

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**REALIZAÇÃO DA GERÊNCIA DISTRIBUÍDA DE REDES  
UTILIZANDO SNMP, JAVA, WWW E CORBA**

**André Mello Barotto**

Florianópolis, Abril de 1998.

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**  
**Programa de Pós-Graduação em Ciência da Computação**

**REALIZAÇÃO DA GERÊNCIA DISTRIBUÍDA DE REDES  
UTILIZANDO SNMP, JAVA, WWW E CORBA**

**André Mello Barotto**

**Prof. Dr. Carlos Becker Westphall**  
Orientador

Dissertação submetida à Universidade  
Federal de Santa Catarina para  
obtenção do grau de Mestre em  
Ciência da Computação.

Florianópolis, Abril de 1998.

# REALIZAÇÃO DA GERÊNCIA DISTRIBUÍDA DE REDES UTILIZANDO SNMP, JAVA, WWW E CORBA

André Mello Barotto

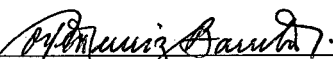
Esta dissertação foi julgada adequada para a obtenção do título de

## MESTRE EM CIÊNCIA DA COMPUTAÇÃO

especialidade SISTEMAS DE COMPUTAÇÃO e aprovada em sua forma final pelo  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO.



Prof. Dr. Carlos Becker Westphall  
Orientador

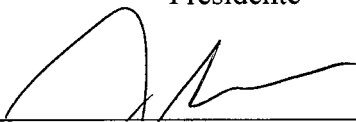


Prof. Dr. Jorge Muniz Barreto  
Coordenador do Curso de Pós-Graduação  
em Ciência da Computação

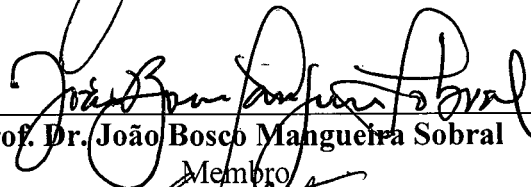
### BANCA EXAMINADORA:



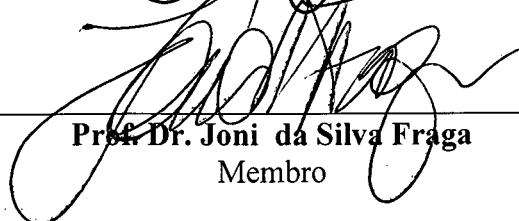
Prof. Dr. Carlos Becker Westphall  
Presidente



Prof. Dr. José Marcos Nogueira  
Membro



Prof. Dr. João Bosco Manguerra Sobral  
Membro



Prof. Dr. Joni da Silva Fraga  
Membro

Florianópolis, Abril de 1998

## **Agradecimentos**

É com grande estima que agradeço a todas as pessoas que direta ou indiretamente auxiliaram neste trabalho.

Agradeço aos meus pais por apoiarem os meus estudos e por apresentarem um exemplo de caráter e honra que eu sempre procurei seguir.

Agradeço a minha futura esposa, Ana Lúcia Kretzer, que me apoiou e sempre esteve ao meu lado nos momentos difíceis.

Agradeço ao Prof. Carlos Becker Westphall, pois além de me orientar e apoiar neste trabalho, foi também um grande amigo.

Agradeço a Jussara e Fernando que nos momentos mais difíceis sempre estiveram firmes ao meu lado, demonstrando o verdadeiro sentido de uma grande amizade. Agradeço também a Katyra, Adriano, Kormann e Socorro, pela amizade e contribuições a este trabalho.

Agradeço aos funcionários do NPD, em especial a Edson Melo e Marcio Cledes, pela amizade e apoio ao desenvolvimento deste trabalho.

E finalmente agradeço a Deus, pois sem Ele, nada disso seria possível.

# Sumário

<b>Lista de Figuras</b> .....	<b>iii</b>
<b>Lista de Quadros</b> .....	<b>iv</b>
<b>Resumo</b> .....	<b>v</b>
<b>Abstract</b> .....	<b>vi</b>
<b>1 Introdução</b> .....	<b>1</b>
1.1 Objetivos .....	2
1.2 Justificativa.....	3
1.3 Organização do Trabalho .....	4
<b>2 Revisão Bibliográfica</b> .....	<b>6</b>
2.1 Tecnologia Web .....	6
2.2 Java.....	7
2.3 CORBA .....	8
2.3.1 ORB.....	9
2.3.2 Arquitetura de um ORB na definição do CORBA 2.0 .....	10
2.3.3 Invocação Estática e Dinâmica de Métodos .....	13
2.3.4 GIOP E IIOP .....	14
2.3.5 A Utilização do CORBA na Gerência de Redes .....	15
2.4 SNMP .....	17
2.4.1 GetRequest .....	20
2.4.2 GetNextRequest .....	20
2.4.3 SetRequest.....	21
2.4.4 Trap .....	22
<b>3 Ferramentas de Gerenciamento Integradas à Tecnologia Web</b> .....	<b>24</b>
3.1 Advent SNMP API.....	24
3.2 Bojangles.....	27
3.3 Ambiente de Desenvolvimento .....	32
<b>4 Ambiente de Gerência de Redes utilizando <i>Browsers Web</i></b> .....	<b>35</b>
4.1 Introdução .....	35
4.2 Métodos para Obtenção de Informações .....	36
4.2.1 Métodos Tradicionais de Obtenção de Informações .....	37
4.2.2 Consultas SNMP através de <i>Applets</i> e <i>Applications</i> Java.....	37
4.3 Acesso a uma Base de Dados Relacional através de Aplicativos Java.....	39
4.4 Independência de Plataforma .....	41
4.5 Um Exemplo de Integração de <i>Applets</i> e <i>Applications</i> Java com um Servidor de Banco de Dados Relacional.....	41
4.6 Geração de Gráficos em uma Interface Web.....	43
4.7 Conclusão.....	43
<b>5 Aplicações Desenvolvidas utilizando o Ambiente de Gerência</b> .....	<b>45</b>
5.1 Software para a Análise de Tráfego em Links TCP/IP através de <i>Browsers Web</i> .....	45
5.1.1 Modelo Lógico .....	47
5.1.2 Estudo dos <i>Softwares</i> Similares ao <i>WebVis</i> .....	48
5.1.3 Problemas encontrados em <i>Softwares</i> Similares ao <i>WebVis</i> .....	51
5.1.4 Objetivos Traçados para a Implementação do <i>WebVis</i> .....	52
5.1.5 Implementação .....	54
5.1.5.1 Módulo de Configuração.....	54
5.1.5.2 Módulo para Obtenção de Informações.....	55
5.1.5.3 Módulo para Visualização de Informações em <i>Browsers Web</i> .....	56
5.1.6 Conclusão .....	58
<b>6 Sistema de Gerência Distribuída</b> .....	<b>60</b>
6.1 Estrutura Lógica.....	61
6.2 Objetos Gerenciados Distribuídos .....	63
6.2.1 Objeto <i>Computer</i> .....	63
6.2.1.1 Interface com o Sistema Operacional.....	64
6.2.1.2 Implementação do Objeto <i>Computer</i> .....	66

6.2.2	Objeto Cluster .....	67
6.2.2.1	Implementação do Objeto <i>Cluster</i> .....	70
6.3	Aplicação de Gerência CORBA .....	71
6.3.1	Implementação de um Protótipo da Aplicação de Gerência.....	72
6.3.2	Módulos que Compõem o Protótipo da Aplicação de Gerência.....	73
6.4	Agente de Contabilização.....	76
6.5	Agente SNMP .....	78
6.5.1	Implementação do Agente SNMP .....	80
<b>7</b>	<b>Conclusão .....</b>	<b>83</b>
7.1	Alcance dos Objetivos.....	85
7.2	Dificuldades Encontradas.....	89
7.3	Perspectivas Futuras .....	90
<b>8</b>	<b>Anexo A: Código Fonte da Classe <i>SNMPMan</i> .....</b>	<b>92</b>
<b>9</b>	<b>Anexo B: Código Fonte da Classe <i>Linha</i> e <i>CadastroLinhas</i> .....</b>	<b>94</b>
<b>10</b>	<b>Anexo C: Código Fonte da Classe <i>ComputerImpl</i> e <i>ClusterImpl</i>.....</b>	<b>104</b>
<b>11</b>	<b>Bibliografia .....</b>	<b>109</b>
<b>12</b>	<b>Glossário.....</b>	<b>112</b>

## Lista de Figuras

Figura 1	Instanciação de Objetos utilizando o CORBA .....	9
Figura 2	Comunicação Transparente entre Objetos através dos ORBs .....	9
Figura 3	Arquitetura do ORB em um Cliente .....	11
Figura 4	Arquitetura do ORB em um Servidor .....	13
Figura 5	Componentes para a Invocação de Métodos Estática .....	14
Figura 6	Interoperabilidade entre ORBs utilizando o GIOP e IIOP .....	15
Figura 7	Comunicação entre Gerentes e Agentes.....	18
Figura 8	Localização do Objeto Gerenciado sysDescr na MIB .....	19
Figura 9	Formato das PDUs <i>GetRequest</i> , <i>GetNextRequest</i> , <i>GetResponse</i> e <i>SetRequest</i> .....	22
Figura 10	Formato da PDU Trap.....	23
Figura 11	Consultas SNMP ao Agente através de <i>Applets</i> e <i>Applications</i> .....	25
Figura 12	Modelo de utilização do <i>Bojangles</i> .....	27
Figura 13	Comunicação entre Agentes e Objetos Gerenciados utilizando o JORB .....	32
Figura 14	<i>Cluster</i> de Estações .....	34
Figura 15	Modelo de Obtenção e Visualização de Dados.....	36
Figura 16	Realização de Consultas SNMP através de <i>Applets</i> .....	38
Figura 17	Comunicação entre um <i>Applet</i> e o Banco de Dados .....	40
Figura 18	Menu Principal do <i>WebVis</i> .....	47
Figura 19	Modelo Lógico Apresentado pelo <i>WebVis</i> .....	47
Figura 20	Gráfico Gerado pelo <i>Router-Stats</i> .....	49
Figura 21	Funcionalidade do <i>Router-Stats</i> .....	51
Figura 22	Obtenção e Armazenamento de Informações do Tráfego .....	56
Figura 23	Gráfico gerado pela Classe <i>Calendario.class</i> .....	57
Figura 24	Gráfico Representando o Tráfego Médio por Hora em um Determinado Dia .....	58
Figura 25	Estrutura Lógica do Sistema de Gerência .....	62
Figura 26	Operação Básica do Objeto <i>Computer</i> .....	64
Figura 27	Instanciação do Objeto <i>Computer</i> .....	67
Figura 28	Objeto <i>Cluster</i> .....	68
Figura 29	Tráfego entre o Gerente e o Objeto Gerenciado .....	69
Figura 30	Janela Principal da Aplicação de Gerência .....	72
Figura 31	Taxa de Ociosidade de um Máquina apresentada pelo Objeto <i>Janela_CPU</i> .....	75
Figura 32	Taxa de Ociosidade de Todas as Máquinas do Ambiente de Teste .....	76
Figura 33	Funcionalidade do Agente de Contabilização .....	78
Figura 34	Funcionalidade do Agente SNMP.....	79

## Lista de Quadros

Quadro 1 Obtenção do Valor da Variável <i>sysDescr</i> (1.3.6.1.2.1.1.1.0) .....	20
Quadro 2 Interface do Objeto .....	29
Quadro 3 Geração de <i>Stubs</i> e <i>Skeletons</i> .....	30
Quadro 4 Aplicação Cliente .....	30
Quadro 5 Inicialização do JORB .....	31
Quadro 6 Registrando uma Implementação .....	31
Quadro 7 Conexão com o Banco de Dados através da Classe <i>Banco.java</i> .....	39
Quadro 8 Execução de um Comando do Sistema a partir de uma Aplicação Java .....	64
Quadro 9 Execução de um Comando do Sistema a partir de um Programa implementado em C .....	65
Quadro 10 Interface do Objeto <i>Computer</i> .....	66
Quadro 11 Interface do Objeto <i>Cluster</i> .....	70
Quadro 12 Arquivo HTML que invoca a Aplicação de Gerência .....	74
Quadro 13 Invocação do Método <i>getClusterHostnames</i> na classe <i>ID_Janela</i> .....	74
Quadro 14 Criação das Instâncias da Classe <i>SnmAPI</i> e <i>SnmSession</i> .....	80
Quadro 15 Obtenção do Valor de uma Variável SNMP no Método <i>getValue</i> .....	81
Quadro 16 Invocação dos Métodos <i>consultaObjetoCluster</i> e <i>getCluster</i> .....	82



## Resumo

A linguagem Java, o WWW (*World Wide Web*) e o CORBA (*Common Object Request Broker Architecture*) são tecnologias complementares que, quando utilizadas em conjunto, oferecem um poderoso mecanismo para o desenvolvimento de aplicações distribuídas. Tais tecnologias podem ser utilizadas com grande sucesso na construção de aplicações distribuídas de gerência sem, no entanto, sobrepor padrões de gerenciamento existentes como o SNMP (*Simple Network Management Protocol*) ou o CMIP (*Common Management Information Protocol*). Este trabalho descreve a utilização dessas tecnologias para a implementação de um sistema de gerência distribuída, que é independente de plataforma, suportando os protocolos SNMP e IIOP (*Internet Inter-Orb Protocol*), permitindo a integração com a maioria dos servidores de banco de dados disponíveis comercialmente e utilizando o WWW para realizar a interface com o usuário final. Para este sistema foi desenvolvido um conjunto de objetos gerenciados distribuídos e agentes multiplataforma, que permitem a implantação de novas funções de gerência, associadas a um *cluster* de estações de trabalho.

## **Abstract**

Java, WWW (World Wide Web) and CORBA CMIP (Common Management Information Protocol) are complementary technologies that, once joined, may offer a powerful mechanism for the development of distributed management applications without necessarily overlapping some already existent standards such as SNMP (Simple Network Management Protocol) and CMIP (Common Management Information Protocol). Within this context, this work describes the usage of these technologies for the implementation of a distributed management system, which is platform independent and provides support for the SNMP and IIOP (Internet Inter-Orb Protocol) protocols. This system integrates with the majority of the data base servers available at the market, and makes use of WWW tools to perform the interface with the user. In addition, a set of managed objects as well as multi-platform agents were specially developed for the system in order to allow the construction of new management functions for the control of a cluster of workstations.

# 1 Introdução

Com a popularização da Internet é cada vez mais comum a utilização de aplicações sobre TCP/IP pela grande maioria dos usuários de redes. Dentre estas aplicações, uma das que mais se destaca é o *browser Web*, que permite ao usuário conectar-se a servidores *Web*, carregar páginas para a visualização de informações e realizar ações variadas como: transações comerciais, localizar novas informações, entre outras.

Com o advento do Java, que permite a construção de programas que podem ser transferidos e executados em *browsers*, as páginas *Web* ganharam uma dinâmica muito maior e o *browser* tornou-se o veículo ideal para a execução de *applets* Java.

A tecnologia *Web* está causando uma profunda mudança ao acesso, à obtenção e ao tratamento de informações na Internet. Uma das possibilidades de utilização desta tecnologia está no acesso e tratamento de informações de gerenciamento [1][2][8][10][12][16].

O modelo de gerência tradicional da Internet baseia-se na comunicação entre gerentes e agentes, em que uma aplicação-gerente, restrita a um número limitado de consoles, consulta e recebe informações dos agentes. No entanto, esta solução é muito restrita, pois praticamente impede que aplicações de gerenciamento possam ser distribuídas em diversas máquinas da rede, de modo automático e independente da sua localização geográfica.

A tecnologia *Web* é capaz de solucionar as principais limitações das aplicações de gerência tradicionais da Internet, permitindo que estas possam ter um mecanismo de instalação automática, através de servidores HTTP, e serem executadas em praticamente qualquer máquina da rede, independente da plataforma ou de sua localização geográfica.

Uma ferramenta que pode ser utilizada com muito sucesso em conjunto com a tecnologia *Web* é o padrão CORBA [1][6][7][15][20][23]. Este padrão permite que uma aplicação-cliente possa instanciar objetos em uma máquina servidora, independente do *hardware* que esta máquina possua ou da linguagem de programação utilizada na implementação dos objetos [20][21][22]. Em consequência deste fato, este padrão pode ser uma ferramenta muito útil para o gerenciamento de redes, permitindo que objetos gerenciados possam ser instanciados de maneira natural e independente do *hardware* utilizado [6][7][15][23]. O CORBA não sobrepõe padrões como o CMIP ou o SNMP [15], permitindo a integração com estes padrões e agindo como uma ferramenta adicional no gerenciamento de redes.

Na Universidade Federal de Santa Catarina (UFSC) existe um conjunto de estações (*cluster*) que estão sendo interligadas por uma estrutura de rede ATM de alta velocidade (155Mb/s) e destinam-se à execução de aplicações na área da computação de alto desempenho. Entre estas aplicações, destacam-se aquelas destinadas ao processamento paralelo e distribuído. Com a finalidade de suprir algumas necessidades deste ambiente de computação científica, este trabalho de dissertação descreve as tecnologias, ferramentas e etapas necessárias à implementação de um sistema distribuído de gerência que permite a um usuário obter informações globais ou parciais deste ambiente, utilizando um *browser Web*. Na implementação deste projeto são utilizados os padrões CORBA e SNMP para gerenciar o ambiente em questão. Deste modo, pretende-se utilizar os recursos oferecidos pela arquitetura CORBA para a instanciação de objetos gerenciados e apresentar, de modo prático, a eficácia de sua utilização no gerenciamento de redes de computadores.

## 1.1 Objetivos

O objetivo geral deste trabalho é implementar um sistema distribuído destinado à gerência de um *cluster* de estações. Este sistema deve ser independente de plataforma, suportar os protocolos SNMP e IIOP/CORBA, permitir a integração com a maioria dos servidores de banco de dados relacionais do mercado e utilizar o WWW para realizar a interface com o usuário final. Somado a isto, o sistema de gerência deve possuir um conjunto de objetos gerenciados distribuídos e agentes multiplataforma que permitam a implantação de novas funções de gerência, associadas a um *cluster* de estações de trabalho.

Para alcançar esta meta, foram definidos os seguintes objetivos específicos:

1. estudo dos principais padrões e ferramentas relacionados com a tecnologia *Web*;
2. estudo do SNMP e CORBA, direcionando o enfoque para o uso destes padrões, associados à tecnologia *Web*, para a gerência de redes de computadores;
3. avaliação das principais ferramentas e protocolos envolvidos no gerenciamento de redes que utilizam a tecnologia *Web*. Nesta fase, deve-se enfatizar os utilitários que permitem uma plena integração com a linguagem Java;
4. criação de um ambiente de gerência de redes, através do desenvolvimento e utilização de um conjunto de ferramentas independentes de plataforma, que permitem a obtenção de informações de gerência, utilizando o protocolo SNMP; o armazenamento destas

informações em uma base de dados relacional e a utilização de uma interface *Web* com o usuário final;

6. criação de uma aplicação modelo para a monitoração do tráfego em *links* TCP/IP, visando demonstrar uma utilização prática do ambiente de gerência citado anteriormente;
7. criação de um sistema de objetos distribuídos e agentes multiplataforma, visando a definição de novas funções de gerência para um *cluster* de estações, através da utilização dos protocolos SNMP e IIOP/CORBA;
8. implementação de uma aplicação modelo, utilizando o ambiente de gerência, os agentes e os objetos gerenciados definidos anteriormente. Esta implementação visa o gerenciamento de um *cluster* destinado ao processamento paralelo e distribuído na UFSC;
9. avaliação dos resultados obtidos e análise das perspectivas futuras para a utilização da tecnologia *Web* e do padrão CORBA aplicados à área de gerência de redes.

## 1.2 Justificativa

A linguagem Java, o WWW (*World Wide Web*) e o CORBA (*Common Object Request Broker Architecture*) são tecnologias complementares que, quando utilizadas em conjunto, oferecem um poderoso mecanismo para o desenvolvimento de aplicações distribuídas [15]. Tais tecnologias podem ser utilizadas com grande sucesso para a construção de aplicações distribuídas de gerência sem, no entanto, sobrepor padrões de gerenciamento existentes como o SNMP (*Simple Network Management Protocol*) ou o CMIP (*Common Management Information Protocol*)[15][17][23].

A tecnologia *Web*, quando aplicada à área de gerência de redes, permite que um administrador possa monitorar e configurar equipamentos conectados em sua rede, a partir de, praticamente, qualquer computador pertencente à Internet [1][16][17][18]. Cada vez mais, surgem equipamentos e *softwares* capazes de serem monitorados e configurados a partir de *browsers Web*, como: *switches*, roteadores, servidores *Web*, entre outros [2][8]. Estas características tornam o *browser* uma preciosa ferramenta para o gerenciamento de redes através da Internet [8][16].

Uma tecnologia que desponta como uma solução muito boa para a criação de objetos distribuídos é o CORBA [20][21]. Este padrão permite que uma máquina cliente possa instanciar objetos em uma máquina servidora, independente do *hardware* ou da linguagem de programação utilizada na implementação dos objetos. Em consequência deste fato, o CORBA pode ser uma poderosa ferramenta para o gerenciamento de redes, permitindo que

objetos gerenciados possam ser instanciados de maneira natural e independente do *hardware* utilizado.

Dentro deste contexto, pretende-se implantar estes novos conceitos no gerenciamento da redeUFSC<sup>1</sup>, permitindo que em um futuro próximo a maior parte de seus recursos computacionais possam ser gerenciados através de *browsers Web*.

### 1.3 Organização do Trabalho

Este trabalho possui a seguinte organização estrutural:

- na primeira etapa é apresentada uma revisão bibliográfica necessária à plena compreensão do restante do trabalho. Os tópicos mencionados nesta etapa são frutos de revisão bibliográfica e englobam o estudo do CORBA, SNMP, Java e da tecnologia *Web*. Estes assuntos são descritos no capítulo 2;
- no capítulo 3 são descritas as principais ferramentas para o gerenciamento de redes integradas com o Java e a tecnologia *Web*. Neste capítulo é apresentada também a estrutura de *software* e *hardware* utilizada na implementação deste projeto Esta etapa foi essencial para a escolha das ferramentas utilizadas na criação de um ambiente de gerência distribuído, que é descrito nos capítulos posteriores;
- no capítulo 4 é apresentada a descrição de um ambiente de gerência que utiliza a tecnologia *Web*. Nesta etapa, é descrito um conjunto de ferramentas implementadas e utilizadas para o desenvolvimento de aplicações de gerência em Java, que possibilitam a obtenção de informações, utilizando o protocolo SNMP e o armazenamento destas informações em uma base de dados relacional;
- no quinto capítulo é apresentada uma aplicação para o gerenciamento de *links* TCP/IP, utilizando o ambiente de gerência definido anteriormente. Esta ferramenta é capaz de obter informações de tráfego dos roteadores, utilizando o protocolo SNMP; armazenar estas informações em uma base de dados relacional; e apresentar as estatística do tráfego

---

<sup>1</sup> Rede baseada no protocolo TCP/IP que interliga os computadores da UFSC

através da apresentação de gráficos, em uma interface Web. Esta aplicação visa demonstrar, de modo prático, os benefícios e eventuais limitações do ambiente de gerência definido no capítulo anterior;

- no capítulo 6 é descrita a utilização do Java, WWW, CORBA e SNMP para a implementação de um sistema de gerência distribuída aplicado a um *cluster* de estações. Neste sistema foram implementados objetos gerenciados distribuídos que podem ser instanciados utilizando o CORBA. Para a integração entre o SNMP e o IIOP/CORBA, foi implementado um agente que realiza a comunicação entre estes protocolos. É apresentada também, a implementação de um agente que permite a contabilização de informações de gerência em uma base de dados de relacional. Descreve-se ainda o desenvolvimento de um *applet* destinado ao gerenciamento do *cluster*;
- no capítulo 7 são descritos os resultados obtidos e as dificuldades encontradas com a utilização do Java, WWW, CORBA e SNMP para o desenvolvimento de aplicações de gerência de redes distribuídas. Também são apresentadas as propostas futuras necessárias à continuidade deste projeto.

## 2 Revisão Bibliográfica

### 2.1 Tecnologia Web

A tecnologia *Web*, no contexto deste trabalho, é uma definição que abrange todas as ferramentas e protocolos envolvidos na comunicação de um *browser Web*, com um servidor HTTP (*Hypertext Transfer Protocol*).

Um conceito primordial na tecnologia *Web* é o HTML (*HyperText Mark-up Language*). O HTML é uma linguagem padronizada que permite a criação das páginas *Web* através da definição de textos, figuras, *tags*, *frames*, entre outros recursos.

As páginas *Web* podem ser armazenadas em uma máquina que esteja executando um servidor HTTP. Um usuário, utilizando um *browser Web* (como o *Netscape* ou *Internet Explorer*), pode iniciar uma conexão com o servidor e, através do protocolo HTTP (*Hypertext Transfer Protocol*), realizar a requisição de páginas ou componentes destas ao servidor HTTP. O servidor HTTP tem a função de receber as requisições proveniente de clientes e, se possível, retornar as informações requisitadas, utilizando o protocolo HTTP.

O HTTP é um protocolo orientado a conexão, que utiliza o TCP como mecanismo de transporte, e tem como função básica requisitar informações ao servidor HTTP e repassar as respostas obtidas ao solicitante do serviço.

Tipicamente, um cliente requisita uma determinada página para um servidor HTTP e recebe a resposta solicitada. O *browser* percorre a página procurando arquivos que devem ser transferidos, a partir de um servidor HTTP. Caso encontre algum arquivo que deva ser transferido, como figuras e *applets*, efetua a conexão com o servidor e requisita a transferência deste arquivo. O *browser* repete os dois últimos passos até que todos os arquivos que compõem a página tenham sido transferidos.

Outro conceito muito utilizado na tecnologia *Web* é o CGI (*Common Gateway Interface*), que consiste em um padrão que permite a execução de programas ou *scripts* em um servidor. O CGI permite a criação de programas capazes de realizar consultas a uma base de dados relacional, realizar transações comerciais, entre outras tarefas. No entanto, o CGI apresenta um comportamento estático, no sentido de que o usuário preenche os parâmetros desejados (como o número de uma conta bancária para um CGI destinado à consulta de saldo) e obtém os resultados. Para realização de tarefas com um comportamento dinâmico, ou seja,



que possam mudar os seus estados de acordo com as ações dos usuários ou de outros eventos, é necessário utilizar a linguagem Java.

## 2.2 Java

O Java é uma linguagem de programação com características muito similares ao C++ cuja utilização avança exponencialmente. O Java apresenta características bem peculiares que justificam a sua popularidade e viabilizaram a implementação deste projeto [4].

Ao ser compilado um programa em Java, é gerado um código para uma máquina genérica denominado *byte code*. Este código pode ser executado tanto por interpretadores, como por *Browsers Web* (que simulam a implementação de uma máquina virtual baseada em *byte code*). Esta característica confere a programas construídos em Java a peculiaridade de serem independentes de plataforma.

Utilizando o Java é possível a criação de dois tipos de programas: *applets* e *applications*. *Applets* são aplicações desenvolvidas em Java, destinadas a serem executadas em *browsers Web*. Neste caso, a classe principal da aplicação deve ser derivada da classe *Applet*. *Application* é uma aplicação Java destinada a ser executada a partir de um interpretador. A execução de um *application* Java inicia na função *main* da classe principal. Outra diferença entre *applets* e *applications* está no fato que *applets* (executados em *browsers Web*) possuem algumas limitações impostas pela segurança, como a impossibilidade de abertura de *sockets* para outras máquinas além da servidora *Web*.

Programas em Java são plenamente integráveis com o sistema de *Interface Web* e oferecem um nível de interatividade e robustez muito maiores que uma interface baseada unicamente em HTML. Para exemplificar estas características, pode-se citar o caso de um usuário que necessite enviar informações de um formulário a um servidor, utilizando um *browser Web*. Para realizar esta operação sem a utilização do Java, o usuário teria que digitar os dados, submetê-los para um servidor utilizando um CGI (*Common Gateway Interface*) e na ocorrência de algum erro, seria retornada uma mensagem indicativa. Neste caso, o usuário teria que corrigir os dados e realizar a submissão novamente. Utilizando o Java, os dados podem ser analisados antes de serem enviados para o servidor. Isto torna a *interface* mais amigável e diminui a carga de processamento relativa ao servidor.

O Java, por ser uma linguagem baseada em objetos, permite a construção de programas legíveis e de fácil manutenção [2]. Estas qualidades, associadas à característica

dinâmica de sua *interface*, permitem que seus programas sejam muito mais fáceis de serem modificados do que um escrito em linguagem baseada em *script*.

### 2.3 CORBA

O CORBA (*Common Object Request Broker Architecture*) consiste na especificação, definida pelo OMG (*Object Management Group*), para uma arquitetura orientada a objetos padronizada. O OMG consiste em um grupo de mais de 700 companhias destinado a promover o desenvolvimento prático e teórico da tecnologia de objetos, para sistemas de computação distribuída. O OMG foi criado em 1989 e em 1990 lançou a primeira versão do CORBA. Em 1995 foi finalizada a versão 2.0, que será utilizada neste trabalho [21].

O CORBA permite a comunicação entre objetos localizados em qualquer máquina de uma rede, podendo residir em diferentes sistemas operacionais e serem implementados através de diferentes linguagens de programação (Figura 1). O segredo do sucesso de tal padrão está no fato de especificar a interface<sup>2</sup> que deve ser oferecida pelo objeto, e não o seu código [20][21][22].

Para realizar a comunicação de um objeto servidor com um objeto cliente utilizando o CORBA, deve-se separar a interface da implementação do objeto. Para a definição da interface, o OMG definiu uma linguagem neutra, denominada IDL (*Interface Definition Language*). Utilizando-se a IDL, pode-se definir a interface do objeto servidor, que é uma das únicas informações necessárias para o cliente instanciar e acessar um objeto servidor. Deve-se ressaltar, que o objeto cliente pode instanciar e acessar um objeto servidor, sem um conhecimento prévio de qualquer detalhe sobre a sua implementação.

---

<sup>2</sup> Dentro do contexto deste trabalho, a interface de um objeto consiste na descrição dos métodos e atributos deste objeto.

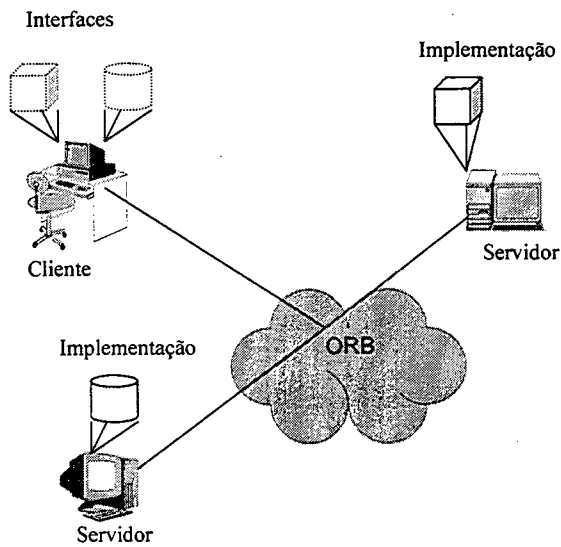


Figura 1 Instanciação de Objetos utilizando o CORBA

### 2.3.1 ORB

O ORB (*Object Request Broker*) é responsável pelo encaminhamento de mensagens entre objetos. Ele permite que um objeto cliente possa invocar, de modo transparente, um método em um objeto servidor, que pode estar na mesma máquina ou em outra máquina qualquer da rede. O ORB intercepta as invocações de métodos e é responsável por encontrar o objeto que implementa o método invocado, passar os parâmetros fornecidos, executar o método e retornar os resultados para o objeto cliente. O cliente não necessita conhecer onde está localizado o objeto servidor, em qual linguagem de programação ele foi implementado ou em que sistema operacional ele reside; necessita apenas, saber qual a interface do objeto (quais são os seus métodos e atributos públicos).

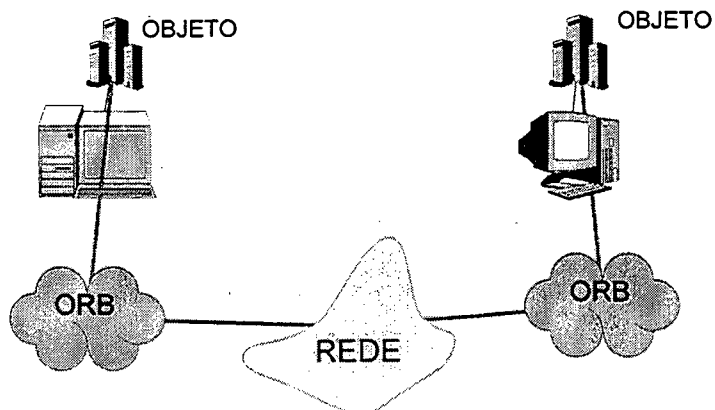


Figura 2 Comunicação Transparente entre Objetos através dos ORBs

### 2.3.2 Arquitetura de um ORB na definição do CORBA 2.0

Um aspecto que deve ser observado, é que utilizando o CORBA, um objeto pode se comportar, ora como servidor, ora como cliente. Quando um objeto invoca um método de outro objeto, ele se comporta como cliente. Quando o contrário ocorre, ele se comporta como servidor.

