

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**SOFTWARE PARA O MONITORAMENTO DE
SEGURANÇA ELETRÔNICA**

Dorinaldo Soares

**Natal/RN
2004**

DORINALDO SOARES

**SOFTWARE PARA O MONITORAMENTO DE
SEGURANÇA ELETRÔNICA**

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

Orientador:

Fellipe Araújo Aleixo, M.Sc.

**NATAL/RN
2004**



Soares, Dorinaldo

Software para o Monitoramento de Segurança Eletrônica/ Dorinaldo
Soares.- Natal: FARN, 2004.

76 p.

Orientador: Fellipe Araújo Aleixo, M.Sc.

Monografia (Graduação em Sistemas de Informação) – Faculdade
Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Software – projeto e análise – monografia I-Título

RN/FARN/BC

CDU 004-4

RB-21.347



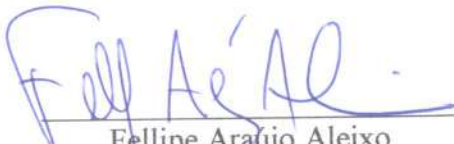
M
004.4
5653n
2004

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SOFTWARE PARA O MONITORAMENTO DE
SEGURANÇA ELETRÔNICA

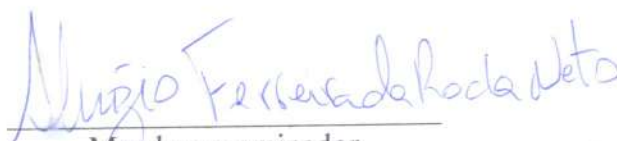
Dorinaldo Soares

Monografia apresentada e aprovada em 11 de DEZEMBRO de 2004 pela banca
examinadora composta pelos seguintes membros


Fellipe Araújo Aleixo
Orientador


Membro examinador

Maristela


Membro examinador

ALUIZIO

Natal, 11 de DEZEMBRO de 2004.

Dedico este trabalho aos meus pais queridos, Manoel Pereira Soares, Maria Pimentel Soares e irmãos, pois nada teria sido feito sem o amor da minha família.

À minha noiva, Karyna Mesquita da Silva, pelo apoio constante.

À meu Diretor e amigo da Garra Vigilância Ltda, Edvaldo Gomes, pela sua compreensão do meu tempo junto a minha vida profissional e educacional.

AGRADECIMENTOS

Em primeiro e principal lugar quero agradecer a Deus por ter me dado o maior e melhor dos talentos, a vida.

A construção desse trabalho foi conseguido por intermédio de várias pessoas, instituições e empresa. Portanto venho aqui expressar meus agradecimentos a todos com as mais possíveis sinceras das palavras.

Aos meus professores que durante o período do curso pode a medida do possível passar todo o conhecimento necessário para o meu conhecimento atual a respeito dessa ciência, em especial ao professor, Felliipe Araújo Aleixo, que me orientou nessa produção, de forma plena e inesgotável prudência e presteza junto a confecção desse software.

Aos meus amigos da Universidade Federal do Rio Grande do Norte (UFRN) e da Faculdade Nalalense para o Desenvolvimento do Rio Grande do Norte (FARN).

Em especial ao meu amigo e supervisor de trabalho da Base Naval de Natal (BNN), MARINHA DO BRASIL, Fabiano Andrade de Azevedo, pelo reconhecimento da minha carga de atividades profissionais e educacionais possibilitando-me tempo para realização adequada das duas de forma equilibrada e justa.

A empresa, Garra Vigilância Ltda, pelo apoio quanto à operacionalização e levantamento técnico no aspecto de necessidades necessário para um melhor atendimento junto aos clientes monitorados de Segurança Eletrônica.

À todas a pessoas de seus Diretores, Coordenadores, professores e demais funcionários dessa Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, pela presteza e suporte que me ofereceram para a construção e conclusão dessa produtiva, incessante e valorosa que é a produção do conhecimento.

E a todos que, de forma indireta, contribuíram na elaboração deste trabalho.

SUMÁRIO

1. INTRODUÇÃO.....	11
1.1 APRESENTAÇÃO.....	11
1.2 PROBLEMA DE PESQUISA.....	12
1.3 OBJETIVO GERAL.....	13
1.4 OBJETIVO ESPECÍFICO.....	13
1.5 JUSTIFICATIVA.....	14
1.6 METODOLOGIA.....	15
2. REFERENCIAL TEÓRICO.....	16
2.1. LINGUAGEM DE MODELAGEM UNIFICADA – UML.....	16
2.1.1 CASOS DE USO.....	18
2.1.2 DIAGRAMA DE CLASSES.....	20
2.1.3 DIAGRAMAS DE INTERAÇÃO.....	24
2.1.4 DIAGRAMA DE COLABORAÇÃO.....	25
2.2 FERRAMENTA DE PROJETO ORIENTADA A OBJETOS – UML.....	26
2.3 HTML.....	26
2.3.1 TABELAS.....	28
2.3.2 FORMULARIOS.....	31
2.4 JAVA.....	35
2.4.1 JAVA DATABASE CONNECTIVITY (JDBC).....	38
2.4.2 JAVA SERVER PAGES (JSP).....	40
2.4.2.1 TAG (S) JSP PARA CONSTRUÇÃO DE SCRIPTS.....	42
2.4.2.2 ELEMENTOS DE CRIAÇÃO DE SCRIPTS.....	42
2.4.2.3 OS FUNDAMENTOS BÁSICOS DE JAVABEANS.....	50
2.5 ACCESS.....	51
2.5.1 TIPOS DE CONSULTAS.....	52
3. ANÁLISE E IMPLEMENTAÇÃO DO SISTEMA.....	57
3.1 ESCOPO DO SISTEMA CASO DE USO.....	57
3.2 TELAS PRINCIPAIS DO SISTEMA.....	59
3.3 DIAGRAMA DE CLASSE DO SISTEMA.....	62
3.4 MODELO DO BANCO.....	63
3.5 ARQUITETURA DO SISTEMA.....	64

3.6 CÓDIGO JAVA.....	64
3.6.1 CÓDIGO JSP.....	68
3.7 ANÁLISE DE RESULTADOS.....	73
4. CONCLUSÃO.....	74
BIBLIOGRAFIA.....	75

LISTA DE FIGURAS

Figura 1. Exemplo de Casos de Uso.....	19
Figura 2. Exemplo de um cenário.....	19
Figura 3. Exemplo de Diagrama de Classes.....	21
Figura 4. Diferentes Alinhamentos.....	30
Figura 5. Plano de Fundo.....	31
Figura 6. Diagrama de Casos de Uso do Sistema.....	57
Figura 7. Tela de Entrada do Sistema.....	59
Figura 8. Entrada do Sistema – Erro Login ou Senha.....	60
Figura 9. Entrada do Sistema – Parâmetro Inválido.....	60
Figura 10. Entrada do Sistema – Tela Principal.....	61
Figura 11. Diagrama de Classe do Sistema.....	62
Figura 12. BD do Sistema.....	63
Figura 13. Modelo de 3 Camadas.....	64

TABELA

Tabela 1. Atributos da Diretiva page.....	45
---	----

RESUMO

Este trabalho irá apresentar um software destinado a tornar mais eficiente o monitoramento de segurança eletrônica. O desenvolvimento desse software foi iniciado com uma análise de requisitos, através de questionários e vivência, pois é o ramo de trabalho do autor do trabalho. Nesta primeira fase foram identificadas as soluções para as diversas situações do ramo do negócio. Situações que uma vez resolvidas trarão uma maior qualidade no serviço de monitoramento junto aos clientes. Assim com o uso desses software, irá sem dúvida aumentar o poder de relacionamento com o cliente, o que é uma das principais forças internas e externas que afetam, desafiam e mudam as organizações. Organizações essas que alcançará o sucesso do mercado, com a integração de todas as atividades tecnológicas.

ABSTRACT

This work will go to present a destined software to become more efficient the monitoring of electronic security. The development of this software was initiated with an analysis of requirements, through questionnaires and experience, therefore it is the branch of work of the author of the work. In this first phase the solutions for the diverse situations of the branch of the business had been identified. Decided situations that a time will bring a bigger quality in the service of together monitoring to the customers. Thus with use of this software, will go without a doubt to increase power of relationship with customer, what she is one of the main internal and external forces that affect, they defy and they change the organizations. Organizations these that the success of the market will reach, with the integration of all the technological activities.

INTRODUÇÃO

Neste capítulo terá, a apresentação, o problema de pesquisa e a exposição do objetivo geral e específico, a justificativa para sua produção e assim qual metodologia utilizada no seu desenvolvimento.

1.1 APRESENTAÇÃO

Este projeto irá estudar e apresentar um Software para melhorar a qualidade do Monitoramento de Segurança Eletrônica. Esse sistema de segurança Eletrônica é composto basicamente com quatro segmentos: os alarmes, os circuitos fechados de TV, os controles de acesso e os sistemas integrados.

A palavra segurança tornou-se mágica perante o assombrado mundo hodierno em todos os setores de atividade humana.

Sente-se que há intensa variedade de segurança. Fala-se em segurança pública, segurança militar, segurança política, segurança aérea, segurança jurídica, segurança nacional, segurança cultural, segurança social, segurança pessoal e muito mais. Muito delas estão correlacionadas com a Segurança Eletrônica Monitorada, pois é essa que sendo eficientemente empregada protege bem as demais seguranças citadas acima. E por que não chamar ela de uma segurança das "seguranças".

Um dos serviços de que mais se ouve falar atualmente é o monitoramento. Em caso de uma invasão e disparo de alarmes dotados de monitoramento, um sinal é enviado para a central da empresa de monitoramento 24horas. Esta acionará sua própria equipe (patrulha) para checar o local e, se necessário, a empresa chamará a polícia, médicos ou bombeiros.

No sistema de Monitoramento 24h de alarmes, o local protegido é dotado de botões de emergência (pânico), que são dispositivos instalados em locais estratégicos. Funcionam até quando o sistema de alarme estiver desativado. A partir do qual o usuário pode acionar silenciosamente, alertando diretamente a central de monitoramento. A central então telefona para o cliente, para checar o problema ocorrido, se o cliente não falar a palavra chave é porque está sob coação de terceiros. Assim a empresa enviará a patrulha ao local.

Um sistema de alarme contra invasão é composto por: uma Central de alarme, Bateria, Sensores, Sirenes, cercas eletrificadas e Acessórios, com e sem fios. São ativados/desativados por meios de teclado (senhas), controle remoto, chave, etc.

No caso de invasão, os sensores captam os movimentos, enviam um sinal ao painel de alarme que:

- 1) Disparam as sirenes, acendem luzes, etc.
- 2) Discam para os números de telefones previamente programados, avisando que houve uma invasão.
- 3) Ou acionam a central de monitoramento 24h, que rapidamente toma as devidas providências.

No caso de o invasor cortar a energia, a bateria sustenta o sistema de 4 a 10 horas de uso, dependendo da quantidade de equipamentos em operação no momento. Assim a Central de Monitoramento, irá mostrar no monitor a partir de um software todos os dados do cliente em questão, informando em destaque o evento do momento.

1.2 PROBLEMA DE PESQUISA

Mas o que não se vê hoje dentro do âmbito dos monitoramentos existentes são as correções quanto aos problemas de :

- ♦ Demonstração quilometragem e tempo médio das patrulhas em serviço do seu ponto de Apoio até o estabelecimento violado;
- ♦ A apresentação de um croquis (planta baixa) do estabelecimento violado, para que assim possa o operador orientar melhor a patrulha;
- ♦ Informações quanto aos procedimentos possíveis a ser(em) tomado(s) pelo(s) operador(es) e supervisor(es) da patrulha;
- ♦ Campo livre para as devidas anotações dos eventos ocorridos;
- ♦ Espaço para preencher os dados dos operadores do dia, com senha, nome hora de chegada e saída e possíveis alterações de serviço passado;
- ♦ A capacidade de comunicação on-line junto aos clientes;
- ♦ Disponibilidade para backup em cd room e/ou outros;
- ♦ Possibilidade de envio de relatórios com seus eventos e anotações respectivas, pelo supervisor do monitoramento;
- ♦ Avisos de alerta quanto aos clientes que faltam fechar e que ainda não abriram

- ♦ Disponibilidade dos clientes averiguarem se foram ativado/desativado seu sistema, que horas ocorreu e por quem?

A resolução desses problemas citados acima nos trará um excelente monitoramento, para evitar acima de tudo a falta de atenção com os bens dos clientes, que depositam toda a segurança em um dos segmentos de Segurança Eletrônica, conhecido como alarme.

1.3 OBJETIVO GERAL

O objetivo desta pesquisa visa estudar, apresentar e implementar um protótipo de um programa para o uso de um monitoramento de segurança eletrônica.

Em geral, o objetivo é um aumento da qualidade de operacionalização, consecutivamente a verificação rápida e precisa na identificação do cliente, análise com o menor tempo possível do evento, para passar para o supervisor da Patrulha e controle automático e avançado, ou seja, uma busca incessante da perfeição da qualidade de atendimento junto aos clientes.

1.4 OBJETIVO ESPECÍFICO

Realizar especificações de funções, categorizar, identificar e relacionar funções e os atributos do sistema;

Demonstrar a partir de uma linguagem gráfica a análise geral do projeto para suprir tais complexidades de conhecimento, isso com a linguagem UML;

Construir diagramas de casos e uso, modelo conceitual utilizando a notação UML;

Fazer classes que farão a manipulação de dados orientados a objetos;

Construir um projeto que enfoque a orientação a objetos, utilizando a linguagem JAVA;

Desenvolver um estudo sobre a linguagem JAVA a fim de utilizar suas ferramentas no que diz respeito a acesso a banco de dados(JDBC) e para a Web (o termo *web* é utilizado como referência a *World Wide Web*) (JAVA BEANS);

Fazer as interfaces gráficas utilizando HTML;

Associar as interfaces com as classes, criando conteúdos dinâmicos utilizando a tecnologia JSP;

Realizar, estudar e utilizar a linguagem SQL SERVER, para gerenciar, processar e manipular comandos dentro do banco de dados do sistema.

1.5 JUSTIFICATIVA

Dentro do âmbito do mundo moderno os assaltos e arrombamentos tem sido freqüentes, ágeis e cada vez mais inteligentes. Em contra partida os níveis de segurança não devem ficar atrás nem igualar-se e sim ultrapassar, para suprir essa “alta” na insegurança nacional.

O sistema terá um acesso via senha lógica, que será utilizada pelo operador do dia de serviço. Assim será anotado sua hora de chegada ou saída dependendo da sua escala de trabalho e principalmente o início de suas responsabilidades.

Com a exposição no sistema da quilometragem e tempo médio de deslocamento das patrulhas o operador terá como monitorar também essa patrulha que está junto no seu dia de serviço, isso será tudo anotado para um futuro relatório e envio para o cliente se assim for solicitado.

A planta baixa do estabelecimento informando o local violado em destaque proporcionará tanto ao operador quanto ao supervisor da patrulha uma melhor análise e/ou averiguação de tal evento ocorrido. Em consequência do evento ocorrido no cliente, a exposição dos principais procedimentos a serem tomados evitará algum tipo de erro por parte do operador junto ao cliente em questão. A principal função do campo livre para as possíveis anotações terá uma excelente importância quanto aos envios de informações e relatórios junto aos clientes solicitantes.

Quanto aos aviso de alerta trará funções como informação dos clientes que estão para fechar ou que ainda não abriram, a primeira informação evitará o esquecimento da ativação do sistema por parte do cliente, a segunda trará uma solução de um possível defeito na comunicação entre a Central de Alarme (no cliente) e a Central de Monitoramento (empresa).

A capacidade de comunicação on-line com os clientes trará uma resolução no problema ou na dúvida no mesmo momento que ocorrerem. Isso fará com que tais clientes possam analisarem como estão sem assegurados, podem saber que horas e quem realizou o ativamento ou desativamento do sistema via Web.

E quanto aos backup's este sistema terá a condição de informar aos clientes um ou dois anos atrás determinados eventos solicitados, e evitará uma possível pane ou incapacidade no processo de armazenamento.

A Segurança Eletrônica existe e é com ela que iremos trabalhar, melhorando seu nível de Monitoramento, no que diz respeito a praticidade no tempo de resposta e análise de risco com o mínimo de erro, a alta satisfação do cliente, a agilidade no tempo de resposta.

