake-on-LAN do cliente, ao receber o pacote de excitação, liga a máquina e as tarefas programadas são iniciadas.

O adaptador de rede Wake-on-LAN monitora a rede continuamente, procurando por pacotes de excitação. Por isso, o adaptador deve possuir fonte de energia constante para inicializar a máquina. Tal energia geralmente é proveniente de uma fonte de alimentação especial que fornece uma determinada quantidade de energia continuamente. Uma outra função desse adaptador é decodificar o pacote recebido para determinar se é uma excitação. Caso o pacote seja de excitação, o endereço MAC será repetido 16 vezes, sem interrupções.

Além de transmitir pacotes de excitação, o *software* de gerência remota também permite que a tecnologia Wake-on-LAN seja desabilitada. Além disso, esse *software* permite a programação das tarefas necessárias e ainda informa à máquina se é para desligar ou entrar em modo de espera quando as tarefas forem encerradas.

A tecnologia Wake-on-LAN foi desenvolvida pelo grupo IBM-Intel Wired for Management. Este grupo se dedica ao desenvolvimento de tarefas automatizadas, tais como instalações de *software*, backups e rastreamento de vírus. Através da programação destas tarefas em horários em que a atividade da rede é mínima, é possível que se economize tempo e dinheiro. O WoL pode ser utilizada em redes Ethernet ou redes Token Ring.

3.3 Protocolos

Antes de falar sobre os protocolos, é interessante falar sobre alguns termos importantes para uma melhor compreensão.

Streaming Media

Streaming Media, ou *time-based media*, é um termo usado para descrever qualquer fluxo de dados que necessite ser recebido e processado em tempo real. É basicamente um fluxo fixo de informação que precisa ser enviado, processado, e apresentado "on the fly" para ser apresentado ao usuário ou registrado em um arquivo. Operações de processamento podem incluir converter os dados em um formato diferente, comprimir ou descomprimí-los, ou até mesmo uní-los a outros fluxos de outras fontes. A qualidade do filme ou canção que se está transmitindo é uma função de vários fatores inclusive a largura de banda, poder de processamento do sistema, e o padrão de compressão em que o dado foi transmitido. Para filmes de alta qualidade, é necessário um maior poder de processamento e largura de banda. A qualidade é determinada por, mas não restrita à, ao número de frames exibidos por um determinado período de tempo.

Os formatos de mídia mais comuns são o CINEPAK, o qual é usado em AVI e arquivos de QuickTime, e o MPEG-1, o qual é usado em arquivos de MPEG. O protocolo mais comum para transmitir mídia é o protocolo de transporte de tempo real (RTP). O RTP pode ser usado sobre uma rede ou na Internet. Pode ser usado com endereços IP unicast ou multicast. Quando é transmitido em cima de unicast, são enviadas cópias separadas das mídias a cada consumidor. Quando é usada uma arquitetura de rede de multicast, os dados são duplicados e enviados aos consumidores enquanto que a fonte só o transmite uma vez. Os pacotes RTP não são ordenados, além de não ter-se a garantia que ele foi recebido, tal qual o UDP. Eles estão sendo transmitidos e é a responsabilidade do receptor apanhá-los e ordená-

los. Alguns pacotes são perdidos ao longo do caminho. O progresso dos dados pode ser monitorado usando o RTCP (Real-Time Transport Control Protocol).

3.3.1 RTSP

O Real Time *Streaming* Protocol (RTSP) [7] é um protocolo em nível de aplicação para o controle da entrega de dados em tempo real. O RTSP fornece uma estrutura extensível para permitir a entrega controlada, sob demanda, de dados em tempo real, tais como áudio e vídeo. As fontes dos dados podem incluir transmissões ao vivo dos dados ou mídias já gravadas. Este protocolo tem a intenção de controlar sessões múltiplas da entrega dos dados, fornecer meios para escolher canais de entrega tais como o UDP, o UDP multicast e o TCP, e fornecer meios para escolher mecanismos de entrega baseados em RTP (RFC 1889, ver 3.3.2 RTP).

O RTSP estabelece e controla *streams* síncronos de mídia contínua tais como áudio e vídeo, tanto provenientes de uma única fonte quanto provenientes de várias fontes. O RTSP não é responsável pelo transporte do conteúdo da mídia em si. Para entender melhor seu funcionamento, pode-se pensar nele como se fosse um "controle remoto" para servidores multimídia. É utilizando o RTSP que um usuário terá o controle sobre o fluxo da mídia. A figura abaixo ilustra esse conceito [8]:

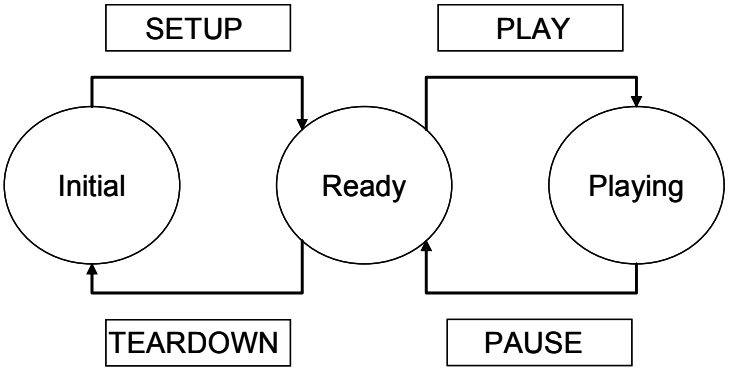

Figura 2: Estados do cliente RTSP

Não existe a noção de uma conexão RTSP. Ao invés disso, o servidor mantém uma sessão etiquetada por um identificador. Uma sessão RTSP não está de maneira alguma amarrada a uma conexão da camada de transporte, como uma conexão TCP. Durante uma sessão RTSP, um cliente RTSP pode abrir e fechar muitas conexões de transporte confiável junto ao servidor para emitir requisições RTSP. Alternativamente, ele pode usar um protocolo não orientado a conexão, como o UDP.

Do ponto de vista do usuário, basta digitar uma URL em seu aplicativo de vídeo, do tipo rtsp://servidor.com/video, e o servidor irá passar a transmitir o vídeo para o cliente, ficando atento para novos comandos.

Na grande maioria das vezes, o transporte dos dados de mídia é feito através do protocolo RTP [3.3.2 RTP], mas a operação do RTSP não depende do mecanismo usado para transportar mídia contínua. O RTP irá criar uma conexão UDP unidirecional para o transporte da mídia e o RTCP cria duas conexões UDP em sentidos opostos para controle de sincronismo no cliente e informação de perda de pacotes para o servidor [8].

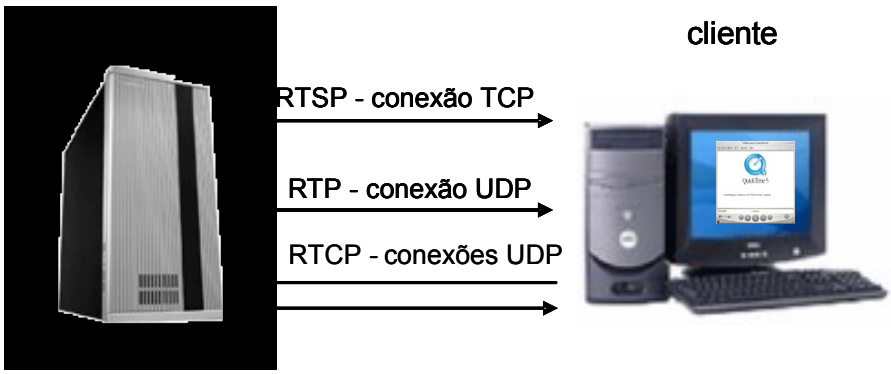

Figura 3: RTSP - Conexões cliente/servidor

O protocolo é intencionalmente similar em termos de sintaxe e operação ao HTTP/1.1 de modo que mecanismos de extensão para o http podem na maioria dos casos também serem aplicados ao RTSP. Para maior detalhamento deste protocolo, sugere-se a leitura da bibliografia indicada na seção Referência Bibliográfica

3.3.2 RTP

O RTP (*Real-time Transport Protocol*) provê serviços fim-a-fim de transporte de dados em tempo real, como áudio e vídeo, tanto em unicast quanto em multicast.

Na grande maioria das implementações, o RTP utiliza-se do protocolo UDP para que possam fazer uso de seus serviços de multiplexação e checksum. Assim, o UDP complementa a funcionalidade de transporte do RTP. Entretanto, o RTP pode ser usado com qualquer outro protocolo de transporte de dados que se mostrar adequado.

A RFC 3550 divide o RTP em duas partes [9]:

- o *Real-Time Transport Protocol* (RTP): é responsável por transportar os dados
- *o RTP Control Protocol* (RTCP): deve monitorar a qualidade do serviço e trazer informações sobre os participantes de uma sessão

O RTP e o RTCP são especificados de tal forma a serem independentes das camadas subjacentes de transporte e de rede.

O RTP por si só não provê nenhum mecanismo para assegurar ou prover qualidade de serviço (QoS), mas confia em serviços das camadas inferiores para fazê-lo. Ele não garante a entrega dos pacotes, mas os envia em seqüência. A numeração seqüencial dos pacotes também serve para que o receptor tenha idéia de quantos pacotes perdeu e, a partir disso, como está a qualidade da recepção. A informação sobre a qualidade da recepção é enviada periodicamente através de relatórios na porta destinada para o controle da sessão.

3.4 Tecnologias Java

3.4.1 J2EE

O J2EE (Java 2 Enterprise Edition) é uma plataforma de programação de computadores que faz parte da plataforma Java. Ela é voltada para aplicações multicamadas, baseadas em componentes que são executados em um servidor de aplicações. A plataforma Java EE é considerada um padrão de desenvolvimento já que o fornecedor de *software* nesta plataforma deve seguir determinadas regras se quiser declarar os seus produtos como compatíveis com Java EE. Ela contém bibliotecas desenvolvidas para o acesso à base de dados, RPC, CORBA, etc. Devido a essas características a plataforma é utilizada principalmente para o desenvolvimento de aplicações corporativas.

A plataforma J2EE contém uma série de especificações, cada uma com funcionalidades distintas. Dentre elas, tem-se:

- JDBC (Java Database Connectivity): utilizado no acesso a bancos de dados;
- Servlets: são utilizados para o desenvolvimento de aplicações Web com conteúdo dinâmico. Ele contém uma API que abstrai e disponibiliza os recursos do servidor Web de maneira simplificada para o programador.
- JSP (Java *Server* Pages): uma especialização do servlet que permite que conteúdo dinâmico seja facilmente desenvolvido.
- JTA (Java Transaction API): é uma API que padroniza o tratamento de transações dentro de uma aplicação Java.
- EJBs: utilizados no desenvolvimento de componentes de *software*. Eles permitem que o programador se concentre nas necessidades do negócio do cliente, enquanto

questões de infra-estrutura, segurança, disponibilidade e escalabilidade são responsabilidade do servidor de aplicações.

- JCA (Java Connector Architecture): é uma API que padroniza a ligação a aplicações legadas

Neste projeto, serão utilizados os conceitos de JDBC, Servlets e JSP.

3.4.2 *Struts* e *Actions*

O *Framework Struts* foi criado por Craig R. McClanahan e doado para a ASF (Apache *Software* Fundation) em 2000. Craig esteve envolvido na especificação de uma grande parte da implementação do *servlet container* TOMCAT 4 [10]. O *Struts* é um *Framework* open *source* do projeto Jakarta que auxilia a construção de aplicações para a Web [11]. Ele é construído em Java, e seu núcleo consiste numa camada de controle flexível baseada nas tecnologias *Java Servlets, JavaBeans, ResourceBundles* e XML.

Aplicações Web baseadas em Java *Server* Pages às vezes misturam código de banco de dados, código de design de página, e código de controle de fluxo. Uma boa prática é dividir a aplicação em camadas segundo o modelo MVC (Model-View-Controller) de arquitetura, conforme será explicado logo adiante. O *Struts* favorece o desenvolvimento de aplicações seguindo o paradigma MVC. O *Struts* fornece um componente Controller e se integra a outras tecnologias para oferecer suporte aos componentes Model (como JDBC, EJB's, etc.), e View (como JSP, XSLT, etc.).

O Design Pattern MVC

No design pattern MVC, o fluxo da aplicação é mediado por um controlador central (Controller). O controlador delega requisições para tratadores apropriados, que estão localizados no modelo (Model), que representa a lógica de negócio e o estado da aplicação. A requisição então é respondida, através do controlador, e apresentada na visão (View), da maneira adequada. No *Struts*, essas respostas são orientadas através de mapeamentos, que são carregados através de um arquivo de configuração (*Struts*-config.xml). Isso faz com que não haja dependência entre a visão e o modelo, que auxilia na criação e na manutenção da aplicação [15].

A figura abaixo, retirada de [10] explica muito bem o fluxo da aplicação segundo o MVC:

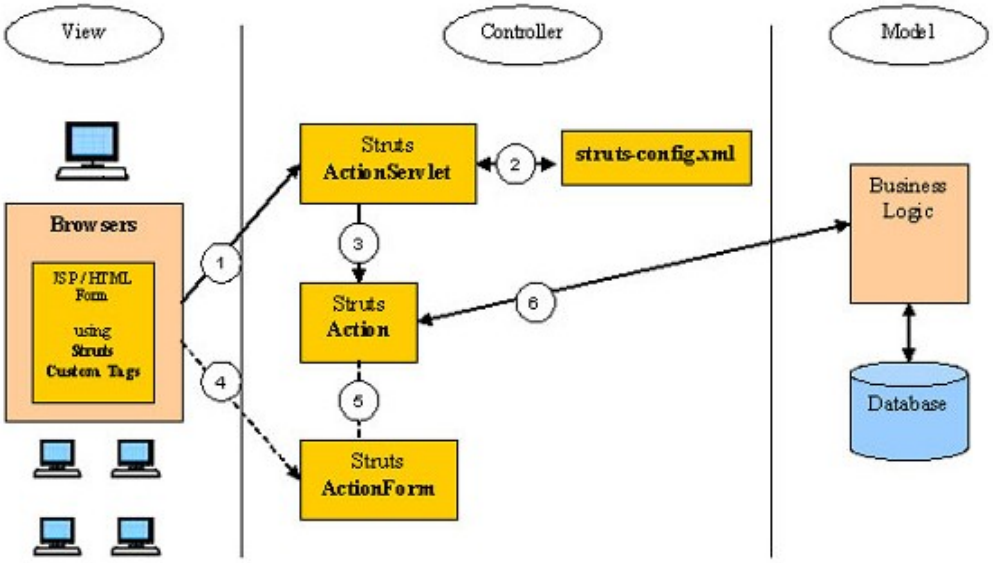

Figura 4: Fluxo de uma aplicação *Struts*

1. Cada solicitação HTTP tem que ser respondida com uma resposta HTTP. Desta forma inicia-se uma aplicação que utiliza o *Struts*, com uma solicitação HTTP. Esta solicitação, normalmente é definida como alguma_coisa.ssp.

2. A solicitação alguma_coisa.ssp é mapeada no arquivo *Struts*-config.xml. Este arquivo é lido por um *Actionservlet* na inicialização da aplicação e cria um banco de objetos com o arquivo de configuração. No arquivo de configuração são definidos os *Actions* para cada solicitação.

3. O *Actionservlet* define o *Action* correspondente para a solicitação. Um Action pode validar a entrada e acessar a camada de negócios para recuperar as informações nos bancos de dados e outros serviços de dados.

4. A requisição HTTP pode ser feita também através de um formulário HTML. Em vez de fazer com que cada Action retire os valores do campo da solicitação, o *Actionservlet* coloca a entrada em um JavaBean , estes JavaBeans são definidos como *FormBeans* no *Struts* e estendem a classe org.apache.*Struts*.*action*.ActionForm.

5. A *Action* pode acessar o *FormBean* efetuar qualquer operação e armazenar o resultado em um *ResultBean*.

6. A *Action* interage com a camada de negócio onde uma base de dados poderá ser atualizada.

O *Struts* envia a solicitação para outro recurso da camada *View* (uma página JSP, por exemplo). Ao completar a lógica de negócio, o Action selecionará e retornará um *ActionForward* para o servlet. Então o servlet usará o caminho armazenado neste objeto para chamar a página e completar a resposta.

Componentes do *Struts [15]*

Componentes Model

Os componentes Model englobam dois conceitos: a lógica de negócio da aplicação e seu estado. As tecnologias utilizadas nessa camada, são JavaBeans, e freqüentemente, Enterprise JavaBeans e JDBC.

Através destas tecnologias, deverá ser construída a lógica de negócio da aplicação.

Componentes View

Os componentes View representam a visão da aplicação, ou seja, a maneira como o sistema interage com o usuário. A tecnologia mais comumente utilizada nessa camada é Java *Server* Pages.

Nessa camada, o *Struts* oferece suporte a dois importantes aspectos: internacionalização e construção de interfaces JSP através de custom tag's.

Outro importante recurso é a disponibilidade de custom tag's do *Struts*, que permitirá uma série de funcionalidades, que, na grande maioria dos casos, dispensará a utilização de código Java dentro da página (aliás, esse é um dos objetivos da arquitetura MVC, pois o fluxo de controle e lógica de negócio da aplicação não devem estar na camada de visão). Além disso, a extensibilidade de Java permite que o próprio desenvolvedor crie suas próprias custom tag's, que, aliás, é uma tarefa bastante simples.

As custom tag's do *Struts* são divididas em cinco bibliotecas:

- html: permite a criação e manipulação integrada de formulários HTML com o *Struts*;

- logic: permite a criação de estruturas de condição e repetição, além da administração do fluxo da aplicação;
- bean: permite a criação e manipulação de JavaBeans dentro da página;
- nested: permite a definição de modelos de objetos aninhados e a capacidade de representá-los e administrá-los;
- template: permite a criação de modelos dinâmicos de páginas JSP que compartilham de um formato comum.

Componentes Controller

Os componentes Controller são responsáveis pelo fluxo da aplicação. O principal componente Controller do *Struts* é a *Actions*servet, que é uma extensão de Servlet, exercendo o papel de controlador principal da aplicação. Sua principal função é fazer o mapeamento das requisições do servidor.

Para isso, é necessário criar um arquivo de configuração, denominado *Struts*-config.xml, que é usado para mapear a navegação da aplicação. Depois disso, deverão ser criadas as classes Action (que são extensões da classe Action do *Struts*), que conterão as ações a serem executadas a cada requisição. O objetivo da classe Action é processar a requisição e retornar um objeto da classe ActionForward que identifica para qual componente de visão (normalmente uma página JSP) será gerada a resposta. As classes Action são compostas de um único método, com a assinatura `public ActionForward execute (ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws IOException`, ServletException, que será invocado quando a Action *s*ervlet receber a requisição e processá-la.

O *Struts* vem sendo largamente utilizado por desenvolvedores de aplicações Java para Web no mundo inteiro e traz os seguintes benefícios:

- os desenvolvedores preocupam-se apenas em criar serviços, e não em *layouts* HTML; mudanças no *layout* não requerem necessariamente, mudanças no código;
- o código fica mais limpo, pois não se mistura código Java com código HTML;
- há uma divisão lógica entre as partes do sistema, auxiliando na clareza do código e no desenvolvimento em equipe.

No desenvolvimento de aplicações simples e de pequeno porte, talvez não seja tão vantajosa à utilização de uma arquitetura tão complexa e modular como o MVC, sendo que os maiores benefícios são alcançados em aplicações corporativas de grande porte.

3.4.3 JMF

O JMF (*Java Media Framework*) é uma API que permite a manipulação de áudio, vídeo e outras mídias em aplicações Java como *applications* e *applets* [12]. Com ela é possível capturar stream de áudio e vídeo e codificá-lo em diversos formatos como também a transmissão das mídias pelo padrão RTP para o desenvolvimento de aplicações que utilizem vídeo sob demanda.

A tabela abaixo mostra alguns formatos e o modo como o JMF os trata [13].