A invocação de métodos, utilizando as definições do CORBA 2.0, podem ser estáticas ou dinâmicas.

Uma invocação estática é utilizada quando já se conhece a estrutura que o objeto deve oferecer. Neste caso, pode-se definir a interface do objeto utilizando a linguagem IDL e acessar os métodos dos objetos, através de uma estrutura estática e previamente conhecida. Este projeto de dissertação utiliza basicamente invocações estáticas.

As invocações dinâmicas são usadas em serviços em que não se conhece as operações que um objeto pode realizar. Neste caso, o CORBA oferece uma API para que as aplicações clientes possam, em tempo de execução, descobrir qual a estrutura dos objetos de um determinado servidor e invocar os seus métodos. Este é um instrumento muito útil para as ferramentas que necessitam descobrir dinamicamente os serviços oferecidos por um servidor.

A arquitetura de um ORB pode ser dividida em elementos que são necessários para a constituição de um cliente ou de um servidor. Tanto na parte cliente, como na parte servidora, existem elementos para viabilizar a invocação de métodos de maneira estática e dinâmica.

Os elementos utilizados pelo cliente em um ORB são (Figura 3):

- **Stubs:** o *Stub* possui uma interface estática para os objetos, no lado cliente de um ORB. Os *Stubs* definem como os clientes podem acessar os métodos de um objeto em um servidor. Na visão do cliente, o *Stub* age como um *proxy* para objeto servidor, que intercepta invocações de métodos locais e repassa para o objeto servidor. O *stub* é gerado automaticamente a partir da compilação de um arquivo IDL, que define a interface de um objeto. Existe um *Stub* para cada interface de objeto e, normalmente, um *Stub* consiste em um arquivo fonte para uma determinada linguagem de programação (C++, C, Java, entre outras), gerado a partir de um arquivo IDL e que deve ser compilado para a criação do código binário do *Stub* em questão;

- **DII (Dynamic Invocation Interface):** permite descobrir a estrutura de um determinado método e invocá-lo, em tempo de execução. Neste caso, o CORBA define uma API padronizada para: pesquisar a estrutura de objetos em um servidor; obter os parâmetros de um determinado método; permitir a sua invocação e obtenção de resultados em tempo de execução. A DII é utilizada na constituição de uma interface dinâmica no cliente;
- **API do Repositório de Interfaces:** consiste em uma API que permite a um objeto obter e modificar a descrição de todas as interfaces de componentes registrados no repositório de interfaces. O repositório de interfaces é um banco de dados que contém as interfaces dos objetos definidas em IDL;
- **Interface do ORB:** consiste em um conjunto de rotinas para os serviços locais ao cliente, que podem ser de interesse de uma aplicação. Pode-se citar como exemplo, uma rotina para transformar um identificador de um objeto em um *string*.

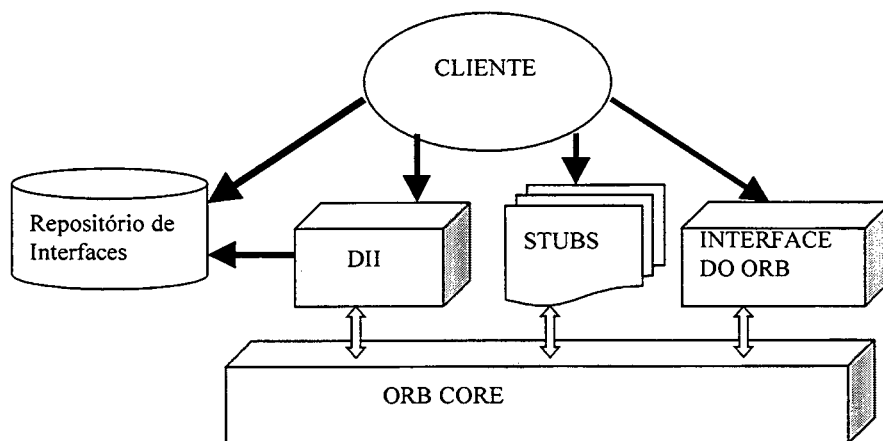


Figura 3 Arquitetura do ORB em um Cliente

Os elementos utilizados pelo servidor em um ORB são (Figura 4):

- **Skeletons:** consiste em uma interface estática, para cada objeto exportado pelo servidor. *Skeletons* são gerados a partir da compilação do arquivo de interfaces em IDL e realizam o mapeamento de um método definido na interface para a sua efetiva implementação. Um *Skeleton* tem um papel semelhante aos *Stubs* em um cliente, permitindo um mapeamento estático, específico para um determinado objeto e ORB. Ao receber a invocação de um método proveniente do cliente (através do *Object Adapter*), o *Skeleton* possui o código necessário para localizar e invocar implementação deste método;
- **DSI (Dynamic Skeleton Interface):** consiste em uma interface que permite a invocação

dinâmica de métodos em um servidor. Ao contrário do *Skeleton* que é baseado em uma definição estática para cada interface de objetos, a DSI permite em tempo de execução receber a invocação de um método (provenientes tanto de *Stubs* como da DII) e identificar o objeto, o método e os parâmetros necessários para acessar a implementação do objeto em questão;

- **Object Adapter:** realiza a interface primária com a implementação no servidor. É o *Object Adapter* que realiza a comunicação direta com os *Skeletons* e a DSI, repassando as mensagens de invocação de métodos, para estes. O *object adapter* é responsável, basicamente, por registrar a implementação das classes em um repositório de implementação; instanciar objetos em tempo de execução; gerar e gerenciar referências para os objetos instanciados; receber invocações de métodos de clientes, repassando-as para os *Skeletons* e para DSI. Existem diversos tipos de objetos e, em consequência disto, o OMG prevê a utilização de diversos *Object Adapters* por ORB, que podem ser otimizados para cada implementação ou ORB. No entanto, o OMG define como um requerimento a ser implementado, que todo ORB possua um *object adapter* básico denominado BOA (*Basic Object Adapter*). O BOA pode ser utilizado para a maioria dos objetos e permite instanciar objetos, repassar requisições para estes e associar referências únicas para estes objetos;
- **Repositório de Implementação:** é responsável por armazenar em um servidor a implementação das classes, as instâncias destas classes e as referências aos objetos instanciados. O repositório de implementação pode armazenar informações adicionais necessárias ao correto funcionamento do ORB;
- **Interface do ORB:** consiste em conjunto de rotinas para os serviços locais ao servidor que podem ser de interesse de uma aplicação. Os serviços oferecidos por esta interface são idênticos àqueles oferecidos em um cliente.

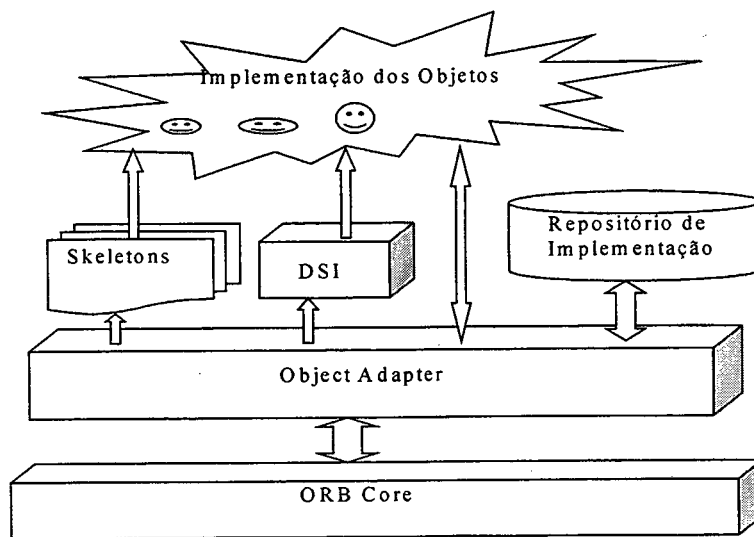


Figura 4 Arquitetura do ORB em um Servidor

### 2.3.3 Invocação Estática e Dinâmica de Métodos

Utilizando o CORBA, pode-se realizar invocações estáticas ou dinâmicas de métodos. As invocações estáticas podem ser utilizadas quando a estrutura do objeto é conhecida e apresenta um comportamento estático, ou seja, não está sujeita a adição, remoção ou modificação de métodos. Com isto, pode-se criar uma definição da interface do objeto através da linguagem IDL e, utilizando um compilador para tal linguagem, gerar *Stubs*, *Skeletons*, arquivos para o Repositório de Interfaces e um modelo de arquivo de implementação, específicos para a linguagem de programação (como C++, Java, Cobol, etc) e ORB utilizados (Figura 5). Pode-se preencher o arquivo de implementação e então compilar os arquivos fontes dos *Skeletons*, dos *Stubs* e da implementação. A seguir, deve-se registrar a implementação no repositório de implementação e, finalmente, instanciar e realizar consultas ao objeto implementado a partir de um cliente, utilizando-se do *Stub* gerado para realizar a comunicação com o ORB.

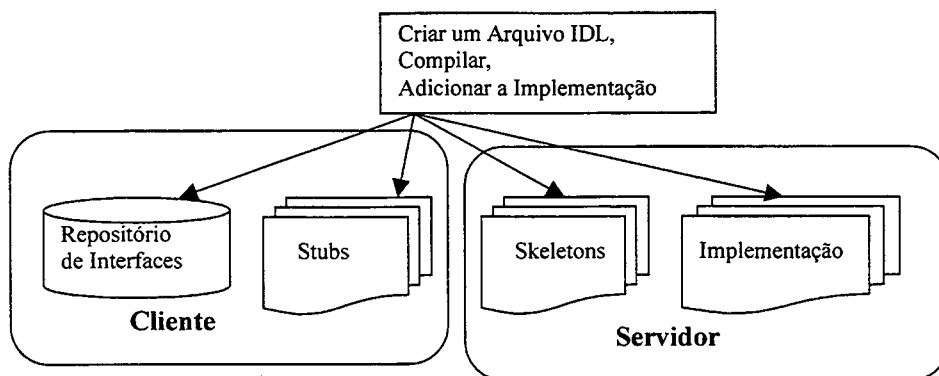


Figura 5 Componentes para a Invocação de Métodos Estática

Quando não existe um conhecimento prévio da estrutura de um objeto, deve-se utilizar a invocação dinâmica de métodos. Utilizando-se este tipo de invocação, o cliente deve consultar um repositório de interfaces e descobrir em tempo de execução a estrutura do objeto. De posse da estrutura do objeto, o cliente pode verificar os serviços que este oferece, montar uma lista de argumentos e realizar a invocação de um método desejado.

Deve-se ressaltar que a invocação estática de métodos é mais eficiente, simples e produz programas mais legíveis que a invocação dinâmica. Por isto, sempre que for possível, deve-se optar pela invocação estática [21].

Um aspecto que deve ser analisado é que para um servidor não é possível diferenciar mensagens provenientes de invocações estáticas ou dinâmicas, pois estas apresentam o mesmo formato. Em consequência deste fato, mensagens provenientes de invocações dinâmicas ou estáticas são tratadas do mesmo modo pelo servidor.

### 2.3.4 GIOP E IIOP

O GIOP (*General Inter-ORB Protocol*) e o IIOP (*Internet Inter-ORB Protocol*) são protocolos requeridos na especificação do CORBA 2.0 e que permitem a interoperabilidade entre ORBs.

O GIOP especifica um conjunto de formatos de mensagens e uma representação de dados única entre ORBs. O GIOP foi projetado para operar sobre um protocolo da camada de transporte orientado a conexão, como o TCP (*Transmission Control Protocol*) ou IPX (*Internet Packet Exchange*). O GIOP apresenta apenas sete formatos de mensagens e não



realiza nenhum tipo de negociação de formatos. Os tipos de dados mapeados a partir da linguagem IDL são transmitidos utilizando o mapeamento *CDR* (*Common Data Representation*); o que permite que estes tipos de dados possam ser interoperáveis e tenham uma representação independente do *hardware*, ou do sistema das máquinas envolvidas.

O IIOP é o protocolo que define como as mensagens GIOP são transmitidas sobre o protocolo TCP/IP (Figura 6). O IIOP permite que a Internet seja utilizada como um poderoso meio de comunicação entre ORBs, possibilitando a comunicação entre ORBs que podem estar localizados em diferentes partes do planeta.

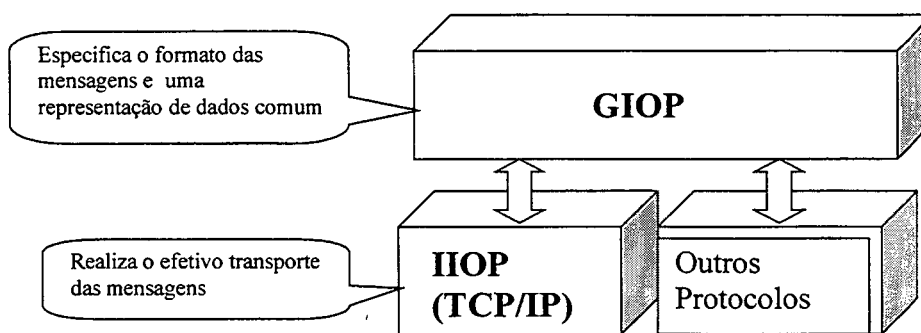


Figura 6 Interoperabilidade entre ORBs utilizando o GIOP e IIOP

### 2.3.5 A Utilização do CORBA na Gerência de Redes

O CORBA é um padrão cada vez mais utilizado na área de gerência de redes [15]. Para entender a razão desta utilização é necessário, primeiramente, verificar algumas características e recursos oferecidos por este padrão.

A filosofia básica do CORBA está na separação entre a interface e a implementação do objeto. Isto possibilita que uma máquina cliente possa instanciar objetos, cuja implementação reside em outra máquina da rede. Esta característica pode ser de grande importância para o desenvolvimento de aplicações para a gerência de redes.

Aplicações de gerência de redes geralmente são complexas e exigem uma grande capacidade de memória e CPU das máquinas em que estão instaladas. Estas aplicações também devem possuir um nível de segurança razoável, por acessarem recursos vitais ao pleno funcionamento da rede.

Uma solução emergente para a gerência de redes distribuída é a utilização da tecnologia *Web*, que permite a transferência automática de uma aplicação gerente (implementada como um *applet* Java) para um *browser Web* e possibilita que tal aplicação possa ser executada em qualquer plataforma, utilizando uma interface conhecida e amigável para o usuário.

Entretanto, sem a utilização de uma ferramenta como o CORBA, a criação de aplicações de gerência mais complexas, utilizando a tecnologia *Web*, fica limitada. Os principais fatores que podem limitar a utilização desta tecnologia são:

- uma aplicação de gerência complexa via *Web*, exige a utilização de um número muito grande de classes Java, algumas com grande quantidade de código. Com isto, a transferência de uma aplicação de gerência de um servidor *Web* para o *browser*, pode ser uma operação extremamente demorada;
- os equipamentos utilizados para o acesso a páginas *Web*, normalmente não possuem um bom desempenho, quando comparados a um servidor de processamento dedicado a função de gerência. Some-se a isso o fato de que um *applet* Java possui um desempenho inferior a uma aplicação nativa de uma máquina e pode-se perceber os fatores que tornam lento o processamento de aplicações de gerência em *browsers Web*, podendo até mesmo inviabilizar esta tarefa;
- se uma aplicação gerente for totalmente transferida para o *browser Web*, o cliente terá acesso a toda implementação desta aplicação. Esta implementação pode conter informações como: senhas de acesso ao banco de dados, mecanismos de criptografia e outros detalhes sigilosos. Uma vez de posse dos arquivos com a implementação completa da aplicação de gerência (definidos em *byte code*), um usuário pode utilizar programas para gerar os fontes, em java, correspondentes aos arquivos em *byte code*<sup>3</sup>.

Com a utilização do CORBA, a interface do objeto é separada de sua implementação propriamente dita. O cliente, possuindo somente a definição da interface, pode instanciar um objeto cuja a implementação pode estar sendo executada em qualquer máquina na rede. Desta forma, a implementação de uma aplicação de gerência pode ser executada em um servidor dotado de um grande poder de processamento e, se necessário, instanciar objetos em outros servidores, distribuindo a carga de processamento entre diversas máquinas da rede. Somado a isto, a implementação do objeto poder ser realizada em praticamente qualquer linguagem de programação (sempre que necessário, pode-se utilizar uma linguagem que permita um bom

desempenho computacional, como C ou C++). A aplicação de gerência, construída como um *applet* Java, só necessita obter a definição da interface dos objetos para instanciá-los. Em outras palavras, no *browser* é executado apenas o código necessário para a realização da interface com o usuário final. A implementação da aplicação de gerência pode ser executada em um, ou vários servidores distribuídos na rede. Estas características permitem os seguintes benefícios:

- diminuição do número e tamanho das classes a serem transferidas para o *browser Web*, permitindo um acesso mais rápido às aplicações de gerência;
- a implementação é executada em um ou vários servidores da rede, sendo que o cliente executa apenas a interface com o usuário, permitindo que aplicações de gerência possam ser executadas com uma performance adequada nos *Browsers Web*;
- não é transferido qualquer detalhe de implementação para cliente, permitindo aumentar o nível de segurança da aplicação;
- como a implementação está sendo executada no servidor e, não no *browser*, ela não fica sujeita às limitações de segurança impostas aos *applets*, para abertura de *sockets* e arquivos. Isto permite que o modelo tenha a mesma flexibilidade de uma aplicação tradicional de gerência.

O CORBA permite, através do protocolo IIOP, o instanciamento e invocação de métodos de objetos, residindo em diferentes máquinas. Este fato, aplicado à área de gerenciamento de redes, permite o instanciamento de objetos gerenciados e a troca de mensagens com estes, de maneira bastante simples e natural. Através do CORBA, pode-se, por exemplo, instanciar um objeto *Computer* em uma máquina qualquer da rede e, através da invocação de métodos deste objeto, obter e modificar informações de gerência desta máquina.

## 2.4 SNMP

O SNMP (*Simple Network Management Protocol*) é um protocolo largamente utilizado para o gerenciamento de redes na Internet. Como o próprio nome já indica, o SNMP é extremamente simples (*"Simple" Network Management Protocol*), e esta talvez seja a característica que mais influenciou na sua popularidade. Outras características muito importantes, oferecidas pelo SNMP, são a extensibilidade, que permite aos fornecedores de

---

<sup>3</sup> Um exemplo de programa para decodificação de arquivos em *byte code* é o *NMI's Java Code Viewer*. Este *software* pode ser encontrado no endereço <http://huizen.dds.nl/~nmi/codeview.html>.

equipamentos de rede adicionarem novas funções de gerência aos seus produtos, e a independência do *hardware* utilizado.

No modelo de gerenciamento SNMP existem basicamente agentes e gerentes (Figura 7). Os agentes são processos relativamente simples cuja função é receber solicitações do gerente, realizar o processamento desejado e enviar as respostas obtidas. O processo agente pode também enviar notificações de um evento qualquer, como a mudança de *status* na interface de um roteador, através da emissão de *traps*. O processo agente pode residir em qualquer equipamento de uma rede, como: estações, hubs, *switchs*, roteadores, entre outros. A aplicação gerente basicamente é capaz de enviar as solicitações para os agentes, obter as respostas e realizar o tratamento de *traps*.

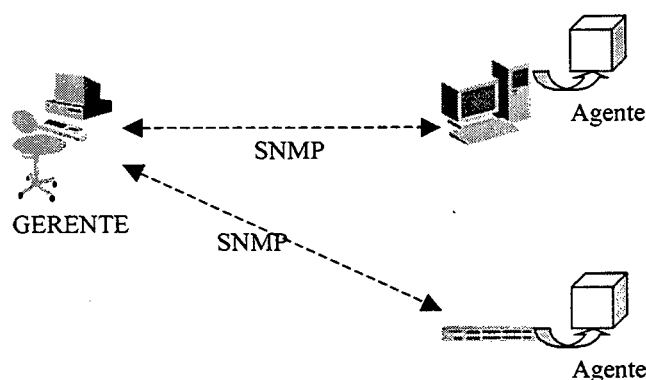


Figura 7 Comunicação entre Gerentes e Agentes

A comunicação entre o agente e o gerente ocorre através da troca de *PDU*s (*Protocol Data Unit*). O SNMP define somente cinco tipos de *PDU*s: *GetRequest*, *GetNextRequest*, *GetResponse*, *SetRequest* e *Trap*. Todas as *PDU*s, exceto a *Trap* (Figura 10), possuem um mesmo formato (Figura 9). As mensagens SNMP transportam o nome da comunidade e o número da versão do SNMP. A identificação da versão garante que o gerente e o agente estão trocando mensagens compatíveis. Mensagens entre o agente e o gerente, com diferentes identificadores de versão, são simplesmente descartadas sem sofrer qualquer forma de processamento. O nome da comunidade age como um mecanismo simplificado de senha entre gerentes e agentes, ou seja, quando o agente recebe uma solicitação do gerente, ele confere se o nome das comunidades são iguais, caso não sejam, retorna um *trap* indicando uma falha de autenticação.

As exigências do SNMP para o protocolo de transporte são modestas. Em virtude deste fato, o protocolo de transporte utilizado pelo SNMP é o UDP (*User Datagram Protocol*),

que é um protocolo não orientado à conexão e que não possui correção de erros.

No modelo SNMP, o gerente envia mensagens para ler (*get* e *get-next*) e alterar (*set*) os valores de variáveis SNMP. Variáveis SNMP são constituídas pela concatenação do identificador do objeto gerenciado (OID) e o identificador de uma instância deste objeto. Um objeto gerenciado é formalmente descrito pela linguagem ASN.1 (*Abstract Syntax Notation One*), que define o tipo (inteiro, *String*, etc), as restrições de acesso, a localização do objeto dentro de uma MIB (*Management Information Base*), entre outras informações. Uma MIB consiste em um conjunto de objetos gerenciados organizados em uma estrutura de árvore, em que cada ramo desta árvore é identificado por um valor inteiro positivo. Um identificador de um objeto gerenciado é obtido pela concatenação dos identificadores de cada sub-ramo da árvore, iniciando da raiz, até alcançar o objeto desejado. Na Figura 8 é apresentado um exemplo do objeto gerenciado *sysDescr* que apresenta o seguinte identificador (OID): 1.3.6.1.2.1.1.1.

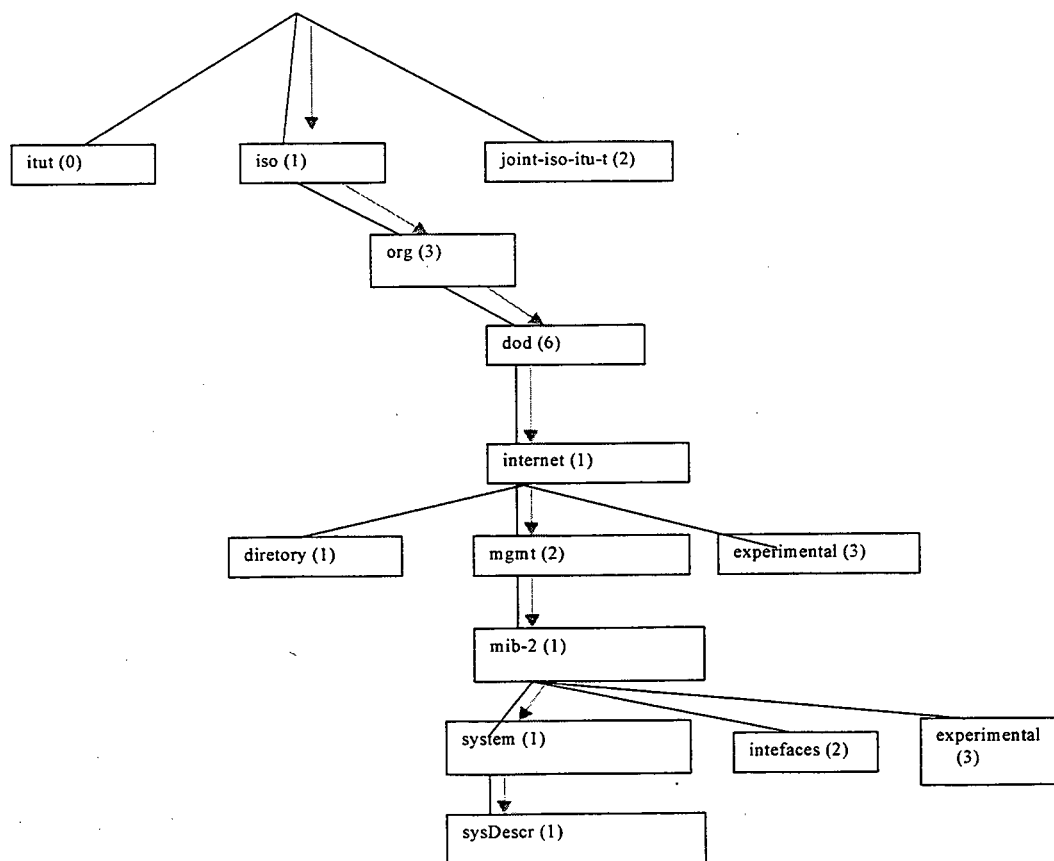


Figura 8 Localização do Objeto Gerenciado *sysDescr* na MIB

Um objeto gerenciado pode possuir múltiplas instâncias. A cada instância está associado um valor. Para obter o identificador de uma variável SNMP, basta concatenar o identificador do objeto com o identificador da instância. Na Figura 8 é apresentado um exemplo de obtenção do valor da variável *sysDescr* utilizando o utilitário *snmpinfo* do sistema operacional AIX 4.1.5 da IBM. Para a obtenção do valor da variável é fornecido o identificador da variável (1.3.6.1.2.1.1.0) que é composto pela concatenação do identificador do objeto (1.3.6.1.2.1.1.1.) com o identificador da instância (0).

```
snmpinfo -m get -h 150.162.1.232 1.3.6.1.2.1.1.0
1.3.6.1.2.1.1.0 = "Cisco Internetwork Operating System Software
IOS (tm) GS Software (GS7-J-M), Version 11.1(8), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1996 by cisco Systems, Inc.
Compiled Thu 05-Dec-96 17:04 by tamb"
#
```

Quadro 1 Obtenção do Valor da Variável *sysDescr* (1.3.6.1.2.1.1.0)

#### 2.4.1 GetRequest

A PDU *GetRequest* é utilizada para a obtenção do valor de uma variável SNMP. Quando o gerente envia uma PDU *GetRequest* para o agente, na máquina gerenciada, deve também, fornecer uma lista de variáveis SNMP. Em condições isentas de erros, é retornada uma PDU de *GetResponse* com a lista de variáveis e seus respectivos valores. No entanto, se ocorrer algum erro, pode retornar uma PDU *GetResponse* com os seguintes *status* de erro:

- caso a variável não seja encontrada na posição especificada, ou o identificador oferecido como parâmetro não seja uma instância de um objeto, é retornada uma PDU com o *status noSuchName*;
- se o tamanho da mensagem extrapola o limite suportado pelo agente, é retornada uma PDU com o *status tooBig*;
- caso ocorra um outro erro, qualquer é retornada uma PDU com o *status genErr*.

#### 2.4.2 GetNextRequest

A PDU *GetNextRequest* é semelhante a *GetRequest*. A diferença básica está no fato de que a *GetNextRequest* retorna o valor do próximo objeto gerenciado na MIB (o próximo em ordem lexicográfica de identificador). Outra diferença é que a PDU *GetNextRequest* não

exige que o identificador fornecido referencie uma instância de um objeto, pois caso o identificador não referencie uma instância, é retornado o valor do próximo objeto, que pode ser uma instância ou não. Com isto, pode ser retornada uma PDU *GetResponse* com o identificador dos próximos objetos, que podem apresentar um valor associado (caso sejam uma instância) ou não. A PDU *GetNextRequest* é muito útil para obter os valores de objetos tabulares, em que se pode fornecer o identificador do objeto gerenciado e, através de sucessivas invocações de *GetNextRequest*, obter-se os valores das instâncias deste objeto. Caso ocorra algum erro é retornado uma PDU *GetResponse* com os possíveis *status* de erro:

- caso o objeto fornecido como parâmetro ou o próximo objeto não possam ser encontrados, é retornada uma PDU com o *status noSuchName*;
- se o tamanho da mensagem extrapola o limite suportado pelo agente, é retornada uma PDU com o *status tooBig*;
- caso ocorra um outro erro qualquer, é retornada uma PDU com o *status genErr*.

### 2.4.3 SetRequest

A PDU *SetRequest* é utilizada para alterar o valor associado às instâncias de um ou mais objetos gerenciados. Caso não ocorram falhas, o agente retorna uma PDU *GetResponse* com o *status noError*. Se algum erro ocorrer, pode ser retornada uma PDU *GetResponse* com um dos seguintes *status* de erro:

- se o objeto referenciado não existir, é retornada uma PDU com o *status noSuchName*;
- algumas implementações de agentes podem retornar o *status readOnly*, caso o acesso a variável desejada seja somente para leitura;
- caso a lista de variáveis e valores não sejam coerentes com os tipos, tamanhos e valores, definidos na MIB (pela linguagem ASN.1) é retornada uma PDU com o *status badValue*;
- se o tamanho da mensagem extrapola o limite suportado pelo agente, é retornada uma PDU com o *status tooBig*;
- caso ocorra um outro erro qualquer, é retornada uma PDU com o *status genErr*.

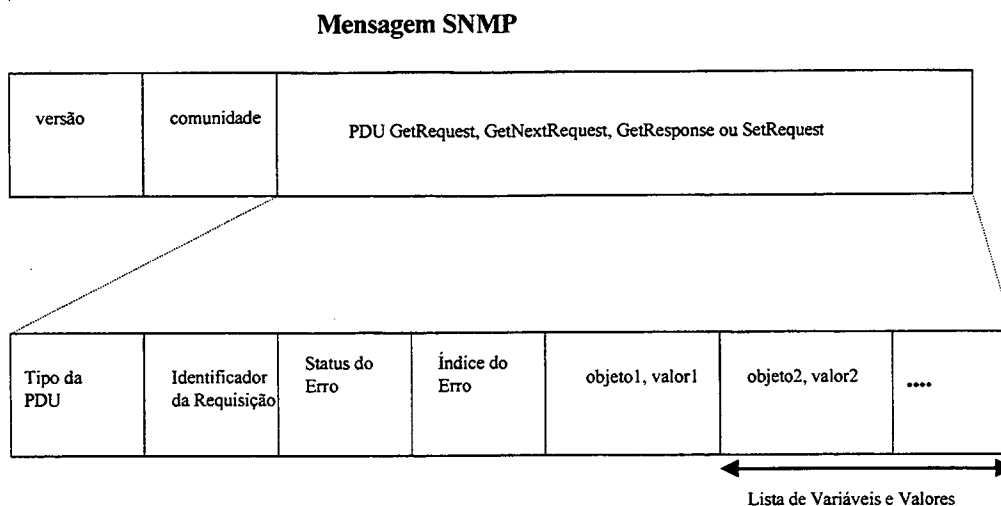


Figura 9 Formato das PDUs *GetRequest*, *GetNextRequest*, *GetResponse* e *SetRequest*

#### 2.4.4 Trap

A PDU de *trap* é enviada por um programa agente para notificar ao gerente a ocorrência de algum evento. Esta PDU tem um formato diferente das outras PDUs SNMP, como pode ser visto na Figura 10. O primeiro campo identifica o tipo da PDU, que neste caso é *trap*. O campo *Empresa* identifica a empresa dentro do ramo da MIB que possui a autoridade sobre o *trap*. O campo *endereço do agente* especifica o endereço IP do agente que gerou o *trap* (caso seja utilizando outro protocolo, além do IP, este campo terá o valor 0.0.0.0). O campo *tipo do trap genérico* especifica o tipo do *trap* genérico, que pode possuir um destes sete valores pré-definidos:

- ***coldStart***: indica que a comunicação agente foi reinicializada e a sua configuração pode ter sido alterada;
- ***warmStart***: indica que a comunicação com o agente foi reinicializada, mas a sua configuração não foi alterada;
- ***linkDown***: falha na comunicação com uma interface;
- ***linkUp***: indica o restabelecimento da comunicação com uma interface;
- ***authenticationFailure***: o agente recebeu uma mensagem com o nome de comunidade incorreto;
- ***egpNeighborLoss***: falha na comunicação com o vizinho EGP;



- *enterpriseSpecific*: indica que é um *trap* específico definido com mais detalhes nos campos *tipo do trap específico* e *Empresa*.