Assim com o uso desses benefícios, irá sem dúvida aumentar o poder de relacionamento com o cliente, o que é uma das principais forças internas e externas que afetam, desafiam e mudam as organizações. Organizações essas que alcançarão o sucesso do mercado, com a integração de todas as atividades tecnológicas.

1.6 METODOLOGIA

O desenvolvimento desse projeto realizou-se com uma forte análise de requisitos, através de vivência pois, é meu ramo de trabalho. A primeira parte foi extraindo benefícios e soluções para as diversas situações do ramo do negócio.

A coleta de informações foi sem dúvida as primeiras linhas da minha análise de requisitos, onde foi utilizado a Linguagem Unificada de Modelagem.

O desenvolvimento do estudo desse projeto classifica-se quanto ao tipo exploratório e descritiva.

A exploratória foi devido a busca de tornar o sistema mais explícito possível com o alto levantamento bibliográfico e a experiência prática com o problema pesquisado. O tipo descritivo nos torna capaz de expor a descrição das características principais do sistema abordado e a realização de pesquisas sociais preocupadas com a atuação prática, dentro das empresas comerciais.

A pesquisa bibliográfica foi utilizada para o desenvolvimento desse protótipo de software, pois se faz necessário o uso de materiais já elaborados e constituintes principalmente de fontes bibliográficas.

Foi utilizando um estudo de campo, com a comunidade de trabalho.

Basicamente, esta pesquisa foi desenvolvida por meio de observações e vivências como o ramo do negócio, especificamente com análise de documentos e ações desenvolvidas no dia a dia.

REFERENCIAL TEÓRICO

2.1 LINGUAGEM DE MODELAGEM UNIFICADA – UML

Diante do crescimento e complexidade das aplicações e de um novo gênero de linguagens de programação (orientadas a objeto), tornou-se necessária a expansão dos métodos de análise projeto, com intuito de suprir tais complexidades. Foi entre 1970 e 1990, que surgiu essa nova metodologia de análise e projeto chamada de Linguagem de Modelagem Unificada (UML) orientada a objetos. Resultado da fusão de vários métodos existentes para análise e projeto orientada a objetos.

Ela foi oficialmente criada, quando Rumbaugh se juntou a Booch na Rational. Em seguida, Jacobson se associou a Rational e o escopo do projeto da UML foi expandido.

A expansão teve início com a criação de um consórcio de UML, onde várias empresas se uniram com o objetivo de unir recursos, a fim de trabalhar a favor de uma definição mais forte e completa dessa Linguagem de Modelagem Unificada. Tal consórcio resultou em uma primeira Linguagem chamada de UML 1.0. Bem definida, expressiva, poderosa e que poderia ser aplicada a uma vasta variedade de tipos de problemas.

Quando a UML surgiu, os desenvolvedores da área da orientação a objetos ficaram entusiasmados já que essa padronização proposta pela UML era o tipo de força que eles sempre esperaram.

A UML é muito mais que a padronização de uma notação. É também o desenvolvimento de novos conceitos. Por isso e muitas outras razões, o bom entendimento da UML não é apenas aprender a simbologia e o seu significado, mas também significa aprender a modelar orientado a objetos no estado da arte.

A UML é uma tentativa de padronizar a modelagem orientada a objetos de uma forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com consistência, simples de ser atualizado e compreensível.

Para Larman (2000), os projetos de forma geral devem apresentar um mapa detalhado das suas atividades como os requisitos, a análise, a projeto e a codificação. A ênfase nas habilidades necessárias para essas atividades quanto a responsabilidade e colaboração são:

- ♦ Expressar modelos de análise e de projeto, notação diagramática que está se tornando padrão para sistemas orientados a objetos;
- ♦ Aplicar padrões para atribuir responsabilidades e colaborações ao projeto;
- ♦ Construção de arquiteturas em camadas e;
- ♦ Mapear modelos de projeto para Java.

A análise modela o problema e consiste das atividades necessárias para entender o domínio do problema (o que deve ser feito). É uma atividade de investigação. Analisar não é definir como o sistema será desenvolvido, mas sim investigar o problema.

O maior problema do desenvolvimento de sistemas utilizando a orientação a objetos é que não existia uma notação padronizada e realmente eficaz que abrangesse qualquer tipo de aplicação que se deseje.

Cada metodologia existente possui seus próprios conceitos, gráficos e terminologias, resultando numa grande confusão, especialmente para aqueles que querem utilizar a orientação a objetos mas não sabem criar modelos para ajudá-los a construir e manter sistemas cada vez mais eficazes.

As várias metodologias de modelagem orientada a objetos que existiam até o surgimento da UML, causavam uma guerra entre a comunidade de desenvolvedores que utilizam a orientação a objetos. A UML acabou com esta divergência trazendo as melhores idéias de cada uma destas metodologias, e mostrando como deveria ser a migração de cada uma para a UML.

Os objetivos da UML são:

- ♦ A modelagem de sistemas (não apenas *software*) usando os conceitos da orientação a objetos;
- ♦ Estabelecer uma união fazendo com que métodos conceituais estejam bem mais próximos dos métodos executáveis;
- ♦ Criar uma linguagem de modelagem usável tanto pelo homem quanto pela máquina.

Assim como a UML aborda o caráter estático e dinâmico do sistema a ser analisado; levando em consideração, já no período da modelagem, todas as futuras características do sistema em relação à implementação do mesmo.

Portanto, a Linguagem de Modelagem Unificada (UML) está emergindo como a notação padrão para ilustrar modelos de análise e projeto.

Larman (2000) afirma, os autores da UML – Booch, Jacobson e Rumbaugh – prestaram um grande serviço à comunidade da tecnologia orientada a objetos ao criar uma linguagem de modelagem padronizada que é elegante, expressiva e flexível.

O mesmo autor acima citado, diz que a UML é uma linguagem para modelagem; ela não guia o desenvolvedor em como fazer análise e projeto orientados a objetos, ou qual o processo de desenvolvimento a ser seguido.

Segundo FURLAN (1998, p.33),

A UML é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação. Buscou-se unificar as perspectivas entre os diversos tipos de sistemas e fases de desenvolvimento de forma que permitisse levar adiante determinados projetos que antes não eram possíveis pelos métodos existentes.

2.1.1 Casos de Uso

Em cada caso de uso possui um cenário, composto por uma sequência de passos ilustrando a interação entre um usuário e o sistema.

Apresentam em sua estrutura:

- ♦ Atores: são papéis a serem desempenhados pelos usuários de um caso de uso, ou seja, o ator é quem solicita os serviços disponíveis em um caso de uso. Um ator pode ser: uma pessoa que interage com o sistema, um *hardware* que interage com o sistema ou um outro sistema que tenha a necessidade de utilizar o caso de uso.
- ♦ Casos de uso: uma descrição narrativa e textual da sequência de eventos e ações que ocorrem quando um usuário participa de um diálogo com um sistema, durante um processo que tem significado, ou seja, o que os usuários estão aptos a fazer com o sistema.
- ♦ Interação: representa a ligação entre um ator e um referido processo em um determinado caso de uso.

Os diagramas de casos de uso tem por objetivo especificar, visualizar e documentar o comportamento que cada parte do sistema deve conter.

Na figura 1, a seguir é mostrado um exemplo de um caso de uso.

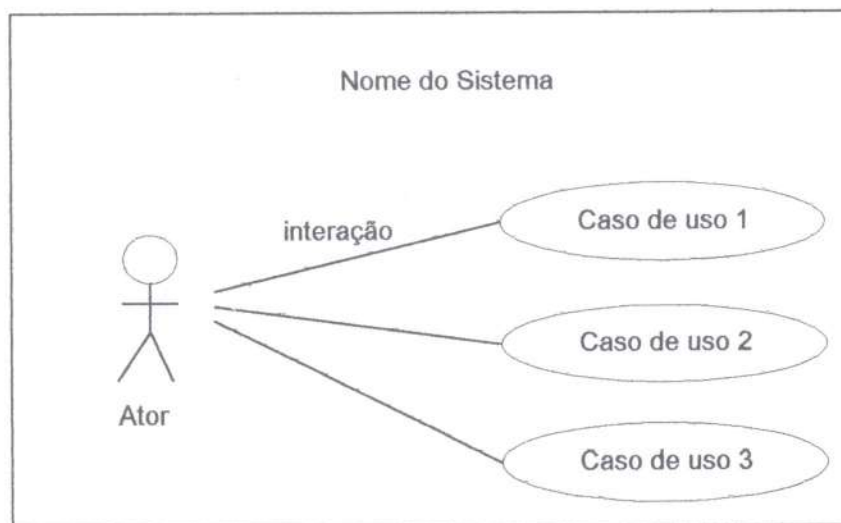


Figura. 1 Exemplo de Casos de Uso

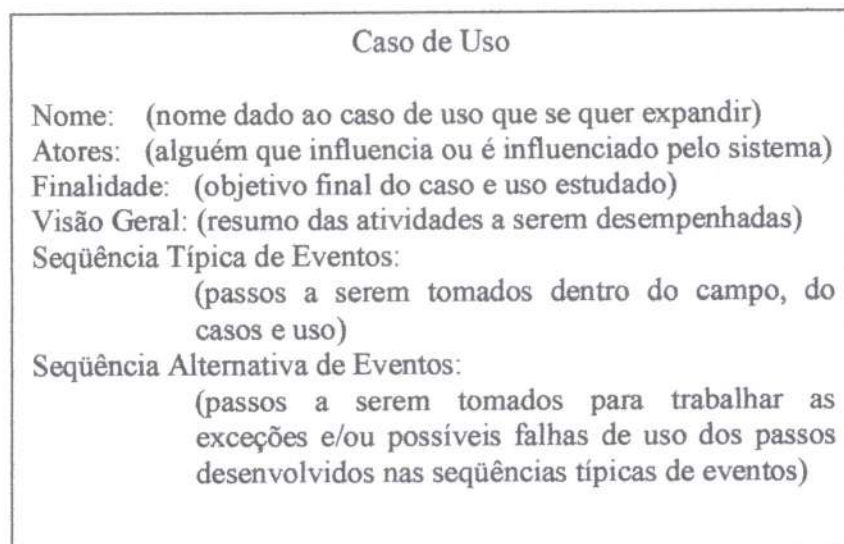


Figura. 2 Exemplo de um cenário

Através de expansão de cada um dos casos de uso ou cenários (ver figura 2), listamos nestes a sequência, em detalhes, a interação entre o referido ator e o sistema.

2.1.2 Diagrama de Classes

O diagrama de classes representa o modelo da estrutura de um sistema orientado a objetos. Esse tipo de diagrama é o centro dos principais conceitos da modelagem orientada a objetos.

Características dos diagramas de Classe:

- ♦ Permite visualizar o modelo da estrutura de um sistema;
- ♦ É um dos diagramas mais utilizados na modelagem orientada a objetos;
- ♦ Demonstra as classes e os relacionamentos (agregação, composição, generalização, etc).

As classes representam os blocos de construção utilizados nesse diagrama. Uma classe é a descrição de um tipo de objeto. Todos os objetos são instâncias de classes, onde a classe descreve as propriedades e comportamentos daquele objeto. Objetos só podem ser instanciados de classes. As classes são usadas para classificar os objetos que identificamos no mundo real.

Na figura 3, é mostrado um modelo de um diagrama de classes. A seguir, serão descritos cada um dos integrantes do modelo:

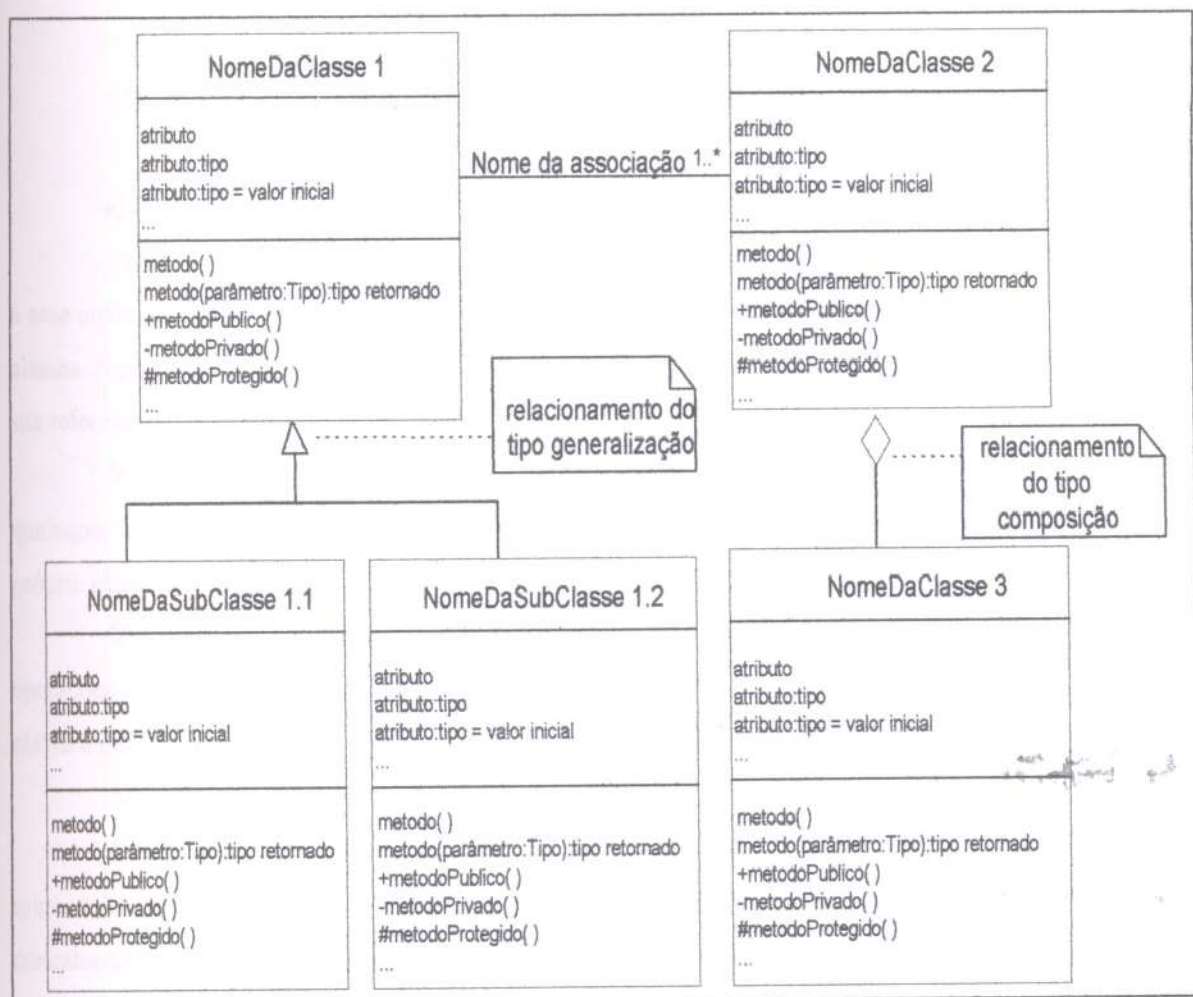


Figura. 3 Exemplo de Diagrama de Classe

Em UML as classes são representadas por um retângulo dividido em três compartimentos:

- O compartimento do nome, que contém apenas o nome da classe modelada. Todas as classes devem ter um nome que as diferencie das demais. Usamos uma sequência de caracteres para identificá-las, cuja sintaxe é: NomeDaClasse.
- O de atributos, que contém a relação dos atributos que a classe possui em sua estrutura interna. São representados logo após o compartimento referente ao nome da classe. Um atributo é uma propriedade de uma classe, representa o tipo de dados que cada item de uma classe pode assumir. Uma classe pode

é ter um número qualquer de atributos ou nenhum atributo. A sintaxe padrão de um atributo:

<Visibilidade> <nomeDoAtributo> : <TipoDoAtributo> = <ValorInicial>

a) A visibilidade é definida como:

Visibilidade pública (+) (valor padrão) - significa que todas as classes têm acesso a esse atributo, podendo o atributo ser utilizado por operações declaradas dentro de outras classes. É parte da interface de uma classe e é visível por qualquer porção do ambiente de sua referência (escopo).

Visibilidade privada (-) - declaração que é parte de uma classe mas não é visível a quaisquer outras classes, o atributo pode apenas ser acessado por operações dentro da própria classe, apenas objetos da mesma classe tem acesso.