Tipo de Mídia	JMF 2.1.1 *Cross Platform Version*	JMF 2.1.1 Solaris/Linux	JMF 2.1.1 Windows *Performance Pack*
AIFF (.aiff)	leitura/escrita	leitura/escrita	leitura/escrita
AVI (.avi)	leitura/escrita	leitura/escrita	leitura/escrita
GSM (.gsm)	leitura/escrita	leitura/escrita	leitura/escrita
HotMedia (.mvr)	leitura	leitura	Leitura
MIDI (.mid)	leitura	Leitura	leitura
MPEG-1 Video (.mpg)	-	leitura	leitura
MPEG Layer II Audio (.mp2)	leitura	leitura/escrita	leitura/escrita
QuickTime (.mov)	leitura/escrita	leitura/escrita	leitura/escrita
Sun Audio (.au)	leitura/escrita	leitura/escrita	leitura/escrita
Wave (.wav)	leitura/escrita	leitura/escrita	leitura/escrita

Tabela 1: Tratamento dos Formatos de Vídeo no JMF

A arquitetura JMF é composta por um conjunto de componentes que desempenham funções distintas na captura, processamento, distribuição (por RTP) e apresentação de feixes de mídia contínua (principalmente áudio e vídeo). Alguns dos objetos usados na arquitetura são:

- Fontes de dados (*Data source*) – encapsulam a fonde de dados de áudio, vídeo, ou ambos combinados. Pode obter os dados de um arquivo ou da rede, fornecendo-os a outros componentes da arquitetura.
- Dispositivo de captura (*Capture device*) – representa o hardware usado para capturar a mídia (microfones, câmaras de vídeo, etc.).
- Reprodutor (*Player*) – um *player* processa um fluxo de uma fonte de dados e o renderiza em tempo real. Um *player* é o que estará entre a tela e a *webcam*.

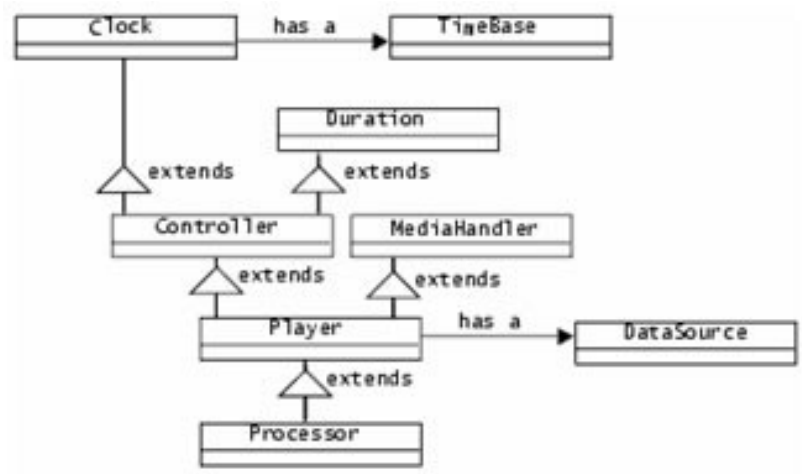

Figura 5: Esquema Básico de uma Aplicação JMF

A seqüência de métodos a invocar para apresentar esses dados é: *realize(), prefetch(),* e *start()*. Vários estados ocorrem antes que o *player* possa ser executado, seguindo a seguinte ordem:

- ***Unrealized***: estado inicial do *player* quando o objeto é instanciado;
- ***Realizing***: determinado os recursos necessários e obtendo informações sobre a mídia;
- ***Realized***: todos os recursos são exclusivos (já está ligado à fonte);
- ***Prefetching***: está a realizar a operação prefetch() (pré-carregamento de dados, obter dispositivos de uso exclusivo);
- ***Prefetched***: está pronto para apresentar os dados;
- ***Started***: *player* está em execução.

Considerando que um *player* é responsável por processar e entregar um fluxo constante de dados, ele tem vários estados:

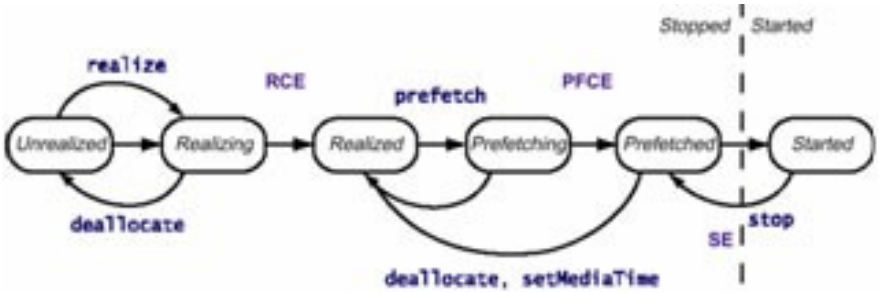

Figura 6: Estados de um Player

- Processador (*Processor*) – é um tipo de *player*. Além de renderizar os dados de mídia, um processador poder enviar a mídia através de uma outra fonte de dados para ser usado por um outro *player* ou processador. Um processador pega o fluxo de dados de entrada, executa um processamento sobre a mídia, e então produz os dados de saída. Ele pode, por exemplo, enviar os dados para um outro dispositivo ou para uma fonte de dados. Um processador tem vários estados, que podem ser divididos em dois estados principais: *unrealized* e *realized*. Para colocar um processador no estado *realized*, deve-se primeiro configurá-lo. Ele então se conecta à fonte de dados acessa toda a informação da qual precisa para processar os dados e passa para o estado *realized* e começa a adquirir a mídia. Neste momento, pode-se começar o processamento. Um processador tem uma operação extra, *configure*(), invocada antes do *realize*(), que dá origem a dois estados adicionais, entre o estado *Unrealized* e o *Realizing*:
    - *Configuring* – está realizando a operação *configure*()
    - *Configured* – já tem toda a informação.

Uma mídia pode conter vídeo e áudio. Neste caso são divididas em stream de mídia chamada de tracks. Uma mídia pode conter múltiplos tracks onde cada uma pode representar o áudio, o vídeo e dados midi. No caso de uma transmissão via rede de um filme, as tracks são transmitidas separadamente o que faz necessário separá-las antes da transmissão. Para isso fazemos utilização da classe javax.media.Processor a qual é responsável por definir o módulo de processamento e controle dos dados da mídia. O processador é uma "caixa preta" com a única finalidade de preparar os dados para os destinos. Enquanto o processador estiver no estado configurado, nós podemos decidir em nossas opções de processamento usar o objeto de controle de track.

O *Processor* está dividido em três estágios:

- Demultiplexação: separa as tracks existentes na mídia para ser processado individualmente.
- Transcodificação: cada track pode ser codificada em outros formatos.
- Multiplexação – as tracks separadas podem ser transformadas em uma única mídia outra vez contendo a nova codificação.

- Escoamento de dados (*Data sink*) – Interface base para objetos que recebem os fluxos.
- Formato (*Format*) – objeto que representa o formato dos dados multimídia. A classe *format* inclui duas subclasses: *AudioFormat* e *VideoFormat*. Por sua vez, estas incluem os vários tipos de formatos de áudio e vídeo suportados.

Uma das limitações do JMF é a reduzida lista de codecs de vídeo disponibilizados, que podem obrigar ao desenvolvimento de codecs para aplicações mais profissionais.

- Gestor (*Manager*) – um objeto intermédio que facilita a interligação entre objetos JMF, validando a compatibilidade entre interfaces (*Manager*); pesquisando os dispositivos de captura disponíveis, alto-falantes, etc. (*CaptureDeviceManager*); interligando a aplicação a *plugin*s para JMF (*Plugin*Manager).

Associado a cada um dos objetos JMF, existem um conjunto de funções para configurar o seu estado, e de eventos, gerados assincronamente para notificar mudanças de estado (e.g. Um controlador gera eventos do tipo *ControllerEvent*, que inclui como subtipos, *StopEvent, StartEvent, ControllerErrorEvent, RateChangeEvent*, etc. Os estados de *Configuring* e *Configured* são necessários para obter as informações de entrada antes de iniciar a programação. Quando o processador estiver configurado o *ControllerListener* é chamado passando o evento *ConfigureCompleteEvent* para indicar que a configuração foi finalizada.

A figura abaixo mostra um simples esquema de processamento de um stream de dados.

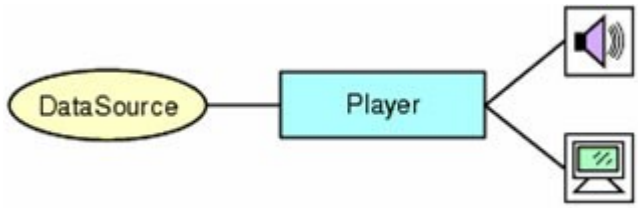

Figura 7: Esquema Simplificado do Processamento de um Stream de Dados

- Controles - controles proporcionam um modo de monitorar e controlar o progresso de mídia que estão sendo baixadas. O JMF vem com controles padrões, mas o usuário tem o poder de definir seus próprios controles.

3.4.4 Hibernate

O Hibernate é uma ferramenta de mapeamento objeto/relacional para Java. Ela transforma os dados tabulares de um banco de dados em um grafo de objetos definido pelo desenvolvedor. [14]

O objetivo do Hibernate é facilitar a construção de aplicações Java independentes de bases de dados relacionais e facilitar o desenvolvimento das consultas e atualizações dos dados. Usando o Hibernate, o desenvolvedor se livra de escrever muito do código de acesso a banco de dados e de SQL que ele escreveria não usando a ferramenta, acelerando a velocidade do seu desenvolvimento de uma forma fantástica. O uso de ferramentas de mapeamento objeto relacional, como o Hibernate, diminui a complexidade resultante da convivência de modelos diferentes: o modelo orientado a objetos (da linguagem Java) e o relacional (da maioria dos SGBDs).

O Hibernate é responsável apenas pelo mapeamento das tabelas do modelo relacional para classes da linguagem Java. As questões relacionadas ao gerenciamento de transações e a tecnologia de acesso à base de dados são de responsabilidade de outros elementos da infra-estrutura da aplicação. Apesar da API do Hibernate possuir operações para o controle transacional, por exemplo, ele simplesmente delega estas tarefas para infra-estrutura no qual foi instalado. No caso de aplicações construídas para serem executadas em servidores de aplicação, o gerenciamento das transações é realizado, normalmente, segundo o padrão JTA.

Nas aplicações *standalone* (que são independentes de um servidor de aplicação), o Hibernate delega o tratamento transacional ao dever JDBC.

O acesso ao banco, independentemente de como a aplicação é construída, é sempre realizado através de *drivers* que suportam o padrão JDBC. Mas o *Framework* não é uma boa opção para todos os tipos de aplicação. Sistemas que fazem uso extensivo de *stored procedures*, *triggers* ou que implementam a maior parte da lógica da aplicação no banco de dados, contando com um modelo de objetos "pobre" não vai se beneficiar com o uso do Hibernate. Além disso, aplicações que tem uma alta freqüência de "movimentos em massa", como seqüências de "*inserts*", "*updates*" e "*deletes*" terminaria por perder performance, graças a instanciação e carregamento de diversos objetos na memória. Ele é mais indicado para sistemas que contam com um modelo rico, onde a maior parte da lógica de negócios fica na própria aplicação Java, dependendo pouco de funções específicas do banco de dados.

O Hibernate trabalha através da interação das classes java (*.java) com arquivos de mapeamento Hibernate (*.hbm.xml). Geralmente para cada classe VO existe um xml .hbm.xml relacionado. Este arquivo de formato xml contém informações sobre a tabela e campos do banco de dados a serem vinculados à classe e suas propriedades. Ex.: user.hbm.xml

```
-------------------------------
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping package="br.ufrj.projetofinal.user.domain">
	<class name="UserVO" table="user">

	<!-- Identificador da classe -->
	<id name="idUser">
		<generator class="increment"/>
	</id>

	<!-- Propriedades da classe -->
	<property name="username"/>
	<property name="password"/>
	<property name="name"/>
	<property name="ask"/>
	<property name="answer"/>
	<property name="bloqueado"/>

	<!-- Relacionamento da classe -->
	</class>
</hibernate-mapping>
```

O arquivo hibernate.cfg.xml é utilizado para a configuração do Hibernate, nele são incluídas informações como: o dialeto "falado " pelo banco de dados, a classe Java do *driver*

JDBC, a identificação do usuário e senha para acesso ao banco de dados, e uma lista de arquivos de relacionamentos (*.hbm.xml). Ex.: hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="hibernate.dialect"> org.hibernate.dialect.MySQLDialect </property>
<property name="hibernate.connection.driver_class"> com.mysql.jdbc.Driver </property>
<property name="hibernate.connection.url">
                jdbc:mysql://localhost:3306/projetofinal
        </property>
        <property name="hibernate.connection.username">
                root
        </property>
        <property name="hibernate.connection.password">

        </property>
        <!-- Configuração do c3p0 -->
        <property name="hibernate.c3p0.max_size">10</property>
        <property name="hibernate.c3p0.min_size">2</property>
        <property name="hibernate.c3p0.timeout">5000</property>
        <property name="hibernate.c3p0.max_statements">10</property>
        <property name="hibernate.c3p0.idle_test_period">3000</property>
        <property name="hibernate.c3p0.acquire_increment">2</property>

        <!-- Configurações de debug -->
        <property name="show_sql">false</property>
        <property name="hibernate.generate_statistics">false</property>
        <property name="hibernate.use_sql_comments">false</property>
        <mapping resource="user.hbm.xml"/>
        <mapping resource="computer.hbm.xml"/>
        <mapping resource="permission_xref.hbm.xml"/>
        <mapping resource="video_xref.hbm.xml"/>
        <mapping resource="webcam_xref.hbm.xml"/>
        <mapping resource="video.hbm.xml"/>

        </session-factory>
</hibernate-configuration>
```

Para este projeto, as seguintes propriedades foram utilizadas:

- hibernate.dialect: é a implementação do dialeto SQL específico do banco de dados a ser utilizado
- hibernate.connection.driver_class: é o nome da classe do driver JDBC do banco de dados que está sendo utilizado
- hibernate.connection.url: é a URL de conexão específica do banco que está sendo utilizado
- hibernate.connection.username: é o nome de usuário com o qual o Hibernate deve se conectar ao banco

- hibernate.connection.password: é a senha do usuário com o qual o Hibernate deve se conectar ao banco.

Essa segunda parte do arquivo são as configurações do "*pool*" de conexões escolhido para a nossa aplicação. Neste projeto, o pool utilizado é o C3P0, mas existem outros *pools* oferecidos no Hibernate. Na terceira parte, estão algumas opções para ajudar a debugar o comportamento do Hibernate, a propriedade "show_sql" faz com que todo o código SQL gerado seja escrito na saída default, "hibernate.generate_statistics" faz com que o Hibernate gere estatísticas de uso e possa diagnosticar uma má performance do sistema e "hibernate.use_sql_comments" adiciona comentários ao código SQL gerado, facilitando o entendimento das queries.

A última parte do arquivo é onde são indicados os arquivos de mapeamento que o Hibernate deve processar antes de começar a trabalhar. Se algum arquivo de mapeamento de qualquer classe não for indicado, essa classe não vai poder ser persistida pela engine do Hibernate. Outro detalhe importante, é que quando você usa mapeamentos com Herança, o mapeamento pai deve sempre vir antes do filho.

No Hibernate, assim como no JDBC, existem os conceitos de sessão e transação. Uma sessão é uma conexão aberta com o banco de dados, onde nós podemos executar queries, inserir, atualizar e deletar objetos, já a transação é a demarcação das ações, uma transação faz o controle do que acontece e pode fazer um roolback, assim como uma transação do JDBC, se forem encontrados problemas.

A classe para configurar e abrir as sessões do Hibernate possui um código bastante simples (usado neste projeto):

```
//Arquivo HibernateUtility.java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtility
 {
        private static SessionFactory factory;
        static
         {
                Try
                 {
                        factory = new Configuration().configure().buildSessionFactory();
                }
                catch (Exception e)
                {
                        e.printStackTrace();
```