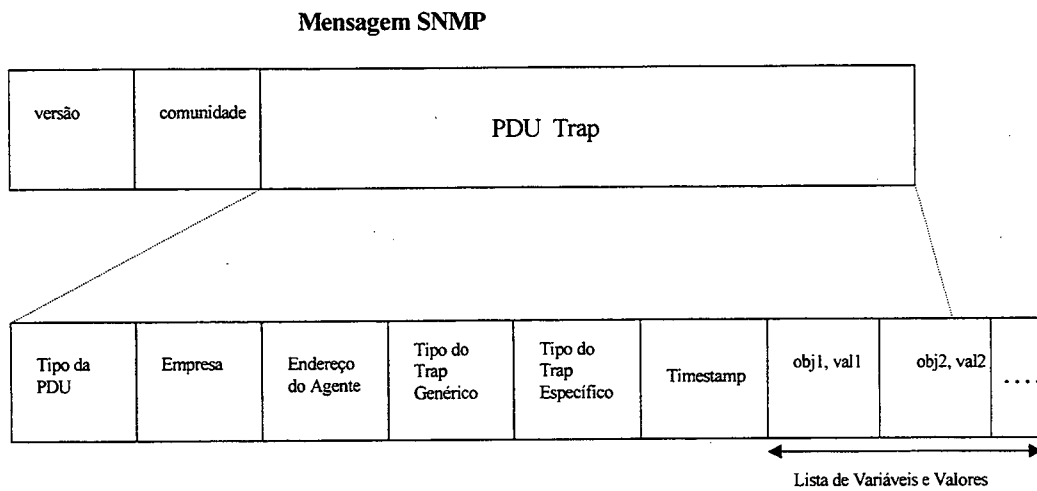


Figura 10 Formato da PDU Trap

O campo *Timestamp* indica o tempo decorrido entre a última reinicialização do agente e o instante da emissão do *trap*.

O campo *tipo do trap específico* define os *traps* não genéricos. Quando um agente deseja enviar um *trap*, basta preencher os campos citados anteriormente e enviá-lo.

### 3 Ferramentas de Gerenciamento Integradas à Tecnologia Web

Neste capítulo são apresentadas algumas ferramentas de gerenciamento de redes que permitem uma plena integração com a tecnologia Web. Uma característica comum a todas é permitir a integração do Java com o protocolo SNMP ou IIOP/CORBA.

#### 3.1 Advent SNMP API

A *Advent SNMP API* consiste em um conjunto de classes implementadas em Java que permitem o envio e recebimento de primitivas SNMP, através de comunicações síncronas e assíncronas com o agente sendo gerenciado.

A *Advent SNMP API* está disponível de modo público para instituições educacionais (<http://www.Adventnet.com>), possui uma boa documentação e um grande número de aplicações que exemplificam operações como: obter o valor de uma variável SNMP utilizando a primitiva *get*, apresentar o valor das variáveis de uma sub-árvore da MIB através da primitiva *get-next*, manipulação de *traps*, implementação de um agente SNMP, entre outras.

A *Advent SNMP API* pode ser utilizada tanto em *applets*, como em *applications* Java. *Applications* Java podem realizar conexões TCP/IP para qualquer equipamento da rede, sem restrições de segurança. No entanto, *applets* (sendo executados em *browsers Web*) só podem realizar conexões TCP/IP com a máquina onde está localizado o servidor *Web* responsável pela transferência do *applet*. Em virtude deste fato, a *Advent SNMP API* possui uma aplicação implementada em Java denominada *SNMP Applet Server* (SAS). Esta aplicação permite que mensagens SNMP, provenientes do *applet*, possam ser repassadas para o processo agente e vice-versa (Figura 11). Através deste artifício, é possível utilizar um *applet* Java para realizar consultas SNMP a qualquer equipamento desejado.

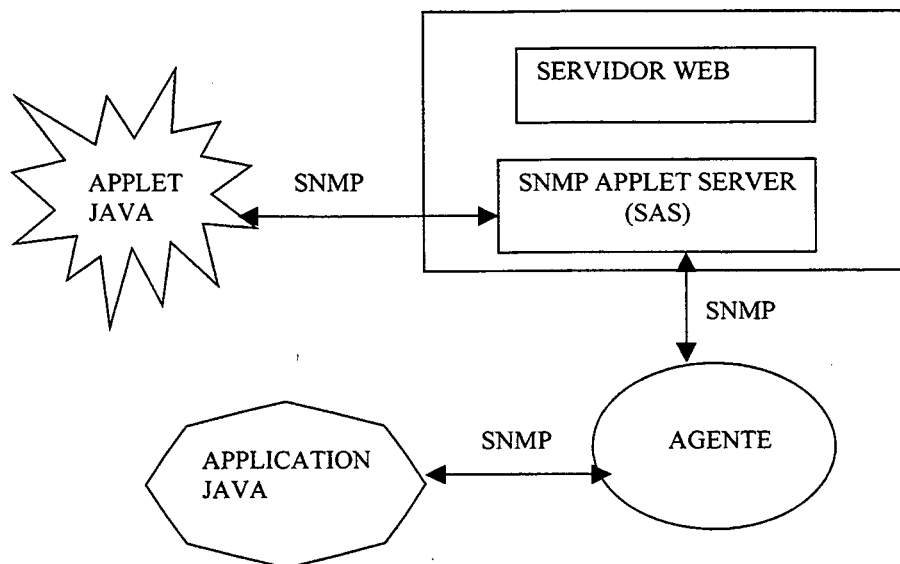


Figura 11 Consultas SNMP ao Agente através de *Applets* e *Applications*

Para a utilização da *Advent SNMP API* visando o envio e recebimento de primitivas SNMP utilizando métodos de comunicação síncronos ou assíncronos entre a aplicação e o processo agente, deve-se basicamente seguir os seguintes passos:

**1. Inicializar uma sessão SNMP para enviar e receber PDUs:** qualquer aplicação que necessite inicializar uma sessão SNMP, para enviar requisições e receber respostas de um agente, deve primeiramente instanciar a classe *Snmplib* e inicializar o *thread* que gerencia e monitora sessões SNMP. Para inicializar uma sessão SNMP deve-se seguir os seguintes passos:

1. Instanciar a classe *Snmplib* (`Snmplib api = new Snmplib();`);
2. Inicializar um *thread* API para monitorar sessões SNMP (`api.start();`);
3. Instanciar a classe *SnmplibSession* para o objeto *api*;
4. Informar o endereço do *host* onde se localiza o agente, o nome da comunidade, a porta remota, a porta local e valores de *timeout*. O endereço do *host* é um parâmetro obrigatório, no entanto, se não forem informados os demais valores, serão atribuídos alguns valores *default*.

**2. Construir uma PDU para ser enviada para o agente:** para enviar uma requisição SNMP, é necessário instanciar o objeto *PDU*. Deve-se informar ao objeto *PDU*, qual a

primitiva SNMP que ele irá transportar (por exemplo: *GET\_REQ\_MSG* para a primitiva *get*) e objeto gerenciado que irá ser consultado (utilizando o objeto *OID*). Pode-se resumir a tarefa de construir uma PDU à execução dos seguintes passos:

1. Instanciar a classe *SnmpPDU* (*SnmpPDU pdu = new SnmpPDU(api)*);
2. Informar qual a primitiva SNMP que a PDU deverá transportar (*pdu.command = GET\_REQ\_MSG*);
3. Instanciar o objeto *SnmpOID* (*SnmpOID oid = new SnmpOID("IdentificadorDoObjetoGerenciado", api)*);
4. Adicionar o objeto *oid* para a lista de objetos da PDU (*pdu.addNull(oid)*).

3. **Enviar e receber mensagens SNMP de modo síncrono:** para enviar e receber mensagens SNMP de modo síncrono, primeiramente deve-se inicializar uma sessão SNMP (*session.open()*). Pode-se então, enviar uma PDU e esperar até que uma resposta seja recebida, ou o valor de *timeout* seja alcançado. É possível enviar uma PDU de modo síncrono, através do comando: "*rpdu = session.syncSend(pdu)*". Neste caso, o programa enviará a PDU e o processamento será interrompido até o retorno de uma resposta, ou serem alcançados os valores de *timeout*.
4. **Enviar e receber mensagens SNMP de modo assíncrono:** no modo assíncrono de recebimento de mensagens, pode-se enviar várias requisições SNMP sem necessariamente, ter que esperar a resposta para cada uma consecutivamente. Existem dois métodos para enviar e receber mensagens SNMP de modo assíncrono: utilizando os métodos *session.send()* e *session.receive()*, ou utilizando o método *callback* da classe *snmpClient*. O método *session.send()* é similar ao *session.syncSend()*, exceto ao fato que ele retorna o controle ao programa imediatamente, ao invés de esperar pela chegada da resposta. Para o recebimento da mensagem, basta utilizar o método *session.checkResponses()* para verificar se existe alguma resposta e o *session.receive()* para receber a mensagem. Utilizando-se o método *callback*, pode-se evitar o laço de repetição para verificar o recebimento de mensagens. Para usufruir deste recurso, basta criar uma classe derivada da classe *snmpClient* e sobrepor o método *callback*, que será invocado sempre que houver o recebimento de uma nova resposta.

Devido às características de facilidade de uso, boa documentação e ao fato da *Advent SNMP API* ser implementada em Java (possibilitando que ela possa ser utilizada em

praticamente qualquer plataforma), esta ferramenta foi escolhida como a mais indicada para a construção de *applications* e *applets* Java que necessitam realizar consultas SNMP. Em consequência destas características, esta ferramenta é utilizada nas demais etapas deste trabalho.

### 3.2 Bojangles

*Bojangles* é uma ferramenta em desenvolvimento da IBM, que consiste em um conjunto de classes e utilitários destinados a construção de aplicações distribuídas, utilizando a linguagem Java. Este *software* oferece uma ótima solução para o desenvolvimento de aplicações utilizando uma interface *Web*.

O *Bojangles* é constituído por um conjunto de classes e aplicações, implementadas em Java, que podem ser divididas basicamente em dois módulos: *JBOF* (*Java Business Object Framework*) e *JORB* (Figura 12). O *JBOF* consiste em um conjunto de classes para a implementação de objetos comerciais. O *JORB* oferece os mecanismos de comunicação entre os objetos, através da utilização do padrão CORBA. O *JORB* possui a implementação de um ORB, construído em Java, que pode ser utilizado em qualquer plataforma que possua um interpretador Java instalado.

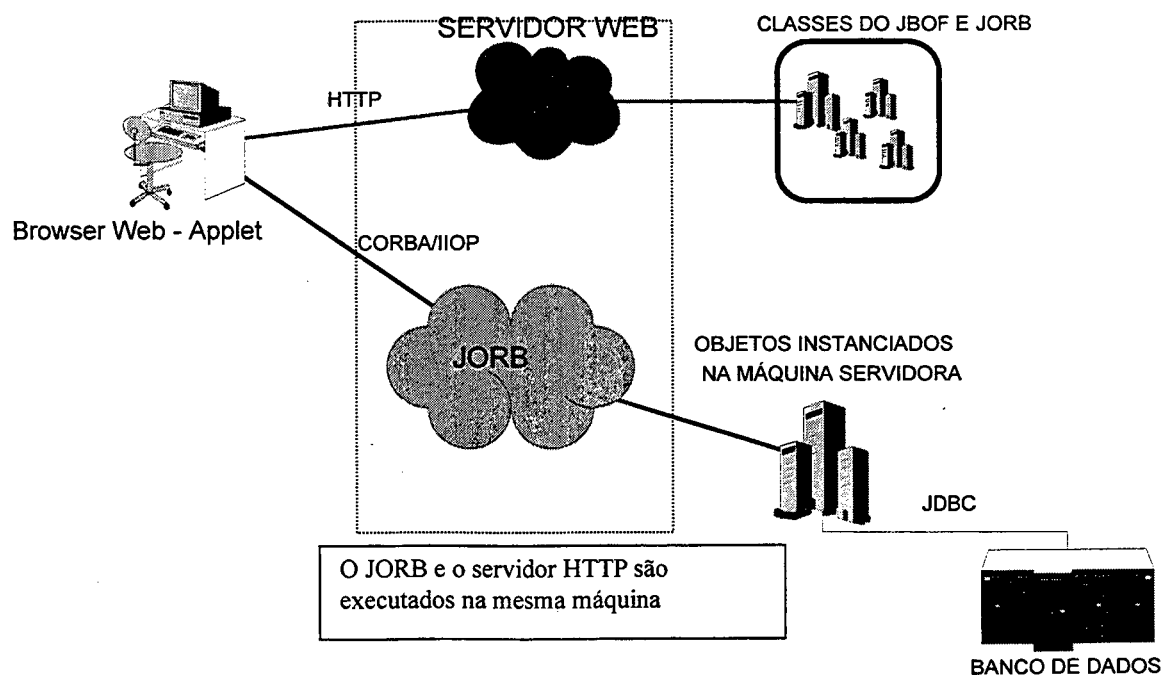


Figura 12 Modelo de utilização do *Bojangles*

O *Bojangles* permite a criação de um modelo cliente/servidor baseado em três camadas, em que *applets* Java instanciam objetos em uma estação servidora, utilizando o protocolo IIOP. Os objetos na estação servidora é que efetivamente farão a comunicação com o banco de dados ou outras aplicações. Este modelo apresenta as seguintes características:

- as aplicações clientes são instaladas, automaticamente, utilizando o protocolo HTTP associado a servidores *Web*;
- as aplicações clientes e servidoras são independentes de plataforma (por serem implementadas em Java);
- o *applet* sendo executado no *browser Web*, só necessita utilizar o protocolo IIOP para a comunicação com o servidor. Isto evita a necessidade de se utilizar novas classes para comunicação com outros dispositivos;
- uma vez que a maior parte de código de uma aplicação está localizado no servidor, e no cliente (*browser Web*) existem apenas os módulos necessários à interface com o usuário, as aplicações tornam-se menores e podem ser transferidas de modo rápido para os *browsers Web*, através do protocolo HTTP;
- neste modelo, os clientes (*applets*) instanciam objetos localizados na servidora (utilizando protocolo IIOP). Tais objetos efetivamente efetuam a comunicação com outros dispositivos. Com isto, o nível de segurança das aplicações aumenta, pois não é necessário transferir todo o código para a máquina cliente. Esta característica evita que se torne visível para a aplicação cliente, características como: senhas para comunicação com o banco de dados ou o código fonte das aplicações (que pode ser obtido decodificando o arquivo em *byte code* das classes implementadas em Java);
- o desempenho apresentado pelas aplicações é muito maior, pois o código pode ser executado diretamente em uma máquina servidora, de forma mais eficiente, quando comparado à sua execução em um *browser Web*;
- uma característica muito importante que o *Bojangles* oferece é a independência de plataforma dos ORBs utilizados. Com isto, independente da plataforma que for utilizada, os mecanismos de instalação e configuração dos ORBS serão os mesmos, não sendo necessário instalar um ORB específico em cada plataforma e viabilizar a comunicação entre eles.

Um exemplo de aplicação utilizando o *Bojangles*, pode ser um *applet* para visualização de *logs* armazenados em uma base de dados relacional. Através do método

tradicional, seria criado um *applet* que poderia acessar o banco de dados, utilizando as classes do JDBC. Neste caso, toda a aplicação seria executada no *browser Web* e a senha do banco e todo o código da aplicação teriam que ser transferidos para este, o que, além de ser um processo demorado, também é inseguro. Utilizando o *Bojangles*, é criada uma classe *Logs*, que acessa o banco e retorna informações de *logs* do sistema. Tal classe pode ser instanciada a partir de um *applet* (utilizando o protocolo IIOP) e ser executada no servidor. Deste modo, o *applet* só contém o código necessário à interface com o usuário. Isto torna o programa cliente menor, mais seguro e evita a transferência de classes desnecessariamente (como a transferência das classes do JDBC).

Pode-se resumir a construção de aplicações, utilizando o *Bojangles*, nos seguintes passos:

1. Criar a interface do objeto separadamente da aplicação. A interface deve ser definida utilizando a declaração *interface* do Java. Tomando como exemplo, a implementação do objeto *Computer*, que é capaz de fornecer informações sobre a ocupação de CPU e do nome do sistema operacional da máquina em que está instanciado, pode-se definir a interface deste objeto através do arquivo *Computer.java* e a sua implementação no arquivo *ComputerImp.java*. No *Quadro 2* é apresentado um exemplo de uma interface, definida no arquivo *Computer.java*;

```
package agente;
import java.lang.Object;
public interface Computer {
    public long GetCPU();
    public String getSystemName();
}
```

Quadro 2 Interface do Objeto

2. Após a criação da interface e da implementação do objeto, deve-se gerar os *skeletons* e *stubs*. O *Bojangles* possui os utilitários *RunStubEmitter* e *RunSkeletonEmitter* para geração de *skeletons* e *stubs*, a partir de uma declaração de interface. Isto permite que um

programador não necessita conhecer detalhes de IDL para construir uma aplicação distribuída, utilizando a linguagem Java. No *Quadro 3* pode-se visualizar um exemplo de criação de *sekeletons* e *stubs* para a interface *Computer*, em que serão gerados os arquivos *ComputerStub.java* e *ComputerSkeleton.java*;

```
java COM.ibm.jaws.tools.emit.RunStubEmitter agente.Computer
java COM.ibm.jaws.tools.emit.RunSkeletonEmitter agente.Computer
```

Quadro 3 Geração de *Stubs* e *Skeletons*

3. Deve-se criar a aplicação cliente e compilar todos os arquivos com a extensão “.java”. Neste caso, utiliza-se o método `CORBA.ORB_init` para inicializar o JORB. Como pode ser visto no *Quadro 4*, deve-se utilizar o método `client.URLToObject (url)` para instanciar o objeto na máquina desejada. A URL contém o seguinte formato: “IIOP://NomeDoServidor/:PortaDoServidor;IdentificadorDaInstância?NomeDaClasse”. O método `client.URLToObject (url)` retorna o identificador da instância. Ao final da utilização do objeto, deve-se invocar o método `cliente.Close()` para desfazer a conexão com o JORB. No *Quadro 4* é apresentado um exemplo de inicialização do ORB, criação de uma instância do objeto *Computer*, invocação de um método deste objeto e finalização da conexão com o servidor JORB;

```
String hostURL = new String("IIOP://"+serverName+"."+serverPort);
Computer computer = null;
ORB client = CORBA.ORB_init(null);
computer = (Computer)client.URLToObject(hostURL+ ";/computer" +
                                         "?agente.Computer");
System.out.println("Sistema é: " + computer.getSystemName());
client.close();
```

Quadro 4 Aplicação Cliente

4. Antes de executar a aplicação cliente, deve-se utilizar o comando demonstrado no *Quadro 5*, para inicializar o servidor JORB;



```
java COM.ibm.corba.IIOP.IIOPServer
```

#### Quadro 5 Inicialização do JORB

5. Pode-se registrar as aplicações utilizando o aplicativo *RegisterImpl*, como pode ser visto no Quadro 6;

```
java COM.ibm.corba.porting.RegisterImpl agente.ComputerImpl agente.Computer
agente.ComputerStub agente.ComputerSkeleton
```

#### Quadro 6 Registrando uma Implementação

6. Finalmente pode-se executar a aplicação cliente.

O *Bojangles* permite a criação de um ambiente de computação distribuída, independente de plataforma. Utilizando o JORB é possível criar agentes e objetos gerenciados, implementados em Java que podem se comunicar entre si (utilizando o protocolo IIOP). Este modelo enquadra-se perfeitamente em um ambiente de *cluster* de estações em que, poderia haver o instanciamento de um objeto gerenciado em cada máquina e ser implementado um agente mestre. Este agente mestre poderia obter informações dos objetos gerenciados através do protocolo IIOP, e repassá-las para uma aplicação gerente através do protocolo SNMP ou IIOP (Figura 13). Uma das informações que poderiam ser fornecidas por um objeto gerenciado seria a taxa de ocupação de CPU de uma máquina. Um agente central poderia coletar estas informações, através da invocação de métodos dos objetos instanciados, e repassar para o gerente a lista das máquinas com a CPU menos utilizada dentro do *cluster* (utilizando o protocolo SNMP ou IIOP).

O *Bojangles* é amigável, possui uma documentação razoável e é implementado em Java. Devido a estas características, o *Bojangles* foi a ferramenta escolhida para permitir a comunicação básica entre agentes e objetos gerenciados neste trabalho de dissertação.

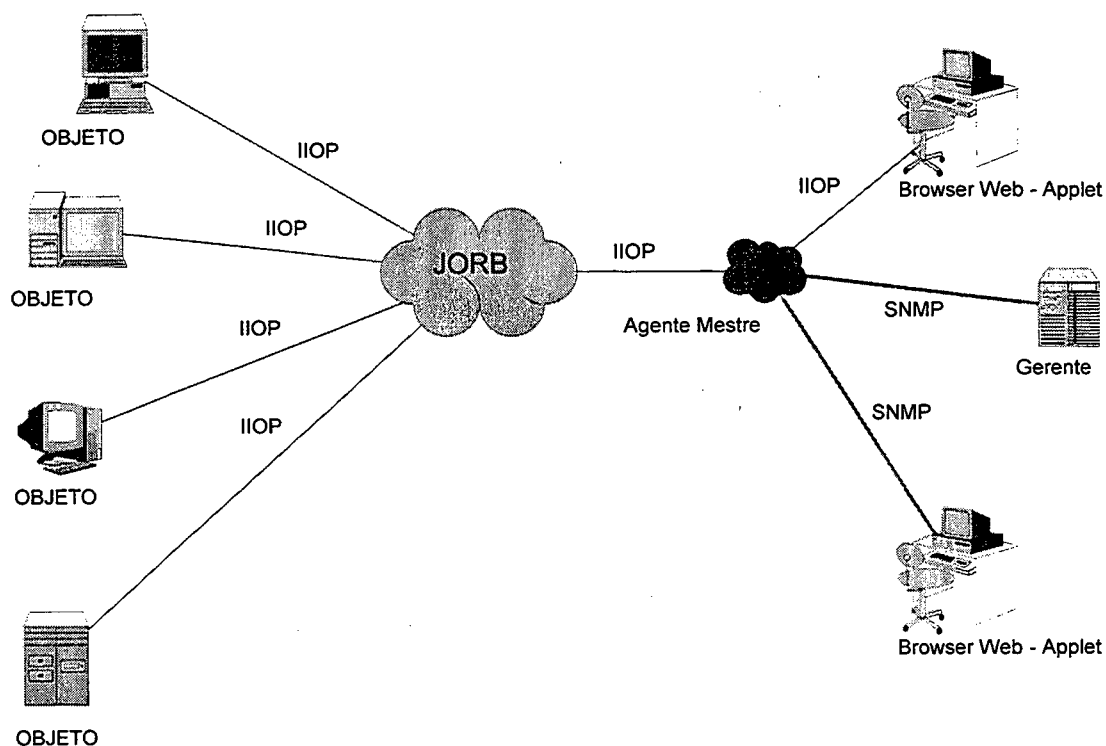


Figura 13 Comunicação entre Agentes e Objetos Gerenciados utilizando o JORB

### 3.3 Ambiente de Desenvolvimento

Na Universidade Federal de Santa Catarina (UFSC) existe um conjunto de estações (*cluster*) que estão sendo interligadas por uma rede de alta velocidade e que destinam-se a execução de aplicações na área da computação de alto desempenho.

Atualmente o *cluster* conta com 22 estações IBM 43P, 2 SUN UltraSparc, 1 IBM F40 e 9 nós de um IBM SP2 que estão interligados por uma estrutura de rede ETHERNET e FDDI. No entanto, este ambiente já está sendo ampliado para ser totalmente interligado por uma estrutura de rede ATM 155 Mb e deve possuir mais 6 estações IBM 43P, 2 Sun UltraSparc e 12 Sun Sparc10/20.

Todas as máquinas possuem um conjunto de contas de usuários comum, permitindo que um usuário possa conectar-se em qualquer máquina do *cluster*. Foram instalados em cada equipamento um conjunto de ferramentas destinadas ao processamento paralelo e distribuído.

Dentre elas, destacam-se o PVM (*Parallel Virtual Machine*), MPI (*Message Passing Interface*), DSOM e *Bojangles*. Para o gerenciamento *batch* das aplicações do *cluster*, foi escolhido o *software* LoadLeveler da IBM.

No ambiente em estudo, há uma grande carência de uma estrutura de gerenciamento adequada para o *cluster*, tanto por parte de administradores, como por parte dos usuários. Esta carência se faz notar principalmente quando necessita-se obter ou alterar um parâmetro genérico que envolva o conjunto de estações de trabalho. Uma situação bastante comum e que ilustra este panorama é a obtenção da taxa de utilização de CPU de cada máquina, visando determinar quais destas estão mais ociosas. Neste caso, o usuário deve conectar-se ou emitir um comando para cada máquina do *cluster*, verificar a taxa de ocupação de CPU e obter as informações desejadas. Outro exemplo ocorre quando um administrador deseja remover arquivos de *logs* nas máquinas do *cluster*. Neste caso, ele deve conectar-se a cada máquina no *cluster* e remover os arquivos desejados. Com os exemplos anteriores, pode-se observar que operações globais ou parciais<sup>4</sup> no *cluster* podem tornar-se extremamente trabalhosas. Visando suprir estas necessidades, este trabalho propõem a implementação de um sistema de gerência que pretende solucionar a maioria dos problemas encontrados na gerência do *cluster*, permitindo que possam ser obtidas e modificadas as informações globais ao *cluster* através de uma única mensagem de gerência (utilizando o protocolo IIOP/CORBA ou SNMP).

As experiências realizadas neste trabalho foram efetuadas utilizando-se as estações de trabalho pertencentes ao *cluster* da UFSC e as aplicações foram compiladas e executadas através do JDK (*Java Developer Kit*) 1.0.2.

---

<sup>4</sup> Dentro do contexto deste trabalho, operações globais são aquelas que afetam todas as máquinas do *cluster* e operações parciais são aquelas que afetam apenas um subconjunto destas máquinas.

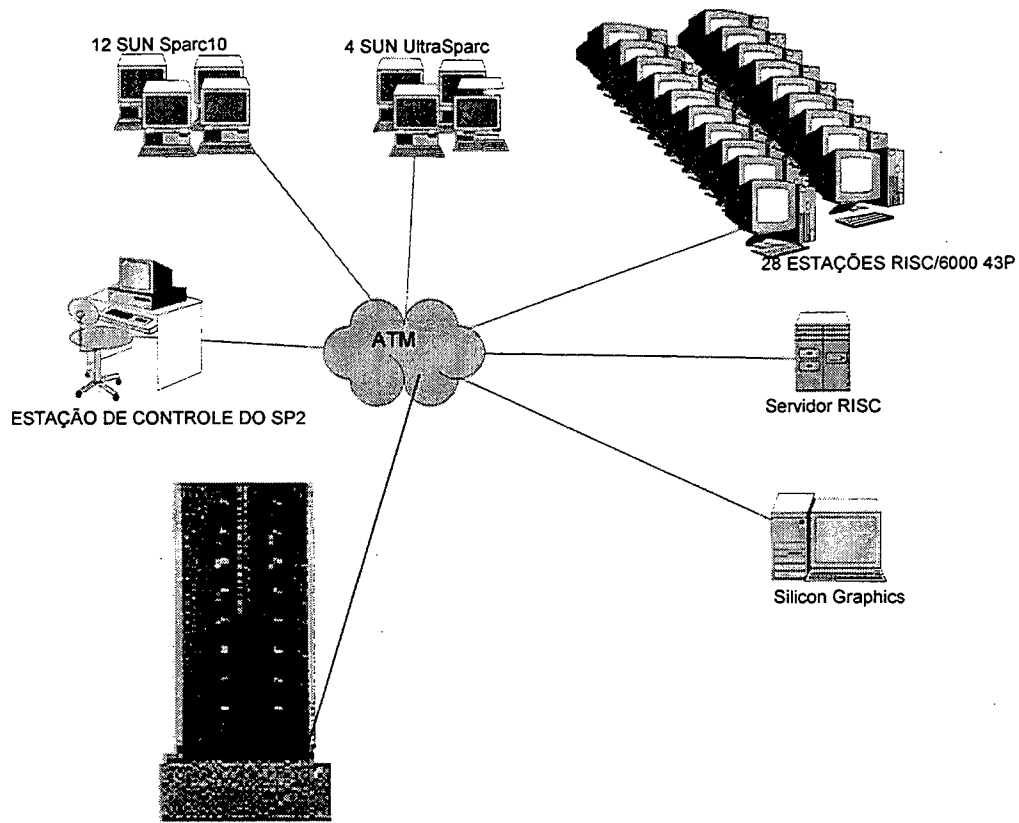


Figura 14 *Cluster* de Estações

## 4 Ambiente de Gerência de Redes utilizando *Browsers Web*

### 4.1 Introdução

O objetivo deste capítulo é descrever o desenvolvimento e a utilização de um conjunto de ferramentas genéricas, que permitem a obtenção, o armazenamento e a visualização de informações referentes a objetos gerenciados, independente da plataforma ou localização geográfica dos equipamentos que realizam tais funções. Estas ferramentas formam um verdadeiro ambiente de desenvolvimento de aplicações de gerência via *Web*.

Um objetivo básico traçado para o ambiente proposto é oferecer uma solução que permita a um usuário visualizar informações de um objeto gerenciado, independente de sua localização geográfica. A utilização do protocolo HTTP associado a *browsers Web*, permite a criação de um mecanismo automático de transferência de *softwares* em uma máquina cliente, ou seja, quando um usuário desejar executar um *applet* para a visualização de informações sobre um determinado objeto gerenciado, ele não precisa instalar este *software* em sua máquina. Para a execução deste *applet*, basta conectar-se a um servidor *Web*, através de um *browser*, e efetuar a transferência do *software* desejado, independente da localização geográfica ou plataforma do equipamento utilizado.

Uma maneira racional para realizar a coleta de informações, pode ser através da utilização de um protocolo padronizado que seja largamente utilizado, tanto na Internet, como em *Intranets*. O protocolo mais indicado para realização de tal função é o SNMP. Neste capítulo são descritas as formas de utilização do SNMP para coletar as informações referentes a objetos gerenciados, indicando também, uma solução independente de plataforma, para programadores que desejam efetuar consultas SNMP, diretamente através de suas aplicações.

Foi definida a utilização de um banco de dados relacional para o armazenamento das informações de gerenciamento, através de um mecanismo seguro e eficiente. Isto possibilita que as aplicações possam armazenar e recuperar informações através de simples consultas SQL. O ambiente proposto apresenta uma forma padronizada de realizar consultas a banco de dados relacionais, através *applets* e *applications* Java, que pode ser aplicada para a maioria dos servidores de banco de dados disponíveis comercialmente e não exige a instalação prévia de qualquer tipo de *driver*, para realizar a conexão com o banco na máquina cliente.

Fundamentalmente, o *Ambiente de Gerência* em questão consiste na recomendação e

implementação de um conjunto de ferramentas que permitem a *applets* e *applications* Java obter informações de gerenciamento, utilizando o protocolo SNMP; armazenar estes dados em uma base de dados relacional; e visualizar as informações obtidas através de gráficos e tabelas, em uma *interface Web*.

#### 4.2 Métodos para Obtenção de Informações

Para o *Ambiente de Gerência* foi empregado um conjunto de ferramentas que permitem utilizar o protocolo SNMP, para a obtenção de informações de gerência através de *applets* ou *applications* Java. Isto permite uma solução genérica e independente de plataforma para a obtenção de informações de gerência de redes.

O *Ambiente de Gerência* permite que informações armazenadas em uma base de dados relacional possam ser recuperadas e visualizadas através de *browsers Web*. Em consequência deste fato, qualquer ferramenta que possa obter informações de gerenciamento e armazená-las em uma base de dados relacional, pode ser empregada como fonte de informações para este ambiente.

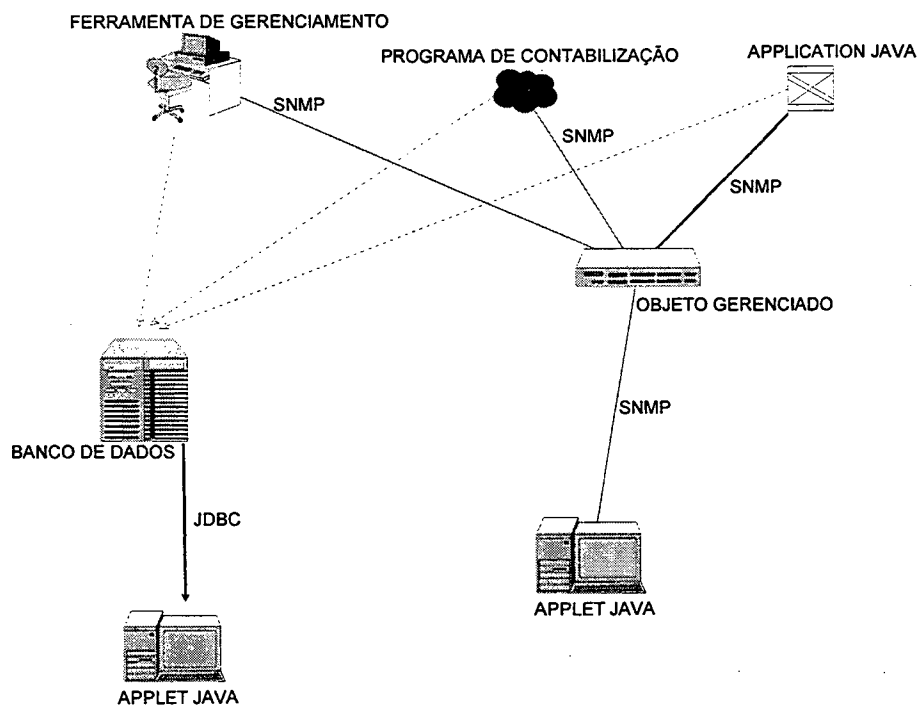


Figura 15 Modelo de Obtenção e Visualização de Dados

#### 4.2.1 Métodos Tradicionais de Obtenção de Informações

Uma vez que este *Ambiente de Gerência* permite que qualquer informação armazenada em uma base de dados Relacional possa ser recuperada e visualizada através de *browsers Web*, é possível utilizar ferramentas tradicionais de gerenciamento de Redes, que possuem integração com bancos de dados relacionais, para a obtenção e o armazenamento de informações de gerência.