Visibilidade protegida (#) - significa que o atributo é acessado através de operações dentro da mesma classe ou de subclasses da referida classe, no escopo no qual a classe é definida.

b) nomeDoAtributo

É uma sequência de caracteres de identificação começando tipicamente com letra minúscula. Se o nome for uma composição de palavras, as demais palavras são concatenadas, preservando-se a primeira letra em maiúscula. Por exemplo: nomeDoIndivíduo.

c) TipoDoAtributo

É uma expressão que depende da linguagem de programação e do tipo de implementação de um atributo. A UML evita especificar regras para construção de tipos de expressão pois as linguagens de programação apresentam uma variedade de regras para declaração de variáveis e parâmetros. Portanto, a UML deixa indefinida a sintaxe real das expressões de tipo.

d) ValorInicial

É uma expressão que também depende da linguagem de programação utilizada para o valor inicial de um objeto criado recentemente. O valor inicial é opcional.

- ♦ O compartimento das operações, representado logo após o compartimento dos atributos, é onde estarão os métodos (conjunto de operações) de manipulação de dados e de comunicação de uma classe com outras do sistema. A sintaxe padrão é:

<Visibilidade><NomeDaOperação><listadeparâmetros>; <ExpressãoDeTipoRetorno >

a) Visibilidade

Definida da mesma forma que para atributos, ou seja: visibilidade pública (+), visibilidade privada (-) e visibilidade protegida (#). Tal como ocorre com os atributos, o marcador de visibilidade pode ser suprimido.

b) NomeDaOperação

Seqüência de caracteres de identificação começando com letra minúscula. Concatenam-se as demais palavras que compõem o nome preservando a primeira letra de cada palavra em maiúscula. Por exemplo: obterNomeDoIndivíduo.

c) Parâmetros

É uma lista de valores separada por vírgula usados nas operações, mensagens e eventos. Trata-se de uma especificação de uma variável que pode ser passada para o método, podem também serem especificados os tipos dos parâmetros.

d) ExpressãoDeTipoRetorno

É uma especificação dependente da linguagem de programação sobre o tipo do valor retornado pela operação. Se o tipo de retorno é omitido, a operação não devolve valor.

A sintaxe usada em cada um destes compartimentos é independente de qualquer linguagem de programação, embora possam ser usadas outras sintaxes como a do C++, Java, ou a própria notação UML.

As associações representam relacionamentos entre objetos de diferentes classes, e são representados graficamente através de uma linha conectando as classes. Uma associação pode ter um nome. E a linha que representa a associação pode ter nome de papéis mostrando como a classe é vista pela outra classe na associação.

A multiplicidade de uma associação especifica quantas instâncias de uma classe relacionam-se a quantas instâncias da classe associada. Uma multiplicidade é representada

por um intervalo de valores possíveis, no seguinte formato: limite_inferior..limite_superior, onde esses limites são valores inteiros.

Esse intervalo pode ser de zero para um (0..1), zero para muitos (0..* ou apenas * - o caracter * é usado para representar “muitas” instâncias), um para muitos (1..*), ou uma sequência especificada (m..n – onde m e n são quaisquer valores inteiros - ex: 3..2). Se não for descrita nenhuma multiplicidade, então é considerado o padrão de um para um.

A composição é uma forma especial de associação que representa o relacionamento “todo-parte” entre objetos. Ela é representada incluindo-se um losango no fim da linha da associação e próximo à classe que representa o todo no relacionamento “todo-parte”. Neste relacionamento, uma classe representa o “todo” e outras classes representam as “partes”. Como por exemplo as classes Carro(todo) e motor, caixa de Marcha, etc (parte).

A generalização é uma ferramenta poderosa para a abstração. A generalização é um relacionamento existente entre uma classe mais geral (superclasse) e uma classe mais específica (subclasse), onde a classe mais específica contém todos os atributos e métodos da super classe e pode ainda adicionar métodos e/ou atributos e sobrescrever métodos. Uma generalização é representada por uma linha com um triângulo que liga a classe mais específica (parte) a mais genérica (todo).

2.1.3 Diagramas de Interação

Os diagramas de interação – formados pelos diagramas de sequência e diagramas de colaboração – representam a dinâmica de um sistema.

Utiliza-se dos diagramas de interação para mostrar como os objetos colaboram entre si para a realização dos comportamentos identificados na análise dos casos de uso.

Características dos diagramas de interação:

- ♦ Representam a dinâmica entre os objetos (instâncias das classes já identificadas);
- ♦ Representam as colaborações entre objetos para cada comportamento do sistema;
- ♦ Oferece fácil identificação de quantas e quais mensagens serão trocadas entre os objetos que compõem o projeto;

Os diagramas de seqüência dão ênfase à ordenação temporal em que os comportamentos acontecem, enquanto os diagramas de colaboração dão ênfase à ordenação estrutural dos objetos que colaboram entre si.

O diagrama de seqüência e o diagrama de colaboração mostram os mesmos aspectos de um modelo, apenas diferenciando como essas informações são apresentadas. Isso quer dizer que os dois diagramas são semanticamente equivalentes, podendo ser usado um ou outro para visualizar modelos sob uma perspectiva dinâmica.

2.1.4 Diagramas de colaboração

Como já foi dito anteriormente, diferentemente do diagrama de seqüência que focaliza a seqüência cronológica do cenário que está sendo modelado, o diagrama de colaboração focaliza o relacionamento entre os objetos, e a compreensão dos efeitos sobre os mesmos da seqüência descrita num cenário.

Os objetos são conectados através de associações e cada associação representa uma instância de associação entre as respectivas classes envolvidas. As associações mostram as mensagens enviadas entre os objetos, e a seqüência destas mensagens é determinada usando-se números seqüenciais. Uma associação especial, chamada de auto-delegação, é a associação de um objeto com ele mesmo, no intuito de modularizar as ações a serem tomadas pelo mesmo.

UML tem uma sintaxe padrão para mensagens:

<retorno> := <mensagem> <parâmetro> : <tipo de retorno>

Onde: retorno é o nome de valor de retorno, mensagem é o nome da mensagem, parâmetro é o nome de um parâmetro da mensagem e tipo de retorno é tipo do valor de retorno.

Uma Colaboração é uma visão de um conjunto de elementos de modelagem relacionados para um propósito particular em apoio a interações.

2.2 FERRAMENTA DE PROJETO ORIENTADA A OBJETOS

JUDE COMMUNITY for UML é uma tecnologia e/ou software de modelagem, que utiliza os padrões definidos pela UML. É de grande utilidade para facilitar o gerenciamento de projetos de *software* orientados a objetos, quando se pretende modelar sistemas a serem implementados com a linguagem Java.

Oferece uma interface amigável, suporta os diagramas de casos de uso, classes, sequência, colaboração, estado, atividade e componentes; além da geração automática de código Java, a partir da geração de modelos de classes.

Uma funcionalidade importante da ferramenta JUDE é a geração automática de críticas sobre os modelos construídos.

2.3 HTML (HyperText Markup Language)

HTML (Hypertext Markup Language) é a linguagem de descrição de documentos usada na World Wide Web para criação de páginas de conteúdo hipertexto.

Para se criar uma página HTML é necessário ter um editor de textos comum, como o bloco de notas do Windows.

Para o Alvarez (2003), a HTML é uma linguagem universal destinada à elaboração de páginas com hipertexto.

O conceito de hipertexto está associado a uma nova forma de documentos, compostos por texto formatado e ligações a outros documentos, os quais poderão ser do tipo hipertexto, ou mesmo imagens, sons vídeo e outros.

Em HTML os comandos, tags informam ao browser como o texto, informações e as imagens que deverão ser apresentadas. Todas as tags são envolvidas pelos sinais $\langle \rangle$ ou $\langle / \rangle$. Entre os sinais $\langle \rangle$ são especificadas as tags propriamente ditas. No caso de tags que precisam envolver um texto, a sua finalização deverá ser feita usando a barra $\langle / \rangle$, indicado o fim do tag. Como sintaxe de uma tag temos:

$\langle \text{NOME DA TAG} \rangle \text{ Texto } \langle / \text{NOME DO TAG} \rangle$

Todo o documento HTML (que deve ter a extensão .html ou .htm) fica contido entre os Tags: <HTML> e </HTML>. Temos duas seções básicas:

HEAD

Contém parâmetros de configuração do documento.

BODY

Contém o documento em si.

A estrutura de um documento HTML é:

```
<HTML>
  <HEAD>
    <TITLE>Título da Home Page</TITLE>
  </HEAD>
  <BODY>
    *** Conteúdo da Home Page ***
  </BODY>
</HTML>
```

Tags mais usadas no documento

<HTML>...</HTML>

Envolvem todas as seções de um documento (HEAD e BODY).

<HEAD>...</HEAD>

Envolvem a seção de cabeçalho do documento.

<TITLE>...</TITLE>

Indica o título do documento para o Browser. Geralmente os Browsers apresentam este título na barra de título da sua Janela no Windows.

<BODY>...</BODY>

Envolvem a seção de corpo do documento. Aqui fica o conteúdo principal da Home Page.

2.3.1 Tabelas

A formatação de tabelas foi adotada bem antes de sua inclusão na definição de HTML. A manipulação de tabelas, mesmo em editores, é trabalhosa; a maior diferença entre tabelas em HTML e em editores como o MS Word, entretanto, é o fato das tabelas em HTML serem definidas apenas *em termos de linhas* e não de colunas.

As tabelas foram uma grande conquista para os autores de documentos para a Web. Tabelas implementam um conceito importante de layout: as “grades”, segundo as quais organizamos textos e ilustrações de maneira harmoniosa.

Como já foi possível perceber, as tabelas contêm textos, listas, parágrafos, imagens, formulários e várias outras formatações - inclusive outras tabelas. Novas versões de HTML e de browsers populares vêm acrescentando diversos atributos às tabelas.

a) Elementos básicos de tabelas

`<TABLE>...</TABLE>` delimita uma tabela. Um atributo básico é `BORDER`, que indica a apresentação da borda.

`<TABLE BORDER="borda">`

...

`</TABLE>`

Títulos, linhas e elementos

`<CAPTION>...</CAPTION>`

define o título da tabela

`<TR>...</TR>`

delimita uma linha

`<TH>...</TH>`

define um cabeçalho para colunas ou linhas (dentro de `<TR>`)

`<TD>...</TD>`

delimita um elemento ou célula (dentro de `<TR>`)

Um exemplo de comandos de tabela simples onde se tem uma borda de tamanho 4, com quatro linhas e duas colunas:

```
<TABLE BORDER=4>
<CAPTION>Primeiro exemplo</CAPTION>
<TR><TH>Coluna 1</TH><TH>Coluna 2</TH></TR>
<TR><TD>linha 1, coluna 1</TD><TD>linha 1, coluna 2</TD></TR>
<TR><TD>linha 2, coluna 1</TD><TD>linha 2, coluna 2</TD></TR>
</TABLE>
```

b) Títulos compreendendo mais de uma coluna ou linha

É possível englobar colunas e linhas, através dos atributos COLSPAN (para colunas) e ROWSPAN (para linhas):

```
<TABLE BORDER=1>
<TR><TH COLSPAN=2>Colunas 1 e 2</TH></TR>
<TR><TD>linha 1, coluna 1</TD><TD>linha 1, coluna 2</TD></TR>
<TR><TD>linha 2, coluna 1</TD><TD>linha 2, coluna 2</TD></TR>
<TR><TH ROWSPAN=3>3 linhas</TH><TD>uma linha</TD></TR>
<TR><TD>duas linhas</TD></TR>
<TR><TD>tres linhas</TD></TR>
</TABLE>
```

Neste exemplo, vemos que o cabeçalho Colunas 1 e 2 compreende duas colunas (COLSPAN=2); o cabeçalho 3 linhas compreende, por sua vez, 3 linhas (ROWSPAN=3).

c) Tabelas sem borda

Usa-se a seguinte declaração:

```
<TABLE BORDER="0">
```

...

```
</TABLE>
```

d) Alinhamentos simples

Os alinhamentos padrão em tabelas são:

- ♦ no sentido horizontal: alinhamento à esquerda;
- ♦ no sentido vertical: alinhamento no centro da célula e
- ♦ As linhas e células podem ter alinhamentos definidos através dos atributos.

ALIGN = alinha horizontal

VALIGN = alinha vertical

Vejamos como esses alinhamentos funcionam nas células:

<TD ALIGN = alinha horizontal> Texto da célula </TD>

<TD VALIGN = alinha vertical> Texto da célula </TD>

Padrão	ALIGN=LEFT	ALIGN=CENTER	ALIGN=RIGHT
Padrão	VALIGN=TOP	VALIGN=MIDDLE	VALIGN=BOTTOM

Figura 4 – Diferentes Alinhamento

Obs.: A tabela anterior foi feita especialmente para mostrar as diferenças entre os alinhamentos. Uma tabela comum ajusta o tamanho de suas células ao conteúdo, da forma a seguir:

e) Alinhamentos combinados

Uma mesma célula pode ter atributos ALIGN e VALIGN:

Por exemplo:

<TD ALIGN=alin_horizontal VALIGN=alin_vertical>Texto da célula</TD>

f) Alinhamentos de linhas

O alinhamento pode ser aplicado a linhas inteiras, com:

Porém, o alinhamento declarado em uma célula prevalece sobre o alinhamento da linha, como se vê no comando:

<TR ALIGN=alin_horizontal VALIGN=alin_vertical>Texto da célula</TR>

g) Imagem de fundo

```
<TABLE BORDER=5 BACKGROUND="imagem">
```

janeiro	fevereiro	março
abril	maio	junho

Figura 5 – Plano de Fundo.

2.3.2 Formulários

Criar um formulário não é uma tarefa tão simples. Existem basicamente duas etapas. A primeira, a qual vamos estudar, é a programação dentro do arquivo HTML. A segunda, é a criação de um *script* (arquivos que organizam os dados de entrada).

A baixo se tem um exemplo simples dos comandos necessários para se criar um formulário em HTML. O comando `<FORM>` é usado para delimitar a área em que estarão os comandos de entrada e formatação do documento.

```
<FORM ACTION="cliente" METHOD="post">
<INPUT TYPE="text" NAME="Nome">
<INPUT TYPE="text" NAME="e-mail">
<INPUT TYPE="submit">
<INPUT TYPE="reset">
</FORM>
```

Entre os comandos `<FORM>` e `</FORM>`, podem existir quaisquer comandos de formatação (parágrafos, imagens, tabelas...) **exceto** outros formulários. Aliás, muitos dos comandos de formatação estudados até agora serão necessários para que se obtenha um

formulário com uma boa aparência. Em especial, colocamos dentro desses comandos (`<FORM>` e `</FORM>`), as formatações para campos de entrada de dados, que são três:

`<INPUT>`, `<SELECT>` e `<TEXTAREA>`.

ACTION: Especifica o endereço URL do *script* ao qual serão enviados os dados do formulário;

METHOD: Especifica o método em que os dados serão enviados do browser para o servidor. Podem ser:

- ◆ **POST**

Ideal para transferir grandes quantidades de dados. Esses dados fazem parte do corpo da mensagem enviada ao servidor.

- ◆ **GET**

Suporta somente 128 caracteres e os dados fazem parte da URL associado à consulta enviada para o servidor.

São basicamente três os tipos de campo para a entrada de dados. `<INPUT>`, `<SELECT>` e `<TEXTAREA>`. Todos esses tipos de campo possuem um atributo em comum. Trata-se do atributo **NAME**, ao qual associamos um nome, que será utilizado pelo *script*. Mas vamos começar pelo primeiro tipo de campo, que é, provavelmente o menos simples.

O campo `<INPUT>` possui um atributo **type**, ao qual atribuímos seis valores diferentes de entrada de dados. São eles:

a) **CAMPOS TIPO TEXTO**

- ◆ **text:** permite a entrada de um texto qualquer. Essa opção é assumida como padrão, ou seja, pode ser omitida dentro do comando `<INPUT>`.

`<FORM>`

Nome: `<INPUT TYPE="text" NAME="Nome">`

`</FORM>`

- password: também permite a entrada de um texto qualquer, porém este será escondido por asteriscos. É usado, geralmente, quando o dado a ser digitado é uma senha.

<FORM>

Login: <INPUT TYPE="text" NAME="Nome" SIZE="40">

Senha: <INPUT TYPE="password" NAME="Senha" SIZE="8" MAXLENGTH="8">

</FORM>

Note que dois novos atributos surgiram. O primeiro, SIZE, determina o tamanho do campo, e o segundo, MAXLENGTH, determina o número máximo de caracteres aceitos por esse campo.