```
                factory = null;
            }
        }

        public static Session getSession()
         {
            return factory.openSession();
        }
}
```

O bloco estático (as chaves marcadas como "static") instancia um objeto de configuração do Hibernate (org.hibernate.cfg.Configuration), chama o método configure() (que lê o nosso arquivo hibernate.cfg.xml) e depois que ele está configurado, criamos uma SessionFactory, que é a classe que vai ficar responsável por abrir as sessões de trabalho do Hibernate.

Existem três meios de se fazer buscas: usando o Hibernate, usando a sua linguagem própria de buscas, a *Hibernate Query Language* (HQL), usando a sua *Criteria Query API* (para montar buscas programaticamente) e usando SQL puro. A maioria das suas necessidades deve ser suprida com as duas primeiras alternativas. Para o resto, sempre se pode usar SQL pra resolver.

A HQL é uma extensão da SQL com alguns adendos de orientação a objetos, nela você não vai referenciar tabelas, vai referenciar os seus objetos do modelo que foram mapeados para as tabelas do banco de dados. Além disso, por fazer pesquisas em objetos, você não precisa selecionar as "colunas" do banco de dados, um select assim: "select * from UserVO" em HQL seria simplesmente "from UserVO", porque não estamos mais pensando em tabelas e sim em objetos.

Criteria Query API é um conjunto de classes para a montagem de queries em código Java, você define todas as propriedades da pesquisa chamando os métodos e avaliações das classes relacionadas.

Outro complemento importante do Hibernate é o suporte a paginação de resultados. Para paginar os resultados, nós chamamos os métodos setFirstResult(int first) e setMaxResults(int max). O primeiro método indica de qual posição os objetos devem começar a ser carregados, o segundo indica o máximo de objetos que devem ser carregados. Estes métodos estão presentes tanto na interface "*Query*" quanto na interface "*Criteria*". Uma propriedade específica da HQL, que foi herdada do JDBC, é o uso de parâmetros nas queries.

4 Detalhamento da Solução

O projeto foi implementado através da linguagem de programação JAVA, totalmente orientado a objetos e seguindo a arquitetura MVC já explicada. Ele foi dividido em quatro módulos:

- Módulo de gerenciamento (interface web/wap): este módulo corresponde a toda parte de administração e gerenciamento do cadastro de usuários e máquinas no sistema. Através deste módulo, o usuário tem acesso às funcionalidades previstas para este projeto, principalmente a transmissão de vídeo em tempo real;
- Serviço que roda nas *workstations* – recebe os pedidos de desligamento, *logoff* e relatório;
- Gravação do sinal de vídeo gerado pela *webcam* – foi desenvolvido um aplicativo com a função de adquirir o sinal da *webcam* e gravar em arquivos de 1h de duração, no formato .**mov** (QuickTime) por questões de compatibilidade com o *Darwin*;
- Tráfego de vídeo em tempo real – Essa parte refere-se a captura e transmissão do vídeo gerado pela *webcam* em tempo real. Foi utilizado o *software* Wirecast para fazer a interface entre a *webcam* e o *Darwin Streaming Server*. Ela faz parte do módulo de gerenciamento, e por isso não será detalhada separadamente.

No *software* desenvolvido, os conceitos de *multi-thread*, *socket*, JSPs, *Actions* e banco de dados foram largamente utilizados. No esquema da Figura 8, pode-se observar a ligação de cada um dos programas com os computadores e a sua localização.

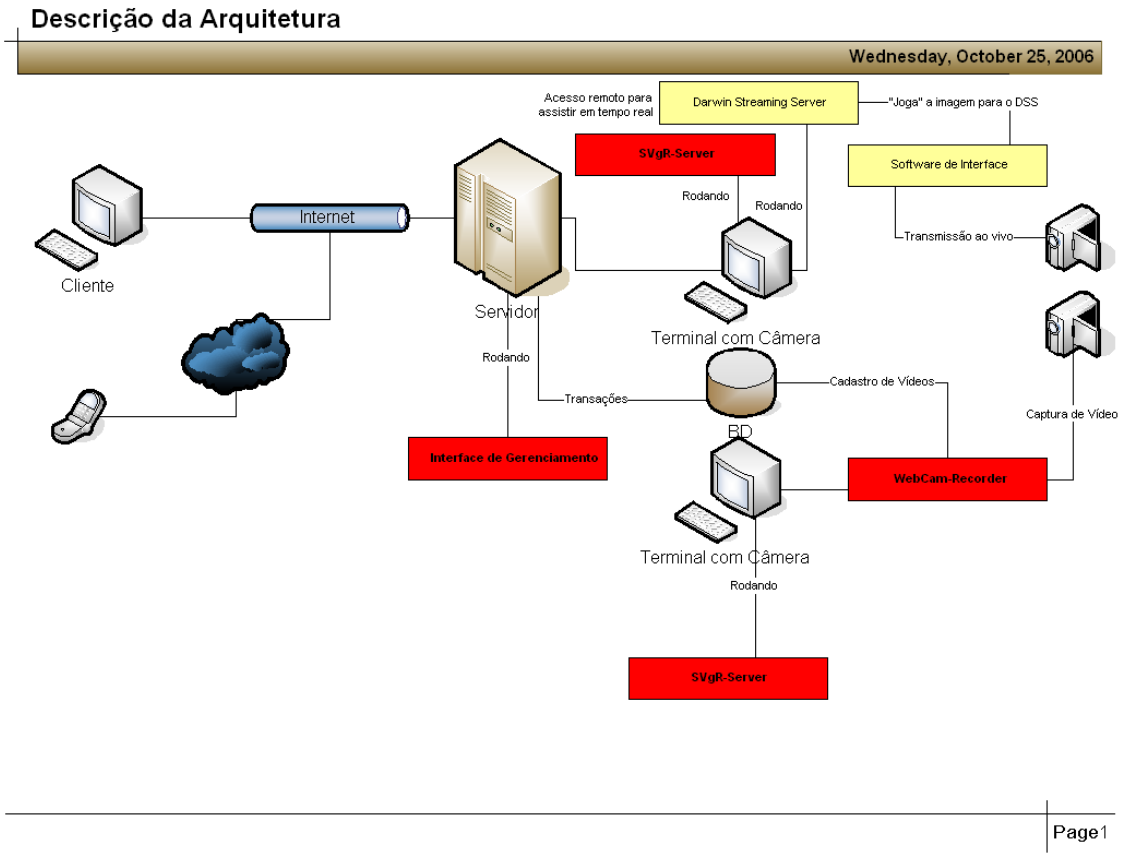

Figura 8: Arquitetura da Solução

Nas subseções a seguir, serão desenvolvidas explicações sobre as tecnologias e classes utilizadas, além de cada programa implementado nas fases descritas acima ser detalhado.

4.1 Interface de Administração

As seguintes tecnologias e *software*s serão utilizados:

- Plataforma: Java. Mais especificamente:
  - Front-end: HTML, Java Servlet Pages (JSP's)
  - Core: Java. Mapeamento de *Actions* através do uso do *Framework Struts*
  - Interface celular: páginas em xhtml geradas dinamicamente por JSP's
- Modelo de Banco de Dados: *DBDesigner 4*
- Banco de Dados: qualquer um, desde que suportado pelo JDBC e pelo Hibernate 3.
- *Application Server*: Apache Tomcat 5.5
- *Streaming*: o vídeo gerado pelo sistema local de vigilância será entregue ao usuário utilizando *streaming*.

A Interface de Administração engloba todo o *software* responsável pelas funções administrativas do sistema. Através desta interface, o usuário terá acesso às seguintes funcionalidades gerais:

- Cadastrar/Remover/Alterar um computador novo (apenas usuário administrador);
- Cadastrar/Remover usuário no sistema (apenas usuário administrador);
- Alterar senha do usuário;
- Setar/Alterar/Remover permissão de um determinado usuário sobre uma determinada máquina: o usuário só poderá atuar sobre um computador para o qual ele tenha permissão (apenas usuário administrador);
- Listagem de todas as máquinas e usuários cadastrados (apenas usuário administrador);
- Desbloquear usuário (apenas usuário administrador);

A aplicação foi desenvolvida para atender aos requisitos básicos de um sistema de gerenciamento de computadores e usuários. Além dos requisitos funcionais especificados, uma boa usabilidade foi também considerada como um requisito do produto. Mais especificamente, o usuário terá as seguintes funcionalidades sobre um computador sobre o qual ele tenha permissão:

- Desligar o computador: o usuário especifica um endereço MAC, IP ou um hostname do computador desejado, desde que ele não seja responsável pela vigilância do ambiente, ou seja, desde que não exista uma *webcam* conectada a ele que esteja sendo utilizada pelo sistema;
- Ligar o computador: o usuário especifica um endereço MAC do computador desejado. Neste caso, não é possível estabelecer conexão via IP com o computador remoto, pois a placa de rede conhece apenas seu endereço MAC e é ela, através da funcionalidade WAKE-UP LAN que irá dar o comando para o *boot* da máquina;
- Acessar a câmera de segurança (*webcam*) de um determinado computador, desde que ele esteja devidamente cadastrado e possua o *Darwin Streaming Server* (DSS) instalado nele, além de um *software* para fazer a interface entre a *webcam* e o DSS. Cada *workstation* com *webcam* instalada por ser um nó independente do sistema de vigilância. O *streaming* permitirá ao usuário acompanhar em tempo real o que está acontecendo na sala específica onde aquela web-cam está.

- Visualizar o arquivo de vídeo de um dado intervalo de hora específico. O arquivo em disco possibilitará que o vídeo seja visto *offline*, ou seja, a qualquer momento, o usuário poderá assistir as imagens de um dado intervalo de tempo. Cada um desses arquivos em disco terá 1h de duração e possuíram nomenclatura com um formato padrão: svgr-dd:mm:aa:hh:mm:ss. Cada vídeo gerado terá uma entrada em uma tabela do banco de dados, para que o usuário possa ver a lista de todos os vídeos, e escolher o desejado.

As duas últimas funcionalidades só seriam acessíveis pela WEB.

O acesso do usuário a esse sistema será feito através de uma autenticação, com login e senha, tanto pela interface WEB (Figura 43) quanto pela interface móvel (Figura 44). Obviamente, por tratar-se de um sistema de segurança, devemos assegurar sua invulnerabilidade, ou, pelo menos, dificultar ao máximo que ele seja invadido e usado por pessoas não autorizadas. Assim, desenvolvemos um sistema de autenticação de usuário baseado no cadastramento de usuários e no uso das variáveis de ambiente da sessão aberta com o servidor.

Haverá uma tela inicial onde o usuário digitará essas informações e, a seguir, elas serão validadas. Em caso positivo, o usuário visualizará o menu do sistema, e suas informações estarão contidas em uma variável de seção, para controle de *logon* e *logout*. Essa validação garante a segurança do sistema. Permite apenas que os usuários cadastrados possam acessar a página do sistema e ganhar direito sobre as demais funções. Como é preciso existir um gerenciamento de todo o sistema, tem-se, por padrão, o usuário admin, criado durante a instalação e terá acesso universal a todas as funções, além de poder atribuir permissões aos demais usuários.

O primeiro passo é o cadastramento de usuários. O sistema tem como único usuário inicial o usuário admin. Ele é o administrador do sistema e tem poder absoluto sobre toda a operabilidade do programa. Cada usuário cadastrado possui uma linha em uma tabela do banco de dados. Essa linha é composta por login, nome do usuário, o md5 da senha do usuário, pergunta e resposta.

Após o devido cadastramento, o usuário está apto a "logar" no sistema, utilizando seu login e a senha cadastrados. A página inicial do sistema pode ser vista na Figura 45.

Também nessa página inicial, o servidor cria uma variável de sessão chamada logged_user, que corresponde a um objeto da classe UserVO, que contém as informações do usuário logado.

Assim, todas as *Actions* e JSP's verificam se este atributo esta setado e podem permitir, ou não, àquele usuário executar uma determinada ação.

As máquinas integrantes do sistema de segurança também devem ser cadastradas. Cada usuário pode ou não possuir permissão para acessar as funcionalidades de uma ou mais máquinas, como ligar e desligar. Essa permissão é setada pelo admin, na opção correspondente do menu principal.

Caso um usuário esqueça sua senha, ele tem a opção de receber uma nova senha, através de um par pergunta-resposta cadastrado no banco de dados. Ou seja, a pergunta cadastrada é mostrada e o usuário, caso saiba a resposta, recebe uma nova senha.

Cada máquina pertencente à rede deve ser devidamente cadastrada no sistema, informando endereço IP, MAC, um hostname e uma senha para essa máquina, além de um status para a *webcam* (presente/ausente). Nessas máquinas, está sendo executado um *software* responsável por fazer as operações que o servidor manda para a máquina (como ligar e desligar), o SVgR-*Server*, que será detalhado na próxima seção.

Assim que um usuário entra no sistema, ele tem acesso ao menu principal, onde ele poderá ter acesso às funcionalidades do programa, de acordo com as permissões setadas para seu tipo de usuário (apenas o admin tem acesso a tudo). Ao escolher um item desse menu, a *action* correspondente é chamada. Quando a *action* terminar de executar a funcionalidade prevista para ela, o resultado será mostrado por um JSP, mapeado através do *Struts* (vide 3.4.2 Struts e Actions).

4.2 SVgR - Server

Desenvolvido usando Java Swing, o SVgR-*Server* é o responsável por executar as ações previstas sobre um computador cadastrado na rede. Ele inicializa junto com o sistema operacional, e deve rodar com permissão de superusuário.

O SVgR-*Server* fica escutando em uma porta definida por uma constante e seu valor padrão é 12345. Essa porta deve ser a mesma configurada no módulo de gerenciamento e é através dela que chegaram os pedidos de processamento remoto enviados pela interface web, através de uma conexão TCP.

Dependendo do pacote TCP que chegar, uma funcionalidade é executada. Esse pacote é enviado pelo módulo de gerenciamento, através do menu Computadores Cadastrados. Esse menu chama uma *action* que possui métodos para se conectar ao banco de dados para recuperar e exibir para o usuário todas as máquinas cadastradas no sistema, sobre as quais ele tem permissão. Para cada computador listado, existe uma lista de ações a serem realizadas, conforme já mencionado. Escolhendo-se um desses itens, o usuário será levado para a *action* ComputerExecuteAction, que receberá como parâmetro a identificação do menu, e entrará no switch apresentado a seguir (simplificado):

```
switch(option)
{
case 1:
{
        ServerVO = computerBO.powerOn(computerVO);
}
break;
case 2:
{
        ServerVO = computerBO.shutdown(computerVO);
}
break;
case 3:
{
        ServerVO = computerBO.cancelShutdown(computerVO);
}
break;
case 4:
{
        ServerVO = computerBO.logout(computerVO);
}
break;
case 5:
{
        ServerVO = computerBO.generateReport(computerVO);
}
break;
}
```

Cada uma dessas funções está dentro da classe ComputerBO. Elas utilizam *socket* e uma conexão TCP para enviar o pedido do procedimento remoto desejado e receber os dados referentes a esse pedido. O parâmetro desses métodos é um objeto da classe ComputerVO, cujos atributos já foram setados previamente.

A função *powerOn*() usa o endereço MAC do computador para ligá-lo (nesse caso, a opção Wake-on-LAN deve estar habilitada na BIOS) através do envio de um pacote apropriado.

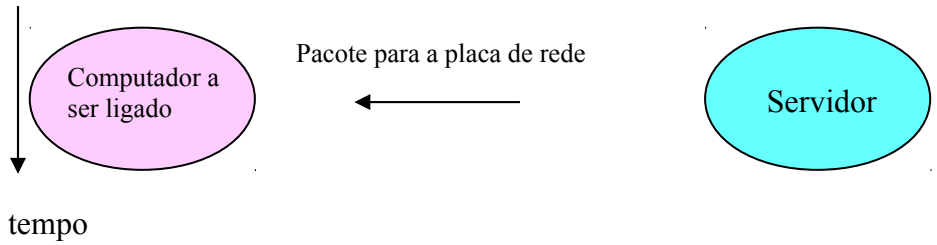

Figura 9: Troca de Pacotes entre Computadores, para Ligar Computador

O endereço IP de cada máquina é usado nas outras funções ao invés do MAC. A função shutdown() manda pacotes para a máquina a ser desligada e recebe pacotes de retorno, o que significa que ocorre uma troca de mensagens entre as duas máquinas, como um meio de autenticação.

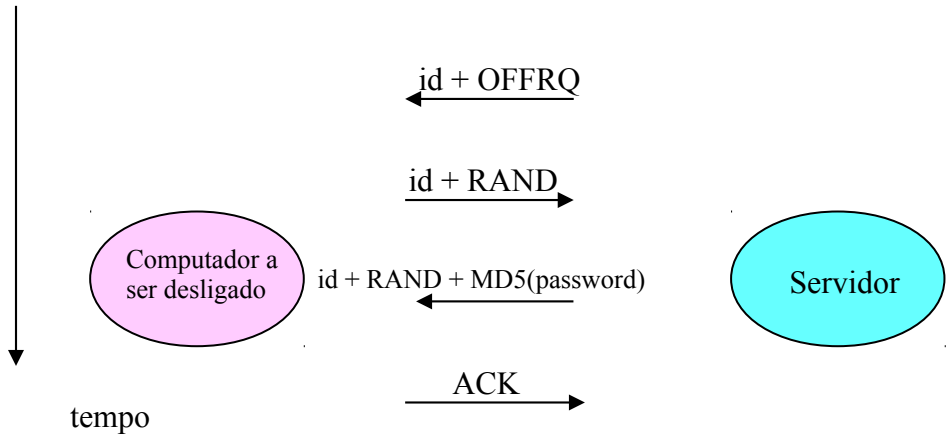

Figura 10: Troca de Pacotes entre Computadores para Desligar Computador

O SVgR-*Server* se conecta ao banco de dados para recuperar a senha do computador no qual está instalado. Assim, quando chega um pedido de procedimento remoto para essa máquina, a senha que está no banco de dados é comparada com a senha enviada juntamente com o comando desejado, liberando, ou não, a funcionalidade requerida.

Esse processo ocorre também para as outras funcionalidades previstas (*cancelShutdown*(), *logout*() e *generateReport*()). A diferença está apenas no tipo de pedido (CCLRQ, LGTRQ e RPTRQ, respectivamente) que é enviado. O SVgR-*Server* possui a inteligência de recuperar esse pedido e executar a ação correspondente. Poder-se-ia implementar uma thread para que o *Server* continuasse apto a receber pacotes. Mas optou-se por não fazê-lo por um motivo bastante simples: evitar que a ação de um usuário cancele a de outro. Assim, o SVgR-*Server* fica "preso" em uma conexão até que ela seja liberada.

De todas as funcionalidades previstas, a única que retorna alguma informação para o cliente (módulo de gerenciamento) é a *generateReport*(). Ela colhe do sistema informações

como sistema operacional e sua versão, usuário logado, versão da JVM instalada, memória RAM e física usadas e disponíveis, dentre outras.

4.3 Webcam – Recorder

Este módulo, desenvolvido em JMF, é o responsável por executar a funcionalidade que, juntamente com a transmissão em tempo real, dá sentido a todo o sistema desenvolvido: adquirir o sinal de vídeo gerado pela *webcam*. Os vídeos gerados, já no formato .mov, são arquivados no diretório */jakarta-tomcat-5.5.17/webapps/svgr*. Assim, basta o cliente acessar este endereço por qualquer *player* que suporte o formato *player*. A automatização do Webcam-Recorder e tal que ele gera arquivos com 1h de duração, nomeados de acordo com o padrão especificado: svgr-AAMMDD_HHMMSS.mov, onde A é o ano, M o mês, D o dia, H a hora, M o minuto, e S o segundo.

A seguir, será apresentada uma pequena explicação sobre os principais métodos utilizados no *software*.

Primeiramente, deve-se acessar a *webcam* e verificar se o JMF a está reconhecendo:

```
ml= new MediaLocator(" vfw://0");
```

A seguir, deve-se criar um *data source* para este dispositivo:

```
setMainCamSource(Manager.createDataSource(ml));
```

Para que este data *source* seja acessado por diversos *players*, deve-se transformá-lo em um data *source* que possa ser clonado:

```
mainCamSource  = Manager.createCloneableDataSource(mainCamSource);
```

Os clones podem ser usados para realizar uma apresentação ou uma transmissão que fosse necessária.

Agora que o *data source* é clonável, é necessário iniciar seu processamento para que os clones possam funcionar. Para isso, será usada a classe camStateHelper que implementa a interface ControllerListener, provida pela Sun e que irá ajudar a controlar os eventos do *player*.

O próximo passo é configurar, realize, e então iniciar o *player*. Uma vez feito isso, é preciso usar um clone para pegar as componentes visuais que serão usadas para mostrar o filme em tempo real na tela:

```
camStateHelper playhelper = new camStateHelper(processor);
playhelper.configure(10000);
processor.setContentDescriptor(null);
playhelper.realize(10000);
processor.start();
```

É importante que o "*content descriptor*" do processador seja setado "null" para impedir que o processador adquirir dados de vídeo sem qualquer tipo de formato (dado "raw"). Quando se está gravando dados em um arquivo, pode-se ajustá-lo ao tipo que for necessário, entretanto, neste momento, não há necessidade de jogar os dados em um outro data *source*. Agora que o clone está ativo e inicializado, sua componente visual será usada para se ter uma prévia do filme na tela:

```
processor.getVisualComponent();
```

Neste momento, é necessário criar um botão associado a um evento que irá invocar um terceiro processador. Este processador irá usar um localizador de mídia e salvar o filme em um arquivo:

```
URL movieUrl = file.toURL();
MediaLocator  dest = new MediaLocator(movieUrl);
```

Em seguida, clona-se um outro *data source*, e invoca-se o processador para agir sobre ele. É necessário agora configurar o processador. Entretanto, antes de realize ele, é necessário obter seus controles de track e ajustar seu formato de vídeo para o formato desejado CINEPAK. Seu *content descriptor* será enviado para um novo arquivo, chamado descriptor.quicktime, e sua taxa será fixada em 15 quadros (*frames*) por segundo.

```
DataSource recordCamSource = dataSource.cloneCamSource();
Processor recordProcessor = Manager.createProcessor(recordCamSource);
camStateHelper playhelper = new camStateHelper(recordProcessor);
playhelper.configure(10000);
VideoFormat vfmt = new VideoFormat(VideoFormat.CINEPAK);
(recordProcessor.getTrackControls())[0].setFormat(vfmt);
(recordProcessor.getTrackControls())[0].setEnabled(true);
recordProcessor.setContentDescriptor(new
```

```
FileTypeDescriptor(FileTypeDescriptor.QUICKTIME));
Control control =
recordProcessor.getControl("javax.media.control.FrameRateControl");
if ( control != null &&
    control instanceof javax.media.control.FrameRateControl )
     ((javax.media.control.FrameRateControl)control).setFrameRate(15.0f);
```

Uma vez feito isso, o *player* pode-se realize o *player*, criar um dissipador de dados (data sink) para o destino, abri-lo, iniciar o processador e iniciar a coleta de dados. A *webcam* agora está ativa. O processador está processando os dados que estão sendo adquiridos dela, e usando um localizador de mídia (*media locator*). O dissipador de dados está transmitindo os dados para o arquivo:

```
playhelper.realize(10000);
DataSink  dataSink = Manager.createDataSink(recordProcessor.getDataOutput(),
dest);
recordProcessor.start();
dataSink.open();
dataSink.start();
```

Para parar a gravação, é necessário fechar o processado e então para e fechar o data sink:

```
recordProcessor.close();
dataSink.stop();
dataSink.close();
```

Agora que a gravação está parada, pode-se abrir o arquivo e assistir ao filme recém criado. Mesmo o processador tenha sido parado, ele é apenas um clone. Os dois originais continuam ativos. A fim de parar o processo inteiro, é necessário fechar o *data source* original.

5 Conclusão

Atualmente, a segurança de computadores de uma intranet é um fator bastante importante a ser considerado. Com isso, foi proposto como Projeto Final um mecanismo de segurança com o objetivo de monitorar um recinto que acomode micros integrantes de uma rede local. Tal projeto foi realizado com a preocupação de se atender situações comuns a laboratórios de universidades e outras instituições de ensino, pequenas, médias ou grandes empresas e domicílios. Em todos os ambientes citados, verificamos o interesse na vigilância de cômodos, para que se possa garantir a segurança não apenas de microcomputadores, mas também de outros equipamentos eletrônicos.

O sistema de segurança foi projetado de forma a ser escalável, com funcionalidades diversificadas e que integrasse hardware e *software*. O sistema apresenta diversas vantagens, como, por exemplo, *software* multiplataforma, mecanismo eficiente de autenticação nas JSP's e *Actions*, custos reduzidos, facilidades na instalação, entre outros.

Apesar das dificuldades, o projeto pôde ser concluído com êxito. Sendo assim, a produção dos vídeos no formato padrão, a configuração do servidor *Darwin Streaming Server* e o desenvolvimento de uma aplicação cliente, possibilitaram a criação de uma solução completa de segurança por *webcam*.

Como trabalhos futuros, diversas possibilidades podem ser enxergadas. A primeira possibilidade, e a que seria mais interessante, consistiria em se ter outros formatos disponíveis para se fazer *streaming*, o que levaria à necessidade de se encontrar um outro servidor de *streaming* ou até mesmo a se construir um servidor próprio. Uma outra possibilidade, também muito interessante, seria criar um mapa de características dos celulares que permitisse entregar o conteúdo na forma mais adequada para o aparelho. Isso significaria criar um banco de dados com dezenas de informações referentes a cada um dos dispositivos móveis existentes e criar-se um método que codificasse as características relevantes de forma que diferentes celulares que possuíssem características semelhantes tivessem a mesma codificação. De posse desse código, o sistema encaminharia o usuário para a página no melhor formato para o dispositivo que ela está usando para acessar o sistema.

Bibliografia

- Java, Como Programar - Harvey M. Deitel & Paul J. Deitel, 6ª Edição, Prentice-Hall
- Tutorial Sun: http://java.sun.com/developer/onlineTraining/J2EE/Intro2/j2ee.html
- Tutorial Ant Demystified: http://www.sitepoint.com/article/apache-ant-demystified
- Tutorial do Apache Ant: http://ant.apache.org/manual/using.html. Acesso em 05 de jul. 2006.
- Java Media *Framework* API (JMF): http://java.sun.com/products/java-media/jmf/

Referência Bibliográfica

[1] Módulo Security, disponível em http://www.modulo.com.br. Acesso em 22 de nov. de 2006

[2] APPLE COMPUTER. *Darwin Streaming Server*. Disponível em: http://developer.apple.com/opensource/*Server*/*streaming*/index.html. Acesso em 20 de ago. 2006

[3] Página da disciplina Engenharia de *Software*, disponível em http://www.del.ufrj.br/~ac/eel873.htm Acesso em 23 de set. 2006.

[4] HotWork Solution - Log4J User Guide. Disponível em: http://hotwork.sourceforge.net/hotwork/manual/log4j/log4j-user-guide.html

[5] WireCast. Disponível em: www.vara*software*.com/products/wirecast/. Acesso em 10 de set. 2006

[6] Nokia Tools and SDKs Mobile Applications Development. Disponível em: http://forum.nokia.com/main/resources/tools_and_sdks/index.html Acesso em 18 de set. 2006

[7] IETF. RFC 2326: Real Time *Streaming* Protocol (RTSP). [S.L.], 1998. Disponível em: http://www.ietf.org/rfc/rfc2326.txt. Acesso em 29 de set. 2006

[8] TOPIC, M. Streaming Media Demystified. New York: McGraw Hill, 2002

[9] IETF. RFC 3550: RTP: A Transport Protocol for Real-Time Applications. [S.L.], 2003. Disponível em: www.ietf.org/rfc/rfc3550.txt. Acesso em 29 de set. 2006

[10] Desmistificando o *Framework* Jakarta *Struts*. Disponível em http://www.javafree.org/content/view.jf?idContent=22. Acesso em 05 de jul. 2006.

[11] *Struts Framework* – http://*Struts*.apache.org. Acesso em 05 de jul. 2006.

[12] Java Media *Framework* API - http://java.sun.com/products/java-media/jmf/

[13] *Framework* Java Para Manipulação de Mídias. Disponível em http://www.plugmasters.com.br/sys/materias/209/1/*Framework*-Java-para-manipula %E7%E3o-de-m%EDdias,-parte-1

[14] Introdução ao Hibernate 3. Disponível em http://www.guj.com.br/content/articles/hibernate/intruducao_hibernate3_guj.pdf

[15] Portal JAVA. Disponível em http://www.portaljava.com.br. Acesso em 22 de nov. 2006

Apêndice – Especificação de Requisitos de *Software*

1 Introdução

1.1 Finalidade

O objetivo deste apêndice é descrever as Especificações de Requisito de *Software* a fim de facilitar o entendimento do sistema pela equipe de desenvolvimento e por seus usuários, bem como para os próprios avaliadores.

1.2 Escopo

O aplicativo em desenvolvimento é o Sistema de Vigilância Remota Através de uma Interface WEB/WAP, cujo objetivo é monitorar um determinado ambiente utilizando uma *webcam* através de uma interface de gerenciamento de fácil utilização e que permita o gerenciamento através de dispositivos móveis.

Seu benéfico está em aumentar a segurança do ambiente em questão, garantindo segurança de dados e patrimônio.

1.3 Definições, Acronismos e Abreviaturas

- ERS – Especificação de Requisitos de *Software*
- DD – Dicionário de Dados
- DER – Diagrama de Entidade Relacionamento
- DFD – Diagrama de Fluxo de Dados
- DTE – Diagrama de Transição de Estados

1.4 Referências

Protocolos RTSP, formato .mov, Apache Tomcat 5.5, Java 1.5, Hibernate 3.0, *Darwin Streaming Server*

1.5 Resumo

Nas seções seguintes da ERS serão detalhados seus requisitos e suas funções, bem como caracterizados o público alvo e as restrições do projeto.

2 Descrição Geral

2.1 Perspectiva do Produto

Este *software* depende de um servidor de *streaming*. No caso, o servidor de *streaming* escolhido foi o *Darwin Streaming Server*, da Apple. O vídeo gerado pela *webcam* é arquivado para posterior exibição, ou transmitido em tempo real para o usuário final através deste servidor de *streaming*.

2.2 Funções do Produto

- Cadastrar/Remover/Alterar um computador no sistema
- Cadastrar/Remover usuário no sistema
- Alterar senha do usuário
- Setar/Alterar/Remover permissão de um determinado usuário sobre uma determinada máquina: o usuário só poderá atuar sobre um computador para o qual ele tenha permissão.
- Listagem de todas as máquinas e usuários cadastrados
- Bloquear/Desbloquear usuário
- Desligar o computador
- Ligar o computador
- Acessar a câmera de segurança (*webcam*) de um determinado computador
- Visualizar o arquivo de vídeo de um dado intervalo de hora específico.

Cabe lembrar que o usuário que acessar via celular não terá acesso a essas duas últimas funcionalidades.

2.3 Características do Usuário

Este sistema é destinado a administradores com conhecimentos avançados em tecnologias de transmissão de vídeo por *streaming* e em aplicativos JAVA, de maneira que eventuais problemas de configuração possam ser resolvidos de forma rápida ou precisa e a usuários que saibam navegar pela Internet.