Um dos meios de se obter informações de gerência de redes é através da utilização de uma ferramenta de gerenciamento tradicional como: *NetView 6000*, *SunNet Manager*, *HP OpenView*, entre outras. Estas ferramentas geralmente permitem a integração com bancos de dados relacionais e realizam a obtenção de informações através do protocolo SNMP. Esta solução apresenta como pontos positivos, o fato ser plenamente integrável com o sistema de gerência central e de utilizar uma ferramenta já implementada e plenamente testada para executar esta tarefa. No entanto, *softwares* de gerência possuem mecanismos de instalação, configuração e utilização relativamente complexos, apresentam um elevado custo de aquisição e não obtêm a mesma flexibilidade que o desenvolvimento de programas de contabilização específicos para uma determinada função.

Existem diversos sistemas operacionais que oferecem recursos para o armazenamento de informações de contabilização e *logs*, em uma base de dados relacional. Este recurso é comum em máquinas UNIX e permite que informações, como o tempo de uso de CPU por usuário ou a taxa de ocupação do disco, possam ser armazenadas em uma base de dados relacional e visualizadas através de *browsers Web*.

A integração com outras ferramentas de gerência é essencial para propiciar uma solução completa para visualização de informações de gerência via *Web*.

#### 4.2.2 Consultas SNMP através de *Applets* e *Applications Java*

Afim de possibilitar uma solução genérica e independente de plataforma, foi desenvolvida uma classe (*SNMPMan.java*) que permite a obtenção de informações referentes a objetos gerenciados, a partir de *applets* e *applications Java*. Esta classe foi construída utilizando-se a *Advent SNMP API* (<http://www.adventnet.com>), que consiste em uma API de domínio público, destinada a realizar consultas SNMP através de conexões síncronas e assíncronas entre as aplicações e o processo agente (Capítulo 3). Esta classe pode ser utilizada

basicamente, para duas finalidades:

- Para a monitoração de recursos que exigem a aquisição de informações em tempo de execução, deve-se utilizar *applets* para obter, continuamente, informações através de consultas SNMP. No entanto, por razões de segurança, um *applet* Java só pode realizar conexões TCP/IP destinadas à máquina onde está instalado o servidor HTTP responsável pelo *download* do mesmo. Por esta razão, é necessária a instalação de uma aplicação Java que age como um servidor de mensagens, destinado a repassar mensagens SNMP que trafegam entre o *applet* Java e o processo agente. Esta aplicação deve ser executada na mesma máquina em que reside o servidor *Web*. Um exemplo de obtenção de informações, em tempo de execução, é a criação de um *applet* para detectar o *status* da *interface* de um roteador. Neste caso, as informações devem ser obtidas de um roteador, durante a execução do *applet* destinado a visualização do *status*;

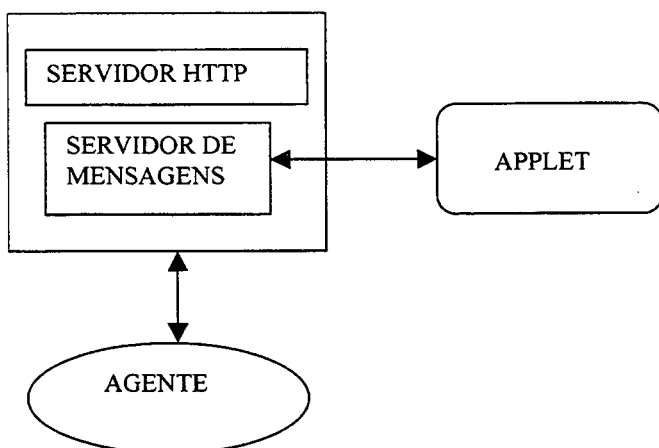


Figura 16 Realização de Consultas SNMP através de *Applets*

- Para a obtenção e armazenamento de informações em um banco de dados relacional, pode-se utilizar um *application* Java para coletar as informações SNMP, sem a necessidade de utilizar um servidor de mensagens, destinado a repassar as primitivas deste protocolo entre a aplicação e o processo agente (visto que um *application* Java pode estabelecer conexões TCP/IP, sem as mesmas restrições de segurança de um *applet*). Pode-se citar como exemplo da utilização deste método, a obtenção de informações do tráfego de *links* Internet, objetivando a visualização destas informações através de *browsers* Web. Neste exemplo, não há a exigência de que o *applet* obtenha informações diretamente do objeto gerenciado, pois estas podem ser armazenadas antecipadamente e o



*applet* necessita apenas consultar o banco de dados, para recuperar as informações desejadas (sem a necessidade de realizar consultas SNMP).

A classe *SNMPMan.java* foi implementada para a realização de consultas SNMP, através de *applets* e *applications* Java. O código fonte desta classe pode ser encontrado no *Anexo A* deste trabalho.

### 4.3 Acesso a uma Base de Dados Relacional através de Aplicativos Java

Um dos objetivos deste projeto é permitir que as informações de gerenciamento possam ser armazenadas em uma base de dados relacional. Isto permite que informações como: tráfego nos *links* (obtido por um programa de contabilização), informações sobre *traps* (fornecidos por um programa gerente) ou dados sobre os recursos utilizados por usuários em uma estação de trabalho, possam ser armazenadas em uma única estrutura de base de dados. Se estas informações estiverem armazenadas em um banco de dados relacional, pode-se facilmente construir aplicações, em uma linguagem de programação qualquer, para emitir comandos SQL visando manipular estes dados, permitindo que auditorias ou análises do comportamento da rede possam ser realizadas, através de simples consultas ao banco [11].

```
class Banco{
    Config conf;

    static {
        try {
            // Registra o driver do servidor de banco de dados
            Class.forName("ibm.net.sql.DB2Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // realiza conexao com o banco de dados
    private Connection ConectaAoBanco() throws SQLException
    {
        String url = conf.retornaURL();
        String userid = conf.retornaUsuarioDoBanco();
        String password = conf.retornaSenhaDoBanco();
        Connection con = DriverManager.getConnection(url,
userid, password );
        return con;
    }
};
```

Quadro 7 Conexão com o Banco de Dados através da Classe *Banco.java*

O acesso a uma base de dados relacional através de um aplicativo Java, é realizado através do JDBC, que consiste em uma API implementada em Java que permite um acesso uniforme e padronizado a qualquer servidor de banco de dados relacional, que ofereça suporte ao *driver* JDBC. O JDBC é um pacote padrão que já é incluído no JDK (*Java Development Kit*), a partir da versão 1.1 (pode ser adquirido no endereço: <http://www.javasoft.com>).

Utilizando o JDBC, é possível realizar a comunicação com os principais servidores de banco de dados relacionais disponíveis comercialmente, através de uma *interface* padronizada e sem a necessidade de uma instalação prévia de nenhum programa ou *driver* para a comunicação com o banco, nas máquinas clientes. Este fato, ocorre em consequência de que o *driver* JDBC, assim como as aplicações, são transferidos para a máquina cliente, utilizando o protocolo HTTP (como qualquer *applet*).

Este trabalho utiliza o JDBC para obter um acesso padronizado ao banco de dados, sem qualquer tipo de *software* para a comunicação com o banco previamente instalado na máquina cliente e independente do sistema operacional ou *hardware* que este cliente possua. Para tanto, foi implementado um conjunto de classes, utilizando o JDBC, afim de realizar consultas e armazenar informações em uma base de dados relacional.

Para efetuar a conexão com o banco de dados (Quadro 7) e realizar consultas a uma base de dados, foi implementada classe *Banco.java*. O código fonte desta classe pode ser encontrado no endereço <http://npd01.cluster.ufsc.br/WebVis>.

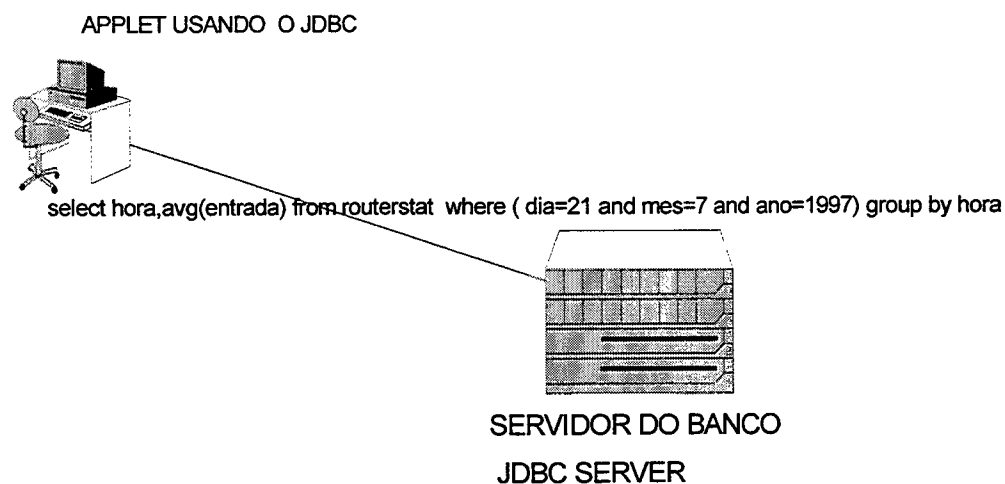


Figura 17 Comunicação entre um *Applet* e o Banco de Dados

#### 4.4 Independência de Plataforma

O meio utilizado para que todos os programas instalados sejam independentes de plataforma, foi implementá-los através do Java, como linguagem de programação [18].

Ao ser compilado um programa em Java, é gerado um código para uma máquina genérica denominado: *byte code*. Este código, pode ser executado tanto por interpretadores como por *browsers Web*, que simulam a implementação de uma máquina virtual baseada em *byte code*. Esta característica confere a programas construídos em Java, a peculiaridade de serem independentes de plataforma, o que consiste em um requisito essencial na construção do *Ambiente de Gerência*.

O Java, por ser uma linguagem baseada em objetos, permite a construção de programas legíveis e de fácil manutenção [2]. Estas qualidades, associadas a característica dinâmica de sua *interface*, permitem que programas escritos em Java sejam muito mais fáceis de serem expandidos, do que um programa escrito em uma linguagem baseada em *script*.

Com isto, pode-se verificar que o Java é uma ferramenta essencial para dotar o *Ambiente de Gerência* com características como: legibilidade, *interface* aprimorada, expansibilidade e independência de plataforma.

#### 4.5 Um Exemplo de Integração de Applets e Applications Java com um Servidor de Banco de Dados Relacional

A forma de utilização do JDBC, para o acesso a qualquer servidor de banco de dados, é basicamente a mesma. A única mudança significativa é o nome do *driver* JDBC a ser utilizado. Neste projeto foram efetuados testes com o Sybase e o DB2 [11] como servidores de banco de dados. Nesta etapa, é apresentado um exemplo demonstrando os passos necessários para a comunicação de um aplicativo Java com o servidor de banco de dados DB2 da IBM.

O primeiro aspecto a ser analisado quando se deseja a implementação de *applets* Java, integrados com o DB2, é que o servidor *Web* e o servidor de banco de dados devem ser instalados na mesma máquina.

O Suporte básico oferecido pelo DB2 para a integração com *applets* é efetuado através do pacote *temjava.sql*. Este pacote consiste em uma versão renomeada do JDBC disponível para domínio público (<http://www.javasoft.com>).

Para a construção de *applications* ou *applets* Java que utilizem o JDBC, para a comunicação com o DB2, devem ser seguidos os seguintes passos:

1. Importar as classes Java apropriadas (*Java.net.URL*, *java.sql.\**, e *ibm.sql* para *applications* ; *tempjava.sql.\** e *ibm.netsql.\** para *applets*);
2. Carregar o *driver* JDBC apropriado (*ibm.sql.DB2Driver* para *applications* e *ibm.netsql.DB2Driver* para *applets*);
3. Realizar a conexão com o banco de dados, utilizando uma *URL* para a localização do servidor. Para a construção de *applets*, é necessário apresentar o identificador do usuário, senha, nome do *host* e porta do *applet Server*;
4. Enviar comandos SQL para o servidor de bando de dados;
5. Receber os resultados obtidos;
6. Encerrar a conexão.

Para desenvolver um *applet* Java, utilizando o JDBC , deve-se definir a classe em um arquivo HTML. Por exemplo, para definir uma classe *teste*, pode-se colocar a seguinte declaração em um arquivo HTML:

```
<applet code="teste.class" width=325 height=275>
```

Para executar um *applet* Java em um máquina cliente, é necessária apenas a utilização de um *browser Web* habilitado para execução de *applets* Java. Quando o arquivo HTML é mapeado em um *browser Web*, as classes Java definidas em *byte code* são transferidas para a máquina, juntamente com o *driver* JDBC oferecido pelo DB2. Quando o *applet* Java utiliza a API JDBC para se conectar ao DB2, o *driver* JDBC estabelece uma conexão proprietária com o DB2, realizando a comunicação com o servidor JDBC .

Para inicializar o Servidor JDBC, deve-se instalar o DB2 e emitir o seguinte comando:

```
> db2jstrt NumeroDaPorta
```

No presente trabalho optou-se pela utilização da porta 6767 para realizar a conexão com o servidor JDBC. Outro procedimento efetuado, foi definir a variável de ambiente "CLASSPATH" para a localização de todos os diretórios de classes do JDBC e demais pacotes instalados.

Um exemplo prático de integração do banco de dados com *applets* e *applications* Java, pode ser encontrado na classe *Banco.java*, que pode ser encontrada no endereço <http://npd01.cluster.ufsc.br/WebVis/>.

## 4.6 Geração de Gráficos em uma Interface Web

A cada dia surgem novos pacotes para geração de gráficos implementados em Java e, muitos destes, são disponíveis para o domínio público. Em consequência deste fato, decidiu-se não implementar uma classe gráfica, mas sim adaptar as classes gráficas existentes para o projeto em questão. Optou-se por definir uma interface padronizada e, através da utilização de classes abstratas, permitir a substituição de uma classe, por outra com recursos melhores, sem a necessidade de modificar o restante do programa.

No presente projeto, foi utilizada a classe *BarGraf.java* (<http://www.javasoft.com>), que consiste em uma classe para gerar gráficos de barra, cujo código fonte foi modificado para atender as necessidades referentes ao projeto do *Ambiente de Gerência*. Para a geração de gráficos mais complexos, será utilizado o pacote gráfico *Netcharts 2.0* (<http://www.netcharts.com>), composto por um conjunto de classes implementadas em Java, que permitem gerar uma variedade de gráficos com avançados recursos de *interface*. A classe implementada para a geração de gráficos denomina-se *Grafico.java* (o código fonte desta classe pode ser encontrado no endereço <ftp://npd01.cluster.ufsc.br/WebVis/>) e define a *interface* básica para a geração de gráficos, em aplicações que utilizam o *Ambiente de Gerência*.

## 4.7 Conclusão

As ferramentas implementadas e utilizadas para o *Ambiente de Gerência*, proposto neste capítulo, possibilitam que aplicações de gerência possam ser construídas de maneira muito simples.

O Java foi a principal ferramenta que viabilizou a implementação deste projeto. Através desta linguagem de programação, pode-se construir aplicações independentes de plataforma e *applets*, que podem ser transferidos automaticamente para uma máquina cliente, sem a necessidade de uma instalação prévia. O Java permite a criação de *interfaces* amigáveis, dinâmicas e com um elevado grau de interatividade com o usuário final. Este fato, somado a característica de que o Java é uma linguagem baseada em objetos, permite que aplicações desenvolvidas, utilizando tal linguagem, possam ser facilmente expansíveis. Pôde-se comprovar estas características, através da implementação do *WebVis* (*software* destinado monitorar o tráfego em *links* TCP/IP), que foi realizada utilizando as ferramentas disponíveis

no ambiente de gerência. Este *software* atualmente pode monitorar a taxa em *bits/s* do tráfego em *links* TCP/IP. No entanto, a sua expansão para monitorar características adicionais (como o tráfego individual por aplicação em um *link* TCP/IP) pôde ser facilmente implementada, graças às características de reusabilidade e expansibilidade oferecidas pelo Java. O *WebVis* é descrito com mais detalhes no próximo capítulo.

As classes implementadas neste trabalho e fornecidas pela *Advent SNMP API* possibilitam um mecanismo simplificado para o gerenciamento de redes, utilizando *applets* e *applications* Java. Esta característica permite a criação de aplicações que podem ser acessadas de qualquer parte do mundo (utilizando a Internet), para monitorar recursos de rede e, se necessário, corrigir as possíveis falhas encontradas.

A utilização de um banco de dados relacional oferece um mecanismo seguro e eficiente para o armazenamento e recuperação de informações de gerência. As classes oferecidas pelo JDBC, aliadas a classe *Banco.java* (implementada neste projeto), oferecem um mecanismo eficiente para a criação de aplicações que acessam uma base de dados relacional, sem a necessidade da instalação adicional de qualquer *software* ou *driver* em máquinas clientes (pois estes podem ser transferidos automaticamente para o *browser Web*). No entanto, após alguns testes práticos, pôde-se observar que as classes do JDBC (devido ao tamanho dos arquivos envolvidos) são um pouco lentas para serem transmitidas para a máquina cliente. Por isto, em redes de baixa velocidade, recomenda-se a utilização de um serviço de *caching WWW*, para agilizar o processo de *download* do *software*, em máquinas clientes. Outro procedimento que pode ser tomado, é a utilização do CORBA a fim de evitar o acesso direto a uma base de dados relacional através de um aplicativo Java.

Todas as ferramentas descritas no *Ambiente de Gerência* formam um precioso instrumento para a construção de aplicações de gerenciamento via *Web*. Os testes realizados com este ambiente vieram a confirmar uma tendência em utilizar a tecnologia *Web* para o gerenciamento de redes de computadores [1] [6] [8] [10] [21], comprovando de modo prático a flexibilidade e eficiência oferecidas por este ambiente.

Pode-se afirmar que os objetivos propostos na introdução deste capítulo foram alcançados. Isto pode ser confirmado no próximo capítulo, onde são apresentadas algumas experiências práticas, utilizando o *Ambiente de Gerência*.

## 5 Aplicações Desenvolvidas utilizando o Ambiente de Gerência

Neste capítulo são apresentadas algumas aplicações desenvolvidas seguindo o modelo definido pelo *Ambiente de Gerência*, citado no capítulo anterior. Tais aplicações caracterizam-se por realizar a obtenção de informações utilizando o protocolo SNMP, armazenar as informações obtidas em uma base de dados relacional e por permitir uma plena integração com a tecnologia *Web*, através da utilização de *applets*.

Este capítulo visa demonstrar e exemplificar o potencial de desenvolvimento de aplicações apresentado pelo *Ambiente de Gerência*, através da implementação dos diversos módulos que compõem um *software* para a análise de tráfego em *links* TCP/IP.

### 5.1 Software para a Análise de Tráfego em Links TCP/IP através de Browsers Web

Os *links* entre instituições, nas redes baseadas em TCP/IP, normalmente são realizados através do aluguel de linhas privativas, junto a uma empresa de telecomunicações. Tais *links* são taxados proporcionalmente à banda de tráfego suportada. Em consequência deste fato, surge a necessidade de se conhecer o comportamento exato do tráfego em um *link* Internet, detectando momentos de ociosidade e sobrecarga. Com estes dados pode-se justificar uma possível expansão ou redução na banda de tráfego suportada para o *link* em questão.

É comum, tanto para os administradores, como para os usuários de uma rede, surgirem questões referentes a um *link* TCP/IP, como: qual o comportamento habitual do tráfego durante determinados períodos de tempo (dia, mês ou hora); quais os momentos de interrupção; qual a taxa de erros; qual a taxa de ocupação; quais são as aplicações e *hosts* que geram mais tráfego; entre outras. As respostas a tais questões podem auxiliar um administrador a diagnosticar problemas com um *link* Internet, ou permitir a um usuário saber quais são os melhores horários para utilizar a rede.

Uma das formas de solucionar as questões citadas anteriormente, seria ativar uma aplicação de gerenciamento e, utilizando o protocolo SNMP, obter as respostas para as questões desejadas. No entanto, esta solução possui como pontos negativos, o fato de que

aplicações de gerência de redes são caras, exigem um grande conhecimento técnico para manuseá-las e não permitem, de modo direto, tornar as informações públicas.

Uma solução mais racional seria obter um *software* facilmente instalável e configurável, que pudesse coletar e armazenar em uma base de dados relacional todas as informações desejadas, disponibilizá-las em um *site Web* e que tivesse a característica de ser independente de plataforma, convivendo assim, de modo harmônico, com o ambiente heterogêneo apresentado pela Internet e por Intranets.

Existem diversos *softwares* já implementados, que cumprem parcialmente os requisitos acima citados (como o *Router-Stats*: <ftp://ftp.scn.de/pub/tools/router-stats>). Tais *softwares*, normalmente, coletam as informações utilizando o SNMP, armazenam estas informações em arquivos seqüenciais, geram gráficos descrevendo o tráfego durante um determinado período de tempo, em arquivos de imagem, e constróem as páginas HTML para apresentação dos gráficos gerados. De modo geral, são implementados usando uma linguagem baseada em *script*, associada a diversos utilitários de domínio público (como o *perl*, *gnuplot*, *giftool*, *netpbm*, *tcl*, etc). Os tópicos observados anteriormente oneram o *software* em características tais como: difícil manutenção, consumo excessivo de espaço em disco e CPU, pouca expansibilidade e interface pobre com o usuário. Além disso, apresentam complicados mecanismos de instalação e não permitem integração com servidores de banco de dados relacionais.

Visando solucionar as principais limitações encontradas nos *softwares* similares, foi desenvolvido o *WebVis*: um *software* independente de plataforma, que pretende atender as principais necessidades dos usuários para a análise de tráfego em *links* TCP/IP (Figura 18).

Neste capítulo, são descritas ferramentas, soluções e protocolos utilizados na construção do *WebVis*. São realizadas uma análises de *softwares* similares, seguidas de uma descrição do estágio de desenvolvimento atual de tal *software* (<http://www.pop-sc.rnp.br/WebVis>), bem como, dos futuros planos de expansão.



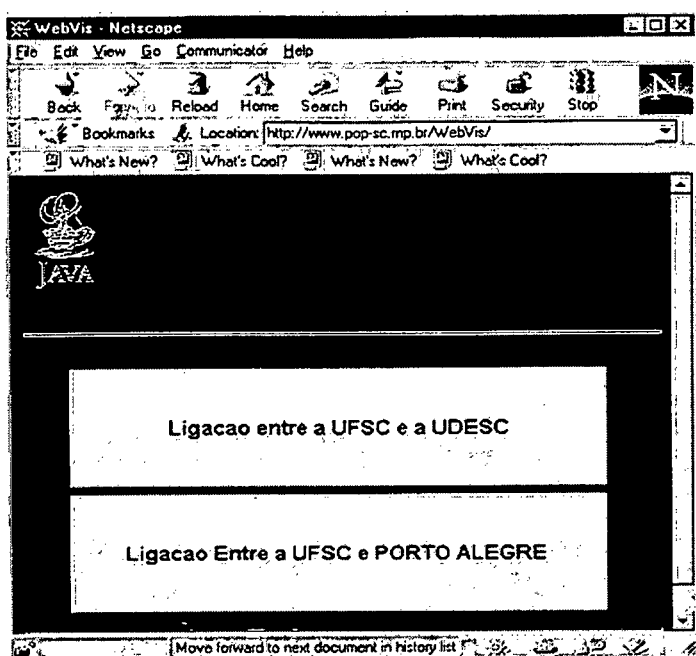


Figura 18 Menu Principal do *WebVis*

### 5.1.1 Modelo Lógico

Para realizar a análise de qualquer tipo de informação referente a *links* TCP/IP, primeiramente deve-se obtê-las. Um dos modos mais práticos de obter tais informações, seria através de um protocolo padronizado, que fosse utilizado pela maioria dos roteadores disponíveis no mercado. Para solucionar tal questão, o protocolo indicado certamente é o SNMP (*Simple Network Management Protocol*), que permite (utilizando as primitivas *get* e *get-next*) obter o valor das variáveis pertencentes a uma MIB (*Management Information Base*) [24] [25], através de consultas a um processo agente (Figura 19).

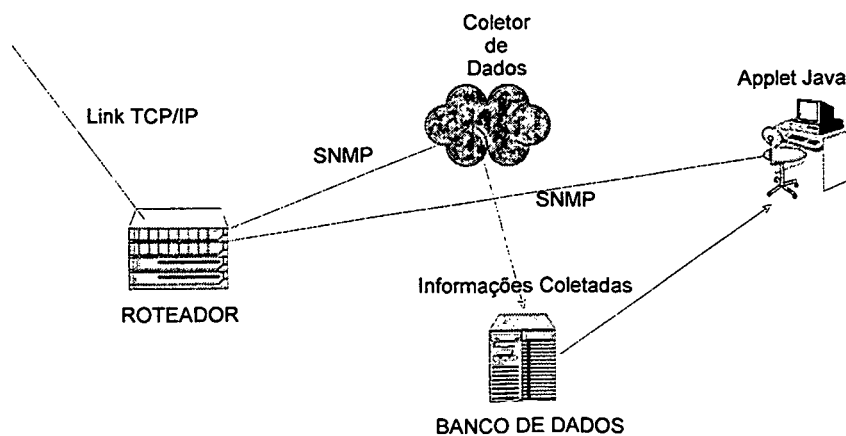


Figura 19 Modelo Lógico Apresentado pelo *WebVis*

Neste trabalho optou-se por utilizar somente MIBs padronizadas, a fim de obter uma solução genérica para o problema em questão. Para a obtenção de informações primordiais, como o tráfego em um *link* TCP/IP, o número de pacotes descartados, entre outras, pode-se utilizar a MIB-II, que é suportada pela maioria dos roteadores disponíveis comercialmente. Para a obtenção de informações mais complexas, pode-se utilizar os padrões RMON e RMON-II [24] [25], que permitem obter detalhes como, por exemplo: quais são as aplicações e *hosts* que geram mais tráfego em uma interface TCP/IP.

Para o armazenamento das informações obtidas através do protocolo SNMP, optou-se pela utilização de um servidor de banco de dados, que permite o armazenamento e recuperação de informações, através de um meio eficiente, flexível e padronizado.

Outro ponto fundamental neste projeto, é o desenvolvimento de programas para recuperar as informações e apresentá-las de uma forma amigável ao usuário, através da utilização de gráficos e tabelas. A recuperação de informações pode ser realizada através de simples consultas SQL ao servidor de banco de dados. Para a geração de gráficos e tabelas, pode-se utilizar classes já implementadas, que ofereçam os recursos desejados, e desenvolver um conjunto de classes abstratas para criar uma interface única com o restante do programa. Isto possibilita a utilização de uma classe com recursos gráficos mais evoluídos (se o usuário desejar), sem que seja necessário modificar o restante do programa.

Um requisito essencial para se adequar com a característica de heterogeneidade da Internet e Intranets, é que todos os programas desenvolvidos neste projeto possam ser utilizados em, praticamente, qualquer plataforma.

Pode-se claramente observar que as soluções oferecidas pelo *Ambiente de Gerência* (descrito no capítulo anterior) viabilizam a implementação do *WebVis*, apresentando soluções para a realização de consultas SNMP, armazenamento de informações em uma base de dados relacional e geração de gráficos em *browsers Web*, através de programas implementados em Java (Figura 19).

### 5.1.2 Estudo dos Softwares Similares ao *WebVis*

Existem diversos aplicativos (alguns de domínio público) para obtenção do tráfego em *links* TCP/IP. Tais *softwares* são extremamente úteis para analisar o comportamento destes *links*, possibilitando visualizar quais os momentos de maior ou menor tráfego, possíveis interrupções na transmissão e identificar o comportamento habitual do tráfego durante um determinado período de tempo.

Na Universidade Federal de Santa Catarina foi instalado o *software Router-Stats* (<http://www.scn.de/~iain/router-stats>). Tal *software* foi implementado através da linguagem *Perl* e utiliza-se de diversas outras ferramentas de domínio público para realizar as tarefas a que se propõem, como: *Gnuplot*, *Expect*, *Tcl*, *Cmu-Snmp*, *Netpbm* e *Giftool*. Todos os *softwares* analisados neste trabalho utilizam a mesma filosofia do *Router-Stats*.

A maioria dos pontos de presença da RNP no Brasil (POPs) possui o *Router-Stats* instalado para obtenção das estatísticas de seus *links* TCP/IP. Esta também é uma prática comum em diversas partes do mundo.

Na UFSC (Universidade Federal de Santa Catarina) o *Router-Stats* foi instalado visando monitorar 28 *links* estaduais (interligando a UFSC à instituições como a UDESC, UNOESC, TVE, FURB, entre outras) e um interestadual (interligando a UFSC à Porto Alegre).

O *Router-stats* realiza a coleta das taxas de entrada e saída do tráfego, em um determinado *link*, utilizando o protocolo SNMP. Para executar tal tarefa, ele necessita que seja habilitado o gerenciamento SNMP nos roteadores e que sejam fornecidos o endereço IP e o número da *interface* do roteador a qual o *link* está conectado. A cada 30 minutos são calculados os tráfegos médios e gerado um gráfico representativo (Figura 20). Diariamente é gerada uma página *Web*, com o calendário de dias em que foram coletadas informações. Com isto, o usuário pode se conectar a um servidor *Web* e visualizar o gráfico representando o tráfego em um *link* TCP/IP, durante um determinado dia. Na Figura 20 é apresentado um gráfico representando o tráfego médio de entrada e saída no *link* que interliga a UFSC a Porto Alegre, no dia primeiro de setembro de 1997.

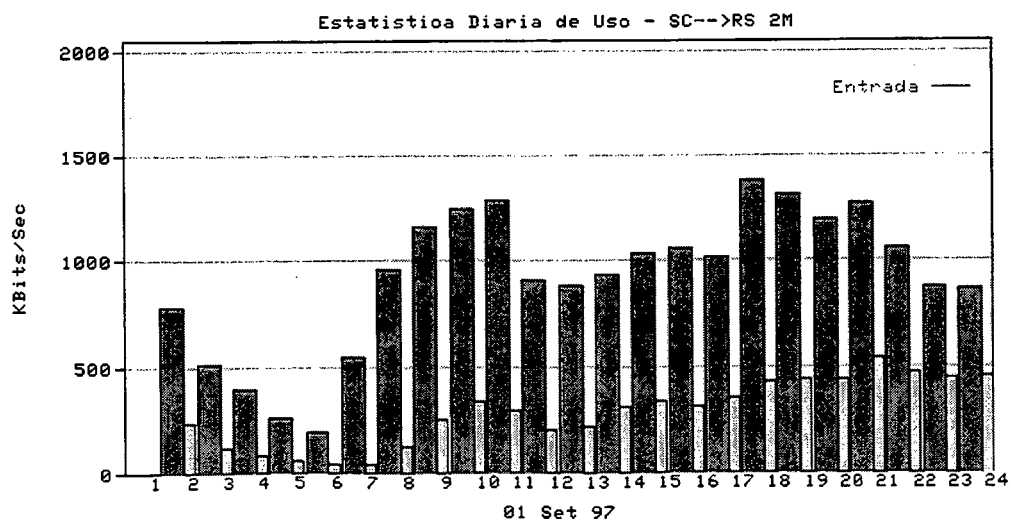


Figura 20 Gráfico Gerado pelo *Router-Stats*

O *software Router-Stats* possui basicamente dois módulos: um para obtenção de informações dos roteadores e outro para geração dos gráficos e páginas *Web*.

O módulo para obtenção de informações dos roteadores utiliza-se de uma MIB proprietária da CISCO (que oferece a taxa do tráfego de entrada ou saída em *bits/s*) ou variáveis da MIB-II (que retornam um contador de octetos que saem e entram em uma determinada interface do roteador, exigindo cálculos adicionais para a transformação destes valores em *bits/s*). Tal *software* foi adaptado para utilizar o utilitário *snmp-info* (*software* nativo do *AIX*) afim de obter o tráfego em *bits/s* dos diversos *links* TCP/IP (não foi utilizado o *Cmu-Snmp*, em virtude da dificuldade em compilar tal *software* para a plataforma desejada, na época da Instalação). Os valores das taxas de entrada e saída do tráfego, em um determinado *link* TCP/IP, são coletados em intervalos de 5 minutos e são armazenados em um arquivo sequencial.