Existem ainda um quarto atributo. Através dele é possível atribuir um valor padrão a qualquer tipo de campo. Trata-se do atributo VALUE.

Nome: <INPUT TYPE="text" NAME="Nome" VALUE="Seu nome">

b) CAMPOS TIPO MÚLTIPLA ESCOLHA

- checkbox: insere um botão para escolha de opções. Pode ser escolhida mais de uma alternativa.

<FORM>

<INPUT TYPE="radio" NAME="Browser" VALUE="Netscape"> Netscape

<INPUT TYPE="radio" NAME="Browser" VALUE="Explorer"> Internet Explorer

<INPUT TYPE="radio" NAME="Browser" VALUE="Outro"> Outro qualquer

</FORM>

c) CAMPOS DE AÇÃO

submit: Insere o botão que causa o envio do formulário. Sua sintaxe mais simples é:

<INPUT TYPE="submit">

FORMULÁRIOS (ENTRADA DE DADOS – SELECT E TEXTAREA)

Para criar uma entrada de dados como a mostrada abaixo, temos que usar o comando:

<SELECT>

```
<FORM>
<SELECT NAME="browser">
<OPTION SELECTED>Netscape
<OPTION>Internet Explorer
<OPTION>Outro Borwser
</FORM>
```

Observando bem este código-fonte você notará que o comando SELECT tem uma estrutura diferente do INPUT. Neste caso é necessário usar um comando </SELECT> para "fechar" a seleção e entre esses dois comandos existirão todas as opções de seleção. Note que em um dos comandos <OPTION> existe uma diretriz SELECTED, que indica que esse item será adotado como padrão.

Agora só nos resta uma forma de entrada de dados. É a <TEXTAREA>, que de acordo com os atributos para linha e coluna cria uma área para entrada de texto. No exemplo baseado no código-fonte ao lado também foi atribuído uma mensagem inicial.

```
<FORM>
<TEXTAREA NAME="comentario" COLS="35" ROWS="4">
Deixe aqui seu recado :-)
</TEXTAREA>
</FORM>
```

2.4 JAVA

Esta linguagem exibe importantes características que fizeram com que fosse escolhida para o desenvolvimento desse projeto, tais como: puramente orientada a objetos, independente de plataforma, não utilização de ponteiros.

Segundo H. M. Deitel (2001, p.65), “Em maio de 1995, a Sun anunciou Java em uma conferência importante. Normalmente, um evento como esse não teria gerado muita atenção. Entretanto, Java gerou interesse imediato na comunidade comercial por causa do fenomenal interesse pela Word Wide Web. Java é agora utilizada para criar páginas da Web com conteúdo interativo e dinâmico, para desenvolver aplicativos corporativos de larga escala, para aprimorar a funcionalidade de servidores da Web.”

Para Horstmann e Cornell (2001), a mais importante vantagem de linguagem Java é a de ter uma biblioteca em tempo de execução que visa disponibilizar independência de plataforma, tendo a capacidade de se utilizar de um mesmo código em diferentes sistemas operacionais, o que é extremamente necessário para a programação na Internet.

Java consistem em partes chamadas classes. Estas consistem em partes chamadas métodos que realizam tarefas e retornam as informações ao completarem suas tarefas. As bibliotecas de classes são também conhecidas como Java APIs (*Applications Programming Interfaces* – interfaces de programas aplicativos).

Quanto ao ambiente Java, basicamente o sistema Java consistem em várias partes: um ambiente, a linguagem, a interface de programas aplicativos Java e várias bibliotecas de classes. Os programas Java normalmente passam por cinco fases para serem executados. Essas fases são: edição, compilação, carga, verificação, e execução.

Um programa fonte escrito em Java é traduzido pelo compilador para os bytecodes, ou seja, o código de máquina da Máquina Virtual Java (JVM). A máquina virtual Java é um programa capaz de interpretar os bytecodes produzidos pelo compilador, fazendo com que um programa Java possa ser executado em qualquer plataforma, desde que possua uma JVM.

O interpretador de Java pode executar os bytecodes do Java diretamente em qualquer máquina para o qual o interpretador tenha sido transportado.

a) Declaração de Classes

♦ Sintaxe de uma classe Java

```
<modificadores> class <nome>
{
    <declaração de atributos>*
    <declaração de construtores>*
    <declaração de métodos>*
}
```

♦ Exemplo

```
public class Retangulo
{
    int base, altura;
    public int calculaArea() { return base * altura ; }
}
```

b) Declaração de Atributos

♦ Sintaxe de um atributo Java

```
<modificadores> <tipo> <nome> [ = <valor inicial> ];
Onde:
<tipo> = byte | short | int | long | char | float | double
        boolean | <nome de uma classe>
```

♦ Exemplo:

```
public int x;
static double calc = 45.76;
String nome = "Rita Cadillac";
```

c) Declaração de Métodos

Sintaxe de um método Java

```
<modificadores> <tipo retornado> <nome> <parâmetro>*
{
    <expressão>*
}
```

♦ Exemplo:

```
public int calculaArea(int b, int a)
{
    int base = b, altura = a;
    return base * altura ;
}
```

d) Declaração de Construtores

♦ Sintaxe de um construtor em Java

```
<modificadores> <nome da classe> <parâmetro>*
{
    <expressão>*
}
```

Formato de Um Arquivo .java

♦ Sintaxe de um arquivo .java

```
[ <declaração de pacote>]
<declaração de import>
<declaração de classe>
```

♦ Exemplo:

```
package modulos.atual;
import java.awt.*;
import java.util.List;
import java.io.*;
public class InterfaceVendas{ .... }
```

Declaração de Pacote

- ♦ Sintaxe de um pacote em Java

`package <nome>[. <parâmetro>]*;`

- ♦ Exemplo:

`package projeto.interface;`

- ♦ Apenas uma declaração por arquivo;
- ♦ Deve ser a primeira expressão do arquivo;
- ♦ Nomes de pacotes devem ser hierárquicos e separados por um ponto e
- ♦ Se ausente então a classe pertence ao pacote default

2.4.1 JAVA Database Connectivity (JDBC).

É a utilização da linguagem JAVA para possibilitar os acessos a banco de dados.

O JDBC é uma API que possibilitará a adição de comandos SQL em uma aplicação Java.

a) Características:

- ♦ Permite o acesso de programas Java a Base de Dados via SQL (Structure Query Language);
- ♦ Possui suporte a uma grande variedade de Banco de Dados;
- ♦ Atualmente se encontra na versão 2.0 e
- ♦ Esta API é implementada através do pacote `java.sql`.

b) Vantagens:

- ♦ Mesma interface de programação para acesso a diferentes banco de dados;
- ♦ Uso da linguagem SQL de acesso a banco de dados;
- ♦ Utilização de ODBC para acesso a banco de dados e
- ♦ Alternativa: API JSQL.

c) Requisitos Básico.

- ♦ JDK 1.1 ou superior;
 - ♦ Sistema Gerenciador de Banco de Dados e
 - ♦ Driver Específico ou ODBC.
- d) Principais Classes / Interfaces
- ♦ DriverManager = Gerencia todos os drivers instalados na Java Virtual Machine;
 - ♦ Connection = Representa uma conexão ao banco de dados;
 - ♦ Statement = Representa uma requisição que será feita ao banco de dados.
 - ♦ ResultSet = Representa uma coleção de linhas que é o resultado de uma operação SQL do tipo SELECT.

Conexão ao banco.

a) Carga da classe do Driver:

- ♦ Class.forName(String nomeDoDriver);
- ♦ Levanta a exceção ClassNotFoundException caso a classe não possa ser localizada.

b) Criação da conexão ao banco:

- ♦ Connection conexão = DriverManager.getConnection(String url, String user, String password);
- ♦ Levanta a exceção SQLException;

Criação dos Statement

a) Criação do Statement:

- ♦ Statement st = conexao.createStatement();
- ♦ Executa uma operação SQL SELECT através do método "executeQuery(String sql);" que retorna um objeto ResultSet;
- ♦ Executa uma operação SQL UPDATE, INSERT ou DELETE através do método "executeUpdate(String sql)" que retorna um inteiro indicando quantas linhas do banco de dados foram modificadas;

Exemplo:

```
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection agend =
    DriverManager.getConnection      ("jdbc:odbc:Agendamento","","");
    Statement stmt = agend.createStatement();
    String insert = "INSERT INTO empregado (nome,endereco,idade,salario) VALUES
    ('rodolfo', 'rua Santa Clara', 50, 10000)";
    stmt.executeUpdate(insert);

    ResultSet rsLogin = stmt.executeQuery("SELECT nome FROM empregado WHERE
    salario > 5000");

    while(rs.next())      {
        System.out.println( rs.getString(1) );
    }
} catch (java.lang.ClassNotFoundException cnfe) {
    System.err.println("ClassNotFoundException: " + cnfe.getMessage());
} catch(SQLException sqle) {
    System.err.println("SQLException: " + sqle.getMessage());
}
```

2.4.2 JAVA SERVER PAGES (JSP).

Segundo Duane K. Freitas e Mark A. Kolb (2000), "JSP É uma Tecnologia baseada em Java que simplifica o processo de desenvolvimento de sites da web dinâmicos.". O seu ponto forte é utilizar e aproveitar todas as vantagens que a linguagem Java fornece.

Com JSP, os designers da web e programadores podem rapidamente incorporar elementos dinâmicos em páginas da web usando Java embutido e algumas tags de marcação simples. Estas tag(s) fornecem ao designer de HTML um meio de acessar dados e lógica de negócios armazenados em objetos Java sem ter que dominar as complexidades do desenvolvimento de aplicações.

Dentro das vantagens do JSP destacamos a utilização de uma linguagem orientada a objetos java possuindo, encapsulamento, tratamento de exceções e gerenciamento de memória automático, o uso de Java conduz a uma maior produtividade do programador e a código mais robusto. Pelo fato de haver uma API padrão publicada para JSP, e pelo fato do bytecode de Java compilado ser portátil através de todas as plataformas que aceitam uma JVM, o uso de JSP não o restringe ao uso de uma plataforma de hardware, sistema operacional ou software de servidor específico.

A mais disso, a JSP pode se aproveitar de todas as outras APIs de Java padrão, incluindo aquelas para acesso a bancos de dados compatíveis com muitas plataformas, serviços de diretórios, processamento distribuído e criptografia. Esta disponibilidade vem a aumentar uma gama de fontes de dados, para a criação de aplicações baseadas na web e repleta de recursos.

Ao incluir código fonte Java diretamente em páginas da web, ela também inclui um conjunto de tag(s) do tipo HTML para interagir com objetos Java que residem no servidor, em especial, estas tag(s) são projetadas para criar, realizar consultas e modificar JavaBens no servidor.

JavaBens são um conjunto de objetos escritos em Java, com as implementações de acordo com um conjunto de convenções projetadas para promover modularidade e reusabilidade. JavaBens usa um estilo de programação que conduz a pedaços de código de programa independentes de coisas externas, que encapsulam comportamento, funcionalidade ou dados relacionados, que podem ser usados e reutilizados em múltiplos contextos sem que seja preciso conhecer os detalhes de sua operação interna.

A questão de separação de implementação e apresentação se aproveitando do suporte embutido de JSP para JavaBens, é possível manter uma rígida separação entre apresentação de dados *a exibição de informação para o usuário final* e a implementação do programa *o código usado para gerar aquela informação*. A vantagem proveniente dessa separação de apresentação e implementação, é que as mudanças em um podem ser realizadas sem a necessidade de realizar qualquer mudança no outro.

O primeiro requisito para a separação dos dois elementos de aplicação para web é a utilização da sintaxe. Ao buscar a tag(s) do tipo HTML de JSP para acessar JavaBens e suas propriedades, podem ser escritas páginas JSP que não tenham código fonte de Java.

Já para o segundo requisito não deve haver o código HTML no seu objeto JavaBens. A princípio pode aparecer um absurdo, pois o JavaBens é escrito em Java e não em HTML.

Mais nada impede que seja criado um objeto JavaBens cujas propriedades incluam cadeias de código HTML.

2.4.2.1 TAG (S) JSP PARA CONSTRUÇÃO DE SCRIPTS

JSP fornece quatro principais categorias de tag (s) de marcação. A primeira *diretivas* é um conjunto de tag (s) para fornecer ao container JSP instruções específicas de página para como o documento contendo as diretivas deve ser processado. As diretivas não afetam o tratamento de solicitações individuais, mas, ao invés disso, afetam as propriedades globais da página JSP que influencia sua tradução em servlet.

Os elementos de criação de scripts são usados para embutir instruções de programação, escritos na linguagem de criação de scripts designada para a página, que devem ser executadas cada vez que a página é processada para uma solicitação. Alguns elementos de criação de scripts são avaliados puramente pelos seus efeitos colaterais, mas eles também podem ser usados para gerar conteúdo dinâmico que aparece no output da página.

2.4.2.2 ELEMENTOS DE CRIAÇÃO DE SCRIPTS.

Considerando que as diretivas JSP influenciam como a página é processada pelo container JSP, os elementos de criação de scripts permitem que os programadores embutam diretamente código em página JSP, incluindo código que gera output para parecer nos resultados enviados de volta para o usuário. JSP oferece três tipos de elementos de criação de scripts: declarações, scriptlets e expressões.

a) Declarações

As declarações são usadas para definir variáveis e métodos específicos para uma página JSP. Os métodos e variáveis declaradas podem então ser referenciadas por outros elementos de criação de scripts na mesma página. A sintaxe para declarações é:

```
<%! Declarações %>
```

Perceba que múltiplas declarações podem aparecer dentro de uma única tag, mas cada declaração deve ser uma instrução declarativa na linguagem de criação de scripts designada.

Declarações variáveis

As variáveis definidas como declarações se tornam variáveis de instância da classe de servlet na qual a página de JSP é traduzida e compilada. Considere a seguinte declaração de três variáveis:

```
<%! private int x=0, y=0; private String units = "fit ; %">
```

Esta declaração terá o efeito de criar três variáveis de instância no servlet criada para a página JSP, chamadas x, y e units. Estas variáveis podem ser referenciadas por qualquer um e todos os outros elementos de criação de scripts na página, incluindo aqueles elementos de criação de scripts que aparecem mais cedo na página do que a própria declaração.

As variáveis de classe, também chamadas de variáveis estáticas, são variáveis cujos valores são compartilhados entre todas as instâncias de uma classe, ao invés de serem específicas para uma instância individual. Quando a linguagem de criação de scripts é Java, as variáveis de classe são definidas usando-se a palavra chave static, como no exemplo abaixo:

```
<%! static public int counter = 0; %>
```

Declarações de métodos

Os métodos definidos através das declarações se transformam nos métodos da classe de servlet nos quais a página JSP é compilada. Por exemplo, a declaração abaixo define um método para processar fatoriais:

```
<%! public long fact (long x) {  
    if (x=0) return 1;  
    else return x * fact (x-1);  
} %>
```

Assim como acontece com as declarações variáveis, os métodos declarados podem ser acessados por todos e quaisquer os elementos de criação de scripts na página, independentemente da ordem na qual a declaração de métodos ocorre em relação aos outros elementos de criação de scripts.

b) Expressões

As declarações são usadas para adicionar variáveis e métodos a uma página JSP, mas não são capazes de contribuir diretamente para o output da página, que, é afinal de contas, o objetivo da geração de conteúdo dinâmico. No entanto, o elemento de expressões de

JSP é explicitamente objetivado para geração de output. A sintaxe para esse elemento de criação de scripts é a que se segue:

```
<%= expressão %>
```

c) Scriptlets

As declarações e expressões são intencionalmente limitadas nos tipos de código de criação de scripts que elas aceitam. Para a criação de scripts de objetivos gerais, a construção de JSP apropriada é o scriptlet. Os scriptlets podem conter instruções de linguagem de criação de scripts arbitrárias, que como as declarações, são avaliadas apenas pelos efeitos colaterais.