Recomenda-se a instalação desse sistema em locais que exijam um nível mínimo de segurança e controle sobre uso indevido dos equipamentos, pois este *software* não cobre necessidades mais extremas de segurança.

Para a interface móvel, é necessário um usuário que saiba configurar e usar a Internet.

2.4 Restrições

Poucas são as restrições de uso deste aplicativo. Uma delas é a obrigatoriedade do uso do formato de vídeo .mov (Quick Time), suportado pelo servidor *Darwin*. No entanto, essa limitação acaba trocando-se o servidor.

Existe ainda a restrição já mencionada no item anterior quanto as requisito segurança. Não é recomendável o uso deste aplicativo em locais onde um alto grau de segurança é exigido, embora seja um sistema bastante confiável.

Embora o programa principal, que implementa a interface de administração, seja independente da plataforma, o serviço apresentará funcionalidade limitada dependendo da plataforma onde ele esteja instalado. Isso ocorre por causa da *webcam*, que não apresenta suporte para a plataforma Linux, pois não foi encontrado um driver adequando para que o dispositivo funcionasse adequadamente. Também não é possível gerar os vídeos e realizar a transmissão em tempo real a partir de uma mesma máquina, ou seja, se uma máquina está transmitindo em tempo real, então ela não poderá estar gerando os vídeos.

Existem restrições também quanto aos celulares, uma vez que apenas celulares que suportam o formato XHTML poderão acessar.

2.5 Pressupostos e Dependências

No que se refere ao módulo de gerenciamento (vide 4.1 Interface de Administração), pressupõe-se que o aplicativo será instalado em uma máquina com bastante capacidade de armazenamento em disco e em memória RAM, pois será gerada uma enorme quantidade de dados (arquivos de vídeos) e vários serão os aplicativos que deverão estar rodando conjuntamente. Mas especificamente, deve-se ter uma máquina que esteja rodando o Apache Tomcat 5.5, o *Darwin Streaming Server* e que possua a versão 1.5 do Java. Esta máquina também deverá ter seu acesso externo liberado no Firewall inclusive para as operadoras de telefonia móvel.

Quanto as workstations, os requisitos são: possuir a versão 1.5 do Java, para que o SVgR-*Server* possa rodar (vide 4.2 SVgR - Server) e, caso este computador possua uma *webcam*, estar rodando em Windows, possuir o Webcam-Recorder (se for gravar vídeo), Wirecast e o *Darwin* instalados e ter um IP real (esses três últimos requisitos apenas para o caso em que ele vá transmitir vídeo).

3 Requisitos Específicos

3.1 Requisitos Funcionais

Para descrever os requisitos funcionais segundo uma análise essencial, é necessário primeiramente listar os eventos mais básicos do sistema:

Lista de Eventos:

- Administrador cadastra usuário
- Administrador remove usuário
- Administrador altera cadastro de usuário
- Administrador lista todos os usuários
- Administrador cadastra computador
- Administrador remove computador
- Administrador altera cadastro de computador
- Administrador lista todos os computadores
- Administrador cadastra nova permissão de usuário sobre computador
- Administrador remove permissão
- Administrador lista todas as permissões
- Administrador lista as permissões de um dado usuário
- Administrador lista os vídeos de um determinado computador
- Administrador lista todos os vídeos
- Administrador assiste a um vídeo de segurança
- Administrador liga um computador
- Administrador desliga um computador
- Administrador realiza logoff de um computador
- Administrador cancela o desligamento de um computador
- Administrador gera o relatório de um computador
- Administrador acessa a câmera de segurança de um computador ao vivo
- Usuário altera seu cadastro
- Usuário lista suas permissões
- Usuário lista os vídeos dos computadores sobre os quais tem permissão
- Usuário assiste ao vídeo de um dos computadores sobre os quais tem permissão
- Usuário liga um computador sobre o qual tem permissão

- Usuário desliga um computador sobre o qual tem permissão
- Usuário realiza logoff de um computador sobre o qual tem permissão
- Usuário cancela o desligamento de um computador sobre o qual tem permissão
- Usuário gera o relatório de um computador sobre o qual tem permissão
- Usuário acessa a câmera de segurança de um computador ao vivo sobre o qual tem permissão

A arquitetura do sistema será a seguinte:

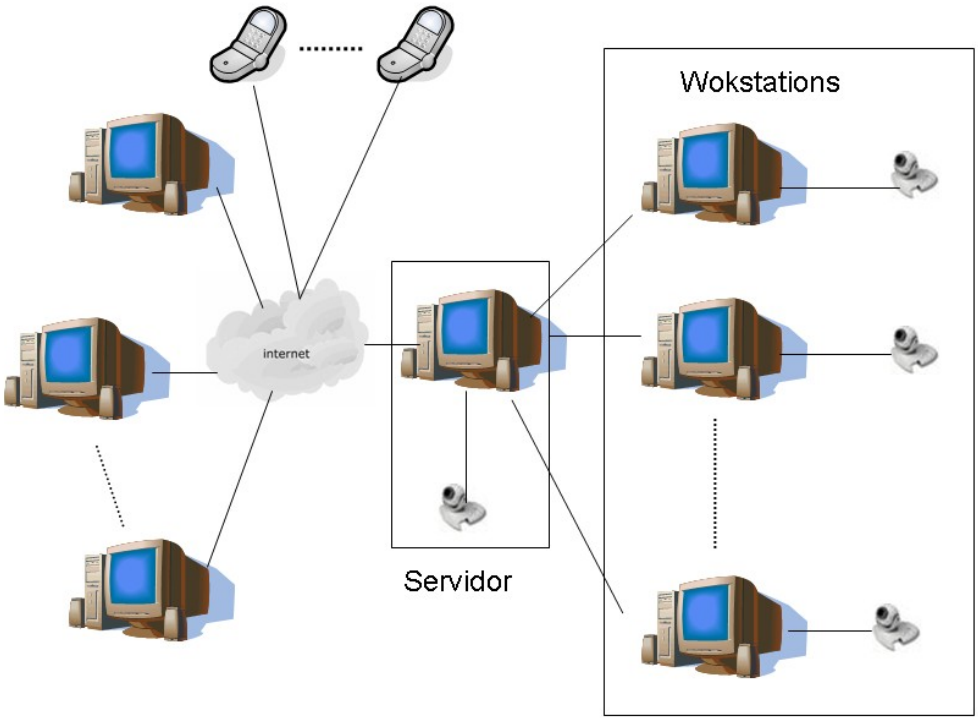

Figura 11: Arquitetura Simplificada do Sistema

Como se pode notar, o sistema é acessível de qualquer computador remoto ou pertencente à rede. O servidor, rodando o apache-tomcat, possui uma *webcam* instalada que é a responsável pro gravar os vídeos de segurança. Todas as máquinas da rede possuem uma *webcam* que provê o acesso ao vivo de um usuário remoto, embora não possuam a capacidade de gravar vídeo. O servidor de banco de dados poderá estar em qualquer computador pertencente à rede.

Cada uma das máquinas deverá estar rodando o SVgR - *Server*, serviço que permite o acesso remoto para execução das ações previstas sobre um computador.

Diagrama de Entidades e Relacionamentos

O sistema usa o modelo lógico de dados que está representado no seguinte DER:

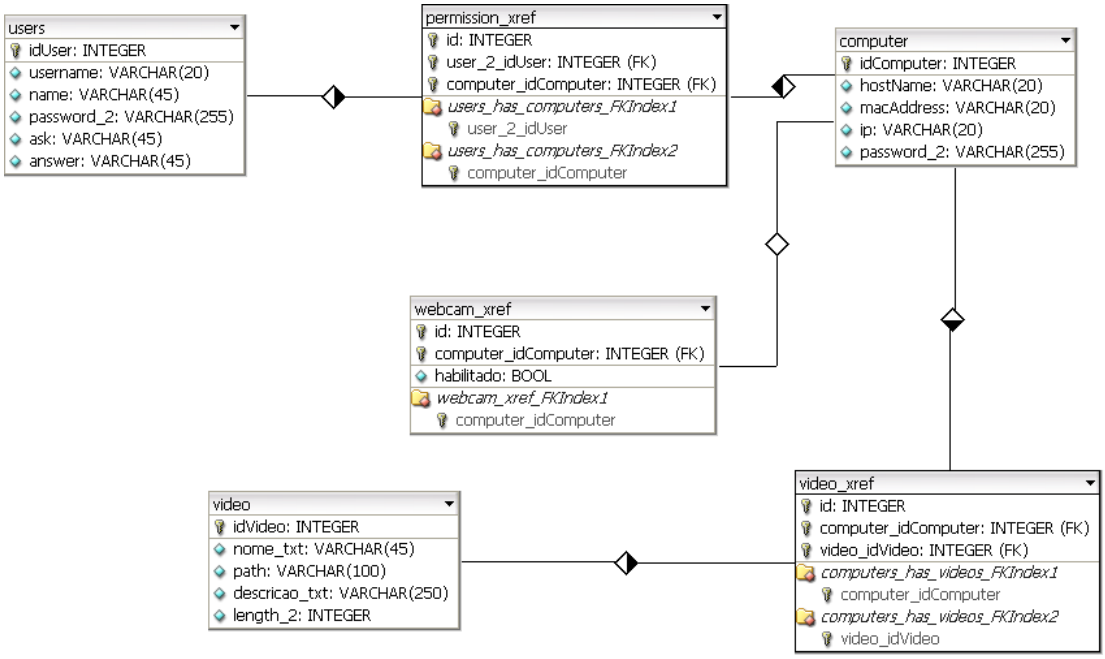

Figura 12: Diagrama de Entidades e Relacionamentos

Para o nosso DER temos o seguinte dicionário de dados:

- **user** – Entidade que define os dados relativos aos usuários
    - **idUser** – código que identifica unicamente um usuário
    - **username** – login da interface de administração
    - **name** – nome do usuário
    - **password** – senha do usuário
    - **ask** – pergunta a ser feita em caso de esquecimento da senha
    - **answer** – resposta à pergunta

- **computers** – Entidade que define os dados relativos aos computadores
    - **idComputer** – código que identifica unicamente um computador
    - **hostName** – hostname do computador
    - **macAddres** – endereço MAC do computador
    - **ip** – endereço IP do computador

    - **password –** senha de desligamento remoto do computador
- **permission_xref –** Entidade associativa que relaciona usuários e computadores
    - **id –** código da permissão
    - **user_idUser –** código do usuário
    - **computer_idComputer –** código do computador

- **video -** Entidade que define os dados relativos aos vídeos
    - **idVideo –** código que identifica unicamente um vídeo
    - **nome_txt –** nome do arquivo de vídeo
    - **path –** caminho do arquivo relativo ao tomcat
    - **descrição_txt –** descrição do vídeo
    - **length –** tamanho em bytes do arquivo

- **video_xref -** Entidade associativa que relaciona vídeos e computadores
    - **id –** código do relacionamento
    - **computer_idComputer** – código do computador
    - **user_idUser** – código do usuário

- **wecam_xref -** Entidade associativa que relaciona *webcam* e computadores
    - **id –** código do relacionamento
    - **computer_idComputer** – código do computador
    - **habilitado** – define se a *webcam* está instalada ou não no computador

Descrição dos DFD's Particionados:

Administrador cadastra usuário

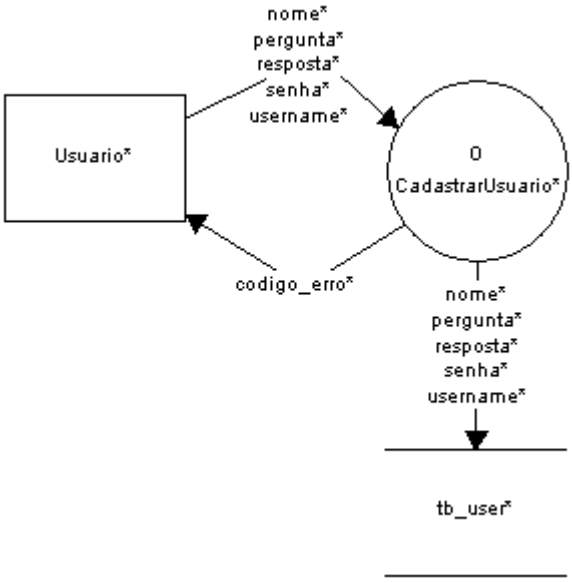

Figura 13: DFD – Administrador Cadastra Usuário

Essa função foi expandida em processos primitivos, mostrados abaixo:

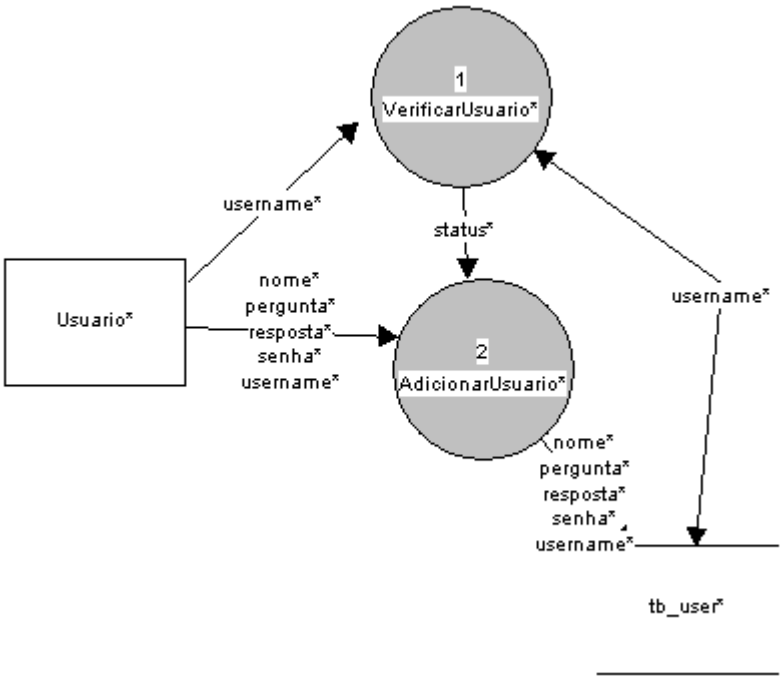

Figura 14: DFD Expandido – Administrador Cadastra Usuário

- VerificarUsuario
    - Terminador: Usuário
    - Depósito de dados: tabela user
    - Objetivos: verificar se um usuário está cadastrado
    - Pré-condições: o username deve ser passado para este processo