O módulo de geração dos gráficos e páginas HTML realiza as seguintes funções (Figura 21):

- utiliza-se dos valores coletados no módulo de obtenção de dados, para calcular o tráfego médio por hora (ou por dia) em um determinado dia (ou mês);
- de posse dos valores do tráfego médio, é utilizado o utilitário *Gnuplot* para gerar um arquivo em formato *PPM*, contendo um gráfico de barras representando o tráfego médio por hora durante um dia (ou por mês durante o ano) das taxas entrada e saída em um *link* TCP/IP;
- o arquivo em formato *PPM* é escalonado e transformado para o formato *GIF*, utilizando-se do utilitário *Netpbm*;
- o programa *GifTool* muda a cor de fundo do gráfico, visando tornar este fundo igual ao da página *Web*;
- é executado um *script*, que utiliza o utilitário *find* do UNIX, para encontrar todos os arquivos com extensão *.gif*. Após esta operação, são gerados os arquivos HTML com a lista de *links* monitorados e os calendários, para permitir ao usuário identificar o dia em que este deseja visualizar o comportamento do tráfego.

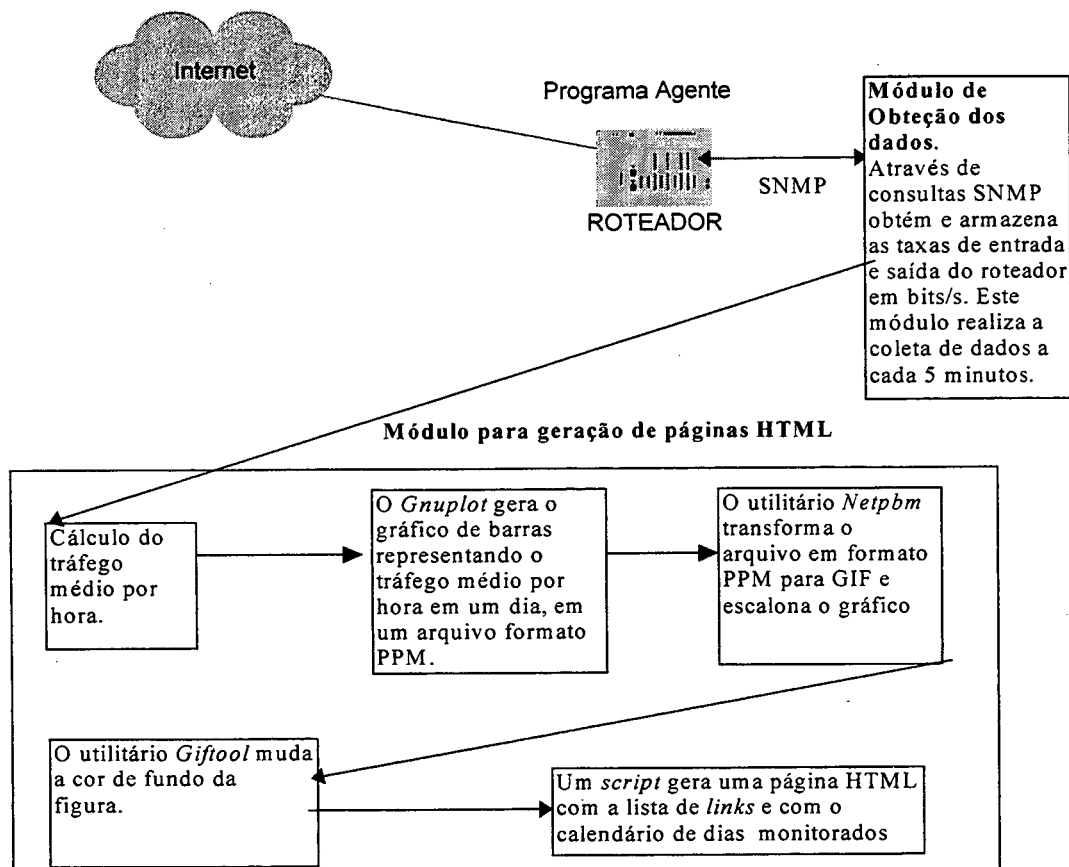


Figura 21 Funcionalidade do *Router-Stats*

### 5.1.3 Problemas encontrados em *Softwares* Similares ao *WebVis*

Todos os *softwares* estudados no presente trabalho, através de uma pesquisa junto a Internet, apresentam características similares ao *Router-Stats*, no sentido de basear-se na geração de arquivos de imagens e páginas HTML estáticas. Em virtude deste fato, pode-se utilizar o *Router-Stats* como modelo para uma análise das características apresentadas por *softwares*, que executam funções semelhantes ao *WebVis*. A análise dos pontos negativos dos *softwares* similares ao *WebVis* é essencial para identificar os problemas apresentados por tais *softwares* e para procurar contorná-los, no projeto do *WebVis*.

Segue abaixo a lista dos principais pontos negativos apresentados pelo *Router-Stats*:

- **dificuldade de instalação:** devido ao grande número de utilitários que devem ser instalados e as personalizações que devem ser realizadas, o usuário deve conhecer

profundamente o funcionamento do programa em questão, muitas vezes, tendo que estudar os códigos fontes e alterar o seu conteúdo. Outro ponto a ser observado, é que utilitários devem ser compilados para a plataforma desejada. Esta é uma tarefa demorada, que exige um minucioso estudo dos documentos de instalação e muita paciência por parte do usuário que a realiza;

- **dependência de plataforma:** todos os utilitários devem ser compilados para a máquina em questão. Caso se deseje portar o *software* para outra plataforma, todos os utilitários devem ser recompilados;
- **carga de processamento exagerada:** para gerar os arquivos *GIF* e *HTML*, são utilizados processos lentos e que consomem muita CPU. Por exemplo: a construção do gráfico exige a geração de um arquivo de imagem, o seu escalonamento, transformação para *GIF* e mudança da cor de fundo. Tudo isto, consome muito tempo de CPU e espaço em memória;
- **não permite a utilização de um banco de dados relacional para armazenar as informações obtidas;**
- **consumo excessivo de espaço em disco:** é gerado um arquivo de imagem, por dia, para cada *link* analisado, acarretando uma enorme quantidade de arquivos armazenados inutilmente;
- **interface pobre e não expansível:** a interface com o usuário é pobre e novos recursos são difíceis de serem adicionados, em virtude da natureza estática de páginas desenvolvidas puramente em *HTML*;
- **dificuldade de configuração:** este *software* utiliza vários arquivos de configuração, que caracterizam-se pela pouca legibilidade e por serem muito pouco intuitivos para a interação com o usuário;
- **dificuldade de manutenção:** este *software*, por ser implementado em *Perl* (que é uma linguagem que caracteriza-se pela pouca legibilidade) e utilizar-se de vários utilitários diferentes, é um programa de difícil manutenção.

#### 5.1.4 Objetivos Traçados para a Implementação do *WebVis*

O objetivo principal, no projeto do *WebVis*, é a construção de um programa implementado em *Java*, que possa obter informações de *links* *TCP/IP* (através de consultas *SNMP*), armazenar estas informações em uma base de dados relacional e visualizar o

comportamento do tráfego no *link* monitorado (através de gráficos e tabelas) em uma *interface Web*. Neste projeto definiu-se que o *WebVis* deve seguir os seguintes princípios:

- **independência de plataforma:** esta característica é obtida em virtude deste *software* ser totalmente implementado em Java;
- **facilidade de uso:** o usuário, através de *applications* ou *applets* Java, pode configurar todos os parâmetros do programa e cadastrar novos *links* TCP/IP, sem a necessidade de editar complicados arquivos de configuração ou conhecer detalhes referentes a implementação do mesmo;
- **legibilidade:** o programa deve ser construído utilizando técnicas de análise baseada em objetos e possuir todos os módulos implementados em Java. Isto permite a construção de um programa cuja manutenção pode ser efetuada de maneira muito simplificada;
- **independência do modelo de roteador utilizado:** o programa deve obter informações referentes a *links* TCP/IP (como taxa de entrada e saída do tráfego em um *link* TCP/IP), através do protocolo SNMP, e utilizar uma MIB padronizada (como a MIB-II). Através destes procedimentos é viabilizado um acesso padronizado às informações, permitindo que, praticamente, qualquer roteador disponível comercialmente possa ser utilizado em tal projeto;
- **armazenamento e recuperação eficiente das informações:** este objetivo pode ser alcançado, através da possibilidade de armazenar as informações em um banco de dados relacional. Isto permite uma recuperação rápida e segura de informações, além de possibilitar a implementação de novas aplicações que utilizem estas informações, de modo fácil, seguro e eficiente;
- **facilidade de instalação e configuração:** para a instalação do *WebVis*, basta descompactá-lo, instalar o JDK 1.0.2 ( ou superior ), configurar a variável *CLASSPATH* e utilizar uma aplicação Java para configurá-lo;
- **interface amigável e expansível:** toda comunicação com o usuário é feita através de uma *interface* desenvolvida em Java. Isto permite a construção de interfaces bem elaboradas, com um alto grau de interação com o usuário e dotadas de uma grande capacidade de expansão (ao contrário das estruturas convencionais baseadas meramente em HTML e CGI).

### 5.1.5 Implementação

O *WebVis* é dividido em três módulos: Módulo de Configuração, Módulo de Obtenção de Informações e Módulo para Visualização de Informações em *Browsers Web*.

#### 5.1.5.1 Módulo de Configuração

O Módulo de Configuração é responsável pela inserção, remoção e modificação das informações de configuração do *WebVis*.

Após a instalação do *WebVis*, o usuário deve utilizar um aplicativo Java para configurar todos os parâmetros necessários ao correto funcionamento deste *software*. Pode-se citar como exemplo de tais parâmetros: o endereço do servidor do banco de dados, a comunidade SNMP utilizada para a obtenção dos dados, o diretório base de instalação, entre outros. Todas estas informações são manipuladas pela classe *Config*. Esta classe tem a função de armazenar e fornecer todas as informações de configuração necessárias ao normal funcionamento das demais classes do *WebVis*. Com isto, para modificar uma informação que será utilizada de modo global por todos os módulos do *WebVis*, basta acessar a classe *Config* e, através de seus métodos, realizar as operações desejadas. Este procedimento facilita a construção de um sistema de configuração global para o *WebVis*, que pode ser realizado através da utilização de uma única classe.

Para a manipulação de informações referentes a *links* TCP/IP foram criadas duas classes: *Linha.class* e *CadastroLinhas.class*. O código fonte destas classes pode ser encontrados no *Anexo B*, deste trabalho.

A classe *Linha.class* tem por objetivo manipular as informações referentes a um determinado *link* TCP/IP. Os principais atributos desta classe são: o número da interface e o endereço do roteador ao qual o *link* esta conectado, a velocidade da linha (64kbps, 2Mbps, entre outras), o texto identificando o *link*, entre outros.

Para o cadastro de *links* TCP/IP foi criada uma classe que representa um conjunto de *links* denominada: *CadastroLinhas.class*. Qualquer informação que necessite ser adicionada ou removida, referente a *links* TCP/IP, deve utilizar esta classe. O objeto *CadastroLinhas* é consultado por todos os módulos do *WebVis* para manipulação das informações referentes a *links* TCP/IP. Por isto, através desta classe, pode-se facilmente construir um aplicativo para a inserção, remoção e modificação de *links* TCP/IP, que pode ser utilizado de modo global pelo *WebVis*, através da consulta a uma única classe (*CadastroLinhas.class*).



### 5.1.5.2 Módulo para Obtenção de Informações

Para a obtenção das informações referentes a *links* TCP/IP, o *WebVis* utiliza o protocolo SNMP. A fim de possibilitar a utilização deste *software* para obtenção de informações referentes a, praticamente, qualquer tipo de roteador, neste projeto são manipuladas apenas MIBs padronizadas. Para a obtenção de informações de tráfego em uma *interface*, o *WebVis* utiliza as variáveis *ifInOctets* e *ifOutOctets* definidas na MIB-II [17]. O Módulo para Obtenção de Informações consiste em um conjunto de aplicações Java que realizam consultas SNMP para os roteadores onde estão conectados os *links* TCP/IP (cadastrados no módulo de configuração), tratando as informações obtidas e armazenando-as em uma base de dados relacional.

Para obter informações referentes a um objeto gerenciável, foi criada uma classe (*SnmMan.java*), derivada das classes da *Advent SNMP API*, que permite enviar mensagens de *get* para o agente e obter as informações desejadas, como definido no *Ambiente de Gerência* descrito no capítulo anterior.

A comunicação com o banco de dados é realizada através da classe *Banco.class*, que consiste em uma classe, implementada em Java, que utiliza o JDBC para realizar a comunicação com o banco de dados (como definido no capítulo anterior).

Para a obtenção e armazenamento de informações de tráfego referentes a *links* TCP/IP, foi implementada uma aplicação Java denominada: *ObtemInf*. Esta aplicação consiste em um *daemon* que, em intervalos regulares (no protótipo em desenvolvimento este intervalo é de cinco minutos), realiza a coleta de informações de tráfego em todos *links* TCP/IP cadastrados, transforma os dados obtidos para *bits/s* e armazena estas informações em uma base de dados relacional (Figura 22).

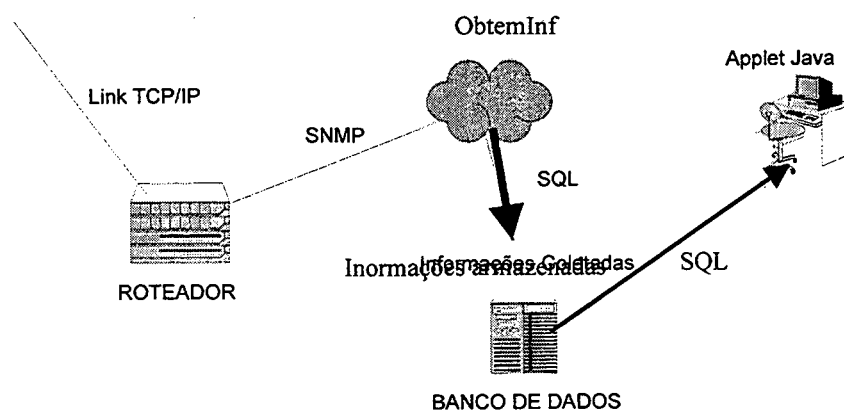


Figura 22 Obtenção e Armazenamento de Informações do Tráfego

### 5.1.5.3 Módulo para Visualização de Informações em Browsers Web

Para a visualização de informações, foi criado um *applet* denominado *WebVis*. Este *applet* representa a aplicação que efetivamente realiza a interface com o usuário, permitindo que este selecione o *link* TCP/IP e o dia em que pretende fazer as análises de tráfego. Após a seleção do *link* e dia, o *WebVis* apresenta ao usuário um gráfico representando o tráfego médio de entrada e saída neste *link* TCP/IP, no dia selecionado.

O *WebVis* utiliza a classe *CadastroLinhas.class* para obter as informações referentes aos *links* TCP/IP. Um vez selecionado o *link* que o usuário deseja monitorar, é apresentado um calendário na qual é possível realizar a escolha do dia, mês e ano para realização da análise de tráfego. Para efetuar esta operação foi criada a classe *Calendario.class*, que consiste em uma classe genérica para a implementação de calendários (Figura 23).

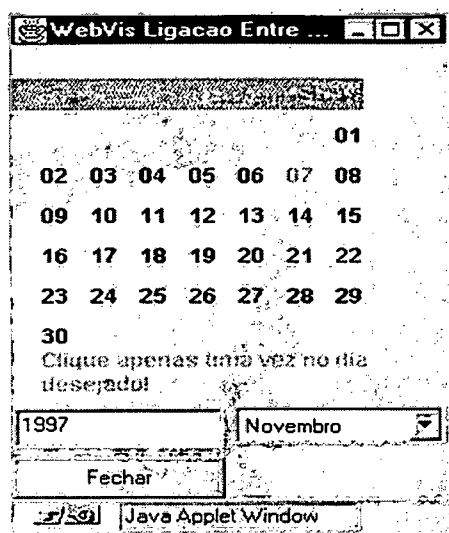


Figura 23 Gráfico gerado pela Classe *Calendario.class*

Para a geração dos gráficos, foi criada uma classe *Grafico.class*, como definida no *Ambiente de Gerência* descrito no capítulo anterior (Figura 24). Esta classe permite que sejam gerados gráficos de barras, representado o tráfego médio por hora durante um determinado dia.

Para obter a informação do tráfego médio por hora em um determinado dia, existem duas possibilidades:

- É possível efetuar uma consulta ao banco de dados diretamente do *applet*. Este processo, no entanto, gera consultas desnecessárias ao banco e exige o *download* de classes adicionais para manipular o banco, no *browser Web*.
- Outra possibilidade é utilizar uma aplicação que, em períodos regulares de tempo, consulte o banco e gere as médias, em arquivo texto, para que os *applets* possam consultar e gerar os gráficos desejados. Esta solução é mais racional que a anterior, visto que as médias só necessitam ser geradas uma vez por um determinado período de tempo (hora, dia ou mês) e o espaço ocupado pelos arquivos textos é desprezível.

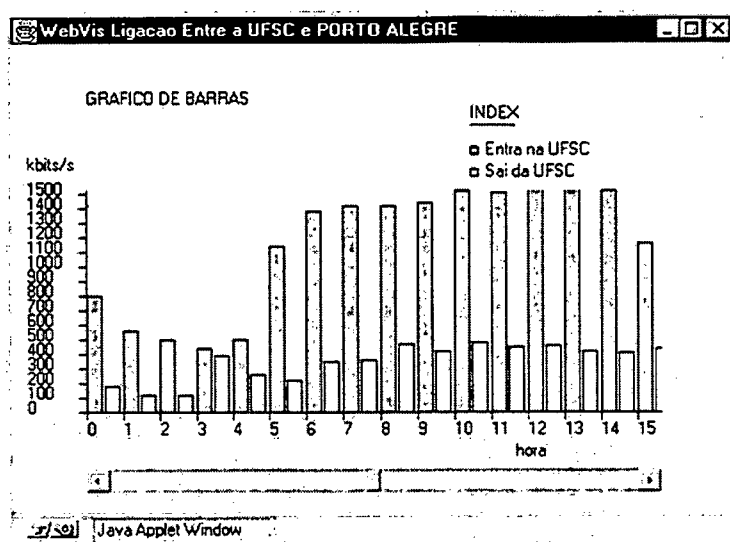


Figura 24 Gráfico Representando o Tráfego Médio por Hora em um Determinado Dia

### 5.1.6 Conclusão

Após os primeiros testes, o *WebVis* provou ser uma alternativa muito boa para o administrador que deseja disponibilizar as estatísticas de tráfego em links TCP/IP, através de um *software* facilmente instalável e configurável.

A instalação do *WebVis* restringe-se, basicamente, a quatro passos e exige somente a utilização do JDK (*Java Development Kit*) como ferramenta adicional ao sistema. Isto torna o processo de instalação extremamente simples, quando comparado a *softwares* similares (como o *Router-Stats*), não exigindo do usuário qualquer conhecimento adicional sobre as formas de implementação do *WebVis* ou de sua interação com o sistema operacional.

Como ponto negativo no processo de instalação, pode-se indicar a dificuldade de instalação e configuração do sistema de banco de dados relacional (que pode ser um tarefa complexa para um usuário leigo em tal área). Para solucionar este problema, está sendo desenvolvida uma classe para manipular arquivos seqüenciais, que possui praticamente a mesma funcionalidade da classe que interage com o banco de dados. Isto, somado ao fato que a comunicação com o banco de dados é realizada através de uma classe abstrata, permite que o usuário possa optar pela sua utilização ou não. Se o objetivo primordial do usuário for a facilidade de instalação em detrimento da segurança, eficiência e reusabilidade que o banco de dados oferece, ele pode optar pela utilização de arquivos sequencias.

O processo de Configuração do *WebVis* é sempre efetuado por aplicações Java, sem a necessidade de editar arquivos manualmente. Com isto, o usuário pode configurar o *WebVis*

através de uma interface amigável e intuitiva, sem a necessidade de saber em quais arquivos ele armazena estes dados.

O *WebVis* foi implementado utilizando a análise e programação baseadas em objetos, conferindo a este *software* características, como legibilidade e expansibilidade que são essenciais para a sua evolução.

O *WebVis* não é uma ferramenta acabada, mas sim um *software* que está em contínua evolução. Após encerrada a fase de testes e documentação, pretende-se disponibilizá-lo para o domínio público.

Nas próximas etapas do projeto, pretende-se utilizar os padrões RMON e RMON-II para obtenção de estatísticas mais detalhadas de tráfego e desenvolver construção de agentes distribuídos utilizando o Java e o CORBA, objetivando transformar o *WebVis* em um sistema distribuído para coleta, armazenamento e visualização de informações em *links* TCP/IP, utilizando *Browsers Web*.

## 6 Sistema de Gerência Distribuída

Neste capítulo é descrito o desenvolvimento de um sistema de gerência distribuída para um *Cluster* de estações de trabalho, que utiliza o CORBA, o WWW, o Java e o SNMP como instrumentos para a sua implementação. Para este sistema foram implementados agentes, objetos gerenciados e uma aplicação que permitem o gerenciamento distribuído do *Cluster*.

As atividades descritas nos capítulos anteriores formam a base para a construção do sistema de gerência do *Cluster*. Com a definição do *Ambiente de Gerência* (Capítulo 4) e com a implementação do *WebVis* (Capítulo 5) foram adquiridos os conhecimentos práticos e teóricos necessários à realização de tarefas como: o armazenamento de informações de contabilização em uma base de dados relacional e a utilização do protocolo SNMP através de aplicativos Java. Tais atividades foram essenciais para o desenvolvimento do agente SNMP, da aplicação de gerência e do agente de contabilização que são descritos neste capítulo. Os estudos teóricos relacionados com o CORBA e a utilização de ferramentas, como o JORB, formam os pré-requisitos necessários à implementação dos objetos gerenciados envolvidos na implementação do sistema de gerência descrito neste capítulo. A união destes conhecimentos viabilizou a utilização conjunta do CORBA, SNMP, WWW e Java para o desenvolvimento de um sistema distribuído multiplataforma, destinado ao gerenciamento de um *cluster* de estações.

No início deste capítulo é descrita a estrutura lógica do sistema de gerência do *cluster*. Nesta descrição é apresentada uma visão geral deste sistema e é abordado, de modo superficial, o papel dos agentes, dos objetos gerenciados e da aplicação de gerência, na composição do sistema, como um todo.

A seguir, é descrito o desenvolvimento dos objetos gerenciados que compõem o sistema de gerência. Nesta fase, são apresentados os detalhes de implementação e a funcionalidade dos objetos gerenciados *Computer* e *Cluster*. Tais objetos são instanciados utilizando-se o CORBA e destinam-se ao gerenciamento das estações pertencentes ao *cluster*.

Após a definição dos objetos gerenciados, é descrita a implementação de uma aplicação de gerência que realiza a comunicação com estes objetos, através do protocolo IIOP. Esta aplicação foi implementada como um *applet* Java e permite a obtenção de

informações sobre o sistema e a taxa de ocupação da CPU de qualquer máquina pertencente ao *cluster*.

A seguir, é descrita a implementação de um agente que permite a interação entre os protocolos SNMP e IIOP/CORBA. Com a utilização deste agente, uma aplicação de gerência SNMP pode acessar um objeto gerenciado, instanciado através do protocolo IIOP/CORBA.

Ao final deste capítulo, é apresentada a implementação de um agente de contabilização. Este agente instancia um objeto gerenciado e, periodicamente, realiza o armazenamento de informações de gerência, em uma base de dados relacional.

## 6.1 Estrutura Lógica

Para o gerenciamento do *cluster*, deve-se definir primeiramente um mecanismo que permita a obtenção e modificação de parâmetros de gerência. Para tanto, foi definido um objeto denominado *Computer* que é instanciado em cada máquina do *cluster* utilizando o protocolo IIOP/CORBA. Como o *cluster* caracteriza-se por ser um ambiente heterogêneo, em que as máquinas podem possuir diferentes *hardwares* e sistemas operacionais, um dos objetivos primordiais deste projeto é definir uma solução independente de plataforma para o ambiente. Em consequência deste fato, todas as aplicações e o ORB, utilizado como mecanismo de comunicação entre os objetos, foram implementados em Java.

Com a finalidade de permitir o gerenciamento de parâmetros relacionados ao conjunto de estações do *cluster*, foi definido um objeto denominado *Cluster*. Este, por sua vez, instancia os objetos *Computer* em cada máquina do *cluster*, permitindo o gerenciamento de todas as máquinas que compõem este ambiente, através do envio de mensagens a um único objeto. Utilizando este mecanismo, um administrador pode obter e modificar informações em todas as máquinas do *cluster*, emitindo uma única mensagem ao objeto *Cluster*. É possível também, construir *applets* que, utilizando o protocolo IIOP, podem instanciar o objeto *Cluster* e obter informações como: quais as máquinas com menor ocupação de CPU dentro do *cluster*. Tais informações podem ser extremamente úteis para um pesquisador decidir em quais máquinas executar os seus programas paralelos ou distribuídos.

Para a execução de tarefas dinâmicas, como o armazenamento de informações de *logs* ou o acompanhamento do *status* das máquinas, foi criado um agente implementado em Java que instancia o objeto *Cluster*. Este agente monitora constantemente o conteúdo de determinados atributos do objeto *Cluster* e, dependendo deste conteúdo, executa determinadas

ações como, por exemplo, iniciar o armazenamento periódico da taxa de ociosidade de CPU das máquinas do *Cluster*, em uma base de dados relacional, permitindo um estudo detalhado da efetiva utilização destas máquinas.

As aplicações de gerência são implementadas como *applets* que utilizam o protocolo IIOP/CORBA para instanciar o objeto *Cluster* e emitir mensagens para este. O usuário pode acessar as aplicações de gerência, independente da localização do seu equipamento ou da sua plataforma, sendo necessário apenas, que este equipamento esteja conectado à rede e possua um *browser Web* com suporte ao Java instalado.

Afim de permitir a integração com outras aplicações de gerência, foi implementado um agente SNMP que age como um *gateway* entre os protocolos SNMP e IIOP. Este agente permite a integração deste ambiente com outras aplicações de gerência.

Com a implementação dos objetos, agentes e aplicações de gerenciamento, pôde-se criar um sistema de gerência distribuída em que um usuário, a partir de um *browser Web* ou de uma aplicação de gerenciamento SNMP, pode obter ou modificar uma informação que esteja relacionada com todas as máquinas do *cluster*, através do envio de uma única mensagem para o processo-agente ou para o objeto *Cluster* (Figura 25).

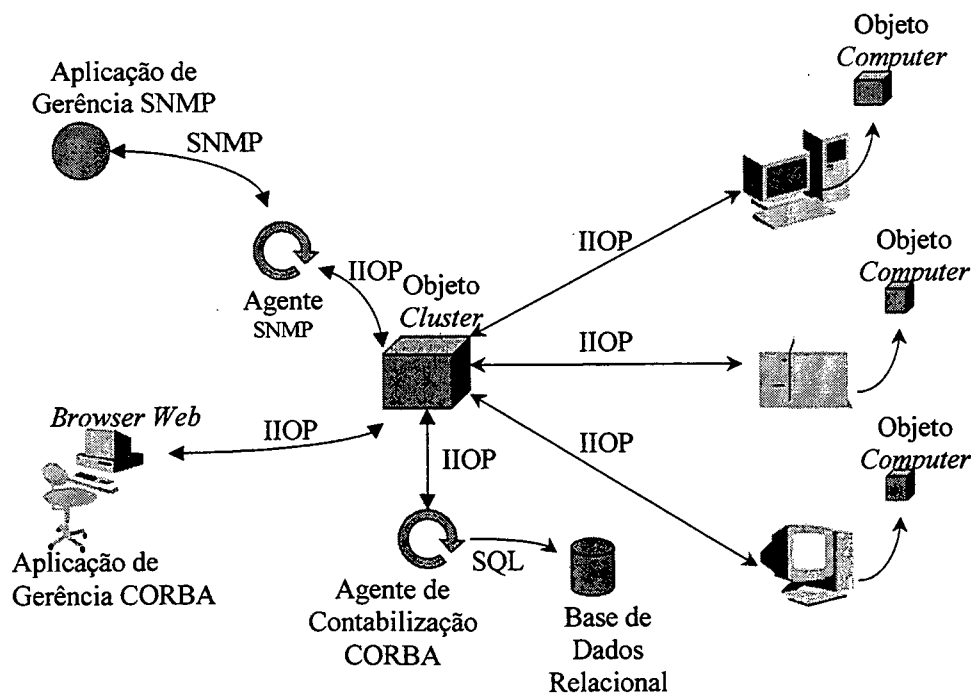


Figura 25 Estrutura Lógica do Sistema de Gerência



## 6.2 Objetos Gerenciados Distribuídos

Os objetos gerenciados, definidos neste projeto, são instanciados utilizando o protocolo IIOP/CORBA e podem ser consultados, utilizando-se os protocolos SNMP ou IIOP.

Para o gerenciamento básico das estações de trabalho pertencentes ao *cluster*, definiu-se o objeto *Computer*. Este objeto pode ser instanciado em cada uma das máquinas pertencentes a este ambiente e é responsável por receber solicitações provenientes do objeto *Cluster*, realizar o processamento desejado e emitir as respostas requeridas. Para o gerenciamento global do *cluster* foi definido o objeto *Cluster*, que é responsável por realizar a interface primária da aplicação gerente com os objetos gerenciados. O objeto *Cluster* instancia o objeto *Computer* em cada máquina do *cluster* e é responsável por receber mensagens provenientes da aplicação gerente, repassá-las para as máquinas desejadas, obter os resultados, processá-los e emitir as respostas encontradas.

### 6.2.1 Objeto Computer

O objeto *Computer* age como um componente intermediário entre o usuário e o sistema operacional da máquina, a qual irá receber solitações, emitir comandos para o sistema e retornar informações sobre a máquina em que está instanciado. Este objeto é implementado em Java, instanciado através do protocolo IIOP/CORBA e é capaz de executar, praticamente, qualquer comando que um usuário poderia emitir através de um *shell Unix*. Utilizando-se deste recurso, é possível obter informações ou executar qualquer comando em uma máquina, através de uma simples invocação de método para o objeto *Computer*.

Como pode ser visto na *Figura 26*, a operação básica do objeto *Computer* resume-se a:

1. receber uma invocação de método através do protocolo IIOP;
2. enviar uma requisição ao sistema na forma de um comando de *shell UNIX*, utilizando-se da classe *Runtime* do Java;
3. redirecionar os resultados obtidos para um arquivo temporário;
4. ler os resultados de um arquivo temporário e processá-los;
5. enviar os resultados encontrados para o cliente que requisitou as informações.

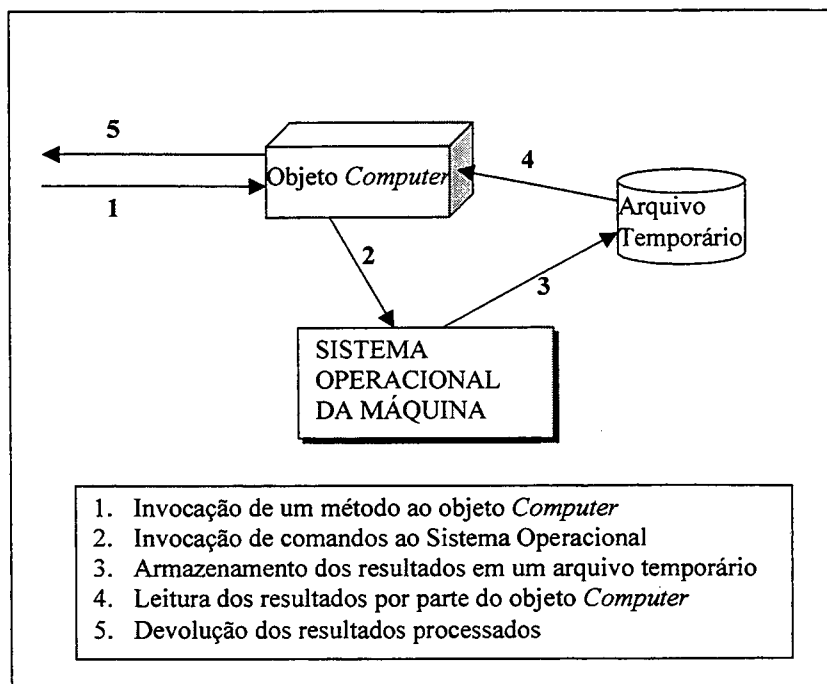


Figura 26 Operação Básica do Objeto *Computer*

### 6.2.1.1 Interface com o Sistema Operacional

O objeto *Computer* utiliza-se da classe *Runtime* do Java para a execução de comandos do sistema operacional. O Quadro 8 apresenta a chamada de execução do programa *obtemCPU*. Este programa consiste em um script UNIX que obtém a taxa de ociosidade da CPU e armazena esta informação em um arquivo temporário. Neste caso, é utilizada a classe *Process*, do Java, para realizar a efetiva execução do programa *obtemcpu*.

```
Process p = Runtime.getRuntime().exec("/u/load/agente/bojangles/agente/obtemcpu 1 1");
```

Quadro 8 Execução de um Comando do Sistema a partir de uma Aplicação Java

Para este trabalho foram empregadas estações dotadas de sistema operacional UNIX. Dentro deste contexto, para a utilização de programas cuja execução é permitida a qualquer usuário, basta invocar o método *getRuntime* do objeto *Runtime* e inicializar um *shell* que execute o comando desejado, redirecionando os resultados para um arquivo qualquer. No entanto, para execução de programas cuja utilização é restrita a usuários administrativos, optou-se pela criação de programas que possam enviar uma requisição de *setuid(0)* ao sistema operacional, visando a troca do proprietário deste processo para o *root*. Após a finalização

desta operação, o programa pode criar um novo processo e executar o comando do sistema cuja a execução é restrita a um usuário administrativo. No Quadro 9 é apresentado um programa, implementado em C, que é utilizado para a execução do utilitário *sar*. Este utilitário, integra o sistema operacional UNIX da IBM (AIX), é responsável por obter informações sobre a taxa de ociosidade da CPU de uma máquina e a sua execução é restrita ao *super-usuário*. Em consequência deste fato, o programa apresentado no Quadro 9 muda o proprietário do processo para o *root* (*setuid(0)*), cria um novo processo (*fork()*) e executa o comando desejado (invocando a função *execl*). Este programa foi utilizado neste projeto para permitir que um usuário comum do sistema possa obter a taxa de ociosidade de CPU de uma máquina.