No entanto, os scriptlets não adicionam automaticamente conteúdo ao output de uma página JSP. A síntese geral para os scriptlets é a que se segue:

```
<% scriptlet %>
```

DIRETIVAS JSP

As diretivas são usadas para demonstrar informações de processamentos especiais a respeito da página para o container JSP. Por exemplo, as diretivas podem ser usadas para especificar a linguagem de criação de scripts para a página, incluir o conteúdo de uma outra página ou indicar que a página usa uma biblioteca de tag (s) personalizadas. As diretivas não produzem diretamente qualquer output que seja visível aos usuários finais quando a página é solicitada; ao invés disso, elas geram efeitos colaterais que mudam o modo que o container JSP processa a página.

a) Diretiva page

A diretiva page é a diretiva JSP mais complicada, principalmente porque ela suporta uma ampla gama de atributos e funcionalidade associada. A sintaxe básica da diretiva page é a que segue:

```
<%@ page attribute1="value1" attribute2="value2" attribute3=... %>
```

O espaço em branco depois da `<%@` de abertura e antes da `%>` de fechamento é opcional, mas é recomendado para melhorar a legibilidade. Assim como todos os elementos de tag (s) JSP, a diretiva page suporta uma linguagem baseada em XML, como se segue:

```
<jsp:directive.page attribute1="value1" Attribute2="value2" attribute3=... />
```

As especificações de atributos são idênticas para os dois estilos de tag (s), e há onze atributos diferentes reconhecidos para as diretivas page. Nos exemplos a seguir usaremos o primeiro estilo, apenas porque ele é ligeiramente menos prolixo e, por conseguinte aparece mais freqüentemente nas páginas JSP que são criadas manualmente. Tenha em mente que estes dois estilos de tag (s) são intercambiáveis.

Atributos aceitos pela diretiva page:

Atributo	Valor	Padrão	Exemplos
Info	Cadeia de texto	None	info="Resgistration form"
languag	Nome da linguagem de criação de scripts	"Java"	language= "java"
Content Type	Tipo MIME, conjunto de caracteres	See first exemple	ContentType=text/html; charset=ISSO-8859-1*
extends	Nome de classe	None	extends="com.taglib.wdjsp. MyJspPage"
import	Nomes de pacote E/ou classe	None	import="Java.netURL" import="Java.útil.* ,Java.text.*"
Session	Flag Boolean	"true"	session="true"
Buffer	Tamanho de Buffer ou false	6kb	buffer="12kb buffer="false"
autoFlush	Flag Boolean	"true"	autoFlush= "false"
IsThreadSafe	Flag Boolean	"true"	IsThreadSafe="true"
errorpage	URL local	None	errorpage="results/Failed.jsp"
IsErrorPage	Flag Boolean	"false"	isErrorPage="false"

Tabela 1- Atributos da Diretiva page.

Atributo info

O atributo info permite que o autor da página adicione uma cadeia de documentação à página que sumariza sua funcionalidade. Esta cadeia será então disponibilizada para uso pelo container JSP ou outras ferramentas de uma maneira programática para exibir a

informação de sumário. Não há restrições quanto ao conteúdo ou comprimento da cadeia de documentação, mas as informações de autor, versão e direito autoral são comumente incluídas, como o exemplo a seguir:

```
<%@ page info="The CLU homepage, Copyright 1982 by Kevin Flynn." %>
```

Atributo language

O atributo language especifica a linguagem de criação de scripts a ser usadas em todos os elementos de criação de scripts na página. Todos os containers JSP devem suportar Java como uma linguagem de criação de scripts e este é o padrão se o atributo language não for explicitamente especificado. O suporte para outras linguagens de criação de scripts é opcional e varia entre as implementações JSP. Eis aqui como o atributo language é usado para especificar Java como a linguagem de criação de scripts:

```
<%@ page language="java" %>
```

Atributo contentType

Esse atributo é usado para indicar o tipo MIME da resposta sendo gerada pela página JSP. Embora MIME signifique *Multipurpose Internet Mail Extensions*, os tipos MIME também são usados para indicar o tipo de informação contida em uma resposta de HTTP, e este é o contexto no qual eles são usados em JSP. Os tipos mais comuns de MIME para JSP são "text/html", "text/xml" e "text/plain", indicando respostas em HTML, XML e simples formatos de texto, respectivamente. Para especificar que um documento JSP está gerando um conteúdo XML, por exemplo, este atributo é especificado como se segue:

```
<%@ page contentType="text/xml" %>
```

Atributos extends

O atributo extends identifica a super classe a ser usada pelo container JSP quando ele está traduzindo a página JSP em um servlet Java, e é especificado como se segue:

```
<%@ page extends="com.taglib.wdjspPage" %>
```

Atributos Import

Diferentemente do atributo `extends`, o uso do atributo `import` é bem comum, porque estende o conjunto de classes Java que podem ser referenciadas em página JSP sem ter que explicitamente especificar os nomes de pacote de classes (Em outras palavras porque economiza digitação).

Para importar uma classe específica, simplesmente especifique seu nome (incluindo o pacote) como o valor do atributo `import`, como o que se segue:

```
<%@ page import= "java.util.list" %>
```

Se esta diretiva estiver presente em uma página JSP, a classe `java.util.list` pode ser mencionada naquela página ao simplesmente usar o nome de classe não qualificado, `List`, também chamado de seu nome base.

b) Diretiva include

A segunda diretiva JSP permite que os autores da página incluam o conteúdo de um arquivo em outro. O arquivo a ser incluído é identificado através de um URL local e a diretiva tem o efeito de substituir a si mesma pelo conteúdo do arquivo indicado. A sintaxe da diretiva `include` é a que se segue:

```
<%@ include file= "localURL" %>
```

OBJETOS IMPLÍCITOS

Como os exemplos apresentados sugerem, os elementos de criação de scripts JSP fornecem grande poder para criar, modificar e interagir com objetos Java a fim de gerar conteúdo dinâmico. As classes específicas de aplicação podem ser instanciadas, e valores das chamadas de método podem ser inseridos no output de JSP. Depósitos e recursos de rede, como banco de dados, podem ser acessados para armazenar e recuperar dados para uso pelas páginas JSP.

a) Objeto request

O objeto `request` representa a solicitação que acionou o processamento da página atual. Para solicitações de `http`, este objeto fornece acesso a todas as informações associadas com uma solicitação, incluindo sua fonte, o URL solicitado e quaisquer cabeçalhos, cookies

ou parâmetros associados com a solicitação. Exige-se que objeto request implemente a interface `javax.servlet.ServletRequest`. Quando o protocolo é http como normalmente é o caso, ele deve implementar uma subclasse desta interface, `javax.servlet.http.HttpServletRequest`.

Dentre os usos mais comuns para o objeto request, encontra-se a procura por valores de parâmetros e cookies. Eis aqui um fragmento de página ilustrando o uso do objeto request para acessar um valor de parâmetro:

```
<% String xStr = request.getParameter ("num") ;
    try { long x = long.parselong (xStr ;) %>
        factorial result: <%= x %>! = <%= fact (x) %> <% } catch
(numberFormatException e) { %>
            Deculpe, o parâmetro<b><%=xStr%> não especifica um valor inteiro.
        <% } %>
```

b) Objeto out

Este objeto implícito representa o fluxo de saída para a página, cujo conteúdo será enviado para o navegador como o corpo de sua resposta. O objeto out é uma instância da classe `javax.servlet.jsp.JspWriter`. Esta é uma classe abstrata que estende a classe `java.io.Writer` padrão, a complementado com diversos dos métodos fornecidos pela classe `java.io.PrintWriter`. Em particular, ela recebe como herança todos os métodos `writer ()` padrão fornecidos por `java.io.Writer`, e também implementa todos os métodos `print ()` e `println ()` definidos por `java.io.PrintWriter`.

Por exemplo o objeto out pode ser usado dentro de um scriptlet para adicionar conteúdo à página gerada, como no fragmento de página seguinte:

```

<P>Counting eggs
<% int count = 0 ;

    while (carton.hasNext() ) { count ++ ; out.print ( ". " ) ; }
%>
<BR>
There are <%= count %> eggs. </P>

```

c) Objeto session

Este objeto implícito de JSP representa a sessão atual de um usuário individual. Todas as solicitações feitas por um usuário, que são parte de única série de interações com o servidor da web, são considerados parte de uma sessão. Desde que novas solicitações por aqueles usuários continuem a ser recebidas pelo servidor, a sessão persiste.

Um dos usos principais para o objeto session é armazenar e recuperar valores de atributos, a fim de transmitir as informações específicas de usuários entre as páginas. Como um exemplo, eis aqui um scriptlet que armazena dados na sessão, na forma de um objeto hipotético

```

UserLogin: <%      UserLogin    UserData = new UserLogin (name, password) ;

session.setAttribut ( "login", userData) ; %>

```

d) Objeto Exception

Assim como o objeto session, o objeto exception não está automaticamente disponível em todas as páginas JSP. Ao invés disso, este objeto está disponível apenas nas páginas que tenham sido designadas como páginas de erro, o objeto exception será uma instância da classe `Java.lang.Throwable` correspondente ao erro não pego que fez com que o controle fosse transferido para a página de erro.

Eis na folha seguinte um fragmento de página demonstrando o uso do objeto exception:

```

<%@      page  isErrorPage="true"  %>
<H1>Warning!      </H1>
The following error has been detected: <BR>
<B><%=      exception      %></B><BR>
<%      exception.printStackTrace      (out) ; %>

```

Neste exemplo, o objeto `exception` é referenciado tanto em uma expressão quanto em um scriptlet.

2.4.2.3 OS FUNDAMENTOS BÁSICOS DE JAVABEANS

JavaBeans são componentes de software escritos em Java. Os próprios componentes são chamados de Beans e devem aderir as especificações delineadas na API de JavaBeans. Eles encapsulam tanto estado quanto comportamento.

Os containers Bean permitem o trabalho com Beans em termos de propriedades atributos nomeado do Bean que mantêm seu estado e controlam seu comportamento. Existem *JavaBeans de dados*, fornecem uma maneira conveniente de acessar dados, os *Beans de visuais*, que são elementos com campos de texto, seletores e outros recursos úteis para criar interface com o usuário. E por último os *Bens de serviço*, também chamados de Beans trabalhadores. Eles recuperam informações de um banco de dados, realizar cálculos ou formatar informações.

Quanto as tag(s) temos:

```
<jsp:useBean>
```

Informa ou diz à página que você quer disponibilizar um Bean para a página. Esta tag cria um Bean ou pega de algum servidor existente.

Duas sintaxe básicas dessa tag:

```
<jsp:useBean id=" bean nome" class="bean class"/>
```

```
<jsp:useBean id=" bean nome" class="bean class">
```

```
    initialization code
```

```
</jsp:useBean>
```

Nessa tag foram utilizada dois atributos o de id (identificador de Java) e a class que é o nome da classe Java.

A tag `<jsp:setProperty>` necessita de três atributos : name, property e value, o atributo name especificado o Bean com qual você esta se trabalhando.

Sintaxe da tag;

```
<jsp:setProperty name = "bean name" property=" property name"/>
```

Exemplo definindo a propriedade daysLeft para um valor de 30.

```
<jsp:setProperty name = "user" property=" daysLeft" value="30" />
```

```
<jsp:setProperty name = "user" property=" daysLeft" value="<%=15 + 2 %>"/>
```

2.5 ACCESS

Um Banco de Dados é um conjunto de informações que estão relacionadas a um tópico ou propósito em particular. O segredo da armazenagem e recuperação eficiente de seus dados é o processo de planejamento. Identificando primeiro aquilo que você quer que o seu Banco de Dados faça, você poderá criar um projeto prático que resultará numa ferramenta de gerenciamento de banco de dados mais precisa e rápida.

O ACCESS ajuda a você a gerenciar o banco de dados fornecendo uma estrutura eficiente para armazenagem e recuperação de informações. O local onde são unidas as informações sobre cada assunto que você decidir acompanhar é uma tabela.

Como o ACCESS é um sistema relacional de gerenciamento de banco de dados, você pode organizar os dados de áreas de assunto diferentes em tabelas, e depois criar relações entre as tabelas. Essa abordagem facilita a reunião dos dados relacionados quando isso for necessário. Estabelecendo relações entre tabelas individuais, em vez de armazenar todas as suas informações em uma tabela grande, você evita excessiva duplicação de dados, economiza espaço de armazenagem no computador e maximiza a agilidade e precisão do trabalho com seus dados.

O Access utiliza uma marca única, chamada Chave Primária, para identificar cada Registro Chave em sua tabela. Assim como o número da placa identifica um automóvel, a Chave Primária identifica unicamente um registro.

Toda tabela deve possuir no mínimo uma Chave Primária. Você seleciona o campo ou os campos que compõem a Chave Primária, pelos métodos clicar ou clicar e arrastar.

O Access é um aplicativo para criação de banco de dados poderosos. Desenvolvido para ser um verdadeiro aplicativo Windows para criação de banco de dados, com finalidade de interagir com outros aplicativos Windows tornando o manuseio do usuário uma tarefa fácil.

2.5.1 TIPOS DE CONSULTAS:

a) Instrução SELECT

Instrui o programa principal do banco de dados para retornar a informação como um conjunto de registros.

Sintaxe

```
SELECT [predicado { * | tabela.* | [tabela.]campo1 [AS alias1] [, [tabela.]campo2
[AS alias2] [, ...]]}
FROM expressão[tabela] [, ...] [IN bancodedadosexterno]
[WHERE... ]
[GROUP BY... ]
[HAVING.. ]
[ORDER BY... ]
[WITH OWNERACCESS OPTION]
```

A execução dessa consulta é da seguinte forma, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

A instrução SELECT **não muda** os dados no banco de dados.

SELECT é normalmente a primeira palavra em uma instrução SQL. A maior parte das instruções SQL são instruções SELECT.

A sintaxe mínima da instrução SELECT é:

SELECT campos FROM tabela

Você pode usar um asterisco (*) para selecionar todos os campos na tabela. O exemplo abaixo seleciona todos os campos na tabela Funcionários:

```
SELECT * FROM Funcionários;
```

ORDER BY é opcional.

Entretanto, se você quiser exibir seus dados na ordem classificada, você deve utilizar ORDER BY. O padrão ordem de classificação é ascendente (A a Z, 0 a 9). Os dois exemplos abaixo classificam os nomes dos funcionários pelo sobrenome.

```
SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome;
```

Esse exemplo seleciona todos os campos da tabela "Funcionários".

```
SELECT Funcionários.* FROM Funcionários;
```

b) Instrução INSERT INTO

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

Sintaxe

Consulta anexação de vários registros:

```
INSERT INTO destino [IN bancodedadosexterno] [(campo1[, campo2[, ...]])]  
SELECT [origem.]campo1[, campo2[, ...]]  
FROM expressãodetabela  
Consulta anexação de um único registro:  
INSERT INTO destino [(campo1[, campo2[, ...]])]  
VALUES (valor1[, valor2[, ...]])
```

Você pode usar a instrução INSERT INTO para adicionar um único registro a uma tabela usando a sintaxe de consulta anexação de um único registro como mostrado acima. Neste caso, seu código especifica o nome e o valor de cada campo do registro. Você precisa especificar cada um dos campos do registro para os quais um valor deve ser designado e um valor para este campo. Quando você não especifica cada campo, o valor padrão ou Null é inserido nas colunas omitidas. Os registros são adicionados no final da tabela.

Você também pode usar INSERT INTO para anexar um conjunto de registros de outra tabela ou consulta usando a cláusula SELECT ... FROM como é mostrado acima na

sintaxe consulta anexação de vários registros. Neste caso, a cláusula SELECT especifica os campos para acrescentar à tabela destino especificada.

INSERT INTO é opcional, mas quando incluída, precede a instrução SELECT.

Exemplo de instrução INSERT INTO

Esse exemplo seleciona todos os registros de uma tabela hipotética "Novos Clientes" e os adiciona à tabela "Clientes" (quando não são designadas colunas individuais, os nomes das colunas das tabelas SELECT devem corresponder exatamente aos da tabela INSERT INTO).

```
INSERT INTO Clientes SELECT [Novos Clientes].*
```

```
FROM [Novos Clientes];
```

Esse exemplo cria um novo registro na tabela "Funcionários"

```
INSERT INTO Funcionários (Nome,Sobrenome, Título)
```

```
VALUES ("André", "Pereira", "Estagiário");
```

Esse exemplo seleciona todos os estagiários de uma tabela hipotética "Estagiários" que foram contratados há mais de 30 dias e adiciona seus registros à tabela "Funcionários".

```
INSERT INTO Funcionários SELECT Estagiários.*
```

```
FROM Estagiários WHERE DataContrato < Now() - 30;
```

c) Declaração UPDATE

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

Sintaxe

```
UPDATE tabela
```

```
SET valornovo
```

```
WHERE critério;
```

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo. O exemplo abaixo aumenta o Valor do Pedido em 10 por cento e o valor do Frete em 3 por cento para embarques do Reino Unido:

```
UPDATE Pedidos SET ValorPedido = ValorPedido * 1.1, Frete = Frete * 1.03
```

```
WHERE PaísEmbarque = 'RU';
```

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

Exemplo de instrução UPDATE

Esse exemplo muda os valores no campo "RelatórioPara" para 5 para todos os registros de funcionários que atualmente têm valores de RelatórioPara de 2.

```
UPDATE Funcionários SET RelatórioPara = 5
```

```
WHERE RelatórioPara = 2;
```

d) Instrução DELETE

Cria uma consulta exclusão que remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.