    - Fluxo básico:
        * recebe o username
        * procura pelo username no banco de dados
        * se encontrar, retorna status true; senão, false

- AdicionarUsuario
    - Terminador: Usuário
    - Depósito de dados: tabela user
    - Objetivos: cadastrar um novo usuário no sistema
    - Pré-condições: VerificarUsuario deve retornar status "false"
    - Fluxo básico:
        * recebe username, nome, pergunta, resposta e senha
        * adicionar essas informações no banco de dados
        * retorna uma mensagem para o usuário

Administrador remove usuário

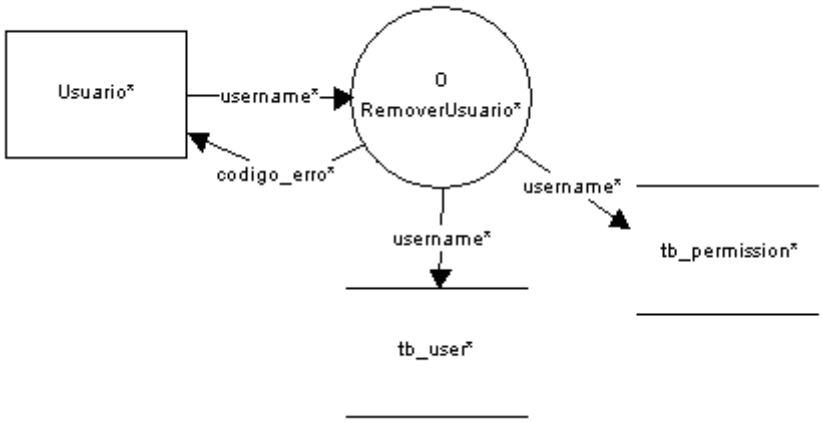

Figura 15: DFD – Administrador Remove Usuário

Essa função foi expandida em processos primitivos, mostrados abaixo:

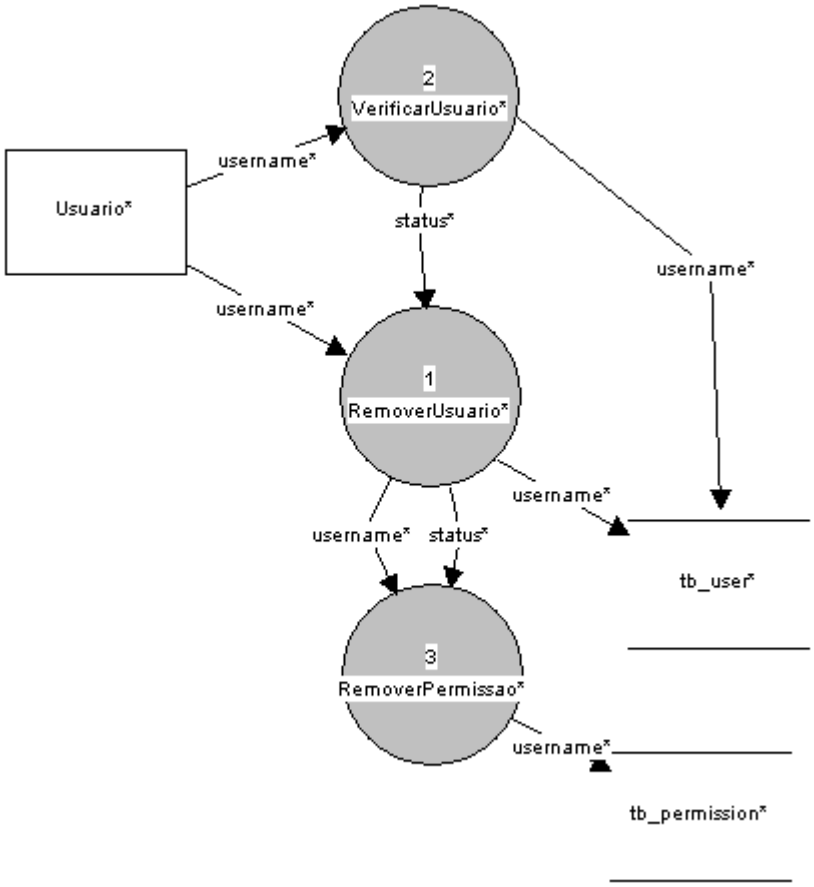

Figura 16: DFD Expandido – Administrador Remove Usuário

- VerificarUsuario: já especificado anteriormente

- RemoverUsuario
  - Terminador: Usuário
  - Depósito de dados: tabela user
  - Objetivos: remover um usuário do sistema
  - Pré-condições: VerificarUsuario deve retornar "true"
  - Fluxo básico:
    - * recebe username
    - * remove username do banco de dados
    - * retorna status de sucesso (ou insucesso)

- RemoverPermissao
  - Terminador: Usuário
  - Depósito de dados: tabela permissões
  - Objetivos: remover a permissão de um usuário sobre um computador
  - Pré-condições: RemoverUsuario deve retornar status de sucesso
  - Fluxo básico:
    - * recebe username
    - * remove todas as permissões associadas a username do banco de dados
    - * retorna uma mensagem para o usuário

Administrador altera cadastro de usuário

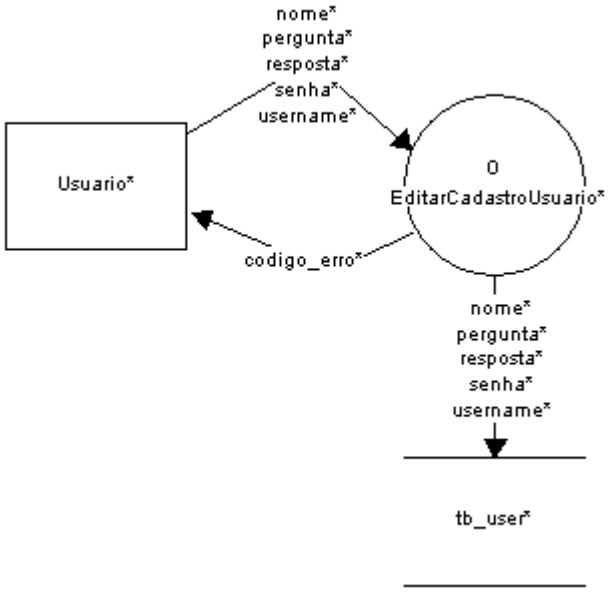

Figura 17: DFD – Administrador Altera Cadastro de Usuário

Essa função foi expandida em processos primitivos, mostrados abaixo:

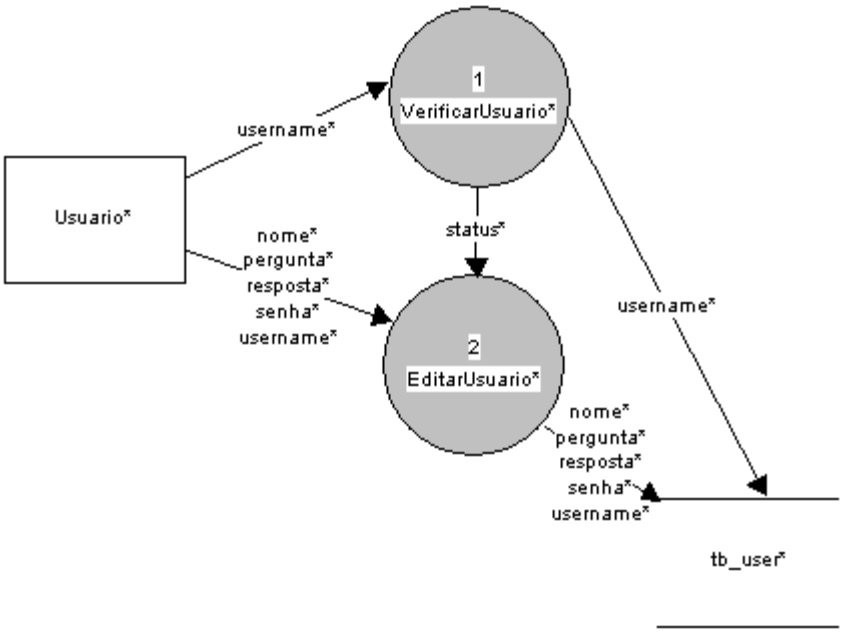

Figura 18: DFD – Administrador Altera Cadastro de Usuário

- VerificarUsuario: já especificado anteriormente

- EditarUsuario
  - Terminador: Usuário
  - Depósito de dados: tabela user
  - Objetivos: editar as informações de cadastro de um usuário do sistema
  - Pré-condições: VerificarUsuario deve retornar "true"
  - Fluxo básico:
    - * recebe username, nome, senha, pergunta e resposta
    - * atualiza as informações do usuário
    - * retorna mensagem para o usuário

Administrador lista todos os usuários

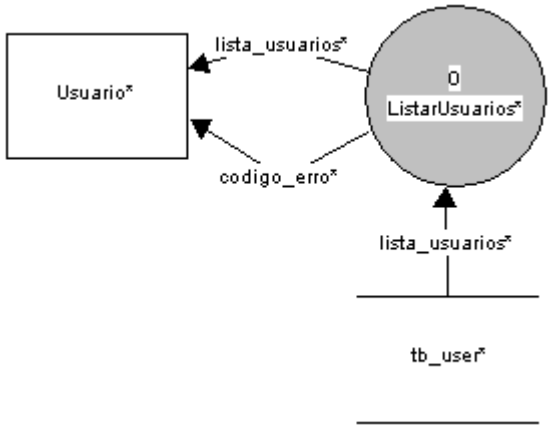

Figura 19: DFD – Administrador Lista Usuários Cadastrados

- ListarUsuarios
  - Terminador: Usuário
  - Depósito de dados: tabela user
  - Objetivos: listar todos os usuários cadastrados no sistema
  - Pré-condições: o usuário deve ser o administrador
  - Fluxo básico:
    - * acessa a tabela, e recebe todas as informações de todos os usuários
    - * mostra a lista para o usuário

Administrador cadastra computador

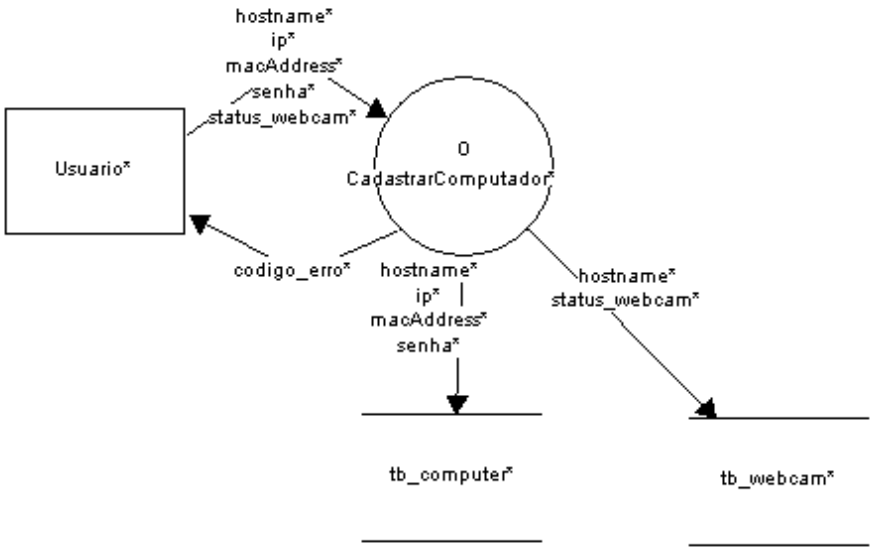

Figura 20: DFD – Administrador Cadastra um Computador

Essa função foi expandida em processos primitivos, mostrados abaixo:

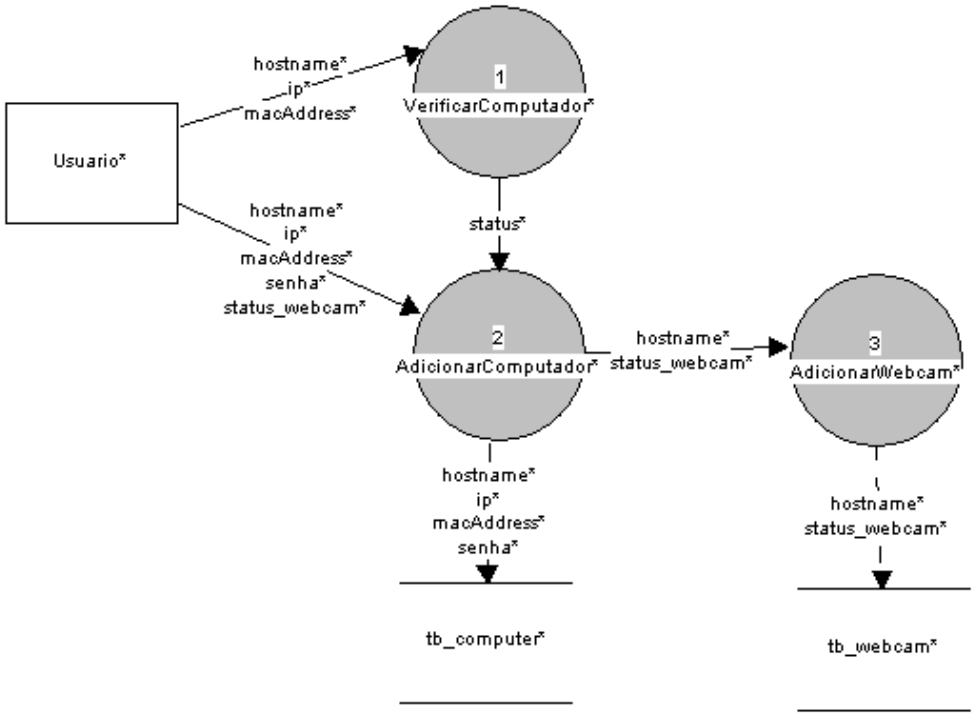

Figura 21: DFD Expandido – Administrador Cadastra um Computador

- VerificarComputador
  - Terminador: Usuario
  - Depósito de dados: tabela computer
  - Objetivos: verificar se um computador está cadastrado no sistema
  - Pré-condições: deve receber hostname, ip ou macAddress
  - Fluxo básico:
    - * pesquisar por hostname, ip ou macAddress no banco de dados
    - * retorna “true” se encontrar ou “false” se não encontrar

- AdicionarComputador
  - Terminador: Usuario
  - Depósito de dados: tabela computer
  - Objetivos: cadastrar um novo computador no sistema
  - Pré-condições: VerificarComputador deve retornar “false”
  - Fluxo básico:
    - * recebe hostname, ip, macAddress e senha do novo computador
    - * retorna “true” se excluir ou “false” se não excluir

- AdicionarWebcam
  - Terminador: Usuario
  - Depósito de dados: tabela *webcam*
  - Objetivos: adicionar uma nova *webcam* no sistema, associada a um computador
  - Pré-condições: AdicionarComputador deve retornar true
  - Fluxo básico:
    - * recebe hostname e status_*webcam*
    - * adiciona uma *webcam* e seu status na tabela *webcam*

Administrador remove computador

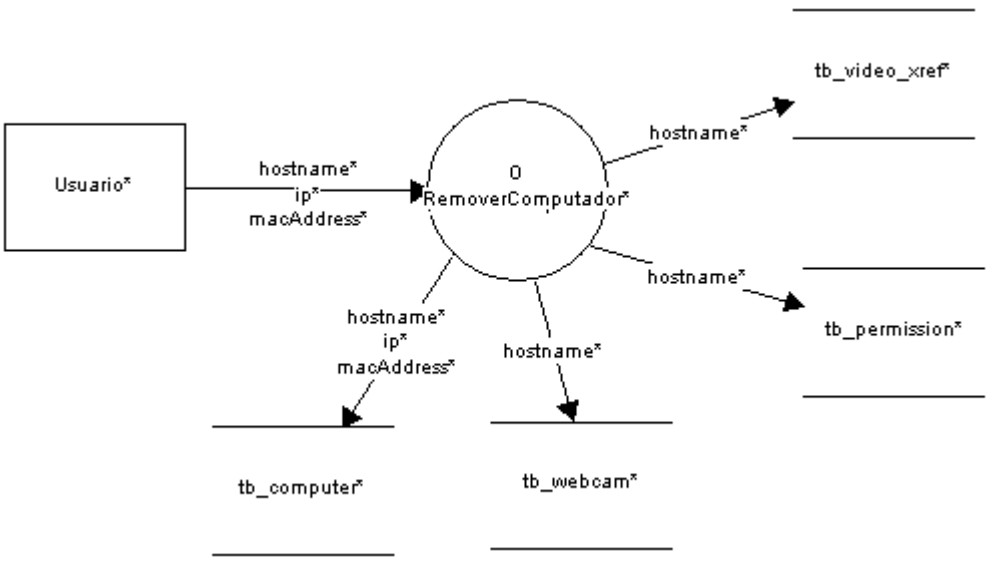

Figura 22: DFD - Administrador Remove Computador

Essa função foi expandida em processos primitivos, mostrados abaixo:

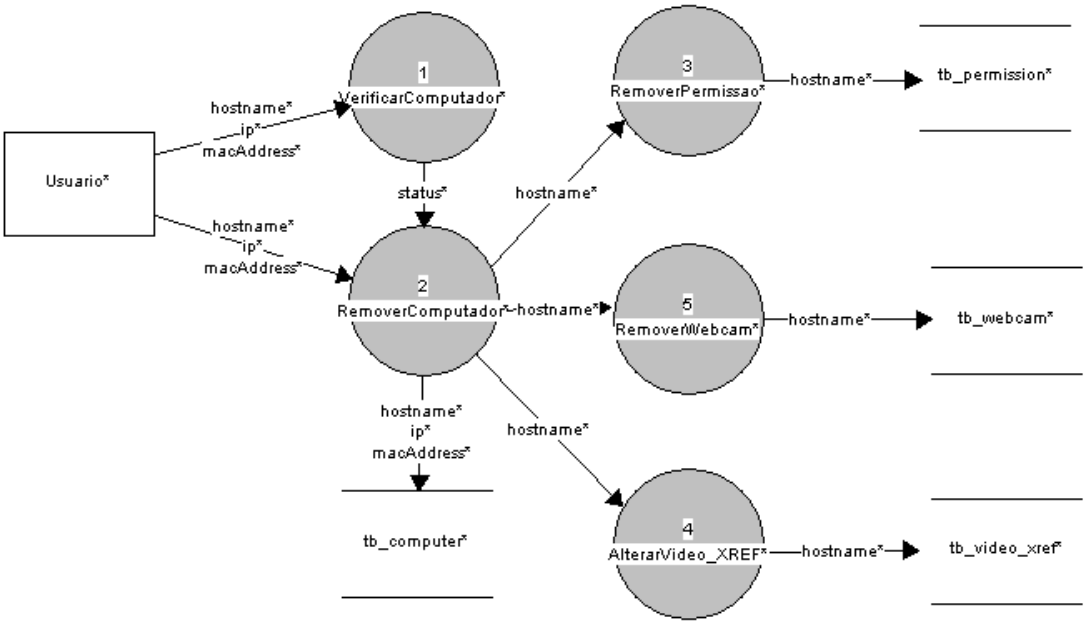

Figura 23: DFD Expandido - Administrador Remove Computador

- VerificarComputador: já especificado anteriormente

- RemoverComputador
  - Terminador: Usuario
  - Depósito de dados: tabela computer
  - Objetivos: remover um computador do sistema
  - Pré-condições: VerificarComputador deve retornar "true"
  - Fluxo básico:
    - * recebe hostname, ip ou macAddress
    - * remove o computador do banco de dados
    - * retorna status "true" ou "false"

- RemoverPermissao: já especificado anteriormente

- RemoverWebcam: já especificado anteriormente

- AlterarVideo_XREF
    - Terminador: Usuário
    - Depósito de dados: tabela video_xref
    - Objetivos: alterar o computador ao qual o vídeo esta associado
    - Pré-condições: RemoverComputer deve retornar “true”
    - Fluxo básico:
        * recebe hostname
        * procura pelo vídeo associado a hostname
        * associa o(s) vídeo(s) encontrados a um novo computador

Administrador altera cadastro de computador

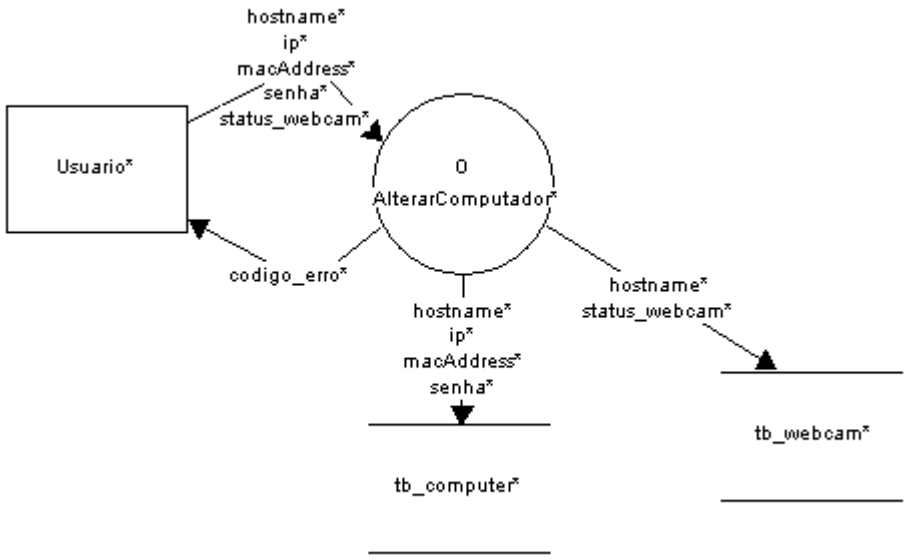

Figura 24: DFD - Administrador Altera Cadastro de Computador

Essa função foi expandida em processos primitivos, mostrados abaixo:

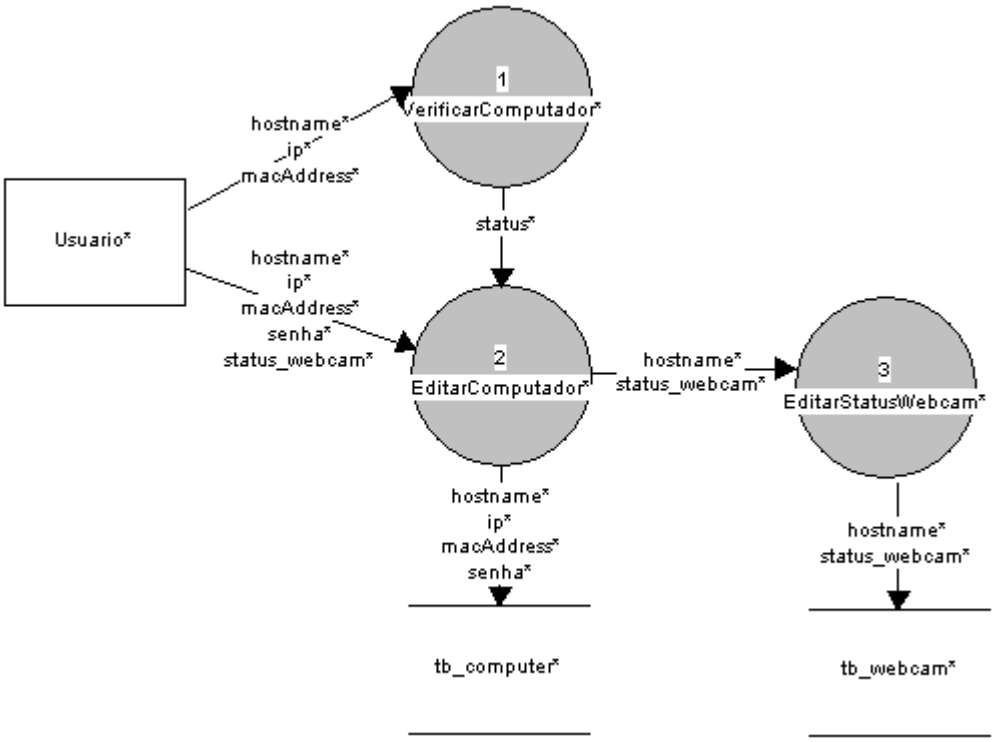

Figura 25: DFD Expandido - Administrador Altera Cadastro de Computador

- VerificarComputador: já especificado anteriormente

- EditarComputador
  - Terminador: Usuário
  - Depósito de dados: tabela computer
  - Objetivos: editar as informações de um computador cadastrado no sistema
  - Pré-condições: VerificarComputador deve retornar “true”
  - Fluxo básico:
    - * recebe hostname, ip, macAddress e senha
    - * atualiza as informações do computador no banco de dados
    - * retorna status “true” ou “false”

- Editar_status_*webcam*

  - Terminador: Usuario

  - Depósito de dados: tabela *webcam*

  - Objetivos: alterar o status (habilitada ou não) de uma *webcam* associada a uma *webcam*

  - Pré-condições: EditarComputador deve retornar "true"

  - Fluxo básico:

    * recebe hostname e o status da *webcam*

    * atualiza o status da *webcam* associada ao hostname

Administrador lista todos os computadores

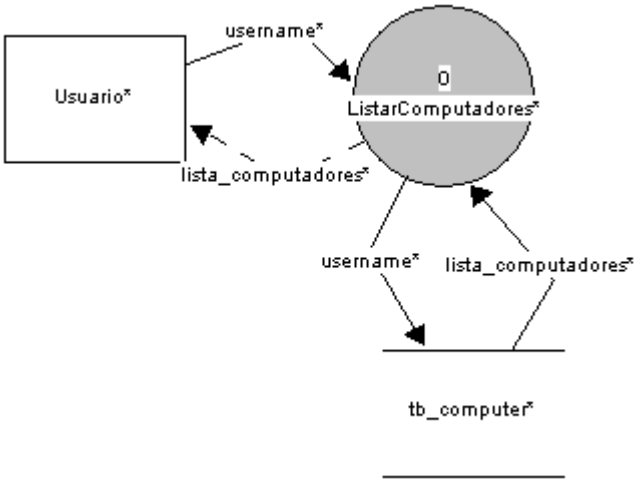

Figura 26: DFD - Administrador Lista Todos os Computadores

- ListarComputadores

  - Terminador: Usuário

  - Depósito de dados: tabela computer

  - Objetivos: listar todos os computadores cadastrados no sistema

  - Pré-condições: não há

  - Fluxo básico:

    * recebe username

    * se username for "admin", retorna todos os computadores; senão, retorna apenas os computadores associados a username.

Administrador cadastra nova permissão de usuário sobre computador

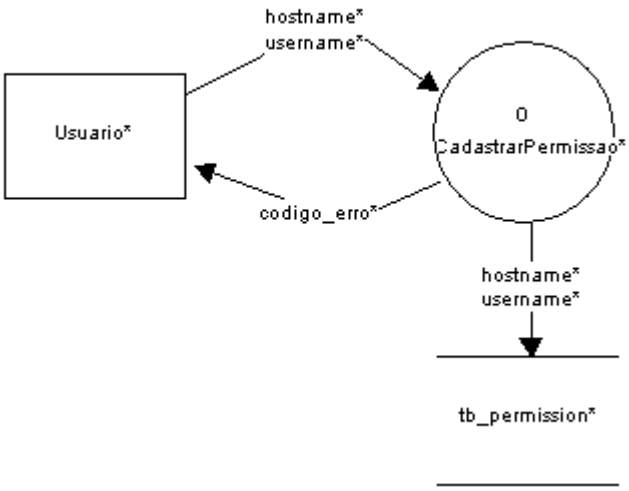

Figura 27: DFD - Administrador Cadastra Nova Permissão

Essa função foi expandida em processos primitivos, mostrados abaixo:

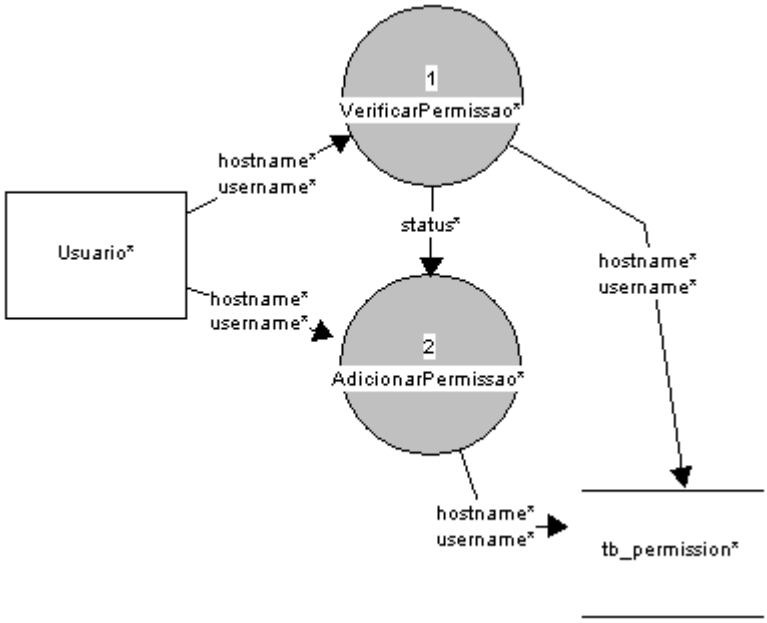

Figura 28: DFD Expandido - Administrador Cadastra Nova Permissão

- VerificarPermissao

  - Terminador: Usuário

  - Depósito de dados: tabela permission

  - Objetivos: verificar se a permissão daquele usuário sobre aquela máquina está cadastrada

- Pré-condições: deve receber hostname e username
- Fluxo básico:
    * recebe hostname e username