A utilização de programas implementados em C para execução de utilitários do sistema operacional, além de permitir um acesso a aplicativos restritos a um usuário administrativo, permite também a criação de uma interface única para o acesso a utilitários que não sejam padrões ao sistema operacional UNIX. Pode-se exemplificar este fato, com a utilização do utilitário *sar*. Neste caso, pode-se criar um programa que invoque o utilitário *sar* e cujo executável seja nomeado *obtemcpu*. Caso seja necessário portar esta função para outra plataforma, basta criar um programa que invoque um utilitário similar ao *sar* e cujo executável apresente o mesmo nome que o anterior (*obtemcpu*). Através deste mecanismo, pode-se obter informações referentes a CPU, de máquinas que possuem diferentes plataformas, sem a necessidade de modificar a implementação do objeto *Computer*.

```
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char **argv)
{
    if (argc==1)
    {
        int ret;
        ret = setuid(0);
        ret=fork();
        if(ret==0)
            execl("/bin/sh", "sh", "-c", "/usr/sbin/sar -u 1 1 >
/tmp/sarT", NULL);
        else
            wait(0);
    }
    else
        printf("\nNumero de parametros invalidos! \n\n");
}
```

Quadro 9 Execução de um Comando do Sistema a partir de um Programa implementado em C

### 6.2.1.2 Implementação do Objeto *Computer*

Como já foi informado anteriormente, o objeto *Computer* tem como função básica: receber uma invocação de método (através do protocolo IIOP), requisitar as informações ao sistema operacional, recuperar os resultados a partir de um arquivo temporário, processá-los e enviá-los para o cliente solicitante. Para implementar esta funcionalidade, foi desenvolvido um protótipo do objeto *Computer* capaz de obter a taxa de ociosidade da CPU de uma máquina, gerenciar um sistema de contabilização e retornar informações sobre o sistema operacional de tal máquina. A implementação deste protótipo foi realizada utilizando-se o Java e pode ser facilmente expandida para atender a novas funções de gerência, uma vez que a funcionalidade básica para a emissão de comandos ao sistema operacional é sempre a mesma.

Para a construção do objeto *Computer*, foi definida primeiramente a interface deste objeto, utilizando-se a declaração *interface* do Java (Quadro 10). A partir desta definição, foram gerados Stubs e Skeletons, através da utilização dos utilitários *RunStubEmitter* e *RunSkeletonEmitter*, pertencentes ao *Bojangles* (Capítulo 3).

```
package agente;

import java.lang.Object;

public interface Computer {

    public boolean getStatusContab();
    public void startContab();
    public void stopContab();
    public boolean estaAtivo(boolean remoto);
    public String GetCPU();
    public String getSystemName();

}
```

Quadro 10 Interface do Objeto *Computer*

Como pode ser visto no Quadro 10, o objeto *Computer* apresenta os seguintes métodos públicos:

- *getStatusContab*: identifica se a contabilização de recursos de CPU está ativa. Esta informação é utilizada pelo agente de contabilização para verificar se deve, ou não, realizar as tarefas de contabilização;
- *startContab* inicializa a contabilização de recursos de CPU;
- *stopContab* finaliza a contabilização de recursos de CPU;
- *estaAtivo*: verifica se o objeto está instanciado, ou não;
- *getCPU*: obtém a taxa de ociosidade de CPU da máquina em que está instanciado;
- *getSystemName*: obtém as seguintes informações referentes ao sistema operacional da máquina: nome do *host*, número identificador da máquina e a versão do sistema operacional.

A definição da interface do objeto *Computer* é localizada no arquivo *Computer.java* (Quadro 10) e a sua implementação no arquivo *ComputerImpl.java*. O conteúdo destes arquivos é apresentado no *Anexo C* deste trabalho.

Após a construção da interface e implementação, pode-se construir um objeto-cliente para instanciar o objeto *Computer* na máquina desejada e realizar a invocação de métodos para este (Figura 27).

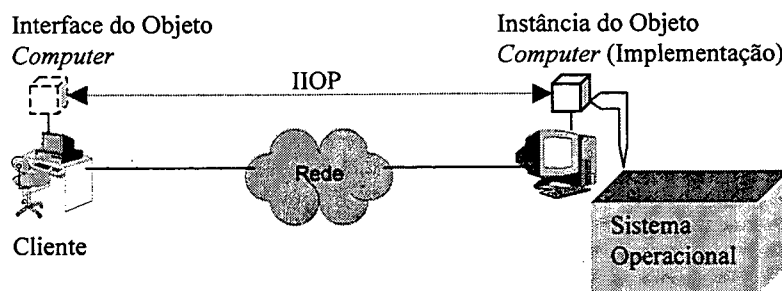


Figura 27 Instanciação do Objeto *Computer*

## 6.2.2 Objeto Cluster

O objeto *Cluster* realiza a interface primária com a aplicação gerente. Através deste objeto, são efetuadas todas as operações necessárias a esta aplicação. O objeto referido instancia o objeto *Computer* em cada uma das máquinas do *cluster* e, através da invocação de métodos, realiza as operações de gerência desejadas. Uma aplicação-gerente pode instanciar o objeto *Computer* diretamente em uma máquina e realizar as operações de gerência desejadas. No entanto, *applets* executados em *browsers Web*, só podem realizar a abertura de *sockets*, com a máquina que possui o servidor Web. Para solucionar esta limitação, pode-se instanciar

o objeto *Cluster* na mesma máquina que executa o servidor Web e, através da invocações de métodos deste objeto, realizar o gerenciamento de todo o *Cluster* (Figura 28).

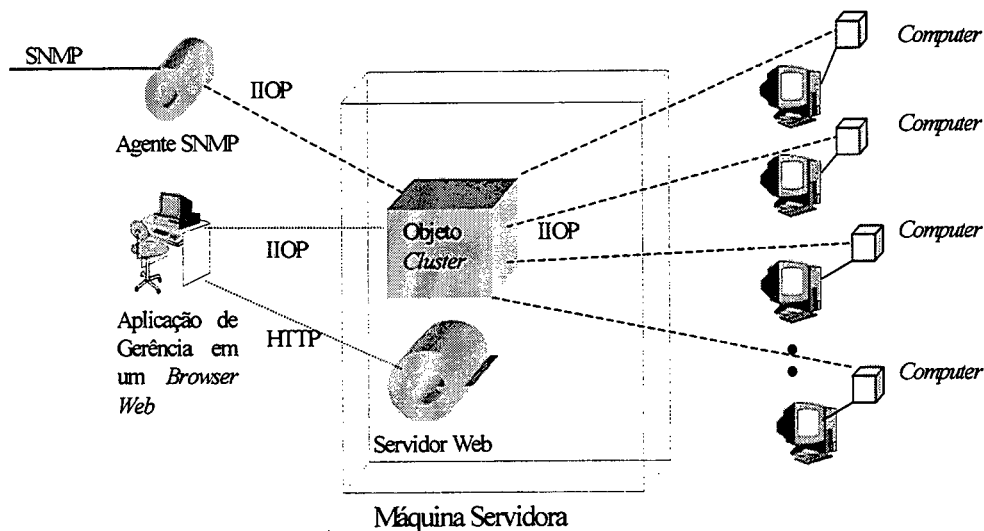


Figura 28 Objeto *Cluster*

Ao receber uma invocação de método, o objeto *Cluster* pode enviar mensagens para todas as máquinas do *cluster*, receber as respostas, processá-las e retornar os resultados encontrados. Por este motivo, para operações globais ou parciais ao *cluster*, ocorre uma diminuição do tráfego gerado e da carga de processamento relativa a aplicação-gerente. Pode-se exemplificar este fato na implementação de uma função de gerência para determinar as máquinas com as CPUs menos utilizadas no *cluster*. Neste caso, a aplicação-gerente pode enviar uma única mensagem requisitando a informação desejada. O objeto *Cluster*, ao receber a invocação de um método, envia uma mensagem para cada máquina, requisitando a taxa de ocupação de CPU. Uma vez de posse das respostas, o objeto *Cluster* processa os resultados e envia para o gerente uma lista das estações mais ociosas dentro do *cluster*. Neste caso, há uma diminuição da complexidade das operações relativas à aplicação-gerente, uma vez que esta envia apenas uma única mensagem, sem a necessidade de realizar qualquer processamento adicional, para determinar as máquinas mais ociosas. Esta característica é fundamental para implementação de aplicações de gerência, utilizando-se *browsers Web*. Sem o uso do objeto *Cluster*, a aplicação-gerente teria que enviar uma mensagem para cada máquina do *cluster*, receber as respostas e processar os resultados. Estes fatos oneram em uma maior complexidade das operações da aplicação-gerente e no aumento o tráfego na rede.

Um aspecto que deve ser observado, é que as estações do *cluster* são interligadas por uma rede de alta velocidade (ATM 155 Mb/s). Por outro lado, aplicações-gerentes, em

*browsers Web*, podem se comunicar com o *cluster* através de redes de baixa velocidade. Por este motivo, o tráfego de gerência dentro do *cluster* pode ser considerado desprezível, enquanto que a diminuição do tráfego entre o *browser* e o objeto *cluster* é de vital importância para a gerência Web deste ambiente. Na Figura 29 pode-se observar que, utilizando-se o objeto *Cluster*, é gerada uma única mensagem entre o *browser* e o objeto gerenciado. Sem a utilização deste objeto, é gerada uma mensagem para cada máquina, o que aumenta em muito o tráfego entre o *browser* e os objetos gerenciados. Supondo que se deseje gerenciar um *cluster* com  $N$  estações de trabalho, em que seja necessário obter informações globais ao ambiente e verificar o tráfego gerado entre o gerente e o objeto gerenciado. Neste caso, utilizando-se o método de gerência SNMP tradicional, seriam enviadas  $N$  mensagens e recebidas  $N$  respostas. Através da utilização do objeto *Cluster* é necessário enviar apenas uma mensagem do gerente para o objeto gerenciado. Neste caso, é gerada apenas uma mensagem de resposta pelo objeto *Cluster*.

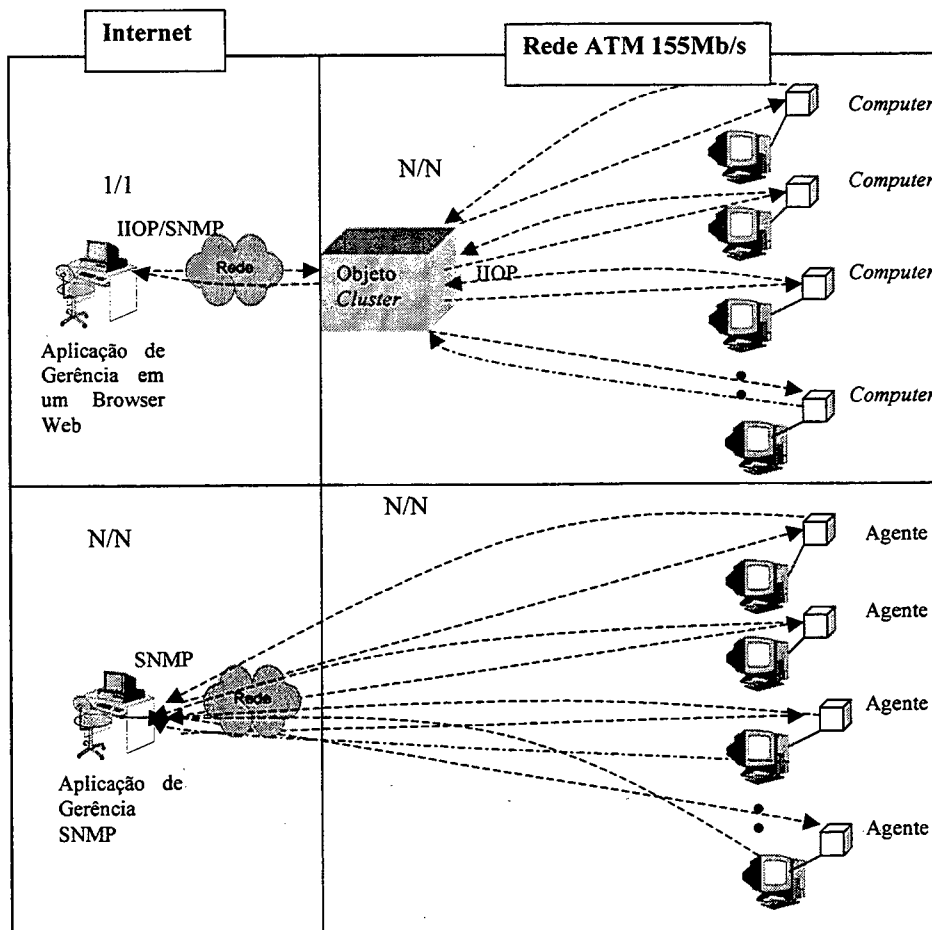


Figura 29 Tráfego entre o Gerente e o Objeto Gerenciado

### 6.2.2.1 Implementação do Objeto *Cluster*

O objeto *Cluster* tem a função de realizar a interface com as aplicações de gerência, permitindo o gerenciamento do *cluster* como um todo ou de estações individuais pertencentes a este ambiente. Afim de demonstrar a funcionalidade básica do objeto *Cluster*, foi desenvolvido um protótipo que permite gerenciar a contabilização de recursos, assim como, obter informações referentes ao sistema e CPUs das máquinas pertencentes ao ambiente. O objeto *Cluster* permite obter informações referentes a uma máquina individual ou do *cluster* como um todo, através de uma simples invocação de método. Este objeto mantém uma lista dos *hosts* pertencentes ao *Cluster* e obtém as informações de gerência através da invocação de métodos ao objeto *Computer*, instanciado nas diversas máquinas do *cluster*.

O objeto *Cluster* foi implementado em Java e, através de sua interface (Quadro 11), foram gerados os *Stubs* e *Skeletons*, a fim de permitir que este objeto possa ser instanciado por aplicações de gerência, utilizando o protocolo IIOP. A interface deste objeto é apresentada no Quadro 11 e a sua implementação consta no *Anexo C* deste trabalho de dissertação.

```
package agente;

import java.lang.Object;

public interface Cluster {

    public boolean getStatusContab();
    public void startContab();
    public void stopContab();

    public String getCPUStation(int Station);
    public String getSystemInfoStation(int Station);

    public String getClusterCPU();
    public String getClusterHostnames();
    public String getClusterSystemInfo();

}
```

Quadro 11 Interface do Objeto *Cluster*

No protótipo implementado, o objeto *Cluster* apresenta os seguintes métodos públicos:



- *getStatusContab*: retorna *true*, caso esteja ativa a contabilização de recursos de CPU. Este método é utilizado pelo agente de contabilização para verificar se deve ou não efetuar a contabilização de recursos de CPU, em uma base de dados relacional;
- *startContab*: inicia o contabilização de recursos de CPU;
- *stopContab*: finaliza a contabilização de recursos de CPU;
- *getCPUStation*: obtém a taxa de ociosidade de uma estação individual do *cluster*;
- *getSystemInfoStation*: obtém informações sobre o sistema de uma máquina individual no *cluster*;
- *getClusterCPU*: obtém um *string* contendo a taxa de ociosidade de todas as máquinas do *cluster* separadas por “;”;
- *getClusterHostnames*: obtém um *string* com a lista de nomes de todas as máquinas do *cluster* separados por “;”;
- *getClusterSystemInfo*: obtém informações sobre o sistema de todas as máquinas do *cluster*.

### 6.3 Aplicação de Gerência CORBA

Para o gerenciamento das estações de trabalho pertencentes ao *cluster*, foi desenvolvida uma aplicação que pode ser distribuída para diversas máquinas da rede, utilizando a tecnologia *Web*. Esta aplicação visa a realização de experiências práticas com a utilização da arquitetura CORBA, da tecnologia WWW e da linguagem Java na implementação de sistemas de gerenciamento distribuído.

A aplicação foi implementada como um *applet* Java (Figura 30). Em consequência deste fato, ela possui a característica de ser independente de plataforma e de poder ser utilizada, a partir de praticamente qualquer computador pertencente a Internet, sem a necessidade da instalação prévia de algum *software* de gerência.

CORBA foi utilizado para realizar o instanciamento do objeto *Cluster*. Com a utilização do CORBA, a implementação do objeto *Cluster* é executada em um servidor e o *applet* necessita, apenas, de conter a definição da interface deste objeto. Por este motivo, a aplicação de gerência torna-se mais simples, contendo apenas o código necessário para realização da interface com o usuário. Outro ponto a ser observado, é que a implementação do objeto *Cluster* está sendo executada no servidor e não no *browser*. Em consequência deste

fato, as operações realizadas pelo objeto *Cluster* não estão sujeitas às limitações de segurança impostas a *Applets*, sendo executados em *browsers Web*.

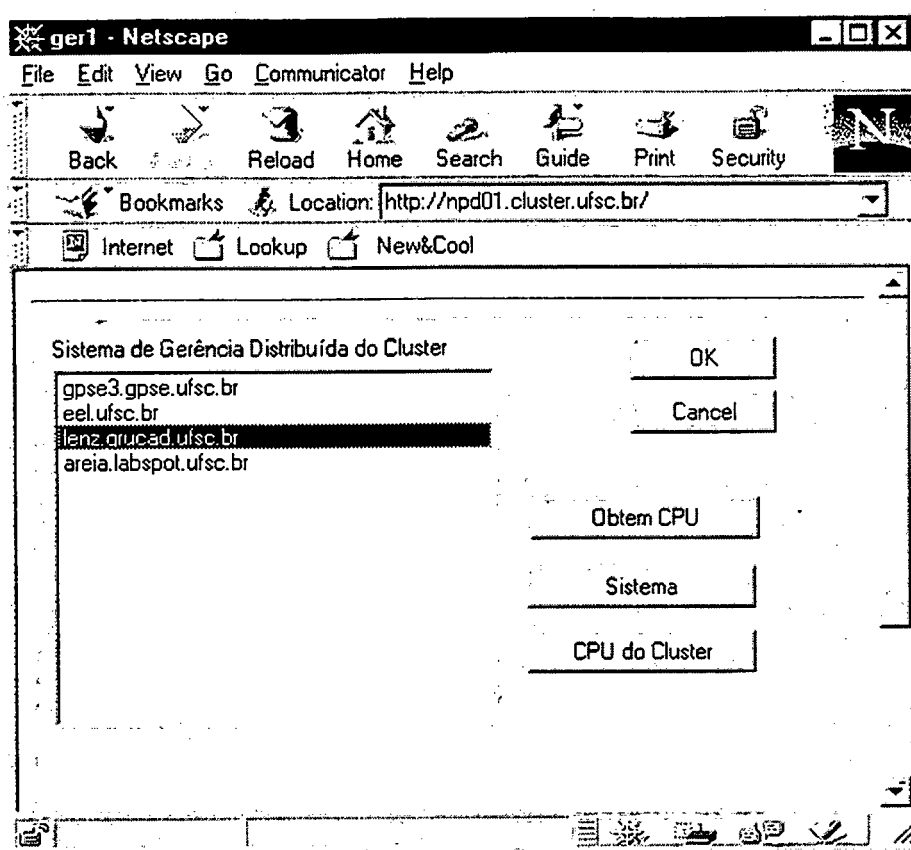


Figura 30 Janela Principal da Aplicação de Gerência

### 6.3.1 Implementação de um Protótipo da Aplicação de Gerência

Afim de apresentar um exemplo prático de uma aplicação de gerência CORBA, foi implementado um protótipo do sistema cuja finalidade é monitorar a taxa de ociosidade de CPU das máquinas do *cluster* e obter informações referentes ao sistema operacional de tais máquinas.

A fim de diminuir a complexidade do ambiente de estudo, os testes foram realizados utilizando apenas quatro estações pertencentes ao *cluster*. No entanto, todos os testes realizados podem, prontamente, abranger um número maior de estações.

A aplicação de gerência visa implantar a funcionalidade básica para a realização de operações de gerência globais ao *cluster* e de tarefas restritas a uma determinada estação de trabalho. Para demonstrar esta funcionalidade, na aplicação de gerência implementada neste

projeto é possível realizar operações para obter a taxa de ociosidade de uma máquina individualmente ou de todas as estações do *cluster* simultaneamente. Ambas as operações exigem que a aplicação-gerente realize apenas uma única invocação de método ao objeto *Cluster*. Nesta aplicação, é possível, também, obter informações sobre o sistema operacional de todas as máquinas do ambiente, através do envio de uma única mensagem ao objeto *Cluster*.

As funções de gerência para o acompanhamento da taxa de ociosidade das máquinas do *cluster* podem ser muito úteis para um usuário observar quais as máquinas que possuem a menor utilização de CPU. Tais funções podem ser úteis, também, para um administrador realizar um estudo sobre o nível de utilização das máquinas e auxiliá-lo a decidir o momento necessário para realização de uma atualização destes equipamentos.

A função para a obtenção de informações sobre o sistema das máquinas do *cluster* pode ser muito útil para a aquisição de parâmetros como o *hostname* e o *hostid* dos equipamentos. Tais informações são necessárias, por exemplo, para a instalação de licenças de *softwares* nas máquinas. Caso o usuário tivesse que realizar esta tarefa manualmente, ele teria que se conectar a cada uma das estações do *cluster* e requisitar as informações desejadas. Esta operação poderia demandar grande quantidade de tempo.

Na implementação da aplicação de gerência foram definidas funções somente para leitura de informações não sigilosas do sistema. Esta limitação ocorre porque, neste trabalho, ainda não foram definidos procedimentos de autenticação e segurança para o acesso às funções de gerência.

Para a implementação deste protótipo foram utilizados os seguintes ambientes de desenvolvimento Java: o Microsoft J++ e o JDK 1.0.2.

### **6.3.2 Módulos que Compõem o Protótipo da Aplicação de Gerência**

Na implementação da aplicação de gerência, foram criadas algumas classes cuja finalidade básica é a realização da interface com o usuário e a comunicação com o objeto *Cluster*.

```

<html>
<head>
<title>ger1</title>
</head>
<body>
<hr>
<applet
  code=ger1.class
  name=ger1
  width=420
  height=340 >
</applet>
<hr>
<a href="ger1.java">The source.</a>
</body>
</html>

```

Quadro 12 Arquivo HTML que invoca a Aplicação de Gerência

A classe *ger1.java* consiste em um *applet* que é invocado a partir de um arquivo HTML, como apresentado no Quadro 12. Este *applet* é responsável por interceptar os eventos ocorridos e instanciar a classe *ID\_Janela*. A janela principal deste *applet* é apresentada na Figura 30.

```

String clusterStations = cluster.getClusterHostnames();
StringTokenizer t;
t= new StringTokenizer(clusterStations, ";");
String n=null;
int j = 0;
while(t.hasMoreTokens()) {
  n = t.nextToken().trim();
  if (j!=0){
    l.addItem(n);
  }
  j++;
}

```

Quadro 13 Invocação do Método *getClusterHostnames* na classe *ID\_Janela*

Para a criação dos botões e da *listbox* pertencente a janela principal do *applet* foi criada a classe *ID\_Janela*. Esta classe também instancia o objeto *Cluster* e emite uma mensagem requisitando os nomes das máquinas que compõem este ambiente (invocando o método *getClusterHostnames*), para serem adicionados ao *listBox* (Quadro 13). Na janela principal do *Applet* o usuário pode selecionar uma máquina na *listbox* e obter as informações sobre a sua taxa de ociosidade. Esta tarefa pode ser realizada, pressionando-se o botão

“Obtem CPU”. Para se obter as informações sobre taxa de ociosidade ou informações do sistema de todas as máquinas listadas na *listbox* simultaneamente, basta pressionar os botões “CPU do *Cluster*” ou “Sistema”. Os eventos relativos aos itens de interface, criados pela classe *ID\_Janela*, são tratados na classe *ger1*. Com o ato de pressionar os botões “Obtem CPU”, “CPU do *Cluster*” e “Sistema” são ocasionados eventos que, uma vez tratados, permitem que sejam instanciados os respectivos objetos: *Janela\_CPU*, *Janela\_CPU\_Cluster* e *Janela\_Show\_System*.

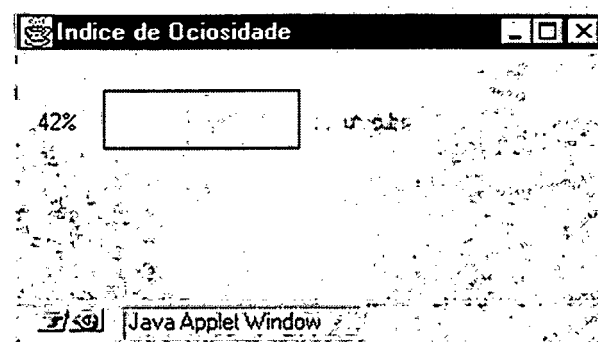


Figura 31 Taxa de Ociosidade de um Máquina apresentada pelo Objeto *Janela\_CPU*

Afim de possibilitar a monitoração da CPU de uma máquina individualmente, foi criada a classe *Janela\_CPU*. Esta classe é derivada da classe *Frame* e instancia o objeto *ShowCPUTread*. O objeto *ShowCPUThread* é derivado da classe *Thread* e, em intervalos regulares, invoca a atualização da taxa de CPU apresentada pelo objeto *Janela\_CPU*. Assim, o objeto *Janela\_CPU* cria uma janela que, através de um gráfico, apresenta a taxa de ocupação de uma máquina, em intervalos regulares de tempo (Figura 31). Este objeto obtém a taxa de ocupação de CPU de uma máquina, através da invocação do método *getCPUStation* pertencente ao objeto *Cluster*.

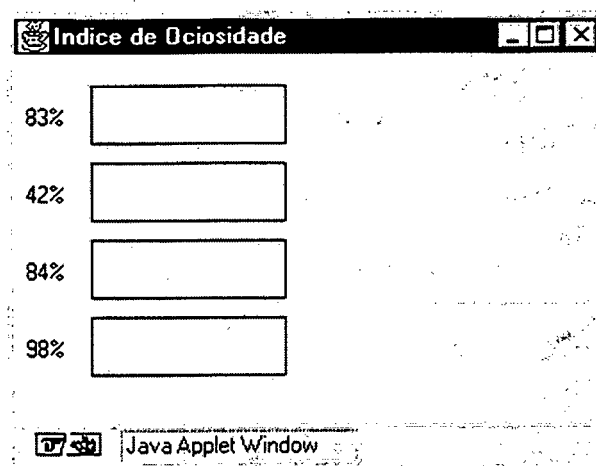


Figura 32 Taxa de Ociosidade de Todas as Máquinas do Ambiente de Teste

A classe *Janela\_CPU\_Cluster* foi criada para possibilitar a monitoração da CPU de todas as máquinas gerenciadas. Esta classe é derivada da classe *Frame* e instancia o objeto *ShowCPUClusterTread*. O objeto *ShowCPUClusterThread* é derivado da classe *Thread* e, em intervalos regulares, invoca a atualização das taxas de ociosidade de CPU apresentadas pelo objeto *Janela\_CPU\_Cluster*. Com este mecanismo, o objeto *Janela\_CPU\_Cluster* cria uma janela que, através de gráficos, apresenta a taxa de ociosidade de CPU de todas as máquinas gerenciadas. Os valores da taxa de ociosidade das CPUs são atualizados em intervalos regulares de tempo, com as invocações realizadas pelo objeto *ShowCPUClusterThread* (Figura 31). A obtenção da taxa de ociosidade de todas as máquinas que compõem o ambiente é realizada através da invocação do método *getClusterCPU*, pertencente ao objeto *Cluster*.

Os fontes de algumas classes que compõem a implementação da aplicação de gerência do *cluster*, podem ser encontradas nos anexos deste trabalho. Porém, todas as classes utilizadas nesta implementação podem ser acessadas através do endereço <ftp://www.cluster.ufsc.br/cluster/fontes>.

#### 6.4 Agente de Contabilização

Para o armazenamento periódico de informações de contabilização, em uma base de dados relacional, foi implementado um Agente de Contabilização. Este agente é utilizado, neste trabalho, para o armazenamento periódico da taxa de ociosidade das máquinas do *cluster*. Estas informações podem ser muito úteis para realizar uma análise do perfil de

ocupação das máquinas, durante um determinado período. Por exemplo, pode-se determinar quais os horários em que as máquinas estão mais ociosas, durante um determinado dia.

A primeira operação que é realizada pelo agente de contabilização é o instanciamento do objeto *Cluster*. Neste caso, deve-se ressaltar que a aplicação de gerência e o agente de contabilização utilizam a mesma instância deste objeto. Por este motivo, se a aplicação de gerência alterar o valor de um atributo da instância utilizada pela aplicação de gerência, esta alteração será percebida, também, pelo agente de contabilização. Utilizando-se desta característica, o agente periodicamente verifica, junto ao objeto *Cluster*, se a função de contabilização está ativa e, caso esteja, realiza o armazenamento de informações em uma base de dados relacional. Dentro deste contexto, o processo-agente realiza basicamente as seguintes operações (Figura 33):

1. instancia o objeto *Cluster*;
2. realiza o invocação do método *getClusterCPU* do objeto *Cluster*, visando verificar se a função de contabilização de recursos de CPU está ativa. Esta função pode ser ativada por uma aplicação de gerência, através da invocação do método *startContab*, pertencente ao objeto *Cluster*. Para desativar esta função, a aplicação de gerência deve invocar o método *stopContab*. Esta interação entre o agente e a aplicação de gerência é possível porque ambos referenciam a mesma instância do objeto *Cluster*;
3. caso função de contabilização esteja desativada, o processo-agente é paralisado por alguns instantes e realiza o passo 2 novamente;
4. se a função de contabilização estiver ativa, o processo-agente invoca o método *getClusterCPU* do objeto *Cluster*, visando obter a taxa de ociosidade de CPU de todas as máquinas gerenciadas;
5. as taxas de ociosidade são armazenadas em uma base de dados relacional, assim como, a data, a hora e o minuto em que foi realizada a aquisição dos dados;
6. o processo fica paralisado por alguns instantes e volta a executar o passo 2.