Sintaxe

```
DELETE [tabela.*]
```

```
FROM tabela
```

```
WHERE critério
```

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

Consulta de exclusão exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para Null.

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

Exemplo de instrução DELETE

Esse exemplo exclui todos os registros de funcionários cujo título seja Estagiário. Quando a cláusula FROM inclui apenas uma tabela, não é necessário indicar o nome da tabela na instrução DELETE.

```
DELETE*FROMFuncionáriosWHERE Título='Estagiário'.
```

ANÁLISE E IMPLEMENTAÇÃO DO SISTEMA

3.1 Escopo do Sistema Caso de Uso.

A partir das análises de requisitos, foi escolhido para se iniciar, a utilização da Linguagem de Modelagem Unificada – UML, pois ela proporcionará uma padronização na modelagem orientada a objetos. Iniciando com o Caso de Uso do Sistema exposto na figura 06 abaixo:

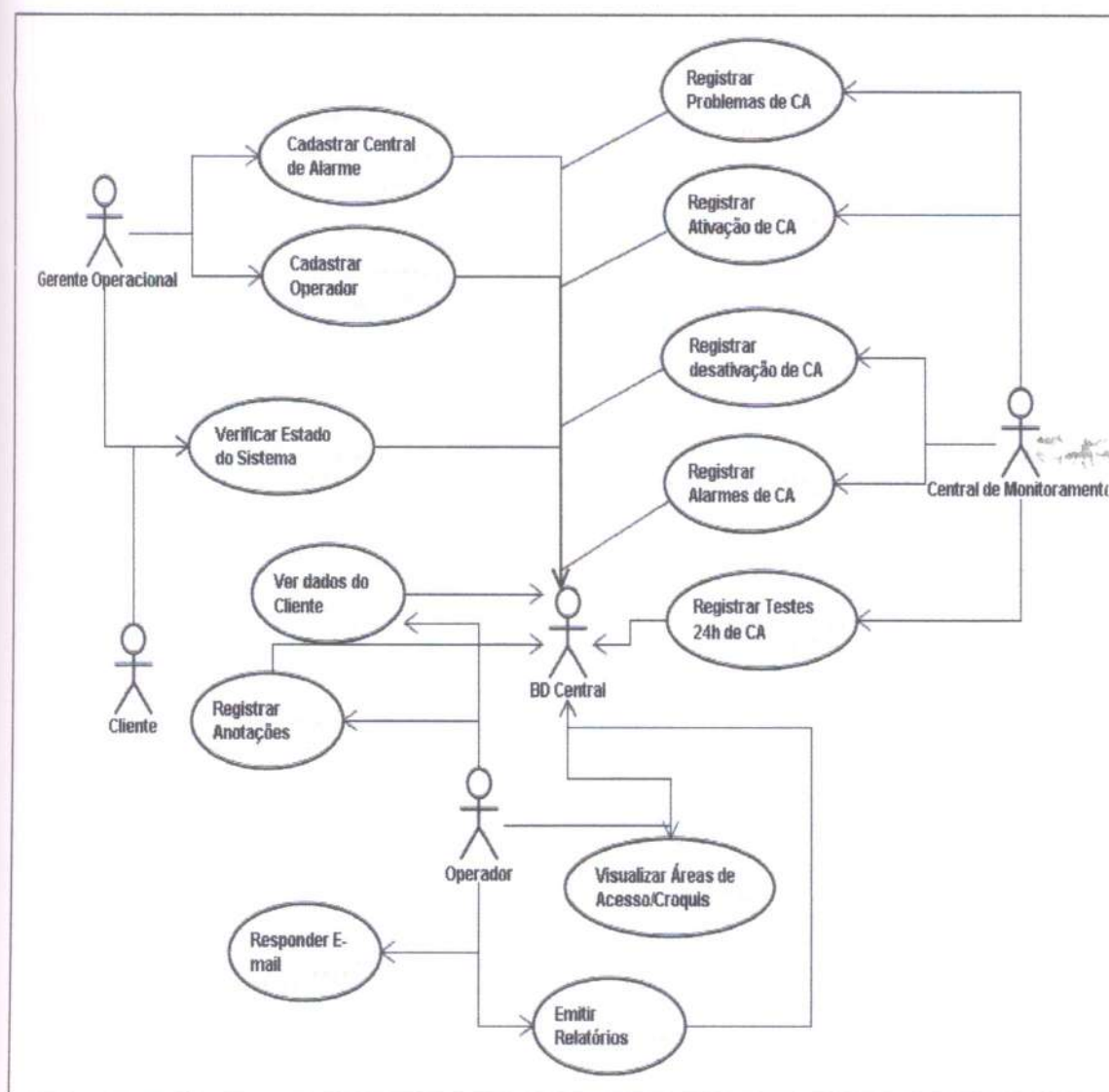


Figura. 6 - Diagrama de Caso de Uso do Sistema

Na sequência desta sessão serão apresentadas as expansões dos casos de uso do tipo primários do sistema. As apresentações são:

1. Cadastrar Central de Alarme;

2. Visualizar Áreas de Acesso e
3. Verificar Estado do Seu Sistema.

As expansões dos Casos de Uso do tipo primário:

1. Cadastrar Central de Alarme.

Caso de Uso: Cadastrar cliente (Central de Alarme)

Ator: Gerente operacional

Finalidade: Cadastrar um cliente no banco de dados do sistema.

Visão Geral: Gerente operacional tem acesso ao sistema de monitoramento e faz o cadastro do cliente, para posteriormente poder utilizar esses dados.

Seqüência Típica de Eventos:

0. Este caso de uso tem início quando o Gerente operacional inicia o processo de cadastramento do novo cliente;
1. Clicar na opção cadastrar cliente;
2. O sistema exibe o formulário para a inserção dos dados do cliente;
3. Preenchimento de todos os campos do formulário; estes incluem: número de identificação nome, razão social, endereço, login e senha;
4. Clicar no botão Enviar (Cadastrar);
5. O sistema verifica todos os campos preenchidos;
6. O formulário é submetido à aprovação do administrador;
7. O sistema retorna a mensagem "Cliente cadastrado com sucesso";

Seqüência Alternativa de Eventos:

- a) Linha 3: Caso o gerente não preencha todos os campos do formulário, o sistema retorna a mensagem "Completar campos não preenchidos";
- b) Linha 5: O sistema verifica o login e senha digitado pelo gerente e caso este já exista retorna a mensagem "login e/ou senha já existente" e solicita ao gerente digitar novo login e/ou senha;
- c) Linha 5: Ao verificar a senha digitada pelo gerente e caso esta contenha caracteres inválidos, o sistema retorna a mensagem "Senha inválida" e solicita ao gerente digitar nova senha;

2. Visualizar áreas de acesso

Caso de Uso: Visualizar áreas de acesso

Atores: Operadores, BD do Cliente

Finalidade: Disponibilizar uma melhor análise do estabelecimento violado para se chegar a uma conclusão mais precisa quanto a área violada e passar para os supervisores da ronda.

Visão Geral: O sistema apresentará uma posição de forma visual (croquis) do local do disparo, para que assim possa verificar as possíveis formas da violação (entrada) de um indivíduo no setor do alarme.

Tipo: Primário e Essencial.

Seqüência Típica de Eventos:

1. Este caso de uso tem início quando um usuário é cadastrado, pois assim é feito um croquis do seu estabelecimento e enviado para o banco de dados do cliente;
2. Todos esses eventos e as tomadas de decisão dos operadores e supervisores de ronda, de acordo com os devidos eventos serão armazenados em um BD do determinado Cliente em questão ;
3. Quando o sistema detecta uma violação o envia para a central de monitoramento onde o operador visualizará. Todas essas informações são gravadas no sistema e assim fica a espera de uma solicitação;
4. Visualizar o campo violado tanto de forma escrita, como de forma ilustrativa, mostrando as possíveis formas de entrada no local em específico.

Seqüência Alternativa de Eventos:

- a) Linha 1: Caso o usuário seja cadastrado e não tenha sido preenchido os campos do Croquis do estabelecimento não será possível mostrá-lo, no momento da violação linha 4. A visualização será apenas na forma escrita.

3. Verificar Estado do Seu Sistema

Caso de Uso: Verificar Estado do Seu Sistema

Atores: Cliente, BD do Cliente

Finalidade: Analisar a situação do Sistema. Seu status.

Visão Geral: Parte do Sistema que possibilitará que o cliente observe os eventos ocorridos no seu sistema isso a partir do dia especializado, e se não especializado o mesmo dará os eventos do dia até momento do acesso.

Tipo: Primário e Essencial.

Seqüência Típica de Eventos:

4. Este caso de uso tem início quando um usuário ativa via teclado o sistema de alarme, assim esse eventos e os demais como alarmes, testes 24h, avisos de problemas e desativação estarão por vir;
5. Todos esses eventos e as tomadas de decisão dos operadores e supervisores de ronda, de acordo com os devidos eventos serão armazenados em um BD do determinado Cliente em questão ;
6. O cliente acessa o sistema para analisar e/ou averiguar o estado do seu sistema, usando seu login e senha ;
7. De acordo com dia o sistema fornecerá ao usuário e aos administradores que horas fechou, abriu, se houve alarme(s) e avisos de problemas com os seus devidos procedimentos de operação, tanto por parte dos operadores como por parte dos supervisores de ronda ;

Seqüência Alternativa de Eventos:

- a) Linha 2: Caso o usuário não preencha os campos de login e senha corretos não terá acesso ao seu sistema. Assim o sistema retorna a mensagem "Senha ou login inválidos" e solicitará ao usuário digitar nova senha;

3.2 Telas principais do Sistema.

Ao entrar no Sistema é solicitado ao visitante sua Senha e Login, para que assim possa lhe dar capacidade de usufruir do sistema, Figura – 07.

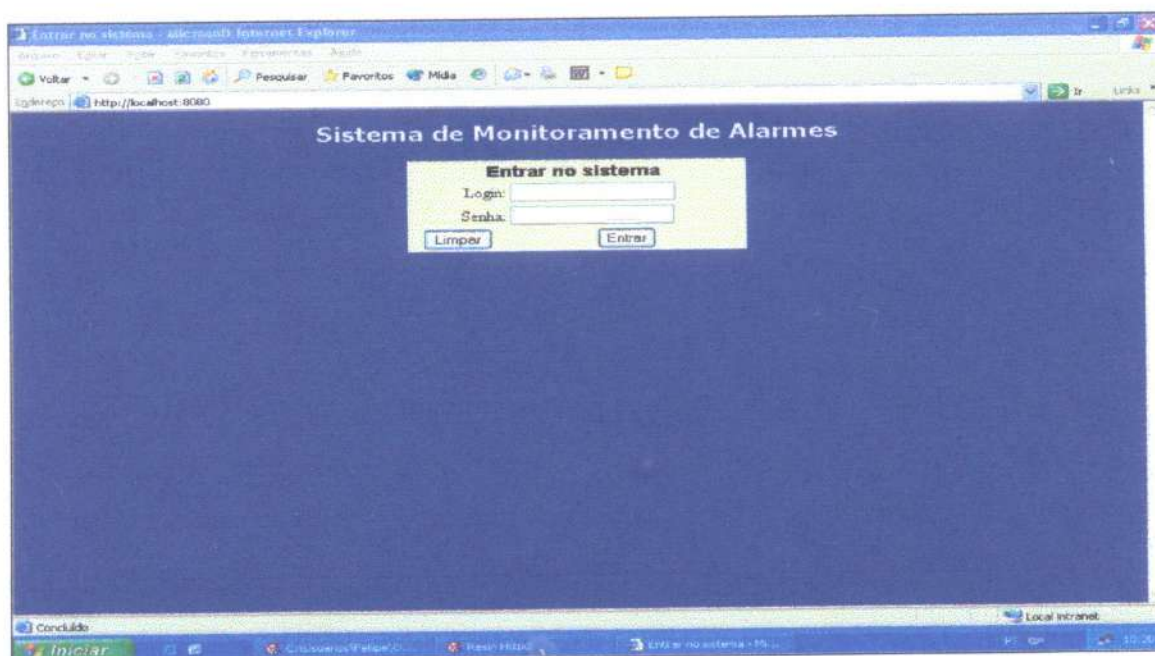


Figura. 7 –Tela de Entrada no Sistema

Após acessar o endereço da página do Sistema o usuário terá que entrar com seu login e senha e em seguida clicar na tecla de entrar. Caso haja erro de senha ou login, será mostrada na tela para o usuário a mensagem de: “erro ao digitar senha ou login”. Figura – 8. Parâmetro inválidos Figura – 9. E assim será solicitada uma nova senha e login.

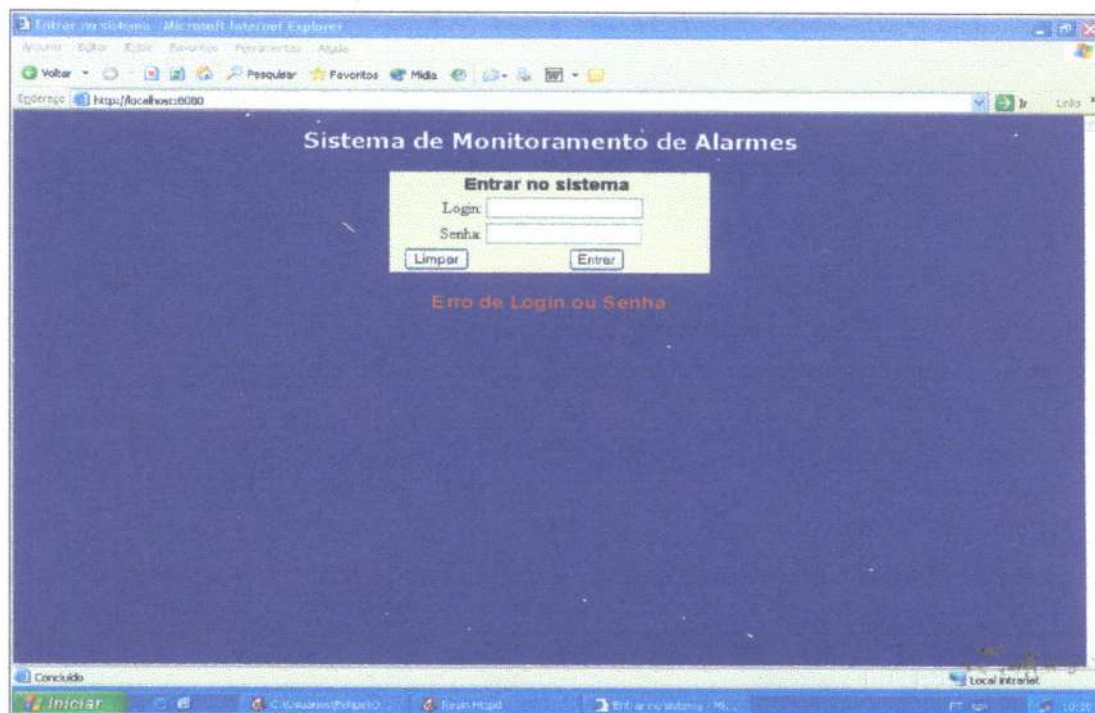


Figura 8 - Entrada no Sistema - Erro Login ou senha

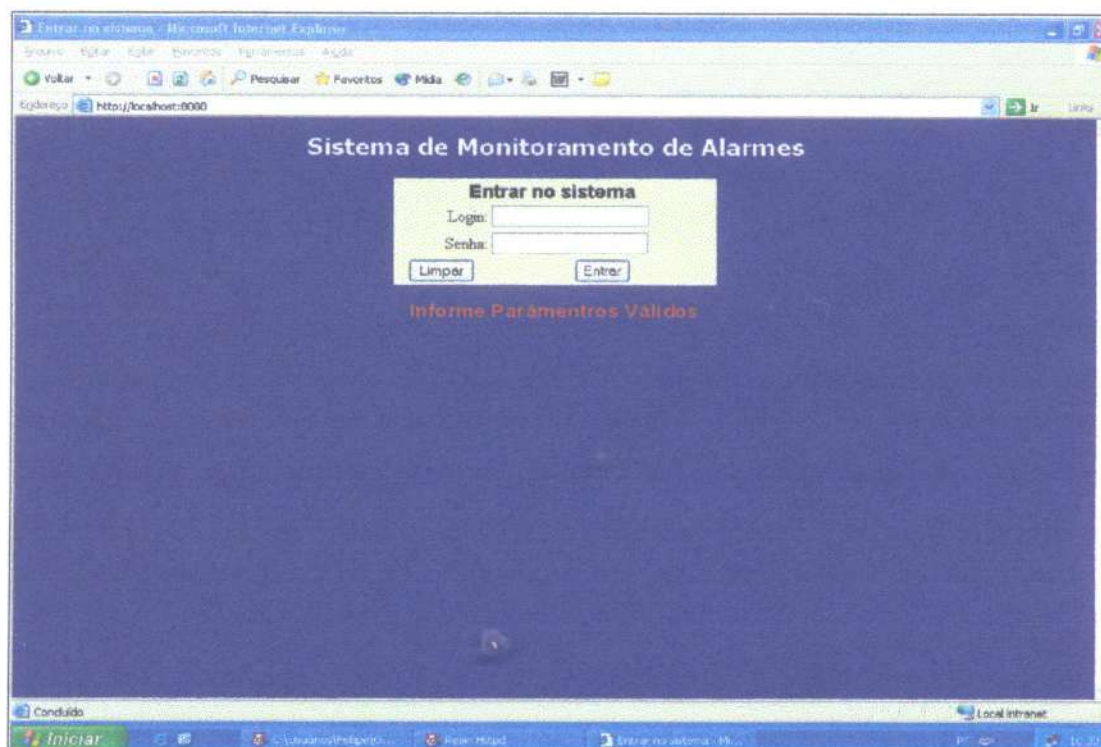


Figura 9 - Entrada no Sistema – Parâmetro Inválidos

Logo após o sucesso na entrada do Sistema será apresentado a tela principal do Sistema de Monitoramento na Figura – 10.