    * faz uma pesquisa no banco de dados por hostname e username
    * retorna “true” se achar e “false” caso contrário

- AdicionarPermissao
    - Terminador: Usuário
    - Depósito de dados: tabela permission
    - Objetivos: cadastrar a permissão daquele usuário sobre aquele computador
    - Pré-condições: VerificarPermissao deve retornar “false”
    - Fluxo básico:
        * recebe username e hostname
        * adiciona uma linha contendo username e hostname no banco de dados
        * retorna uma mensagem de erro/sucesso para o usuário

Administrador remove permissão

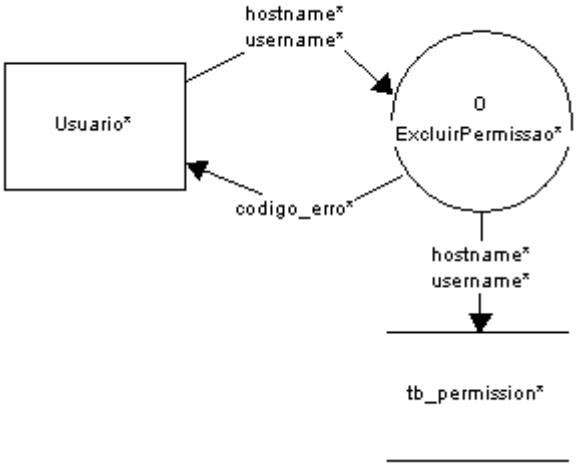

Figura 29: DFD - Administrador Remove Permissão

Essa função foi expandida em processos primitivos, mostrados abaixo:

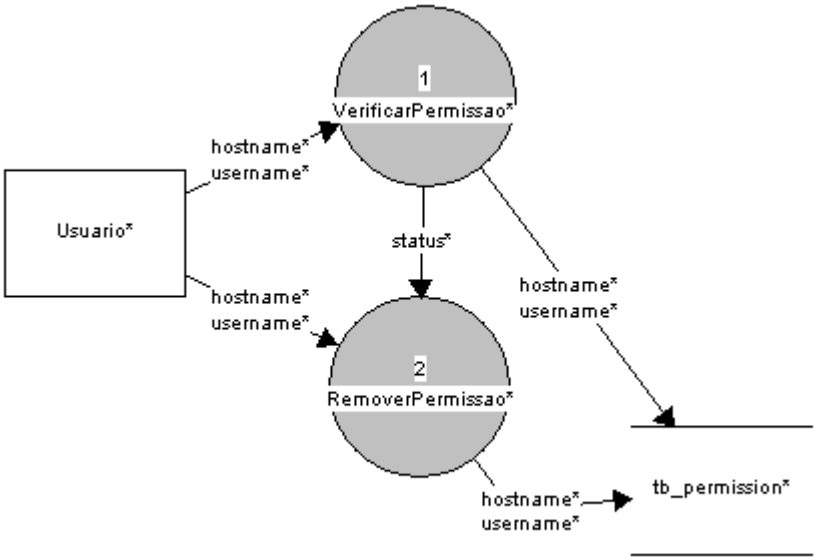

Figura 30: DFD Expandido - Administrador Remove Permissão

- VerificarPermissao: já especificado anteriormente

- RemoverPermissao
  - Terminador: Usuario
  - Depósito de dados: tabela permission
  - Objetivos: remover a permissão daquele usuário sobre aquele computador
  - Pré-condições: VerificarPermissao retorna "true"
  - Fluxo básico:
    - * recebe username e hostname
    - * remove a linha contendo username e hostname do banco de dados
    - * retorna uma mensagem de erro/sucesso para o usuário

Administrador lista todas as permissões

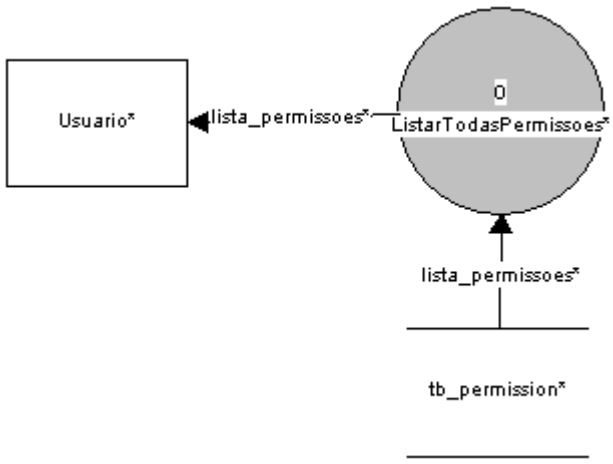

Figura 31: DFD - Administrador Lista Todas as Permissões

- ListarTodasPermissoes
  - Terminador: Usuario
  - Depósito de dados: tabela permission
  - Objetivos: listar todas as permissões cadastradas no sistema
  - Pré-condições: o usuário deve ser o administrador
  - Fluxo básico:
    - * acessa o banco de dados e retorna todas as linhas da tabela permission
    - * retorna a lista de permissões para o usuário

Administrador lista as permissões de um dado usuário

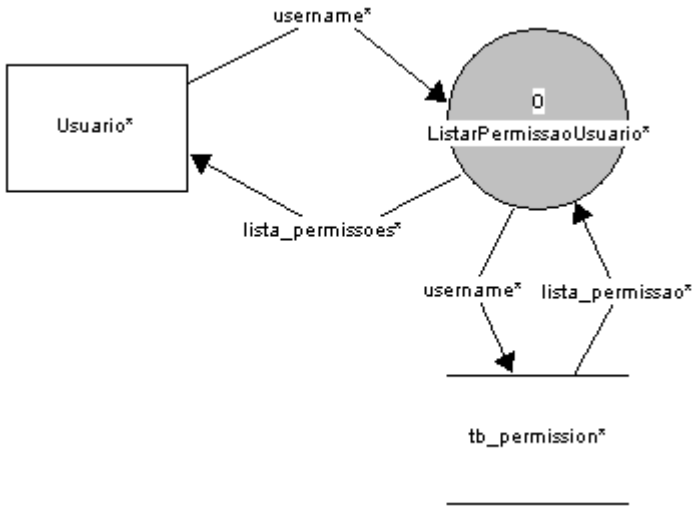

Figura 32: DFD - Administrador Lista as Permissões de um Usuário

- ListarPermissaoUsuario
  - Terminador: Usuario
  - Depósito de dados: tabela permission
  - Objetivos: listar todos os computadores sobre os quais aquele usuário tem permissão
  - Pré-condições: o username deve existir
  - Fluxo básico:
    - * recebe username
    - * acessa o banco e obtém todas as linhas da tabela
    - * retorna a lista para o usuário

Administrador lista os vídeos de um determinado computador

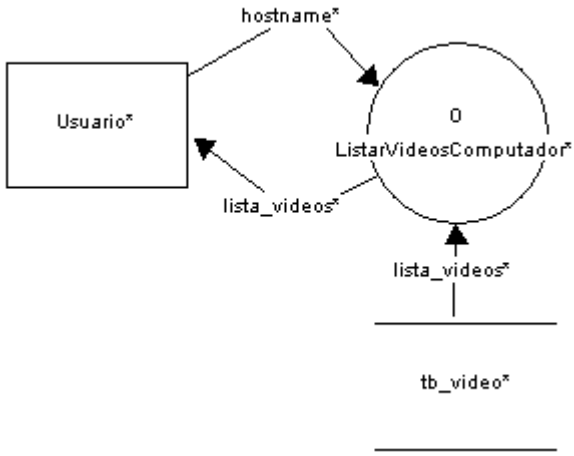

Figura 33: DFD - Administrador Lista os Vídeos de um Determinado Computador

Essa função foi expandida em processos primitivos, mostrados abaixo:

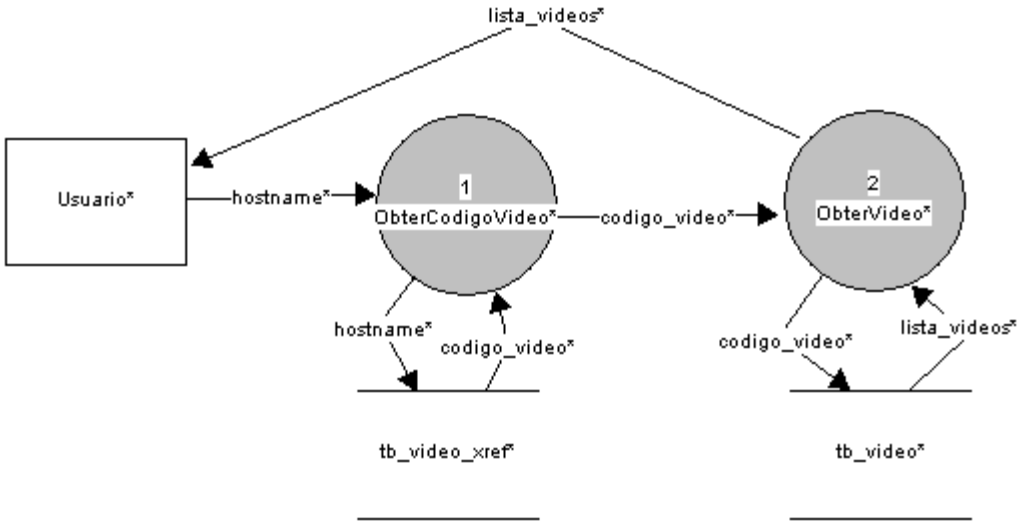

Figura 34: DFD Expandido - Administrador Lista os Vídeos de um Computador

- ObterCodigoVideo
  - Terminador: Usuario
  - Depósito de dados: tabela video_xref
  - Objetivos: obter o código do vídeo desejado
  - Pré-condições: o computador referente a hostname deve existir
  - Fluxo básico:
    - * recebe username e busca na tabela o código de vídeo referente a username
    - * retorna o código de vídeo encontrado

- ObterVideo
  - Terminador: Usuario
  - Depósito de dados: tabela video
  - Objetivos: obter o vídeo desejado
  - Pré-condições: deve receber o código de vídeo
  - Fluxo básico:
    - * recebe o código de vídeo e busca as informações referentes àquele vídeo na tabela
    - * retorna para o usuário a lista com as informações referentes aos vídeos encontrados

Administrador lista todos os vídeos

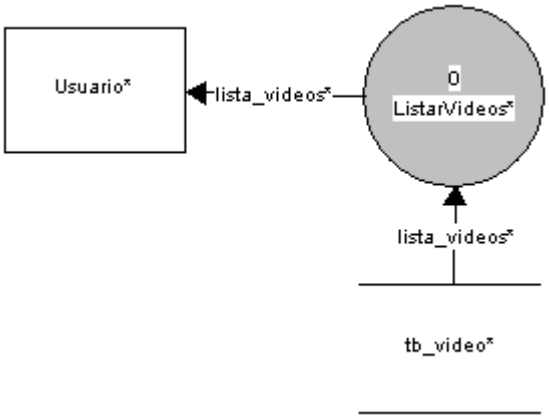

Figura 35: DFD - Administrador Lista Todos os Vídeos

- ListarVideos
  - Terminador: Usuario
  - Depósito de dados: tabela video
  - Objetivos: listar todos os vídeos cadastrados no sistema
  - Pré-condições: o usuário deve ser o administrador
  - Fluxo básico:
    - * busca no banco todos os vídeos cadastrados
    - * retorna a lista de todos os vídeos encontrados

Administrador assiste a um vídeo de segurança

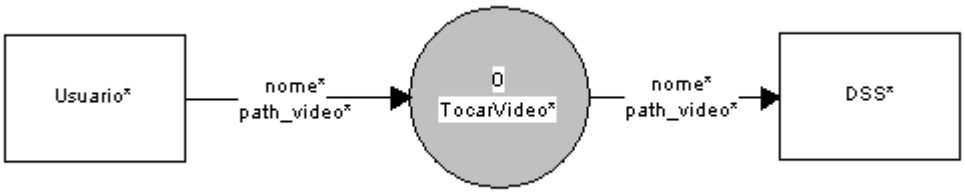

Figura 36: DFD - Administrador Assiste a um Vídeo de Segurança

- TocarVideo

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: tocar o vídeo especificado

  - Pré-condições: o vídeo deve estar armazenado no path indicado, com o nome especificado

  - Fluxo básico:

    * recebe o path e o nome, e passa para o *player* esse path (relativo à instalação do tomcat)

    * *player* começa a tocar o vídeo

Administrador liga um computador

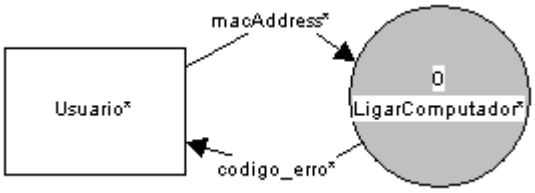

Figura 37: DFD – Administrador Liga um Computador

- LigarComputador

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: ligar um computador remotamente

  - Pré-condições: o computador com macAddress deve estar na rede e estar com a função WAKE-UP ON LAN ativa

  - Fluxo básico:

    * recebe macAddress

    * envia um pacote através do protocolo WOL para broadcast

    * o computador com macAddress ligará

    * retorna uma mensagem de erro/sucesso para o usuário

Administrador desliga um computador

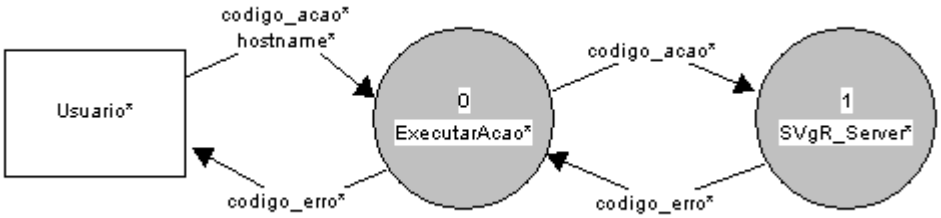

Figura 38: DFD - Administrador Desliga um Computador

- ExecutarAcao

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: desligar um computador remotamente

  - Pré-condições: o computador hostname deverá estar com o serviço SVgR-*Server* ativo

  - Fluxo básico:

    * recebe codigo_acao referente ao desligamento

    * inicia a transação com o computador hostname

    * envia o pedido de desligamento para hostname

    * retorna uma mensagem de erro/sucesso para o usuário

Administrador realiza logoff de um computador

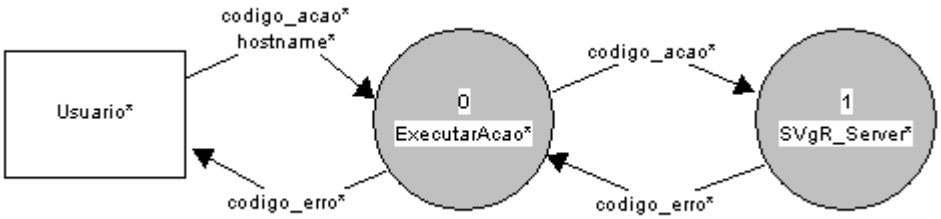

Figura 39: DFD - Administrador Realiza Logoff de um Computador

- ExecutarAcao

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: realizar o logoff de um computador remotamente

  - Pré-condições: o computador hostname deverá estar com o serviço SVgR-*Server* ativo

  - Fluxo básico:

    * recebe codigo_acao referente ao logoff

    * inicia a transação com o computador hostname

    * envia o pedido de logoff para hostname

    * retorna uma mensagem de erro/sucesso para o usuário

Administrador cancela o desligamento de um computador

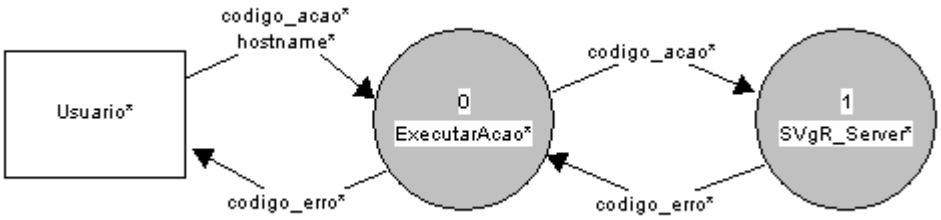

Figura 40: DFD - Administrador Cancela o Desligamento de um Computador

- ExecutarAcao

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: cancelar o desligamento de um computador remotamente

  - Pré-condições: o computador hostname deverá estar com o serviço SVgR-*Server* ativo

  - Fluxo básico:

    * recebe codigo_acao referente ao cancelamento do desligamento

    * inicia a transação com o computador hostname

    * envia o pedido de cancelamento para hostname

    * retorna uma mensagem de erro/sucesso para o usuário

Administrador gera o relatório de um computador

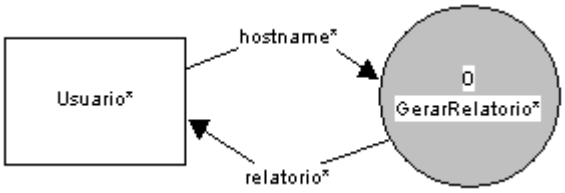

Figura 41: DFD - Administrador Gera o Relatório de um Computador

- GerarRelatorio

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: gerar um relatório com as informações de sistema de um computador remoto

  - Pré-condições: o computador hostname deverá estar com o serviço SVgR-*Server* ativo

  - Fluxo básico:

    * recebe codigo_acao referente ao pedido de relatório
    * inicia a transação com o computador hostname
    * envia o pedido de relatório para hostname
    * retorna uma mensagem de erro/sucesso para o usuário

Administrador acessa a câmera de segurança de um computador ao vivo

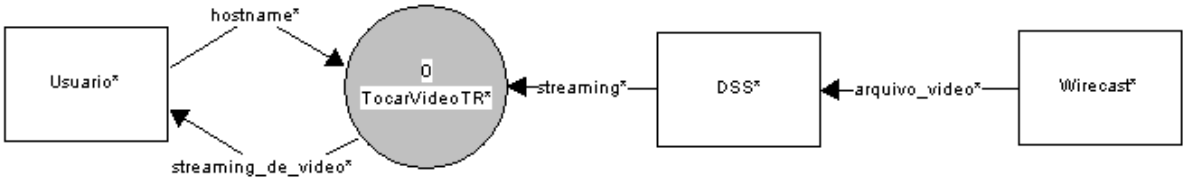

Figura 42: DFD – Administrador Acessa a Câmera de Segurança de um Computador

- TocarVideoTR

  - Terminador: Usuario

  - Depósito de dados: não há