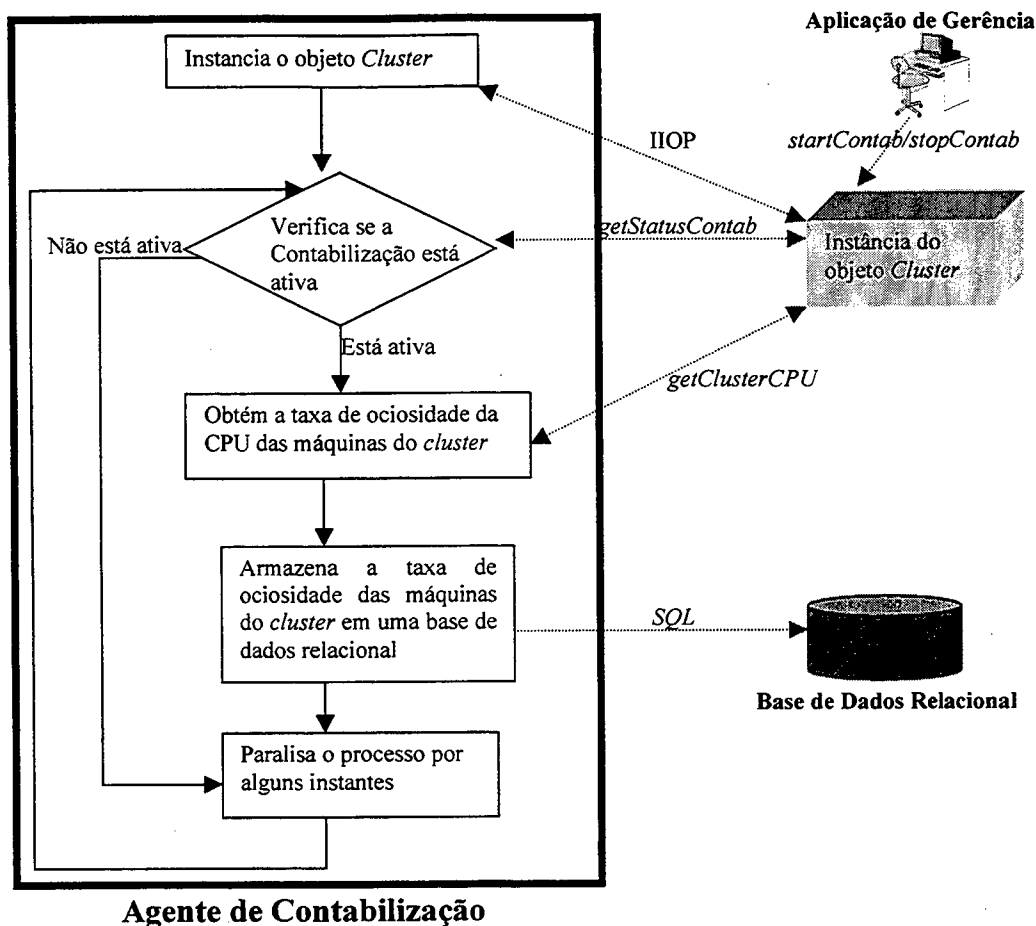


Figura 33 Funcionalidade do Agente de Contabilização

Um fato que deve ser observado é que este modelo de agente pode ser utilizado para a contabilização de qualquer tipo de informações de gerência. A implementação deste agente foi realizada utilizando o Java e os fontes podem ser encontrados no endereço <http://npd01.cluster.ufsc.br/cluster/fontes>.

Caso seja necessário a realização da contabilização de recursos de uma única máquina, pode-se criar um agente que instancie o objeto *Computer* e execute esta função, seguindo a mesma funcionalidade apresentada pelo agente descrito anteriormente.

## 6.5 Agente SNMP

O agente SNMP, implementado neste projeto, apresenta uma estrutura bastante simples e tem a função de permitir a interação entre os protocolos SNMP e IIOp/CORBA. Este agente, ao receber primitivas *get-request*, instancia o objeto *Cluster*, envia as requisições desejadas e transmite uma PDU *get-response* (contendo os resultados obtidos) para a



aplicação gerente. Deste modo, uma aplicação de gerência SNMP pode acessar objetos gerenciados que são instanciados através do protocolo IIOp/CORBA.

A Figura 34 mostra a funcionalidade do Agente SNMP implementado.

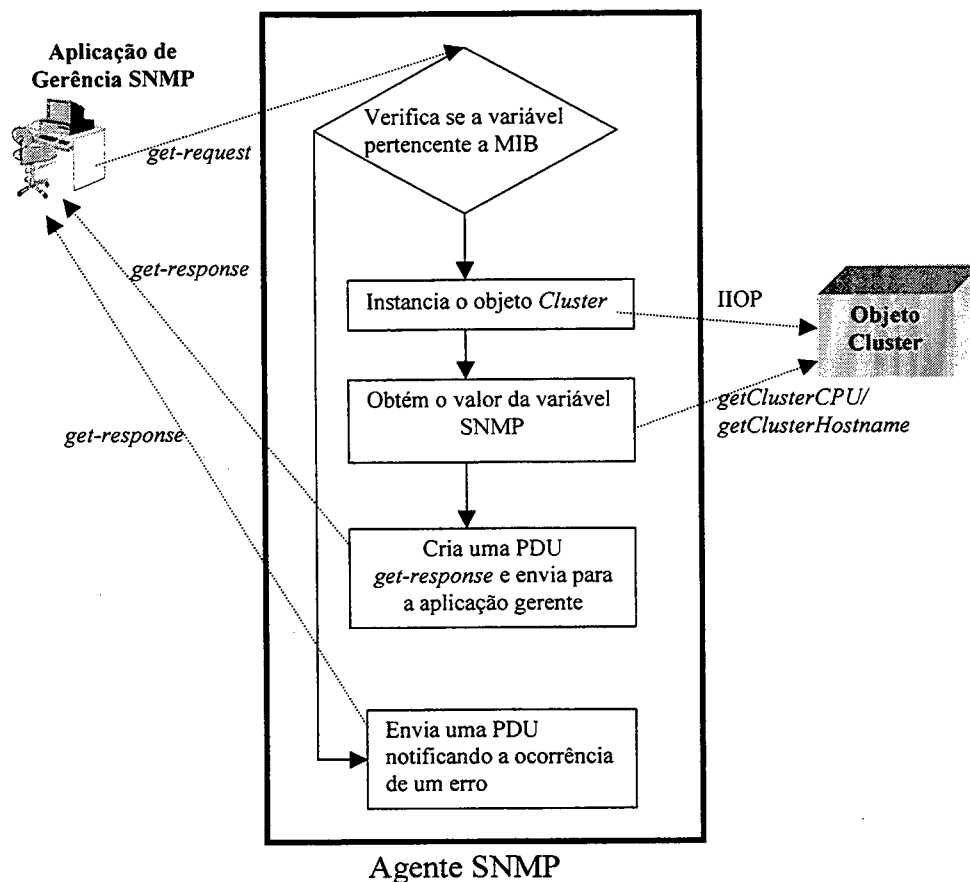


Figura 34 Funcionalidade do Agente SNMP

Para a implementação deste agente foi utilizado um modelo disponível na documentação da *Advent SNMP API*. Este modelo foi adaptado para permitir o instanciamento de objetos gerenciados, através do CORBA, e viabilizar a utilização de uma MIB própria para o *cluster*. O agente SNMP é capaz de retornar informações sobre o nome e a taxa de ociosidade da CPU das máquinas que compõem o *Cluster*. Os nomes das máquinas podem ser obtidos através da variável SNMP “1.3.6.1” e a taxa de ociosidade, através da variável “1.3.6.2”. A identificação destas variáveis não é definitiva e deve ser modificada, em um futuro próximo, para ser integrável com a estrutura da MIB-II. As requisições realizadas para as variáveis SNMP retornam como resultado um *string* contendo os nomes ou as taxas de ociosidade de todas as máquinas separados pelo caracter “;”. Através deste mecanismo, é

possível obter informações globais ao *cluster*, através do envio de uma única primitiva SNMP para o processo agente.

### 6.5.1 Implementação do Agente SNMP

O agente SNMP foi implementado como um *application* Java. Na função *main* deste *application* é criada uma instância da classe *AgenteSNMP* (denominada *sim*) e invocado o método *init* deste objeto. Com a invocação deste método é criada uma instância da classe *SnmpAPI* (denominada *api*) e associado como seu cliente o próprio objeto *sim* (Quadro 14). Com isto, sempre que o agente receber uma PDU SNMP, será invocado o método *callback* do objeto *sim*. A seguir, é criada uma instância da classe *SnmpSession* (denominada *session*), na qual estará associada uma porta TCP/IP, uma instância da classe *SnmpAPI* e o nome da comunidade que este agente pertence. Tanto o nome da comunidade como o número da porta TCP/IP podem ser fornecidos como parâmetros na inicialização do agente. O valor padrão para o nome da comunidade é *public* e para o número da porta é 161. Deste modo a instância da classe *SnmpSession* deve monitorar uma a porta TCP/IP para receber PDUs SNMP endereçadas para a mesma comunidade (Quadro 14).

```

        // Inicia a API SNMP
        api = new SnmpAPI();
        api.start();
        api.client = this; // Permite que o método callback seja invocado
        if (values[0].equals("Set")) api.DEBUG = true;
        // Open session and set local port

        SnmpSession session = new SnmpSession(api);

        if (values[1] != null) session.community = values[1];
        else session.community = "public";

        try { // usa a porta padrão
            if (values[4] != null) session.local_port = Integer.parseInt(values[4]);
            else session.local_port = api.SNMP_PORT;
            System.out.println(" PORTA : " + session.local_port);
        }
        catch (NumberFormatException ex) { System.err.println("Invalid Integer Arg");
        }

        // Inicia a monitoração da porta TCP/IP
        try { session.open(); }
        catch (SnmpException e) {
            System.err.println(e);
            System.exit(1);
        }
    }

```

Quadro 14 Criação das Instâncias da Classe *SnmpAPI* e *SnmpSession*

As classes *SnmpSession* e *SnmpAPI* são derivadas da classe *Thread*. Deste modo, as instâncias destas classes são constantemente executadas e, assim que a instância da classe *SnmpSession* recebe uma PDU SNMP, esta repassa para o objeto *api*, que invoca o método *callback* da classe *AgenteSNMP*.

Como pôde ser observado anteriormente, o método *callback* será invocado sempre que o agente receber uma PDU SNMP. De posse de uma PDU, o método *callback* invoca o método *getValue* para obter o valor de cada variável SNMP e envia os resultados para a aplicação-gerente, na forma de uma PDU *get-response*.

O método *getValue* (Quadro 15) verifica se a PDU contém uma mensagem de *get-request* e se o identificador da variável pertence a MIB. Caso pertença, ele invoca o método *consultaObjetoCluster* para obter o valor da variável SNMP e retorna este valor para o método *callback*.

```

SnmpVar getValue(SnmpVarBind varbind, int req) throws
MibException {
    MibNode node = varbind.oid.node;
    if (req == SnmpAPI.GET_REQ_MSG) { // get request
        String saux= varbind.toString();
        System.out.println("objeto:"+saux+"!!");
        if (estaNaMIB(saux)){
            SnmpString aux= consultaObjetoCluster(saux) ;
            return aux;
        }
    }
}

```

Quadro 15 Obtenção do Valor de uma Variável SNMP no Método *getValue*

O método *consultaObjetoCluster* instancia o objeto *Cluster* e, se a variável SNMP possuir o identificador “.1.3.6.1”, invoca o método *getClusterHostnames*; caso a variável possua o identificador “.1.3.6.2”, invoca o método *getClusterCPU*. A invocação destes métodos retorna um *String* contendo respectivamente os nomes e a taxa de ociosidade de CPU das máquinas do *cluster* (Quadro 16).

```

private Cluster getCluster()
{
    try {
        String URL = new
String("IIOP://" + serverName.trim() + ":" + serverPort);
        String hostURL = new String(URL);
        ORB client = CORBA.ORB_init(null);
        cluster = (Cluster)client.URLToObject(hostURL +
        "/cluster" +
        "?agente.Cluster" );
    } catch(Exception e){
        e.printStackTrace();
    }
    return cluster;
}

SnmString consultaObjetoCluster(String oid)
{
    SnmpString result= null;
    Cluster cluster= getCluster();
    if ( oid.equalsIgnoreCase(".1.3.6.1: NULL") ){
        String clusterStations =
cluster.getClusterHostnames();
        result = new SnmpString(clusterStations);
    }
    if ( oid.equalsIgnoreCase(".1.3.6.2: NULL") ){
        String clusterCPU = cluster.getClusterCPU();
        result = new SnmpString("taxa de ociosidade");
    }
    return result;
}

```

Quadro 16 Invocação dos Métodos *consultaObjetoCluster* e *getCluster*

## 7 Conclusão

O objetivo principal traçado para este trabalho, foi a utilização do protocolo SNMP, da linguagem Java, da tecnologia Web e da arquitetura CORBA na implementação de um sistema de gerência distribuída, para um *cluster* de estações que constituem a Rede UFSC. Como características fundamentais deste sistema, determinou-se que este deveria ser independente de plataforma e suportar a maioria dos servidores de banco de dados disponíveis comercialmente. A fim de alcançar este objetivo, foram traçados vários passos intermediários.

A utilização do SNMP, Java, WWW e CORBA, aplicados à gerência de redes, envolve um domínio vasto de conhecimentos teóricos e a realização de uma variedade de experiências práticas. Neste sentido, iniciou-se uma série de estudos baseados na literatura e em material obtido junto a Internet objetivando o domínio destas tecnologias.

A seguir, foi apresentado o ambiente de desenvolvimento. Nesta etapa, foi descrita a estrutura de *software* e *hardware* do *cluster* de estações e apresentadas algumas necessidades básicas de gerência deste ambiente.

Com a base teórica adquirida e o ambiente de desenvolvimento definido, iniciou-se o estudo de ferramentas que permitem a integração do SNMP e o CORBA com o Java. Como fruto das pesquisas realizadas, foi selecionada a *Advent SNMP API* para a utilização do SNMP integrado ao Java. Outra ferramenta selecionada foi o *Bojangles*, que possui a implementação de um ORB independente de plataforma (JORB) e permite a utilização do CORBA para aplicações desenvolvidas em Java. Com a utilização do *Bojangles*, tornou-se viável a criação de aplicações de gerência e a utilização de um mecanismo de comunicação entre objetos distribuídos que sejam independentes de plataforma. Esta característica foi um dos requisitos básicos traçados para este projeto.

Após os estudos teóricos e a definição das ferramentas utilizadas, foram especificadas e implementadas um conjunto de classes que permitem a construção de aplicações de gerência que podem manipular primitivas do protocolo SNMP e ser integrável com a maioria dos servidores de banco de dados disponíveis comercialmente. Este conjunto de ferramentas (*Ambiente de Gerência*) constituiu a base para a construção da aplicação de gerência e do agente de contabilização implementados neste trabalho. Para a utilização do SNMP em aplicativos Java, foi empregada a *Advent SNMP API* e implementou-se uma classe (*SNMPMan*) que permite a leitura do valor de um objeto gerenciado. Para o acesso a uma

base de dados relacional, através de um aplicativo Java, foi utilizado o JDBC. O JDBC consiste em uma API que permite o acesso a maioria dos servidores de banco de dados, disponíveis comercialmente, e viabilizou a construção de aplicações de contabilização capazes de armazenar informações em uma base de dados relacional. Nesta dissertação foram realizados testes de conexão com os servidores de banco de dados *Sybase* e *DB2*. Para o *Ambiente de Gerência* foram implementadas também classes que permitem a geração de gráficos, através de um *applet* Java.

Utilizando o *Ambiente de Gerência*, foi implementado um software que permite a análise de tráfego em *links* TCP/IP (denominado *WebVis*). Este *software* foi desenvolvido em Java, utiliza o SNMP para obter as informações de tráfego e armazena as informações obtidas em uma base de dados relacional. Como parte integrante do *WebVis*, foi implementado um *applet* que recupera as informações referentes ao tráfego médio em um *link* TCP/IP, durante um determinado período, e apresenta estas informações na forma de gráficos.

Através da criação do *Ambiente de Gerência* e da implementação do *WebVis* foi possível a aquisição das primeiras experiências práticas para a utilização do SNMP, WWW e Java na construção de aplicações de gerência distribuídas. Os conhecimentos adquiridos com esta etapa formaram a base necessária ao desenvolvimento do sistema de gerência distribuída para o *cluster* de estações.

Através dos conhecimentos teóricos e práticos adquiridos com as experiências anteriores, tornou-se viável o início da implementação do sistema de gerência para o *cluster* de estações. Primeiramente, iniciou-se o desenvolvimento dos objetos gerenciados distribuídos. Os objetos gerenciados foram implementados em Java e podem ser instanciados utilizando o CORBA, fazendo uso do JORB como mecanismo de comunicação. Para tanto, foi implementado o objeto *Computer* para o gerenciamento de estações individuais e o objeto *Cluster* para o gerenciamento global do *cluster*. O objeto *Computer* é instanciado em cada máquina do *cluster* e permite a interação com o sistema operacional das máquinas envolvidas. O objeto *Cluster* é instanciado na mesma máquina que possui o servidor Web do *cluster* e permite o gerenciamento de estações individuais ou de todas as máquinas deste ambiente, através da invocação de um único método. O objeto *Cluster* realiza a interface entre a aplicação de gerência e as diversas instâncias do objeto *Computer*, permitindo a diminuição da complexidade das aplicações de gerência, assim como, contornando as limitações de segurança impostas a *applets* sendo executados em *browsers* Web. Estes benefícios ocorrem em consequência do fato de que a implementação do objeto *Cluster* é executada no servidor e

não no *browser*. Neste projeto foram implementados objetos gerenciados que permitem a obtenção da taxa de ociosidade da CPU, das máquinas do *cluster* e de informações referentes ao sistema operacional destas máquinas. Com a implementação dos objetos gerenciados, ficou concretizada a base para a adição de novas funções de gerência, associadas ao *Cluster* de estações.

Para o gerenciamento do *cluster* foi desenvolvido um *applet* que instancia o objeto *Cluster* (utilizando o protocolo IIOP) e permite a obtenção da taxa de ociosidade da CPU de uma máquina individual ou de todas as máquinas pertencentes ao *Cluster*, através de uma única invocação de método. Esta aplicação permite, também, a obtenção de informações referentes ao sistema operacional de todas as máquinas que compõem este ambiente e implementa a estrutura básica exigida para a construção de aplicações de gerência, utilizando o WWW, CORBA e Java. Deste modo, este aplicativo pode agir como um modelo para a construção de outras aplicações de gerência (que podem ser mais complexas).

Para a integração do sistema de gerência distribuída com as aplicações de gerência SNMP, foi implementado um agente que age como um *gateway* entre os protocolos SNMP e IIOP. Este agente permite que uma aplicação de gerência SNMP possa obter o nome e a taxa de ociosidade da CPU, de todas as máquinas do *cluster*, através do envio de uma única primitiva SNMP.

Foi implementado também, um agente de contabilização que periodicamente armazena em uma base de dados relacional, os valores da taxa de ociosidade de CPU de todas as máquinas do *cluster*. As informações armazenadas permitem uma análise detalhada do nível de utilização destas máquinas.

Com os estudos teóricos e as implementações realizadas, pôde-se finalmente efetuar uma análise dos resultados alcançados com a utilização do SNMP, Java, WWW e CORBA na implementação do sistema de gerência distribuída.

## **7.1 Alcance dos Objetivos**

O objetivo geral traçado para este trabalho foi a implementação de um sistema distribuído destinado à gerência de um *cluster* de estações. Definiu-se que este sistema deveria ser independente de plataforma, suportar os protocolos SNMP e IIOP/CORBA, permitir a integração com a maioria dos servidores de banco de dados disponíveis comercialmente e utilizar o WWW para realizar a interface com o usuário final. Somado a isto, o sistema de gerência deve possuir um conjunto de objetos gerenciados distribuídos e

agentes multiplataforma que permitam a implantação de novas funções de gerência, associadas a um *cluster* de estações de trabalho. O objetivo geral definido para este trabalho foi alcançado com sucesso. No entanto, na introdução deste trabalho, especificou-se objetivos intermediários que foram paulatinamente seguidos até a completa implementação do sistema de gerência distribuída. Nesta etapa são verificados os resultados obtidos em relação aos objetivos inicialmente traçados.

Primeiramente foi definida a realização de um estudo dos principais padrões e ferramentas envolvidos com a tecnologia *Web*. Neste estudo pôde-se obter uma visão geral sobre o HTML, HTTP e CGI. Com relação ao Java, foi realizada uma pesquisa bibliográfica mais extensa, uma vez que praticamente todas as aplicações desenvolvidas neste projeto foram implementadas utilizando esta linguagem de programação. Através da pesquisa bibliográfica realizada, pôde-se observar que a tecnologia *Web* pode ser muito útil para o desenvolvimento de aplicações de gerência distribuídas e independentes de plataforma. Após o término destes estudos, pode-se concluir que as metas traçadas para esta fase foram alcançadas.

O próximo passo definido para este trabalho, foi o estudo dos protocolos SNMP e IIOP/CORBA, com o enfoque no gerenciamento de redes de computadores. O estudo do SNMP foi realizado com sucesso. No entanto, as pesquisas relacionadas com o padrão CORBA não atingiram a profundidade desejada, em consequência da complexidade deste padrão e da escassez de tempo para o término deste trabalho. Apesar da superficialidade do estudo do CORBA, pôde-se observar que esta ferramenta pode ser extremamente útil para a realização da gerência distribuída de redes e que os conhecimentos adquiridos foram suficientes para alcançar os objetivos inicialmente propostos para este trabalho.

Outra meta definida foi a avaliação das principais ferramentas e protocolos envolvidos no gerenciamento de redes, utilizando a tecnologia *Web*. Nesta etapa foram pesquisadas várias ferramentas. Dentre elas, foi escolhida a *Advent SNMP API* para realizar o uso do SNMP integrado ao Java. Para a utilização do CORBA escolheu-se o *Bojangles*. Este utilitário permite a utilização do CORBA em aplicativos Java e possui a implementação de um JORB que oferece um mecanismo de comunicação independente de plataforma entre os objetos. Neste trabalho não foram apresentadas todas as ferramentas pesquisadas, no entanto, a documentação relativa a todas as ferramentas estudadas podem ser encontradas no endereço <ftp://npd01.cluster.ufsc.br/ferramentas>. Apesar de não terem sido descritas todas as ferramentas estudadas neste trabalho, pode-se considerar que os objetivos traçados para esta etapa foram cumpridos com sucesso, uma vez que realizou-se a escolha das ferramentas necessárias à continuidade deste projeto.



Após a escolha das ferramentas utilizadas, definiu-se que a próxima meta a ser alcançada neste projeto seria a criação de um *Ambiente de Gerência*, através do desenvolvimento e utilização de um conjunto de ferramentas independentes de plataforma. Estas ferramentas permitem a obtenção de informações de gerência (utilizando o protocolo SNMP), o armazenamento destas informações em uma base de dados relacional e a utilização de uma interface *Web* com o usuário final. Esta etapa foi concluída com sucesso e permitiu a definição de um conjunto de classes para a execução de três tarefas fundamentais para as demais fases deste projeto: o acesso a uma base de dados relacional, a utilização do protocolo SNMP e a geração de gráficos a partir de um aplicativo Java.

Com a finalidade de realizar uma implementação prática, utilizando o Ambiente de Gerência, foi definido como um objetivo a ser alcançado a criação de uma aplicação modelo para a monitoração do tráfego em *links* TCP/IP. Este software foi denominado *WebVis* e foi implementado com sucesso. O *WebVis* está sendo utilizado para monitorar os *links* TCP/IP na UFSC e deve ser disponibilizado, em breve, para o domínio público. Como fruto principal deste projeto, pode-se considerar as experiências práticas adquiridas no desenvolvimento de aplicativos Java, capazes de acessar uma base de dados relacional, utilizar o protocolo SNMP e apresentar informações na forma de representações gráficas.

O próximo passo definido para este projeto foi a criação de um sistema de objetos distribuídos e agentes multiplataforma, visando a definição de novas funções de gerência para um *cluster* de estações (utilizando os protocolos SNMP e IIOP/CORBA). Para este projeto foram implementados os objetos *Cluster* e *Computer*. Tais objetos permitiram a definição de novas funções de gerência para a obtenção da taxa de ociosidade de CPU e de informações referentes ao sistema das máquinas pertencentes ao *cluster*. Com esta implementação foi criada uma estrutura de objetos distribuídos e ORBs multiplataforma que permitem o gerenciamento de todas as máquinas do *cluster*. Esta estrutura pode ser facilmente expandida para conter novas funções de gerência. Com as experiências práticas realizadas, pôde-se observar que o CORBA pode ser muito eficiente para instanciar objetos gerenciados. Após os primeiros testes, pôde-se concluir que a meta de criar objetos gerenciados e novas funções de gerência, utilizando o protocolo IIOP/CORBA, foi realizada com sucesso.

Afim de permitir a integração dos objetos gerenciados, com as aplicações de gerência SNMP, foi estabelecida como uma das metas, para este trabalho, a implementação de um agente que age como um *gateway* entre os protocolos SNMP e IIOP. A implementação deste agente apresenta uma estrutura bastante rudimentar, porém, implementa os requisitos exigidos

no início deste projeto, ou seja, permite a adição de novas funções de gerência SNMP, com o uso de objetos gerenciados acessíveis através do protocolo IIOP.

Visando realizar a contabilização de informações de gerência em uma base de dados relacional, foi definida a implementação de um agente que instancia os objetos gerenciados, obtém as informações e armazena estas em uma base de dados relacional. A implementação deste agente foi realizada com sucesso e possui a funcionalidade para o armazenamento de qualquer informação de gerência em uma base de dados relacional.

A última meta traçada para este projeto foi a implementação de uma aplicação para o gerenciamento do *cluster*. Esta aplicação foi implementada com sucesso e permite a monitoração de informações do sistema e CPU das máquinas deste ambiente. A aplicação de gerência instancia o objeto *Cluster* e gerencia as máquinas, através da invocação de métodos a este objeto. Esta característica permite uma diminuição significativa da complexidade da implementação realizada e do tráfego gerado entre o objeto gerenciado e o gerente. A aplicação de gerência foi implementada como um *applet* Java e, devido a este fato, possui a característica de ser independente de plataforma e poder ser acessada, praticamente, a partir de qualquer computador pertencente a Internet (<http://www.cluster.ufsc.br/cluster>). Com esta implementação foi cumprido com sucesso o objetivo de realizar o desenvolvimento de uma aplicação de gerenciamento utilizando o WWW, CORBA e Java.

Ao final deste trabalho de dissertação pode-se concluir que todos os objetivos traçados foram alcançados. Com os estudos teóricos e as implementações realizadas pôde-se alcançar o objetivo principal de utilizar o SNMP, Java, WWW e CORBA para a implementação de um sistema de gerência distribuída, que seja multiplataforma e permita a integração a maioria dos servidores de banco de dados, disponíveis comercialmente. Este sistema permite o gerenciamento do *cluster* a partir de qualquer máquina conectada a rede e possibilita a integração com o protocolo SNMP. Com a utilização do objeto *Cluster*, este gerenciamento é realizado com uma sensível diminuição de tráfego e da complexidade da aplicação gerente.

Os frutos deste trabalho já poderão ser sentidos de maneira prática pelos usuários e administradores do *cluster* da Rede UFSC. Os usuários poderão obter as informações importantes para a execução de suas aplicações a partir de qualquer ponto da rede. Do mesmo modo, os administradores poderão gerenciar este ambiente a partir da utilização de um *browser Web* instalado em qualquer máquina da rede.

## 7.2 Dificuldades Encontradas

A dificuldade inicialmente enfrentada para o desenvolvimento deste trabalho foi a grande quantidade de conhecimentos teóricos exigidos para sua implementação. O estudo do SNMP, Java, WWW e CORBA envolve a necessidade de uma vasta pesquisa bibliográfica, que, em alguns casos, não pode ser realizada com a profundidade desejada. O exemplo mais marcante deste fato foi o estudo do CORBA. Este estudo exigiu o domínio de uma grande quantidade de conhecimentos práticos e a aquisição de uma ampla base teórica referente a este tema. Outro ponto que dificultou esta etapa, foi a escassez de documentação relacionada a utilização destas tecnologias aplicadas a área de gerência de redes.

Na fase de pesquisa das ferramentas para a utilização do CORBA e SNMP integradas ao Java, foram encontradas algumas dificuldades. A tarefa de pesquisar, instalar e realizar testes com as ferramentas estudadas, exigiu uma grande demanda de tempo e esforço. Como fruto dos testes realizados, constatou-se que o JORB utilizado não alcançou o nível de robustez desejado, apresentando alguns problemas de queda de comunicação, ainda não explicados.

Na implementação do Ambiente de Gerência e do *WebVis*, a maior dificuldade encontrada foi a integração com o servidor de banco de dados. Esta tarefa exigiu a instalação de um servidor, o estudo do JDBC e o domínio de aspectos teóricos na área de banco de dados. Somado a isto, a implementação do *WebVis* envolveu um estudo aprofundado do Java e exigiu a implementação de um grande número de classes. Estas tarefas não apresentaram um nível de complexidade muito grande, porém exigiram uma grande quantidade de tempo para a sua concretização.

A implementação do sistema de gerência para o *cluster* de estações foi uma tarefa relativamente complexa e que exigiu uma grande demanda de tempo. O primeiro fator que dificultou a realização desta tarefa foi a escassez de uma documentação adequada, relatando a utilização conjunta do SNMP, Java, WWW e CORBA aplicados à área de gerência de redes. Outro fator a ser considerado, é que a implementação do sistema de gerência exigiu a realização de várias tarefas cujas soluções não eram conhecidas até então. Uma delas foi a integração entre o Java e o sistema operacional da máquina. Esta tarefa exigiu um grande número de experimentos práticos e pesquisas teóricas. Outra tarefa considerada complexa, foi a implementação dos objetos gerenciados, pois envolveu a utilização prática do CORBA integrado ao Java.

A implementação do agente SNMP também pode ser considerada uma tarefa

complexa. Em virtude deste fato, a funcionalidade deste projeto foi simplificada ao máximo, resultando na definição de um mecanismo bastante rudimentar para a integração entre os protocolos SNMP e IIOP.

Para que a finalização deste projeto pudesse ocorrer em tempo hábil, a abrangência das implementações realizadas foi restringida ao máximo. Em consequência deste fato, não foi implementado nenhum mecanismo de segurança para o sistema de gerência. Devido a ausência de mecanismos de segurança, neste trabalho foram definidas apenas funções de gerência que podem ser consideradas de acesso irrestrito a todos os usuários.

### **7.3 Perspectivas Futuras**

Neste trabalho foram efetuadas pesquisas visando a realização da gerência de redes distribuída, utilizando o SNMP, Java, WWW e CORBA. Os objetivos inicialmente propostos para este trabalho foram alcançados. No entanto, existem ainda muitas pesquisas a serem realizadas nesta área. Algumas destas pesquisas são sugeridas neste capítulo e devem compor as futuras etapas deste projeto.

Uma das necessidades primordiais para o sistema de gerência do *cluster* é a definição de mecanismos de segurança e autenticação. A implementação destes mecanismos é vital para possibilitar a definição de funções de gerência mais complexas.

Outro ponto muito importante a ser realizado, é uma avaliação de outras implementações de ORBs. Nesta avaliação devem ser efetuados testes de performance e robustez, visando determinar, com uma maior segurança, alternativas para a comunicação entre objetos distribuídos.

A definição de outras funções de gerência é um outro ponto importante a ser explorado. Para o gerenciamento do *cluster*, pretende-se implementar funções que permitam a realização de operações como: a instalação automática de *software*, o cadastro de usuários no *cluster*, a remoção de arquivos temporários, entre outras.

Outro ponto a ser explorado é um estudo detalhado do CORBA, enfatizando os aspectos aplicáveis à gerência de redes. Dentre estes aspectos, sugere-se estudos mais avançados sobre a integração do CMIP e SNMP com o CORBA.

Finalmente, uma das pesquisas que são sugeridas neste trabalho é a avaliação da aplicabilidade do sistema de gerência do *cluster* para subredes da Internet. Uma vez que o

*cluster* e as subredes apresentam uma semelhança fundamental: ambos consistem em um conjunto de computadores interconectados a uma rede.

As pesquisas da utilização conjunta do SNMP, Java, WWW e CORBA aplicados à gerência de redes estão apenas iniciando. Dentro deste contexto, os estudos e experimentos desenvolvidos nesta dissertação de mestrado contribuem significativamente para a realização da gerência de redes distribuída.