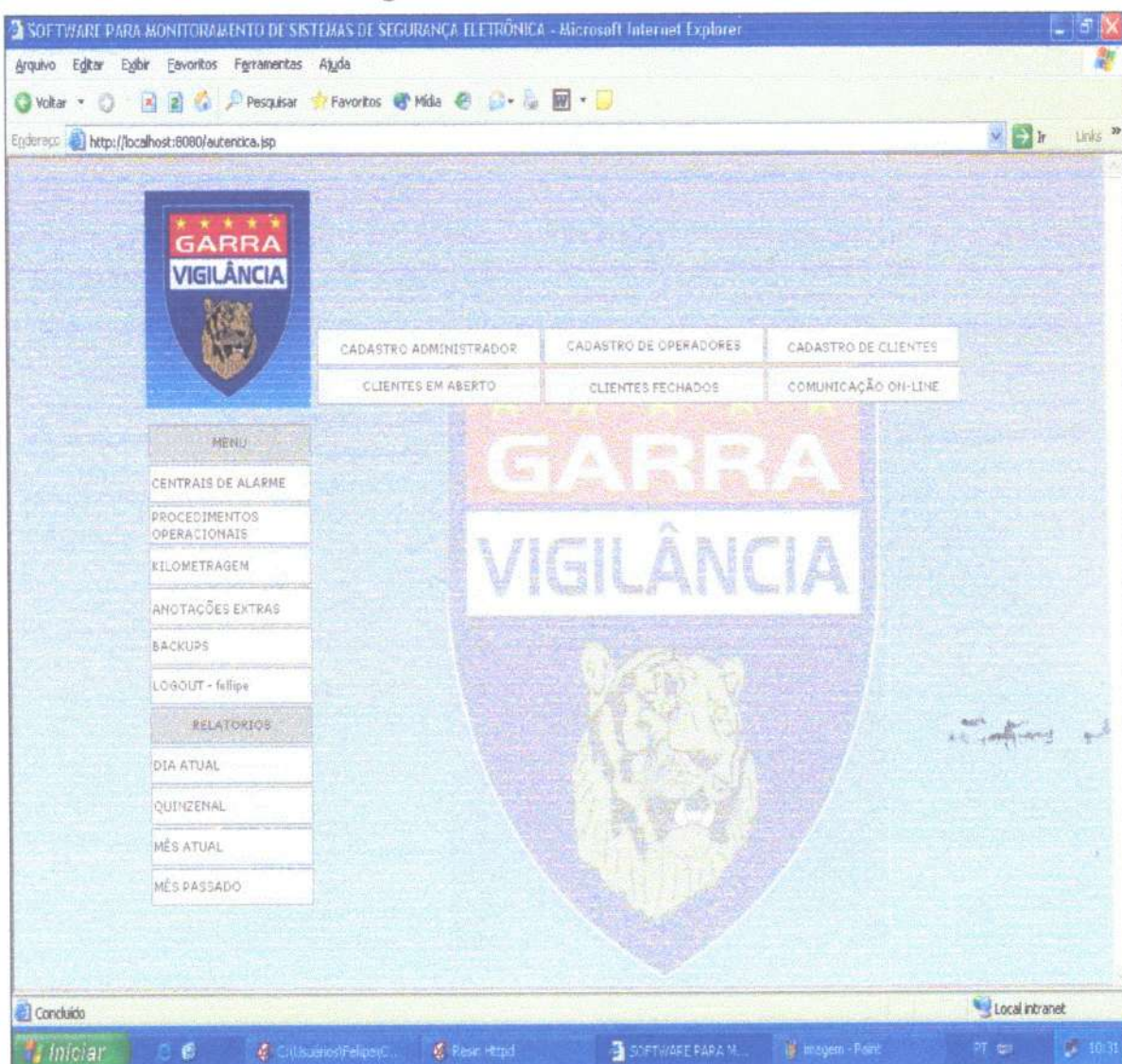


Figura. 10 – Entrada no Sistema – Tela Principal

Nesta tela terá para Clientes, outros links, que proporcionará agilidades quanto as informações úteis do seu sistema, como por exemplo se estar fechado ou aberto qual funcionário realizou tal ação, isso de qualquer lugar do mundo.

Relatórios quinzenal, mês atual ou passado.

Se houve alarme na (s) noite(s) passada (s), pelo link de “Anotações Extras”, que trará todos os procedimentos tomados pelo Setor de Monitoramento e de Patrulhamento, junto ao seu estabelecimento, link de “relatórios”.

Poderá entrar em contato com Monitoramento, via e-mail.

Este software proporcionará para os Operadores o “poder” de analisar dados mais detalhados dos Clientes, terão que se logar durante seu horário de trabalho e assim, serão responsáveis pelas respostas dos e-mail, pelas anotações extras, visualização do croquis dos clientes e possíveis alarmes.

Este programa também informará automaticamente os Clientes que ainda não fecharam ou que ainda não abriram.

3.3 Escopo do Sistema Diagrama de Classe

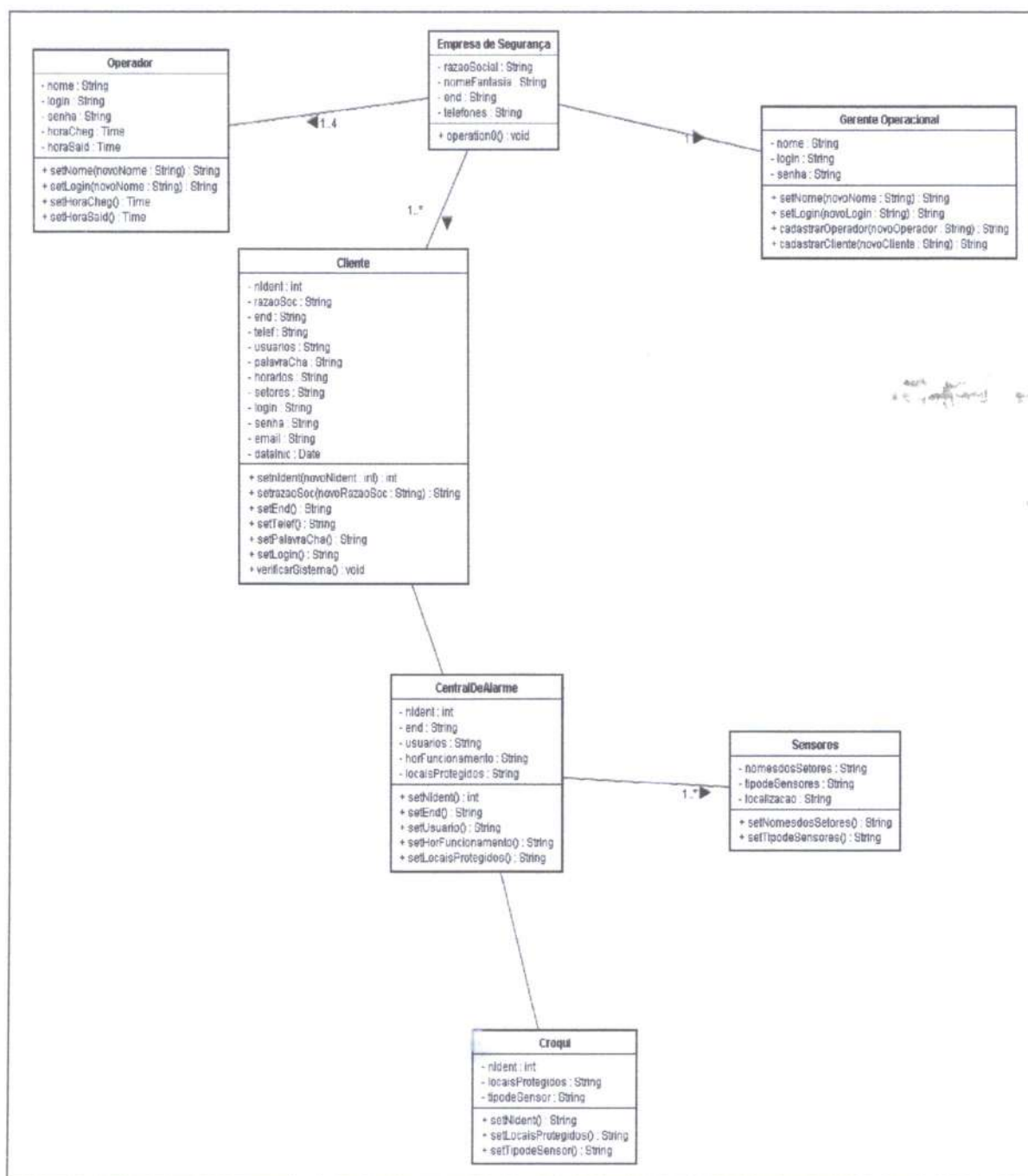


Figura. 11- Diagrama de Classe

As principais tabelas utilizadas até o momento e seus respectivos relacionamentos. Várias Centrais de Alarmes para uma Empresa, muitos usuários para uma Central e mais de um Administrador para uma Empresa de Segurança. Assim se comporta nosso banco até o momento.

3.4 Modelo do Banco

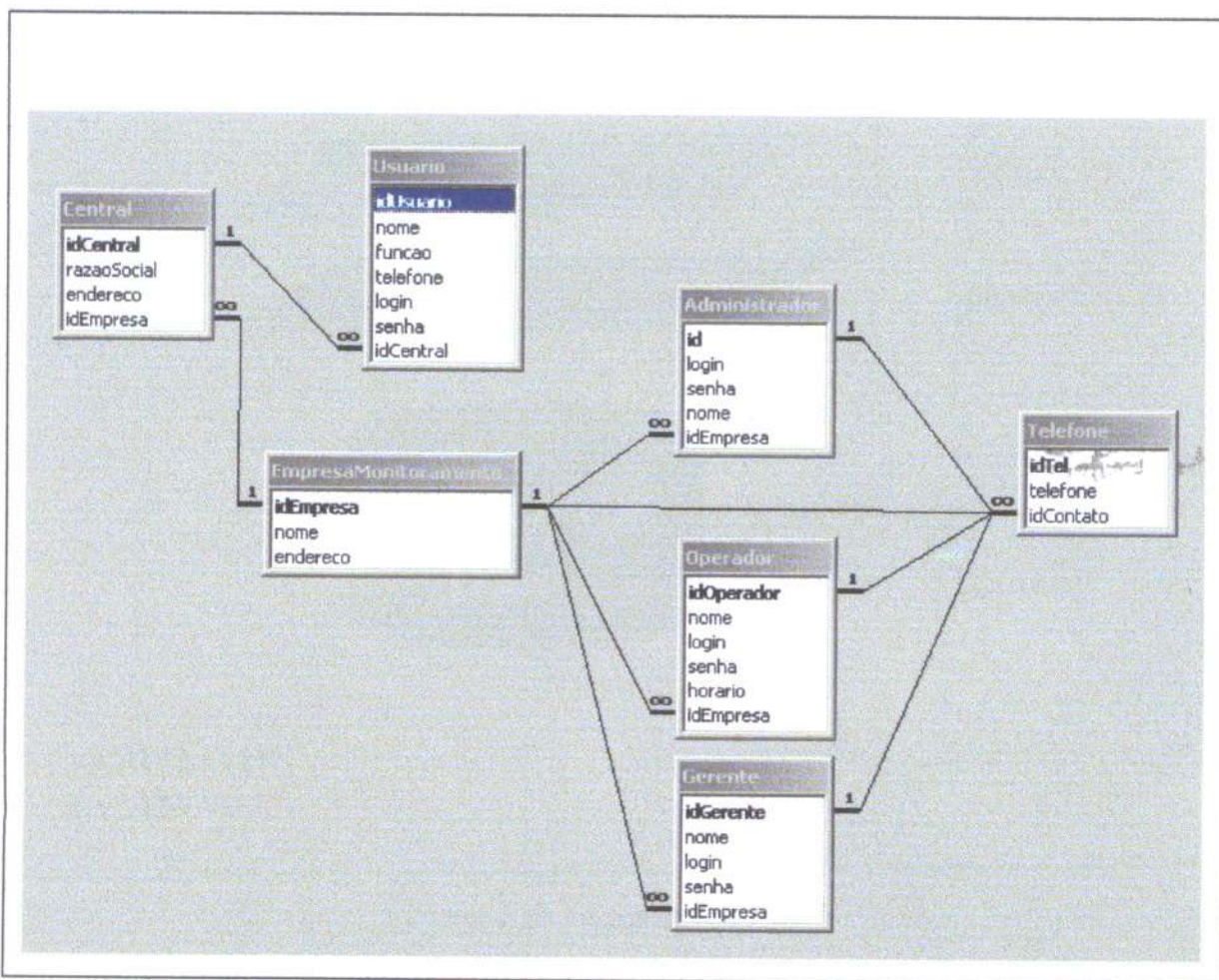


Figura. 12- BD do Sistema

3.5 ARQUITETURA DO SISTEMA

O software tem sua estrutura dentro do método de três camadas, sendo a primeira de Apresentação desenvolvida em JSP, a qual terá a comunicação com o usuário do programa, a segunda de negócio, responsável pelo recebimento e transferência de dados para a terceira camada chamada de Persistência que terá a função de conectar-se com o banco de dados (Access) para as devidas transações.