  - Objetivos: assistir ao vídeo de segurança de um computador remoto em tempo real

  - Pré-condições:

  * o servidor *Darwin* deverá estar ativo

  * a *webcam* deverá estar ligada

  - Fluxo básico:

    * recebe hostname

    * se conecta através do protocolo RTSP ao computador hostname

3.2 Interfaces Externas

3.2.1 Interfaces dos Usuários

A primeira interface com a qual o usuário se depara é a tela de login. Ela possui dois campos: username e password.

Figura 43: Página Inicial - WEB

Figura 44: Página Inicial - WAP

Se o usuário não se lembrar de sua senha, ele acessa o menu "Esqueceu sua senha?", logo abaixo do campo de login. Nessa interface, o usuário digitar seu username, e recebe a pergunta cadastrada. Ele então, digita a resposta para a pergunta e, em de acerto, ele recebe uma nova senha.

O usuário digita essas informações e, se o login for aceito, ele é enviado para a página principal da interface de administração.

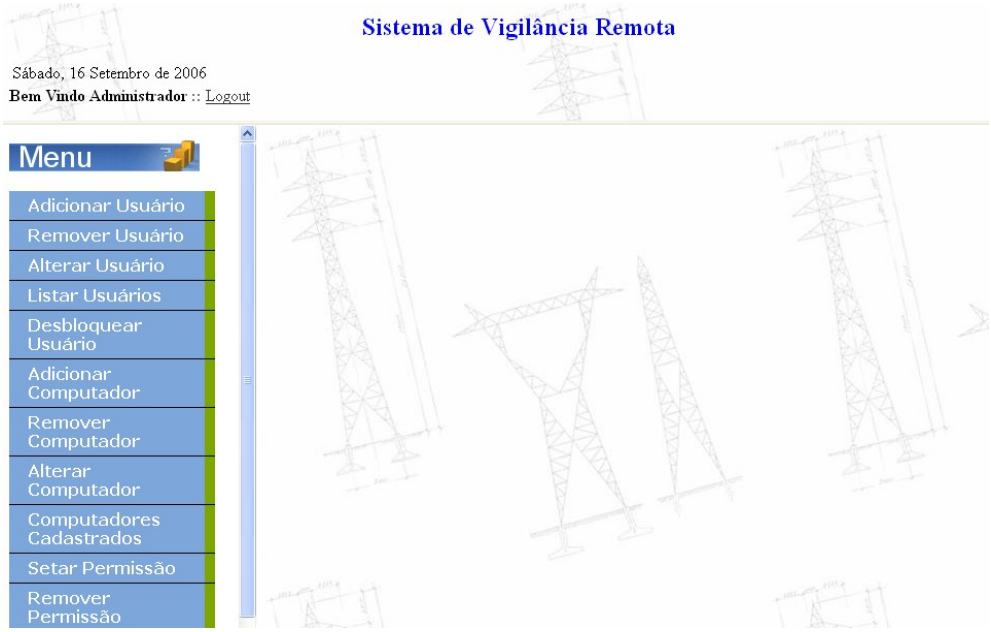

Figura 45: Página Principal do Sistema - WEB

Nessa página, ele terá acesso a todas as funcionalidades previstas, caso ele seja o usuário administrador. Senão, as funcionalidades serão limitadas. A seguir, todas as interfaces disponíveis serão descritas.

Figura 46: Página Principal do Sistema - WAP

3.2.1.1 Adicionar Usuário

Nesta interface, o administrador entra com os dados do novo usuário: nome, username, senha, pergunta e resposta (caso o usuário esqueça a senha).

Figura 47: Adicionar Usuário

3.2.1.2 Remover Usuário

Nesta interface, o administrador digita o username do usuário a ser removido.

Figura 48: Remover Usuário

3.2.1.3 Alterar Usuário

Esta interface é idêntica àquela de incluir usuário. A diferença está quando um usuário comum está acessando esta funcionalidade, pois ele só pode alterar seus dados. Assim, o campo username não estará disponível, evitando que ele altere outro usuário. Já o usuário administrador visualizará esse campo.

3.2.1.4 Adicionar Computador

Nesta interface, o administrador entra com o IP, hostname, endereço MAC e senha do computador a ser cadastrado, e informa ao sistema se o computador terá ou não *webcam* associada a ele.

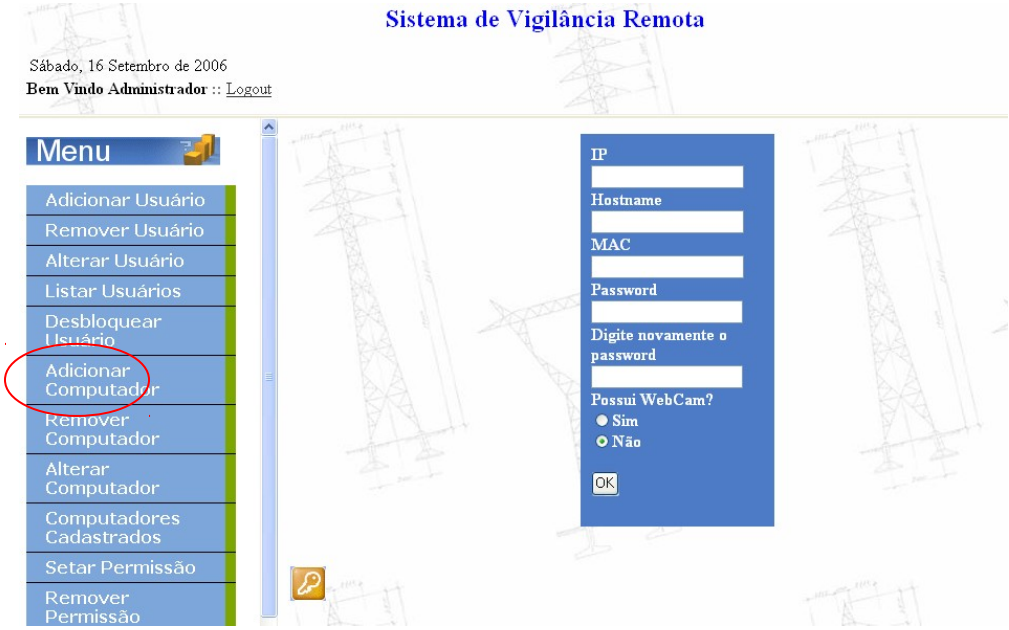

Figura 49: Adicionar Computador

3.2.1.5 Remover Computador

O administrador deverá digitar qualquer uma das três informações que definem o computador a ser removido: IP, endereço MAC ou hostname.

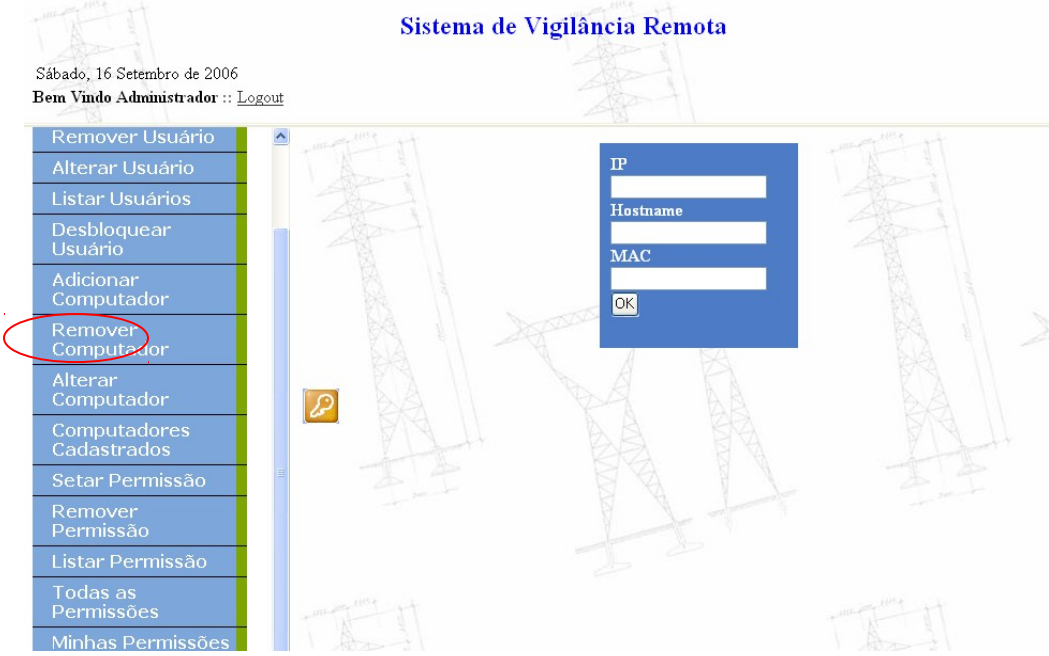

Figura 50: Remover Computador

3.2.1.6 Alterar Computador

Esta interface é idêntica a interface de inclusão de computador.

3.2.1.7 Lista Computadores

Não há entradas de dados. Basta o usuário clicar no item de menu destacado.

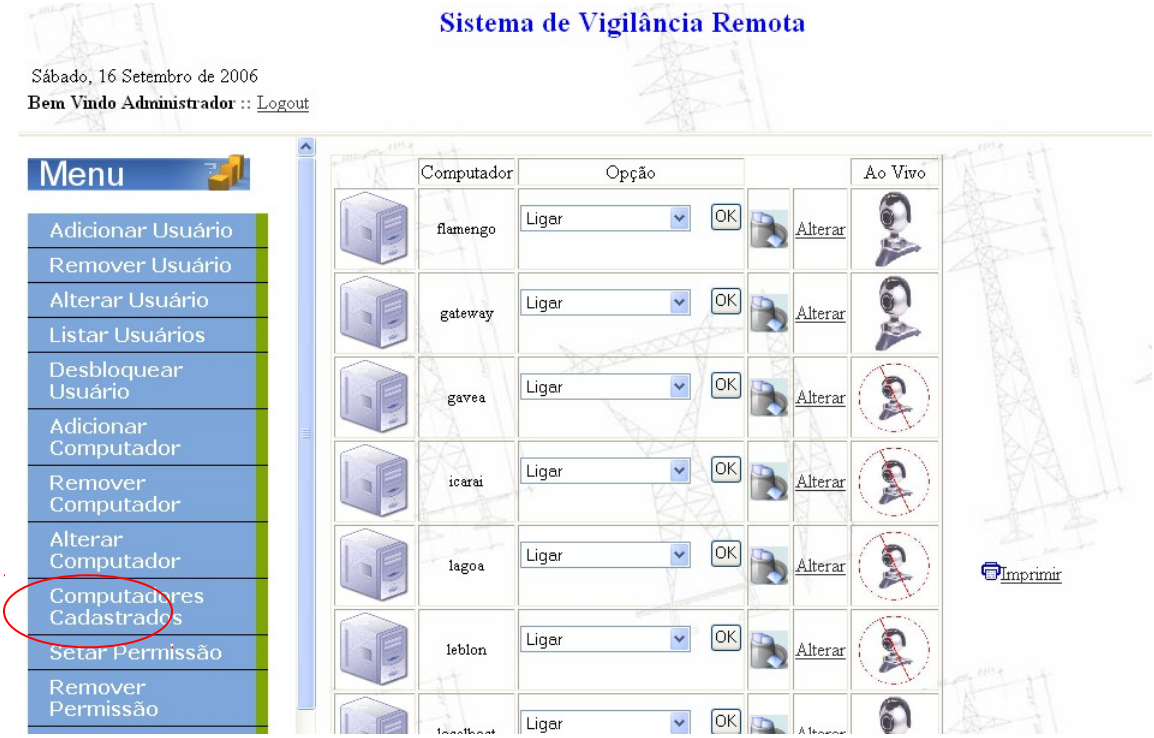

Figura 51: Listar Computadores - WEB

O botão "Imprimir" na direita da tela chama o serviço de impressão do sistema operacional.

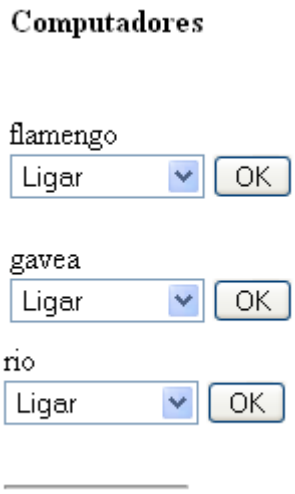

Figura 52: Listar Computadores - WAP

3.2.1.8 Listar Usuários Cadastrados

Não há entradas de dados. Basta o administrador clicar no item de menu destacado.

Figura 53: Listar Usuários

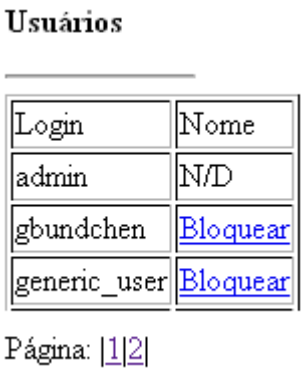

Figura 54: Listar Usuários - WAP

3.2.1.9 Definir Permissão de Usuário sobre Computador

O administrador digita o username e o computador para o qual ele deseja dar a permissão.

Figura 55: Definir Permissão

3.2.1.10 Remover Permissão

Esta interface é idêntica a de inclusão de uma permissão.

3.2.1.11 Listar Permissão de um Usuário Específico

O administrador digita o username do usuário a ser pesquisado.

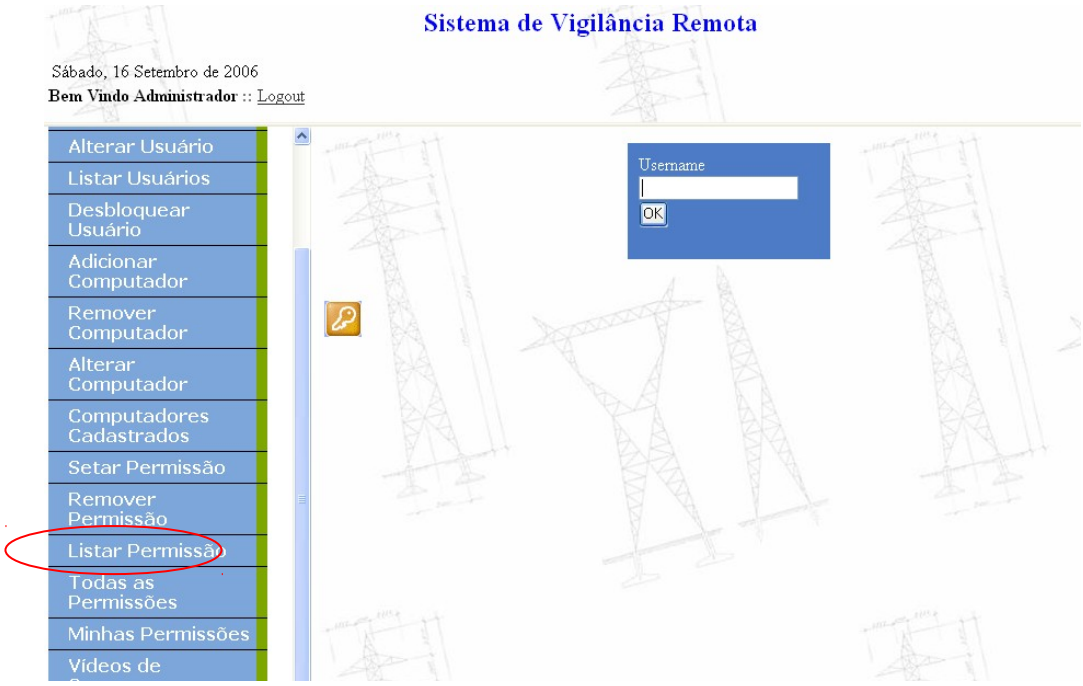

Figura 56: Remover Permissão

3.2.1.12 Listar Todas as Permissões Definidas

Não há entradas de dados. Basta o administrador clicar no item de menu destacado.

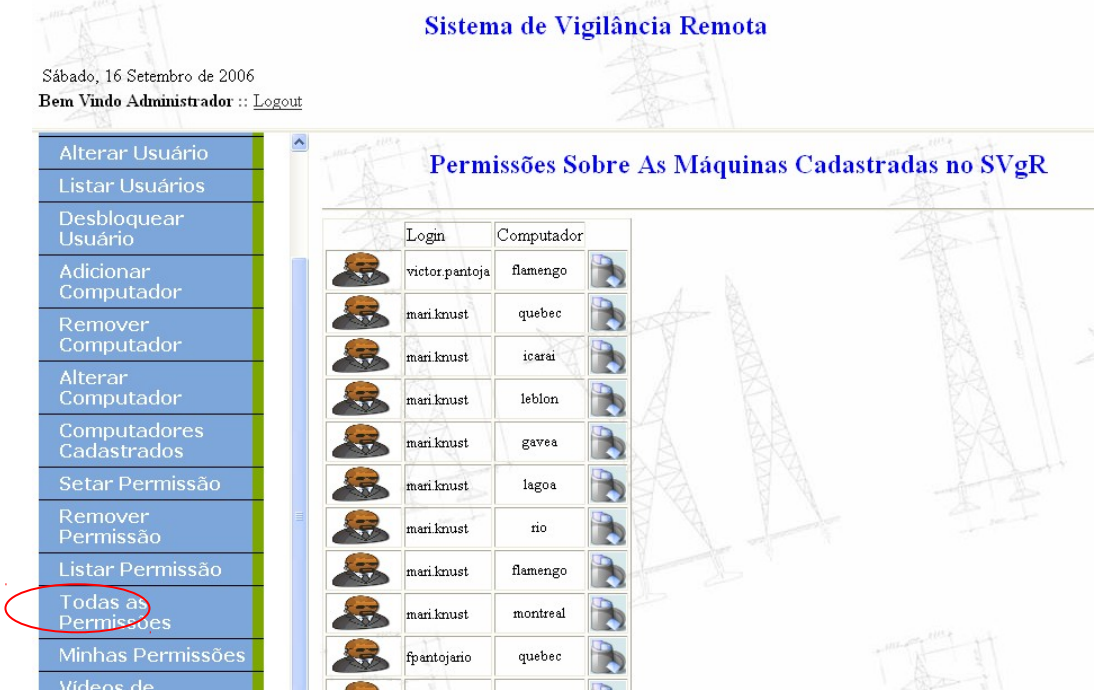

Figura 57: Listar Todas as Permissões

3.2.1.13 Listar os Vídeos Gerados por um Computador Específico

O administrador digita o hostname desejado.

Figura 58: Listar os Vídeos de Segurança de um Computador

3.2.1.14 Lista de Vídeos Cadastrados

Não há entradas de dados. Basta o administrador clicar no item de menu destacado. O usuário comum também pode executar essa ação, mas só visualizará os vídeos de segurança das máquinas sobre as quais ele tem permissão.

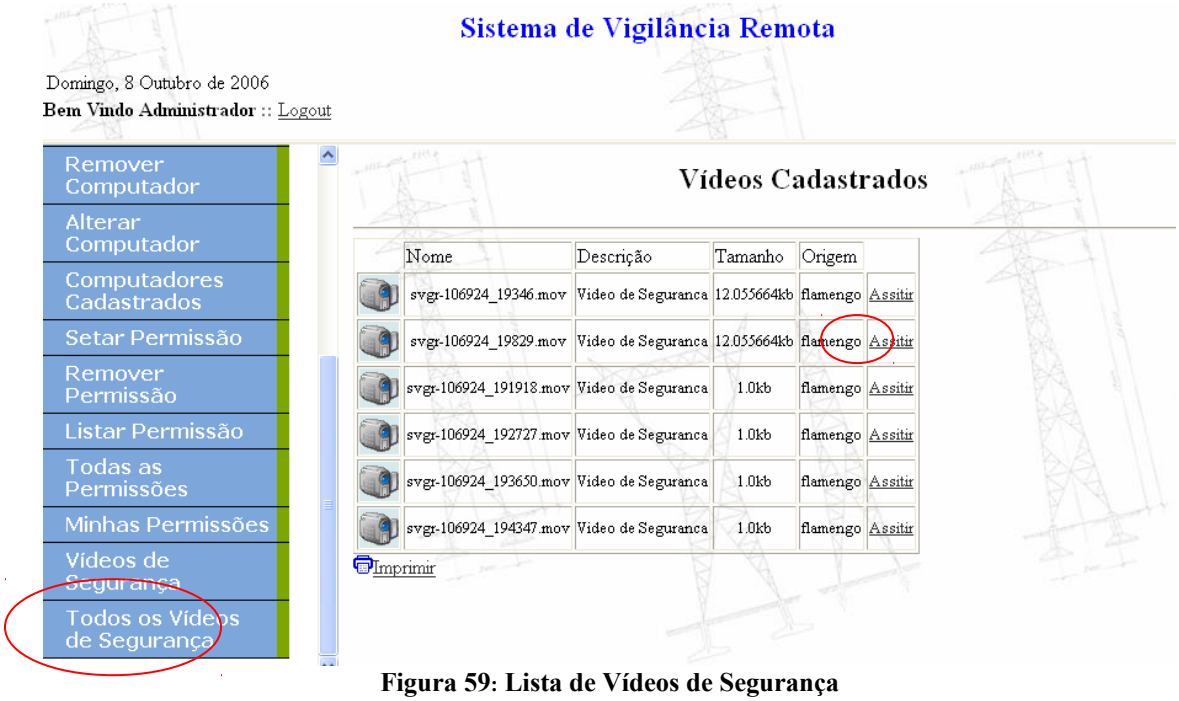

Figura 59: Lista de Vídeos de Segurança

3.2.1.15 Interface de Visualização do Vídeo

O usuário/administrador deve clicar no link "Assistir", conforme destacado na figura 12.

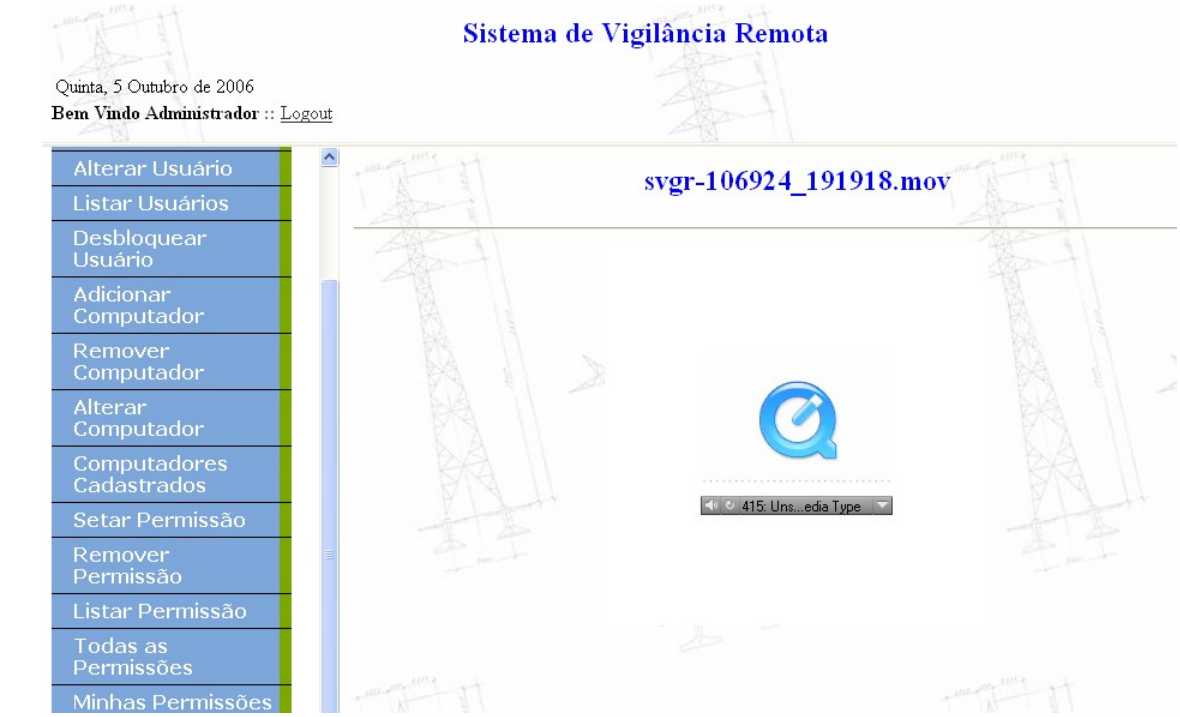

Figura 60: Assistir um Vídeo

3.2.1.16 Ações sobre um Computador

Apenas um detalhamento da figura 6.

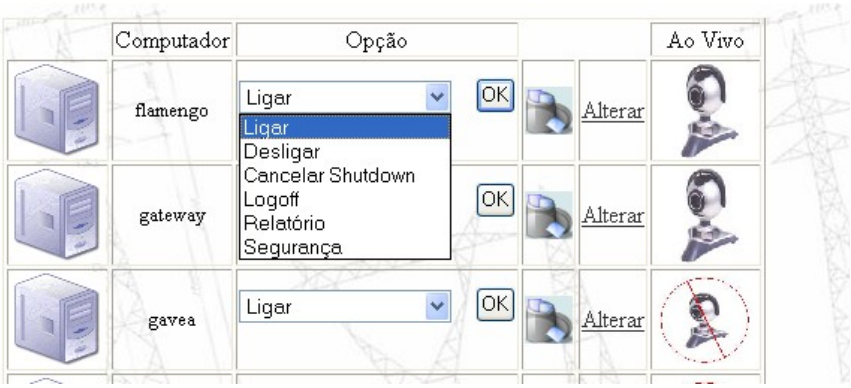

Figura 61: Funcionalidades sobre um Computador

3.2.1.17 Telas de Sucesso e Erro

O usuário/administrador verá essas telas sempre que obtiver sucesso ou insucesso, respectivamente em uma ação executada.

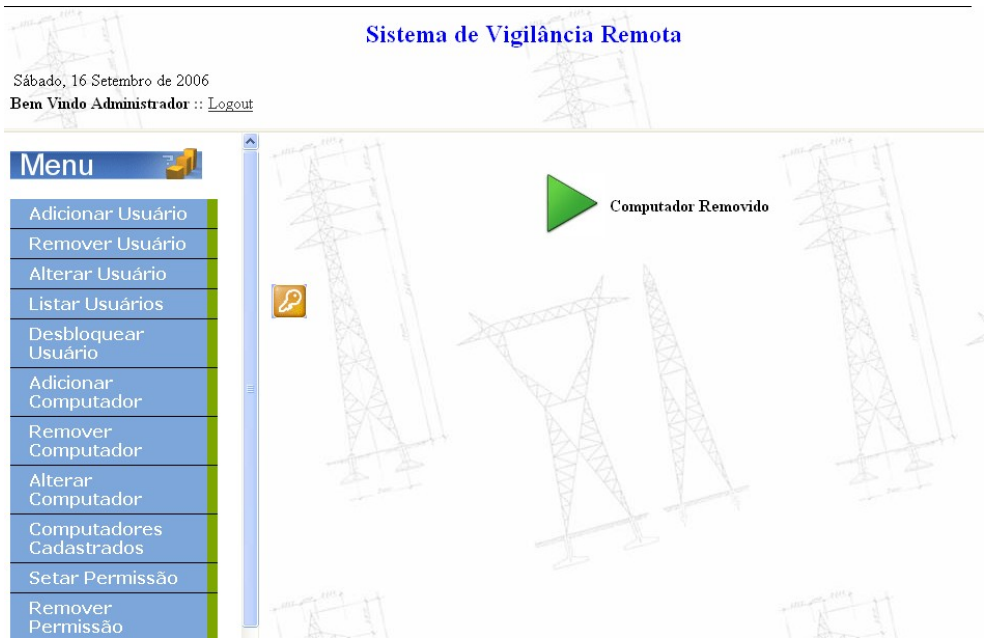

Figura 62: Tela de Sucesso

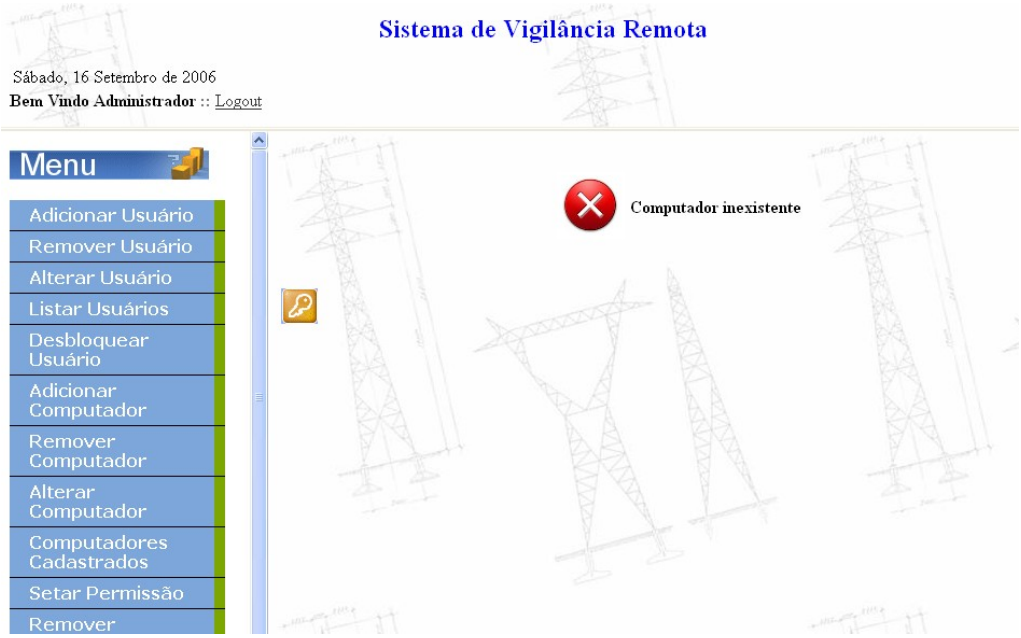

Figura 63: Tela de Erro

3.2.1.18 SVgR – Server

Interface gráfica do serviço que fica rodando nas workstations.

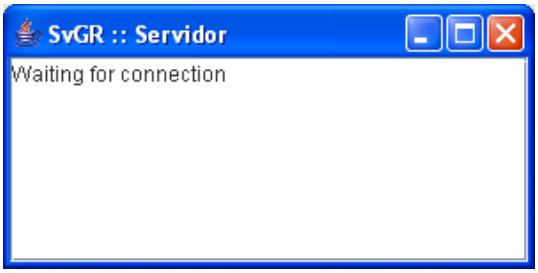

Figura 64: Tela do SVgR - *Server*

3.2.1.19 Tela de Relatório

Tela que o usuário/administrador acessa ao escolher a opção "Relatório" na tela da figura 14.

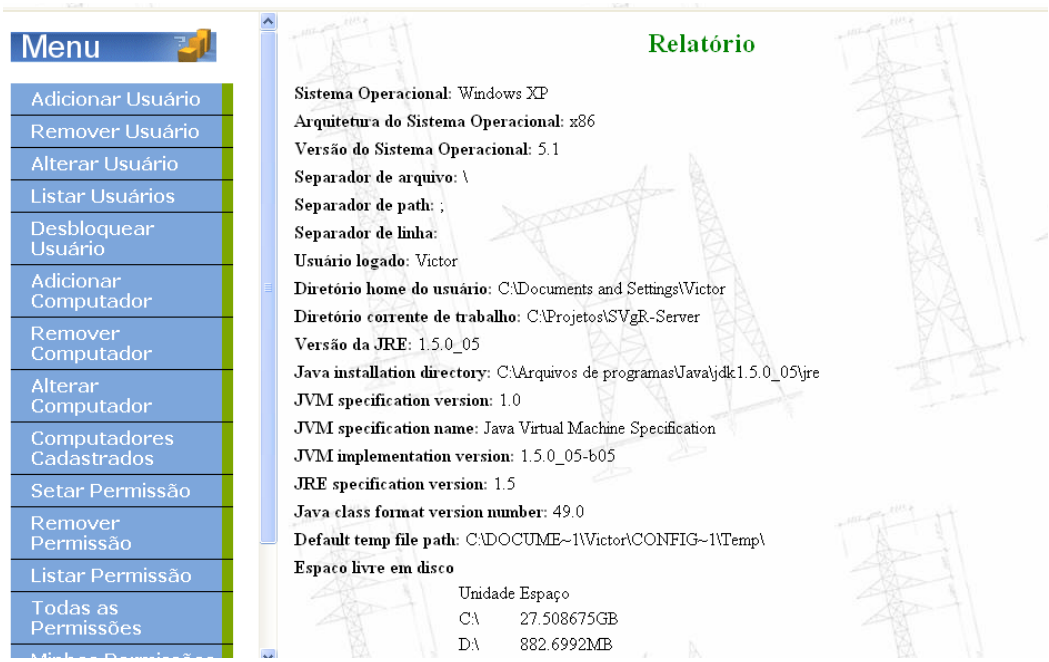

Figura 65: Visualização do Relatório

3.2.1.20 Interface de Gravação de Vídeo da Webcam

Interface gráfica o aplicativo que captura e grava a imagem gerada pela *webcam* e a tela para salvar o arquivo que está sendo gravado, respectivamente.

Figura 66: Tela Principal do SvgR WebCam Recorder

3.2.2 Interfaces de Hardware

Não se aplica.

3.2.3 Interfaces de *Software*s

O aplicativo possui uma interface com o *Darwin Streaming Server*. Essa interface é bastante simples: basta os arquivos de vídeo serem salvos na pasta /movies que o vídeo estará automaticamente pronto para ser transmitido.

3.2.4 Interfaces de Comunicação

O *software* utiliza uma rede Ethernet para comunicação cliente – servidor, isto é, entre o servidor principal (rodando a interface de administração) e o serviço rodando na *workstation*.

3.3 Requisitos de Desempenho

O maior requisito deste projeto é o desempenho na transmissão dos pacotes de *streaming*. Deve-se garantir uma boa qualidade no sinal de vídeo recebido da *webcam*.

3.4 Restrições de Projeto

Deve-se utilizar o formato de vídeo .mov (Quick Time) pois é o formato utilizado pelo *Darwin Streaming Server*. Além disso, o acesso à *webcam* só poderá ser feito em computadores rodando Windows, pois ela não é suportada pelo Linux. Este acesso poderá ser apenas para gravação ou transmissão ao vivo.

3.5 Atributos

- Amigabilidade – para facilitar a administração, precisa ter um ambiente agradável para o usuário, que o faça se sentir bem navegando.
- Segurança – O sistema precisa ser seguro, uma vez que ele guardará informações cadastrais dos usuários, computadores, vídeos etc.
- Portabilidade – Uma vez que o sistema não possui nenhum requisito que o obrigue a rodar apenas sobre Windows ou Linux, ele pode ser compilado em qualquer plataforma fazendo pequenas adaptações. É apenas uma questão de se encontrar o

driver para a *webcam*, e instalar as versões adequadas do servidor de banco de dados e do servidor de *streaming*.

- Manutenibilidade – A manutenibilidade do *software* está muito mais relacionada com a atualização do cadastro de máquinas do que com o *software* em sim.

3.6 Outros Requisitos

Não há outros requisitos de *software* além dos que já foram citados no documento.