## 8 Anexo A: Código Fonte da Classe *SNMPMan*

```

/*
SNMPMan.java
*****
Esta classe a obter o valor de uma variavel SNMP,
atraves do metodo retornaValor

*/
import java.lang.*;
import java.util.*;
import java.net.*;
import Snmp.*;
import ResultString;
import Config;

public class SnmpMan {

    String comunidade = "public";
    SnmpAPI api;
    Config conf;

    public SnmpMan(Config conf_par)
    {
        conf = conf_par;
        api = new SnmpAPI();
        api.start();
        while (api.client == null)
            try { Thread.sleep(10); } catch (Exception ie) {}
    }

    // Este metodo eh utilizado para retornar o valor de uma variavel SNMP,
    // atraves da primitiva get-request
    public ResultString retornaValor(String hostRemoto, String variavel) {
        // Cria o objeto session definindo o host remoto e comunidade

        boolean ERRO=true;
        ResultString res;
        SnmpSession session = new SnmpSession(api);
        session.peername = hostRemoto;
        session.community = conf.retornaComunidade();

        // Constroi uma PDU get-request
        SnmpPDU pdu = new SnmpPDU(api);
        pdu.command = api.GET_REQ_MSG;
        SnmpOID oid = new SnmpOID(variavel, api);
        if (oid.toValue() == null) {
            res= new ResultString(ERRO,"OID Invalido: " + variavel);
            session.close();
            return res;
        }
        else pdu.addNull(oid);

        try {
            session.open();

            // Envia PDU
            pdu = session.syncSend(pdu);

```

```
    } catch (SnmpException e) {
        res= new ResultString(ERRO,"Enviando PDU: "+e.getMessage());
        session.close();
        return res;
    }

    if (pdu == null) { // timeout

        res= new ResultString(ERRO,"timed out na comunicacao para: " +
hostRemoto);
        session.close();
        return res;
    }

        // retorna a o valor da variavel obtida
    if (api.DEBUG) System.out.println("Resposta recebida de " +pdu.address
        + ", community: " + pdu.community);
    if (pdu.errstat != 0)
    {
        res= new ResultString(ERRO,"Erro indicado na resposta: " +
SnmpException.exceptionString((byte)pdu.errstat)
        + "\nErrindex: " + pdu.errindex);
        session.close();
        return res;
    }

    SnmpVarBind varaux= (SnmpVarBind) pdu.variables.firstElement();
    String Result=varaux.variable.toString();
    session.close();
    res = new ResultString(Result.trim());
    return res;
}
}
```

## 9 Anexo B: Código Fonte da Classe *Linha* e *CadastroLinhas*

```
// A classe Linha eh utilizada para representar um link TCP/IP

import java.io.*;
import java.lang.*;
import java.util.*;
import java.net.URL;
import java.util.Date;
import tempjava.sql.*;
import ibm.netsql.*;
import SnmpMan;
import ResultString;
import Config;
import ResultQuery;
import BaseDeDados;

class Linha{
    String roteador;
    int inter;
    int tipo;
    // falta a taxa de transmissao
    String mensagem;
    String labelEntrada;
    String labelSaida;
    boolean primeiraColeta;
    Double valorSaidaAnt;
    Double valorEntradaAnt;
    long tempoAnt;

//Constructor da Classe Linha
public Linha ( String roteador_par, int inter_par, int tipo_par,
               String mensagem_par, String labelEntrada_par, String
               labelSaida_par)

{
    roteador= roteador_par;
    inter = inter_par;
    tipo = tipo_par;
    mensagem = mensagem_par;
    labelEntrada = labelEntrada_par;
    System.out.println (labelEntrada);
    labelSaida = labelSaida_par;
    primeiraColeta= true;
}

public Linha () {
    roteador= "";
    inter = -1;
    tipo = -1;
    mensagem = "";
    labelEntrada = "";
    labelSaida = "";
    primeiraColeta= true;
}
}
```



```

// apresenta o conteudo dos principais atributos da classe Linha
public void mostra() {
    System.out.println("roteador:" + roteador);
    System.out.println("Interface:" + inter);
    System.out.println("Tipo:" + tipo);
    System.out.println("Mensagem:" + mensagem);
    System.out.println("LabelEntrada:" + labelEntrada);
    System.out.println("LabelSaida:" + labelSaida);
}

// Imprime na saida padrao o valor da media de entrada e saida do trafego
no
// link TCP/IP
public void mostraMedia( Config conf) {
    Date data = new Date();
    BaseDeDados base= new BaseDeDados(conf);
    String sInter= "" + inter;
    ResultQuery res = base.ConsultaMediaDiaria(data, roteador,
    sInter.trim());
    if ( res.houveErro() )
        System.out.println("erro na obtencao das medias"+
        res.retornaMensagemDeErro());
    else{
        System.out.println("Media Entrada : " +
        res.retornaMediaEntrada().trim());
        System.out.println("Media Saida : " +
        res.retornaMediaSaida().trim());
    }
}

// Retorna medias de entrada e saida no link TCP/IP
public ResultQuery retornaMedias (Config conf, Date data) {
    BaseDeDados base = new BaseDeDados (conf);
    String sInter="" +inter;

    ResultQuery res = base.ConsultaMediaDiaria (data,
roteador,sInter.trim());
    if (res.houveErro())
        return null;
    else
        return res;
}

// Grava em um arquivo o valor das medias
public void gravaMedias (ResultQuery res, RandomAccessFile file) {
    try {
        String mediaEnt = res.retornaMediaEntrada();
        String mediaSai = res.retornaMediaSaida();
        file.writeUTF(mediaEnt);
        file.writeUTF(mediaSai);
    }
    catch (IOException e) {

```

```

        System.out.println ("Erro em arquivo em gravaMedias!");
    }
}

// le o varlor da media de entrada a partir de um arquivo
public String leiaMediaEntrada(Config conf, Date data) {
    try {
        String media;
        String nomeArquivo = conf.retornaNomeDoArqGrafico (this,data);
        RandomAccessFile arquivo = new RandomAccessFile (nomeArquivo,
"rw");
        media = arquivo.readUTF();
        return media;
    }
    catch (IOException e) {
        return null;
    }
}

// le o valor da media de entrada a partir de uma URL
public String leiaMediaEntradaUrl (Config conf, Date data) {
    try {
        String media;
        URL url = new URL (conf.retornaNomeDaUrlGrafico (this, data));
        DataInputStream file = new DataInputStream (url.openStream());
        media = file.readUTF();
        return media;
    }
    catch (IOException e) {
        return null;
    }
}

// le o valor da media de entrada a partir de uma URL
public String leiaMediaSaida(Config conf, Date data) {
    try {
        String media;
        String nomeArquivo = conf.retornaNomeDoArqGrafico (this,data);
        RandomAccessFile arquivo = new RandomAccessFile (nomeArquivo,
"rw");
        media = arquivo.readUTF();
        media = arquivo.readUTF();
        return media;
    }
    catch (IOException e) {
        return null;
    }
}

// le o valor da media de entrada a partir de uma URL

```

```

public String leiaMediaSaidaUrl (Config conf, Date data) {
    try {
        String media;
        URL url = new URL (conf.retornaNomeDaUrlGrafico (this, data));
        DataInputStream file = new DataInputStream (url.openStream());
        media = file.readUTF();
        media = file.readUTF();
        return media;
    }
    catch (IOException e) {
        return null;
    }
}

public String retornaLabelEntrada() {
    return labelEntrada.trim();
}

public String retornaLabelSaida() {
    return labelSaida.trim();
}

// retorna string com o valor dos atributos separados por |
public String retornaEntradaNaLista() {

    String sa = roteador.trim() + " | " + inter + " | " + mensagem.trim()
                + " | " + labelEntrada.trim() + " | " + labelSaida.trim();
    return sa;
}

public String retornaMensagem() {
    return mensagem;
}

public int retornaInterface() {
    return inter;
}

public String retornaRoteador() {
    return roteador.trim();
}

public int retornaTipo() {
    return tipo;
}

// le o valor dos atributos armazenados em um arquivo sequencial
public void le(RandomAccessFile arq) throws IOException
{
    byte bl[] = new byte [40];
}

```

```

    arq.readFully(b1);
    roteador= new String(b1,0);
    inter = arq.readInt();
    tipo = arq.readInt();
    byte b2[] = new byte [50];
    arq.readFully(b2);
    mensagem = new String(b2,0);
    byte b3[] = new byte [40];
    arq.readFully(b3);
    labelEntrada = new String(b3,0);
    byte b4[] = new byte [40];
    arq.readFully(b4);
    labelSaida = new String(b4,0);
}

// grava o valor dos atributos em um arquivo
public void grava(RandomAccessFile arq) throws IOException
{
    byte b1[] = new byte [40];
    if (roteador!=null )
        roteador.getBytes(0, roteador.length(),b1, 0 );
    arq.write(b1);
    arq.writeInt(inter);
    arq.writeInt(tipo);
    byte b2[] = new byte [50];
    if (mensagem!=null )
        mensagem.getBytes(0, mensagem.length(),b2, 0 );
    arq.write(b2);
    byte b3[] = new byte [40];
    if (labelEntrada!=null )
        labelEntrada.getBytes(0, labelEntrada.length(),b3, 0 );
    arq.write(b3);
    byte b4[] = new byte [40];
    if (labelSaida!=null )
        labelSaida.getBytes(0, labelSaida.length(),b4, 0 );
    arq.write(b4);
}

// obtem o valor da taxa de saida do trafego em um link TCP/IP
private ResultString retornaSNMPSaida(SnmpMan snmp)
{
    String varSaida=".1.3.6.1.2.1.2.2.1.16." + inter;
    ResultString var1= snmp.retornaValor(roteador,varSaida);
    return var1;
}

// obtem o valor da taxa de entrada do trafego em um link TCP/IP
private ResultString retornaSNMPEntrada(SnmpMan snmp)
{
    String varEntrada=".1.3.6.1.2.1.2.2.1.10." + inter;
    ResultString var1= snmp.retornaValor(roteador,varEntrada);
    return var1;
}

```

```

// obtem a taxa de entrada e saida do link, utilizando o protocolo SNMP, e
//armazena em uma base de dados relacional.
public boolean GravaValorSNMP(SnmpMan snmp, Config conf )
{
    System.out.println("INICIO DA CHAMADA SNMP");
    boolean ERRO=false;
    boolean SemErro=true;
    ResultString ea, sa;
    String aux;
    Date now = new Date();
    if (primeiraColeta){
        tempoAnt=now.getTime();
        sa=retornaSNMPSaida(snmp);
        if (sa.erro)
        {
            System.err.println(sa.retornaMensagemDeErro());
            return ERRO;
        }
        ea=retornaSNMPEntrada(snmp);
        if (ea.erro)
        {
            System.err.println(ea.retornaMensagemDeErro());
            return ERRO;
        }
        valorSaidaAnt=new Double(sa.retornaResultado());
        valorEntradaAnt=new Double(ea.retornaResultado());
        primeiraColeta= false;
        return SemErro;
    }
    else{
        long tempoAtual=now.getTime();
        sa=retornaSNMPSaida(snmp);
        if (sa.erro)
        {
            System.err.println(sa.retornaMensagemDeErro());
            return ERRO;
        }
        ea=retornaSNMPEntrada(snmp);
        if (ea.erro) {
            System.err.println(ea.retornaMensagemDeErro());
            return ERRO;
        }
        Double valorSaidaAtual=new Double(sa.retornaResultado());
        Double valorEntradaAtual=new Double(ea.retornaResultado());
        long temp= tempoAtual - tempoAnt;
        long NumeroDeBytesSaida= valorSaidaAtual.longValue() -
            valorSaidaAnt.longValue();
        if ( NumeroDeBytesSaida < 0 ) {
            valorSaidaAnt=valorSaidaAtual;
            valorEntradaAnt=valorEntradaAtual;
            tempoAnt= tempoAtual;
            System.err.println(" Numero De Bytes Saida menor que zero");
            return ERRO;
        }
        long NumeroDeBytesEntrada= valorEntradaAtual.longValue() -
            valorEntradaAnt.longValue();

        if ( NumeroDeBytesEntrada < 0 ) {

```

```

        valorSaidaAnt=valorSaidaAtual;
        valorEntradaAnt=valorEntradaAtual;
        tempoAnt= tempoAtual;
        System.err.println(" Numero De Bytes da Entrada menor que zero:
");
        return ERRO;
    }
    if ( temp < 0 ) {
        valorSaidaAnt=valorSaidaAtual;
        valorEntradaAnt=valorEntradaAtual;
        tempoAnt= tempoAtual;
        System.err.println(" Medida de Tempo menor ou igual a zero");
        return ERRO;
    }
    long taxaSaida = (NumeroDeBytesSaida) * 8 / temp;
    long taxaEntrada = (NumeroDeBytesEntrada) * 8 / temp;
    valorSaidaAnt=valorSaidaAtual;
    valorEntradaAnt=valorEntradaAtual;
    tempoAnt= tempoAtual;
    System.out.println("taxa saida: "+ taxaSaida + " TEMPO:" +temp);
    System.out.println("taxa Entrada: "+ taxaEntrada + " TEMPO:"
+temp);
    System.out.println(now.toString());
    String sInter= ""+ inter;
    BaseDeDados baseDeDados= new BaseDeDados(conf);
    ResultQuery res=baseDeDados.armazenaMedias(
        taxaEntrada,taxaSaida,now,roteador, sInter);
    System.out.println("-----");
}
return SemErro;
}
}

```

---

```

import java.io.*;
import java.lang.*;
import java.util.*;
import java.net.URL;
import java.util.Date;
import tempjava.sql.*;
import ibm.net.sql.*;
import SnmpMan;
import ResultString;
import Config;
import ResultQuery;
import BaseDeDados;
import Linha;

```

```

// classe para manipular um conjunto de links TCP/IP
class CadastroLinhas{
    RandomAccessFile arq;
    DataInputStream data;
    String nomeArq = "saida.dat";
    Vector listaLinhas;
    BaseDeDados baseDeDados;
    Config conf;

// constructor
    public CadastroLinhas()

```

```

    {
        conf= new Config();
        nomeArq = conf.retornaArqLinhas();
        BaseDeDados baseDeDados = new BaseDeDados(conf);
        listaLinhas=new Vector();
        System.out.println("nome arq linhas" + nomeArq );
    }

// adiciona um novo link TCP/IP a lista
public void AdicionaLinha( Linha lin ) {
    listaLinhas.addElement( lin);
}

// apresenta o os atributos de todos os objetos linha
public void mostra() {
    for(Enumeration e=listaLinhas.elements();e.hasMoreElements(); ) {
        Linha lin= (Linha)e.nextElement();
        lin.mostra();
    }
}

// apresenta o trafego medio em todos os links
public void mostraMedia()
{
    for(Enumeration e=listaLinhas.elements();e.hasMoreElements(); ) {
        Linha lin= (Linha)e.nextElement();
        lin.mostraMedia(conf);
    }
}

// Grava o valor dos atributos dos objetos em um arquivo sequencial
public void GravaValorSNMP(SnmpMan snmp)
{
    for(Enumeration e=listaLinhas.elements();e.hasMoreElements(); ) {
        Linha lin= (Linha)e.nextElement();
        if ( !lin.GravaValorSNMP(snmp,conf))
            System.err.println("Ocorreu um erro obtencao de um valor SNMP");
    }
}

// remove um link TCP/IP
public void removeLinha( int i )
{
    listaLinhas.removeElementAt(i);
}

// retorna um identificar do objeto desejado
public Linha retornaLinha(int i) {
    return (Linha)listaLinhas.elementAt(i);
}

// grava os valores dos atributos de todas as linhas em um arquivo
sequencial
public void grava( )
{
    try
    {
        arq = new RandomAccessFile(nomeArq, "rw");
        int ne= listaLinhas.size();
    }
}

```

```

//String s ="Numero de Elementos : " + ne;
//System.err.println( s);
arq.writeInt(listaLinhas.size());
for(Enumeration e=listaLinhas.elements();e.hasMoreElements();) {
    Linha lin= (Linha)e.nextElement();
    lin.grava(arq);
}
arq.close();
}
catch (IOException e){
    System.err.println( e.toString() );
    System.exit( 1);
}
}

// grava o valor do atributo de de todas as linhas em um arquivo sequencial
public void le()
{
    try
    {
        arq = new RandomAccessFile(nomeArq, "rw");
        int tam=arq.readInt();
        for (int i=0;i<tam;i++){
            Linha lin= new Linha();
            lin.le(arq);
            AdicionaLinha(lin);
        }
        arq.close();
    }
    catch (IOException e){
        System.err.println( e.toString() );
        System.exit( 1);
    }
}

// grava o valor do atributo de todas as linhas em um arquivo sequencial
utilizando uma URL.
public void le2( )
{
    try
    {
        URL url = new URL (conf.retornaUrlLinhas());
        DataInputStream file = new DataInputStream (url.openStream());
        int tam = file.readInt();
        System.out.println (conf.retornaUrlLinhas());
        System.out.println (tam);
        //arq = new RandomAccessFile(nomeArq, "rw");
        //int tam=arq.readInt();
        for (int i=0;i<tam;i++){
            Linha lin= new Linha();
            lin.le(file);
            AdicionaLinha(lin);
        }
        file.close();
    }
    catch (IOException e){
        System.err.println( e.toString() );
        System.exit( 1);
    }
    catch (NullPointerException ex) {}
}
}

```



```

// Gera um arquivo texto com as medias graficas para o grafico a ser
plotado
public void GeraArqGrafico (Config conf) {
    for(Enumeration e=listaLinhas.elements();e.hasMoreElements();) {
        Linha lin = (Linha)e.nextElement();
        Date data = new Date();
        ResultQuery res = lin.retornaMedias(conf,data);
        System.out.println ("Gravando medias");
        if (res!=null) {
            try {
                String nomeArquivo = conf.retornaNomeDoArqGrafico
(lin,data);
                System.out.println (nomeArquivo);
                RandomAccessFile arquivo = new RandomAccessFile
(nomeArquivo,"rw");
                lin.gravaMedias (res,arquivo);
                arquivo.close();
            }
            catch (IOException ex) {
                System.out.println ("Erro em arquivo em
GeraArqGrafico!");
            }
        }
    }
}

void AdicionaElementosAListBox( java.awt.List lb)
{
    for(Enumeration e=listaLinhas.elements();e.hasMoreElements();) {
        Linha lin= (Linha)e.nextElement();
        String sa= lin.retornaEntradaNaLista();
        String sal = sa;
        System.out.println(sal);
        lb.addItem(sal);
    }
}

public Vector retornaListaLinhas()
{
    return listaLinhas;
}
}

```

## 10 Anexo C: Código Fonte da Classe *ComputerImpl* e *ClusterImpl*

```
// A classe ComputerImpl contem a implementacao da interface Computer
package agente;
import java.lang.Object;
import java.io.*;
import java.lang.*;
public class ComputerImpl implements Computer {
    public boolean contab=false;
    public ComputerImpl()
    {
    }

    public void startContab()
    {
        contab=true;
    }

    public void stopContab()
    {
        contab=false;
    }

    public boolean getStatusContab()
    {
        return contab;
    }

// obtem a taxa de ociosidade de CPU de uma maquina
    public String GetCPU()
    {
        String aux2=null;
        Integer aux = new Integer(50);
        try {
            Process p = bjecto foi
instancRuntime.getRuntime().exec("/u/loadl/agente/bojangles/agente/obtemcpu
1 1");
            try {
                RandomAccessFile inputFile = new
RandomAccessFile("/tmp/sarT", "r");

                String aux1 = null;

                while ((aux2 = inputFile.readLine()) != null) {
                    aux1=aux2;
                }
                System.out.println(aux1);
                aux2=aux1.substring(aux1.length()-2,aux1.length());
                System.out.println(aux2);
                Integer aux3=Integer.valueOf(aux2);
                aux=aux3;
                System.out.println("Computer.getCPU:"+ aux2);
                inputFile.close();
            } catch (FileNotFoundException e) {
                System.err.println("FileStreamsTest: " + e);
            } catch (IOException e) {

```

```

        System.err.println("FileStreamsTest: " + e);
    }
} catch (Exception e) {
    System.out.println("Computer.getCPU: ERRO!!!!");
    e.printStackTrace();
}

return aux2;
}

public String getSystemName()
{
    //-----
    boolean achou=false;
    String aux2=null;
    String aux1=null;
    try {
        RandomAccessFile inputFile = new
RandomAccessFile("/tmp/nomeSistema", "r");

        aux1 = null;
        aux2 = null;

        while ((aux2 = inputFile.readLine()) != null) {
            aux1=aux2;
        }
        System.out.println(aux1);
        aux2=aux1;
        System.out.println(aux2);
        inputFile.close();
        achou=true;
    } catch (IOException e) {
        System.err.println("FileStreamsTest: " + e);
    }

    return aux2;

    //-----
}
}

```

---

```

// A classe ClusterImpl contem a implementacao da interface Cluster
package agente;
import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;
import COM.ibm.corba.*;
import COM.ibm.corba.IIOP.*;
public class ClusterImpl implements Cluster{
    public boolean contab = false;
    public Vector maquinas;
    public int numeroDeMaquinas=0;
    public String arquivo =
"/u/loadl/agente/bojangles/agente/hostfile.doc";
    String aux=null;

```

```

public ClusterImpl()
{
    maquinas=new Vector();
    getHostnames();
}

public boolean getStatusContab()
{
    return contab;
}

public void startContab()
{
    contab=true;
}

public void stopContab()
{
    contab=false;
}

// obten a taxa de ociosidade de uma estacao individualmente
public String getCPUStation(int station)
{
    String cpu=null;
    boolean remoto=true ;
//    boolean remoto= station!=0;
    String serverName= (String) maquinas.elementAt(station);
    System.out.println("Computer.GetCPUStation: server name:"+
serverName);
    cpu= getCPU(serverName.trim(), remoto);
    return cpu;
}

// obten a taxa de ociosidade de todas as maquinas pertencentes ao cluster
public String getClusterCPU()
{
    String serverName;
    int i=0;
    String cpu=null;
    String cpuCluster="";
    for(i=0; i< numeroDeMaquinas; i++ ) {
        cpu=getCPUStation(i);
        if (cpu==null){
            cpu="-1";
        }
        cpuCluster=cpuCluster+ cpu.trim() + ";";
        System.out.println("getClusterCPU: "+ i);
    }
    return cpuCluster;
}

// obten o nome de todas as maquinas que compoem o ambiente
public String getClusterHostnames()
{
    String aux1="";

```

```

String aux2;
for(Enumeration e=maquinas.elements();e.hasMoreElements(); ) {
    aux2= (String)e.nextElement();
    aux1=aux1.trim()+aux2.trim()+";";
    System.out.println(aux1);
}
return aux1;
}

private void getHostnames()
{
String aux=null;
try {
    File inputFile = new File(arquivo);
    FileInputStream fis = new FileInputStream(inputFile);
    DataInputStream fe = new DataInputStream(fis);
    numeroDeMaquinas=0;
    while ((aux = fe.readLine()) != null) {
        maquinas.addElement( aux.trim());
        numeroDeMaquinas++;
        System.out.println(aux);
    }
    fis.close();
} catch (IOException e) {
    System.err.println("FileStreamsTest: " + e);
}
}

// obtem o valor da taxa de ociosidade de uma maquina
// utilizando o IIOP/CORBA
private String getCPU(String serverName, boolean remoto)
{
String cpu=null;
try
{
String URL=null;
String aux=null;
String serverPort="3333";
URL = new String("IIOP://" +serverName.trim()+":"+serverPort);
String hostURL = new String(URL);
Computer computer = null;
ORB client = CORBA.ORB_init(null);
computer = (Computer)client.URLToObject(hostURL+
"/computer" +
"?agente.Computer" );
System.out.println("Cluster.getCPU:verificando esta ativa"+
serverName);
if (computer.estaAtivo(remoto))
{
System.out.println("Cluster.getCPU: esta ativa "+
serverName);
cpu= computer.GetCPU();
}
else
System.out.println("Cluster.getCPUobjeto nao instanciano na
maquina "+serverName);
}
}

```

```
        client.close();
    }
    catch (Exception e){
        System.err.println("-----Errrrrooooo-----");
        e.printStackTrace();
    }
    return cpu;
}
public boolean estaAtivo(boolean remoto)
{
    boolean ativo=true;
    try {
        RandomAccessFile inputFile = new
RandomAccessFile("/etc/ativo.xyz", "r");

        String aux = null;

        aux = inputFile.readLine();
        if (remoto){
            ativo=false;
        }

        inputFile.close();
    } catch (Exception e) {
        System.out.println("Arquivo ativo.xyz nao foi aberto
");
    }
    return ativo;
}

// FIM -----
}
```

## 11 Bibliografia

1. BARILLAUD, Frank; DERI, Luca; FERIDUN, Metin. Network management using internet technologies. *Fifth IFIP/IEEE International Symposium on Integrated Network Management*, San Diego, CA, USA, p. 61-70, 1997.
2. BERNHARDT, Martin. *Design and implementation of a web-based tool for ATM connection management*. Stuttgart, Aug. 1996. Master's Thesis – Department of Computer Science, University of Stuttgart.
3. COAD, Peter; YOURDON, Eduward. *Análise baseada em objetos*. Rio de Janeiro: 1992.
4. COMER, D. E.; STEVENS, D. L. *Internetworking with TCP/IP*, v. 1. New Jersey: Prentice Hall, 1992.
5. DACONTA, Michael. *Java for C/C++ programmers*. USA: Ellist, 1996.
6. EVANS, Eric; ROGERS, Daniel. Using java applets and CORBA for multi-user distributed applications. *IEEE Internet Computing*, May/June. 1997, <http://computer.org/internet/>.
7. GUNTHER, Oliver; MULLER, Rudolf; SCHIMDT, Peter. MMM: a web-based system for sharing statistical computing modules. *IEEE Computing*; May/June. 1997, <http://comuter.org/internet/>.
8. HERMAN, James. *Web-Based net management is coming*. *Data Communications International*, New York, Oct. 1997, p. 138-141.
9. HUHNS, Michael N.; SINGH, Munindar . Conversational Agents. *IEEE Computing*; Mar./Apr.. 1997, <http://computer.org/internet/>.
10. HUHNS, Michael N.; SINGH, Munindar . Mobile agents. *IEEE Computing*; May/June. 1997, <http://computer.org/internet/>.
11. IBM Database Server for AIX version 4: *Up and Running*. IBM Corporation, 1996,

USA.

12. KINIRY, Joseph; ZIMMERMAN, Daniel. A hands-on look at java mobile agents. *IEEE Computing*; July/Aug. 1997, <http://computer.org/internet/>.
13. KRULWICH, Bruce. Automating the internet: agents as user surrogates. *IEEE Computing*; July/Aug. 1997, <http://computer.org/internet/>.
14. KOCK, Fernando Luiz. *Agentes autônomos para o gerenciamento de redes de computadores*. Florianópolis, dez. 1997. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística – Universidade Federal de Santa Catarina.
15. LAPPINEN, Mika; PULKKINEN, Pekka; RAUTIAINEN, Aapo. Java and CORBA based network management. *IEEE Computer*, June. 1997, <http://dlib.computer.org/>.
16. LARSEN, Amy. The next web wave: network management. *Data Communications*, p. 31-34, jan. 1996.
17. MILLER, Mark A. *Managing interworks with SNMP: the definitive guide to simple network mangement protocol, SNMPv2, RMON, and RMON2*. NewYork, M&T, 1997.
18. MASTON, M. C. Using the world wide web and java for network service management. *Fifth IFIP/IEEE International Symposium on Integrated Network Management*, San Diego, CA, USA, p. 71-84, 1997.
19. NEWMAN, Alexander. *Usando java: o guia de referência mais completo*. Rio de Janeiro: Campus, 1996.
20. ORFALI, Robert; HARKEY, Dan. *Client/Server programming whith Java and CORBA*. New York : John Wiley & Sons, 1997.
21. ORFALI, Robert; HARKEY, Dan; EDWARDS, Jeri. *The essential distributed objects survival guide*. NewYork : John Wiley & Sons, 1996.
22. OTTE, Randy; PAUL, Patrick; ROY, Mark. *Understanding CORBA*. New Jersey : Prenci Hall, 1996



23. REED, B. Distributed systems management on the web. *IFIP/IEEE International Symposium on Integrated Network Management*, San Diego, p. 85-95, 1997.
24. ROSE, M. T. *The simple book – an introduction to management of TCP/IP*. New Jersey, 1991.
25. SNMP-FAQ/PART 1. <http://www.snmp.com/FAQs/snmp-faq-part1.txt>. 1997.
26. SNMP-FAQ/PART 2. <http://www.snmp.com/FAQs/snmp-faq-part2.txt>. 1997.
27. UNIVERSIDADE FEDERAL DO PARANÁ. Biblioteca Central. *Normas para apresentação de trabalhos : referências bibliográficas*. 6 ed. Curitiba, 1996.
28. VIEIRA, Elvis Melo. *Métodos para desenvolver agentes adaptativos em gerência de redes usando redes neurais*. Florianópolis, fev. 1997. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística – Universidade Federal de Santa Catarina.

## 2 Glossário

**Agente** – dentro do contexto deste trabalho, agente consiste em um processo que recebe invocações da aplicação gerente, executa as operações solicitadas e retorna os possíveis resultados para a aplicação solicitante. O agente também pode notificar ao gerente, a ocorrência de algum evento nas máquinas gerenciadas.

**AIX** – Sistema Operacional UNIX desenvolvido pela IBM.

**API** – *Application Programming Interface* – consiste em um conjunto de rotinas para realizar a interface com o usuário.

**Aplicação Java** – dentro do contexto deste trabalho, uma aplicação java pode tanto ser um *applet*, como um *application* java.

**Applets Java** – são aplicações desenvolvidas em java destinadas a serem executadas em *browsers Web*. A classe principal da aplicação deve ser derivada da classe *Applet*.

**Applications Java** - é uma aplicação implementada java, destinada a ser executada a partir de um interpretador java. A execução de um *application* java inicia na função *main* da classe principal.

**AWT** – *Abstract Window Toolkit* - consiste em conjunto de classes para a geração e manipulação de dispositivos de interface em java.

**Browsers Web** – *software* responsável pelo *download* e visualização de páginas *Web*;

**CGI** – *Common Gateway Interface* – é um protocolo que permite criar páginas *Web* dinâmicas baseado nas informações a partir de botões, listas de seleção e outros campos de formulário HTML. O CGI permite a execução de programas em um servidor e a emissão de parâmetros para o mesmo.

**Cmu-SNMP** – *software* de domínio público que permite a emissão e recebimento de primitivas SNMP.

**CORBA** – *Common Object Request Broker Architecture*- é um padrão definido pelo OMG que permite a comunicação entre aplicações localizadas em máquinas diferentes e implementadas, utilizando diferentes linguagens de programação.

**Daemon** – consiste em um programa que fica constantemente sendo executando e que normalmente é implementado através de um *looping* infinito. Geralmente, um *daemon* executam funções de servidores como o: *sendmail*, *named*, entre outros.

**Driver** - Dispositivo de *software* instalado em uma máquina para permitir a

comunicação da máquina com um determinado *software* ou *hardware*.

**GIOP** – *General Inter-ORB Protocol* - consiste em um protocolo que especifica um conjunto de formatos de mensagens e estruturas de dados para a comunicação entre ORBs.

**HTTP** - *Hypertext Transfer Protocol* - é basicamente um serviço TCP/IP orientado a conexão, baseado num mecanismo de transferência de arquivos. O HTTP viabiliza a transferência de páginas *Web* do servidor *Web* para o *browser Web*.

**HTML** – *Hypertext Markup Language* - é uma linguagem especializada, dedicada à exibição e acesso de páginas *Web*.

**IIOP** – *Internet Inter-ORB Protocol* - protocolo que especifica como mensagens GIOP são transmitidas sobre uma rede TCP/IP.

**Internet** – grande rede mundial composta pela união de milhares de subredes através do TCP/IP.

**Intranets** - Termo utilizado para identificar redes locais que utilizam as ferramentas e protocolos oferecidos pela Internet para a sua composição.

**IP** – *Internet Protocol* – Protocolo da camada de rede utilizado na Internet.

**JDBC** - consiste em uma API implementada em Java que permite um acesso uniforme e padronizado a qualquer servidor de banco de dados delacional, que tenha suporte ao *driver* JDBC.

**JDK** – Java Development Kit – consiste em conjunto de ferramentas disponíveis para domínio público, para construção de *applets* e *applications* Java. Pode-se citar como exemplo destas ferramentas: o compilador *javac*, o interpretador *java*, o utilitário *javadoc*, entre outros.

**LAN** – *Local Area Network* - Termo utilizado para identificar uma rede local de computadores.

**NetView 6000** - plataforma de gerenciamento desenvolvida pela IBM.

**Links TCP/IP** – para o presente trabalho, *link* TCP/IP consiste em ligação *Wan* entre instituições na Internet (ou IntraNets), normalmente realizada através do aluguel de uma linha telefônica dedicada e da utilização de algum protocolo de comunicação ponto-a-ponto ( como PPP, HDLC, etc) para realizar a comunicação entre instituições.

**MIB** – *Management Information Base* – Uma coleção de objetos gerenciados que podem ser acessados via protocolo de gerência de Redes.

**MIB-2** – consiste em uma MIB definida pelo RFC 1213 destinada a monitorar equipamentos na Internet, utilizando o protocolo SNMP.

**OMG** – *Object Management Group* – consiste em grupo de mais de 700 companhias destinado a promover o desenvolvimento prático e teórico da tecnologia de objetos para

sistemas de computação distribuída. O OMG é responsável pela definição da padronização do CORBA.

**ORB** – *Object Request Broker* - é o meio pelo qual se estabelece a relação cliente/servidor entre objetos. Usando um ORB um objeto cliente pode invocar um método no servidor de modo transparente. Este servidor pode estar na mesma máquina ou em outro sistema na rede.

**ORB Core** – é o núcleo do ORB e o componente, dentro da arquitetura CORBA, que efetivamente transporta as mensagens entre os objetos.

**PDU** – *Protocol Data Unit* – Unidade básica de um protocolo, usualmente contendo uma informação de controle de protocolo e de dados de usuário.

**RMON** - *Remote Network Monitoring* – é uma MIB destinada ao gerenciamento remoto de redes.

**RMON-2** - *Remote Network Monitoring version 2* – é uma mib para o gerenciamento remoto de redes que permite o gerenciamento da camada de redes, transporte e aplicações.

**Router-Stats** – *software* de domínio público para obtenção de estatísticas de *links* Internet, baseado da geração de arquivos de imagem e páginas HTML.

**SNMP** – *Simple Network Management Protocol* - Protocolo de gerenciamento do modelo de gerência Internet.

**SQL** - *Structured Query Language* – linguagem utilizada para consultas padronizadas a um banco de dados relacional.

**TCP** – *Transmission Control Protocol* - Protocolo de transporte que oferece um serviço orientado a conexão.

**WWW** – *World Wide Web* – consiste em um conjunto de informações, protocolos, *softwares* e *conexões* que permitem o acesso a informações na rede, através de uma *URL*, utilizando técnicas de *multimedia* e *hipertexto*.