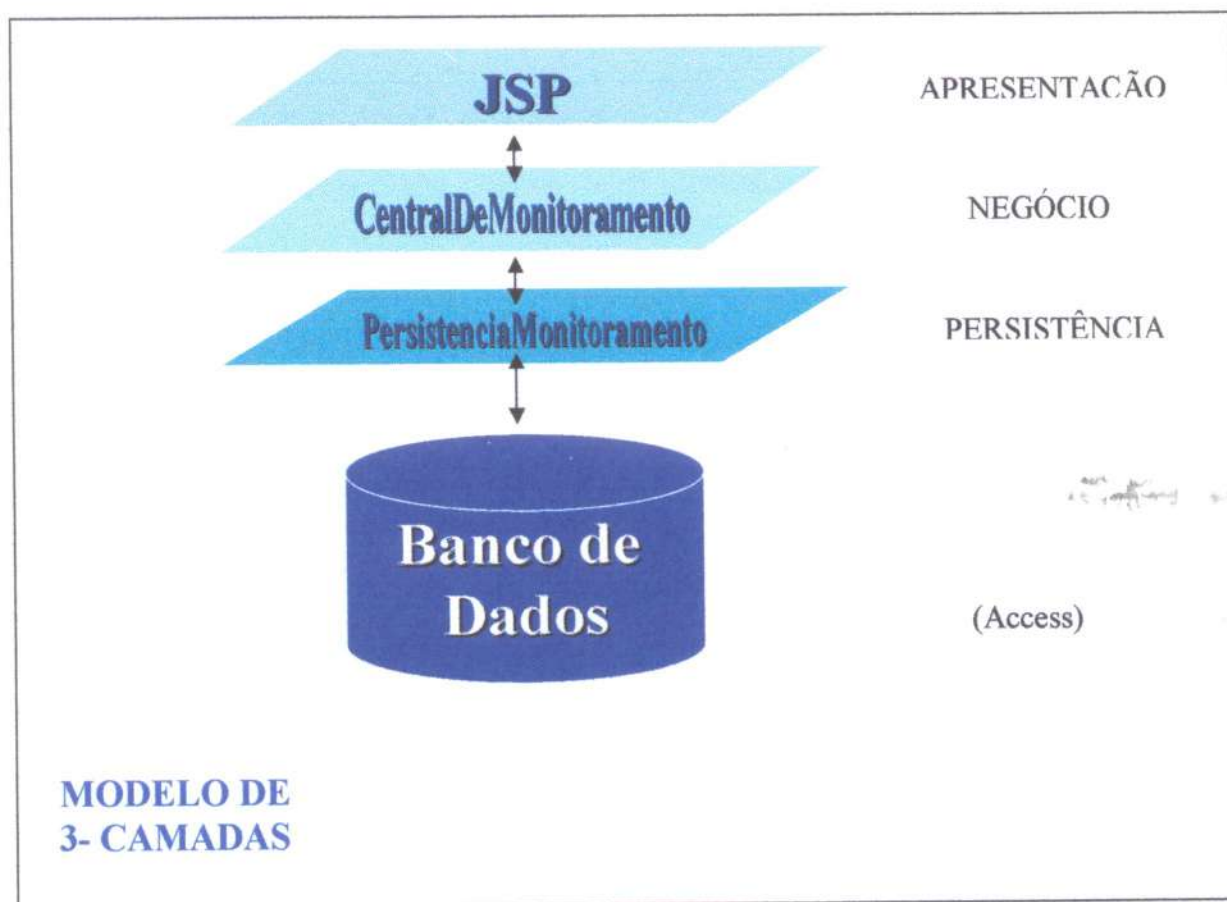


Figura. 13- modelo de 3 Camadas

Vejamos algumas das classes utilizadas no Sistema de Monitoramento:

A classe da Camada de Negócio com suas declarações de pacotes. Para o Sistema é de suma importância, pois é ela que recebe os dados, verifica se existe o contato e assim “conversa” com a 3ª camada, do banco de dado.

Abaixo temos a 2ª camada, que chamamos de negócio:

3.6 Código JAVA

a) Classe CentraldeMonitoramento

```
package br.farn.dorinaldo;

import java.util.*;

public class CentraldeMonitoramento {
    PersistenciaMonitoramento banco;

    public CentraldeMonitoramento() {
        banco = new PersistenciaMonitoramento();
    }

    public boolean autentica(String login, String senha) {
        return banco.autentica(login, senha);
    }

    public Vector lista() {
        return banco.listaCentrais();
    }
}
```

A classe CentraldeMonitorameto com os pacote br.farn.dorinaldo é uma classe de domínio público (modificador de acesso). Um objeto foi instanciado da classe PersistenciaMonitoramento, para se utilizar na autenticação do login e senha do sistema e na exposição através do Vector de uma lista de Centrais.

b) Classe PersistênciaMonitoramento

Nessa classe é utilizada o pacote br.farn.dorinaldo com as declarações de import "java.sql" e "java.util". Esta API é implementada através do pacote java.sql.

Ela será responsável em fazer a comunicação entre a camada de negócio com o banco de dados do Sistema.

```
package br.farn.dorinaldo;

import java.sql.*;
import java.util.*;

public class PersistenciaMonitoramento {
    private Connection con;
    private Statement instr;
    private ResultSet resultado;
    private String erro;

    public PersistenciaMonitoramento() {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con =
DriverManager.getConnection("jdbc:odbc:Monitoramento", "login", "senha");
        } catch ( ClassNotFoundException cnfex ) {
            erro = cnfex.getMessage();
        } catch ( SQLException sqllex ) {
            erro = sqllex.getMessage();
        }
    }
}
```

```

public Vector listaCentrais() {
    String consulta = "SELECT * FROM Central";
    System.out.println(consulta);
    System.out.println("Erro: "+erro);

    Vector resposta = new Vector();
    if ( con!=null ) {
        try {
            instr = con.createStatement();
            resultado = instr.executeQuery(consulta);
            while( resultado.next() ) {
                Vector linha = new Vector();
                linha.addElement(""+resultado.getLong(1));
                linha.addElement(resultado.getString(2));
                linha.addElement(resultado.getString(3));
                resposta.addElement(linha);
            }
            instr.close();
        } catch ( SQLException sqllex ) {
            erro = sqllex.getMessage();
            System.out.println("Erro: "+erro);
        }
    }
    return resposta;
}
}

```

Essa classe tem modificadores de acesso do tipo private, sendo assim só ela se comunica com o banco de dado do Software, nela terá as funções de inserção, modificação e retiradas de clientes, centrais e operadores do Software.

Uma lista de Centrais por completo é solicitada ao banco de dado.

O objeto Statement tem a grande finalidade de enviar as solicitações SQL para o banco de dados do Software.

3.6.1 Código JSP

Neste temos o Código JSP que terá a função de expor uma lista de Centrais pré cadastradas, a utilização de JavaScript para inserir dinamismo e lógica na página na Web.

a) Lista de Centrais

```
<%@ page import="br.farn.dorinaldo.*, java.util.*" %>
<%@ include file="controleSessao.jsp"%>
<html>
  <head>
    <title>SOFTWARE PARA MONITORAMENTO DE SISTEMAS DE SEGURANÇA
    ELETRÔNICA</title>
    <style type="text/css">
      <!--
        .link { font-family: Verdana, Arial, Helvetica, sans-serif;font-size:9px;text-
        decoration:none;color:#666666}
        .link:HOVER { font-family: Verdana, Arial, Helvetica, sans-serif;font-
        size:9px;text-decoration:underline;color:#006600}
        .box { background-color:#F9F9F9;border:1px solid #AAAAAA;height:27}
        .box3 { background-color:#CCCCCC;border:1px solid #AAAAAA;height:27}
        .box2 {
          border-top: 1px none #AAAAAA;
          border-right: 1px solid #AAAAAA;
          border-bottom: 1px solid #AAAAAA;
          border-left: 1px solid #AAAAAA;
        }
      -->
    </style>
    <script language="JavaScript">
      function hiliteOver(tdval)
      {
        tdval.style.backgroundColor = "#EFEFEE";
        tdval.style.border = "1px solid #29166F";
      }

      function hiliteOut(tdval)
      {
        tdval.style.backgroundColor = "#F9F9F9";
        tdval.style.border = "1px solid #AAAAAA";
      }
    </script>
  </head>

  <body background="imagens\fun.do.gif">
```

Nas linhas do código em destaque de azul, foi utilizado JavaScript, para inserir um maior dinamismo na página, pois terá cor no fundo da página e uma categoria de borda na exposição da página.

```
<table width="750" height="100%" border="0" align="center">
  <tr>
    <td valign="top">
      <table width="750" border="0">
        <tr>
          <td>
            <table width="100%" border="0">
              <tr>
                <td width="150"></td>
                <td valign="bottom"><table width="100%" border="0">
                  <tr>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#" class="link">CADASTRO
                        ADMINISTRADOR</a> </div></td>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#" class="link">CADASTRO
                        DE OPERADORES</a></div></td>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#" class="link">CADASTRO
                        DE CLIENTES</a> </div></td>
                  </tr>
                  <tr>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#" class="link">CLIENTES EM
                        ABERTO </a></div></td>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#" class="link">CLIENTES
                        FECHADOS</a> </div></td>
                    <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
                      class="box"><div align="center"><a href="#"
                        class="link">COMUNICA&Ccedil;&Atilde;O ON-LINE</a> </div></td>
                  </tr>
                </table></td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

```
</tr>
</table>
```

```
</td>
```

```
</tr>
```

```
<tr>
```

```
<td><table width="750" border="0">
```

```
<tr>
```

```
<td width="152"><table width="100%" border="0">
```

```
<tr>
```

```
<td bgcolor="#FFFFCC" class="box3"><div align="center"><span
class="link">MENU</span></div></td>
```

```
</tr>
```

```
<tr>
```

```
<td height="27" class="box"
onMouseOver="hiliteOver(this);"onMouseOut="hiliteOut(this);"><div
align="left"><a href="listaCentrais.jsp" class="link">CENTRAIS
DE ALARME</a></div></td>
```

```
</tr>
```

```
<tr>
```

```
<td onMouseOver="hiliteOver(this);"
onMouseOut="hiliteOut(this);"class="box"><div align="left"><a href="#"
class="link">PROCEDIMENTOS
OPERACIONAIS </a></div></td>
```

```
</tr>
```

```
<tr>
```

```
<td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
class="box"><div align="left"><a href="#"
class="link">QUILOMETRAGEM</a></div></td>
```

```
</tr>
```

```
<tr>
```

```
<td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
class="box"><div align="left"><a href="#"
class="link">ANOTA&Ccedil;&Otilde;ES
EXTRAS</a> </div></td>
```

```
</tr>
```

```
<tr>
```

```
<td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
class="box"><a href="#" class="link">BACKUPS</a></td>
```

```
</tr>
```

```
<tr>
```

```
<td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
class="box"><div align="left"><a href="logout.jsp"
class="link">LOGOUT
```

```

- <%= login %></a></div></td>
  </tr>
  <tr>
    <td bgcolor="#FFFFCC" class="box3"><div align="center"
      class="link">RELAT&Oacute;RIOS</div></td>
    </tr>
    <tr>
      <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
        class="box"><div align="center" class="link">
        <div align="left"><a href="#" class="link">DIA ATUAL</a></div>
        </div></td>
      </tr>
      <tr>
        <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
          class="box"><div align="left"><a href="#"
            class="link">QUINZENAL</a></div></td>
        </tr>
        <tr>
          <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
            class="box"><div align="center" class="link">
            <div align="left"><a href="#" class="link">M&Ecirc;S
              ATUAL </a></div>
            </div></td>
          </tr>
          <tr>
            <td onMouseOver="hiliteOver(this);" onMouseOut="hiliteOut(this);"
              class="box"><div align="left"><a href="#" class="link">M&Ecirc;S
                PASSADO </a></div></td>
            </tr>
          </table></td>
        <td valign="top">
<%
  CentralDeMonitoramento controle = (CentralDeMonitoramento)
session.getAttribute("controle");
  Vector centrais = controle.lista();

```

```

if ( centrais.size() > 0 ) {
    out.println("<table whidth=\"100%\" border=\"1\">");

    out.println("<tr>");
    out.println("<th>#Ref.");
    out.println("<th>Raz&atilde;o Social");
    out.println("<th>Endere&ccedil;o");
    Iterator it = centrais.iterator();
    while ( it.hasNext() ) {
        out.println("<tr>");
        Iterator linha = ((Vector) it.next()).iterator();
        while ( linha.hasNext() ) {
            out.println("<td>"+linha.next());
        }
    }
    out.println("</table>");
}

%>

        </td>

    </tr>
</table></td>
<td>&nbsp;</td>
</tr>
</table>
</td>
<background = ====="fundo.gif">

</tr>
</table>
</body>
</html>

```

Dentro do código HTML tem linhas com códigos em destaque de azul para com a função de demonstrar numa tabela uma lista de todas as centrais pré cadastradas no banco do programa, com suas respectivas descrições como número de identificação. Razão social e endereço.

3.7 ANÁLISE DE RESULTADOS

As conseqüências dessa atividade nos proporcionará, várias soluções para o aumento da qualidade de monitoramento como: a determinação do início e término dos responsáveis pelo monitoramento do dia, tempo médio de chegada e quilometragem dos carros de patrulha.

Uma melhor análise nos eventos ocorridos e averiguação do local do disparo, para que assim possa ser anotado na ficha do cliente para futuras produções de relatórios.

Uma exposição de informações e/ou procedimentos para os determinados tipos de eventos para as retiradas de erros cometidas por parte dos operadores e supervisores de patrulha.

Avisos de alerta que trará funções como informação dos clientes que estão para fechar ou que ainda não abriram, a primeira informação evitará o esquecimento da ativação do sistema por parte do cliente, a segunda trará uma solução de um possível defeito na comunicação entre a Central de Alarme (no cliente) e a Central de Monitoramento (empresa).

Uma resolução e/ou dúvida dos clientes serão tiradas on-line, os clientes poderão verificar que horas e quem ativou e desativou seu sistema e terá backup's, para melhor garantia na segurança dos dados atuais e passados dos clientes.

CONCLUSÃO

Com esse software teremos uma maior precisão, rapidez no atendimento e monitoramento real pois terá uma maior e melhor disponibilidade de atendimento do cliente, precisão no atendimento, espaço livre para anotações extras e/ou observações, facilidade para backup's, alerta quanto aos clientes que faltam fechar e que ainda não abriram e Análise do sistema em qualquer lugar do mundo.

A utilização desse software trará um excelente monitoramento, para evitar acima de tudo a falta de atenção com os bens dos clientes, que depositam toda a segurança em um dos segmentos de Segurança Eletrônica., conhecido como alarme.

Assim com o uso desses benefícios, irá sem dúvida aumentar o poder de relacionamento com o cliente, o que é uma das principais forças internas e externas que afetam, desafiam e mudam as organizações. Organizações essas que alcançará o sucesso do mercado, com a integração de todas as atividades tecnológicas.

BIBLIOGRAFIA

- BECEIRO, F. P. Tutorial HTML. Disponível em : <[http:// www.abcdicas.com.br](http://www.abcdicas.com.br)> . Acesso em 11abr. 2004.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**, Guia do usuário. Rio de Janeiro: Campus, 2000.
- CURSO HTML, Disponível em: <<http://www.detetivez.hpg.ig.com.br/html/oquehtml.htm>> Acesso em : 11 abr. 2004.
- CRAIG, L. Tradução de Luiz ^a Meireles Salgado., Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000.
- DEITEL, H. M.; DEITEL, D. J. **Java**, como programar. 3ª edição. Porto Alegre: Bookman, 2001.
- FIELDS, D. K. ; KOLB, M. A. Desenvolvendo na web com **JavaServer Pages**. Rio de Janeiro: Ciências Moderna Ltda, 2000.
- FURLAN, José Davi. **Modelagem de Objetos através da UML**: análise e desenho orientados a objeto. São Paulo: Makron Books, 1998.
- HTML, disponível em:
- <<http://www.icmc.usp.br>> Acesso em : 11 abr. 2004.
- <<http://www.imasters.com.br>> Acesso em 11 abr. 2004.
- Kurniawan, Budi. Java para a Web com Servlets, JSP e EJB. Rio de Janeiro: Editora Ciência Moderna Ltda, 2002.
- LARMAN, Craig. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientados a objetos. Trad. Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.
- LOCUS. Orientação a objetos. Disponível em: <<http://locus.com.br/corporativo/tecnologia/objetos.htm>> . Acesso em 25 abr. 2004.
- MATOS, A. V. de, UML: Prático e descomplicado. 2ª Edição. São Paulo Érica, 2004.
- SOUZA, E. C. de, Programação Orientada a Objetos com Java. Santa Catarina: Relativa Editora Ltda, 2002.

SOLIS, S. M. Introdução ao JSP e WML. Disponível em <http://crieseuwebsite.com/tutjsp01.shtml>. Acesso em 09 mai. 2004.

SOUKUP, R. Desenvolvendo o Microsoft SQLServer. Rio de Janeiro: Ciências Moderna Ltda, 1998.

UML – Unified Modeling Language. Linguagem de Modelagem Unificada. Disponível em: <http://www.umlbrasil.com.br>. Acesso em: 31 jul. 2004.

UML – Unified Modeling Language. Ferramentas. Disponível em: <http://www.uml.com.br/ferramentas/index.asp>. Acesso em: 01 agos. 2002.