UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hugo André Klauck

PROPOSTA DE UM AMBIENTE DE GERÊNCIA DE REDES DE ALTA VELOCIDADE UTILIZANDO CORBA, JAVA E HTML

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Becker Westphall

Florianópolis, Abril de 2000

PROPOSTA DE UM AMBIENTE DE GERÊNCIA DE REDES DE ALTA VELOCIDADE UTILIZANDO CORBA, Java e HTML

Hugo André Klauck

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas do Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Carlos Becker Westphall

Fernando Álvaro Ostuni Gauthier

Banca Examinadora

Carlos Becker Westphall

Bernardo Gonçalves Riso

João Bosco Mangueira Sobral

AGRADECIMENTOS

Agradeço a meus pais Lourdes e Geraldo pela vida, pela educação que me deram e a minha Irma Lílian;

À Claudete, minha amada esposa;

Agradeço a minha tia Antonia, , aos meus tios Vilmar e Salete;

Agradeço ao Professor Carlos Becker Westphall pela orientação;

Aos meus colegas de curso;

Aos meus amigos;

E principalmente, Deus.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Atualmente os sistemas de gerência de redes são poderosos, porém possuem plataformas centralizadas e proprietárias, limitando com isso, o controle em decorrência do grande crescimento das redes de computadores. Este fator gera a necessidade de explorarmos novas tecnologias que podem ser usadas para o projeto de um ambiente de gerência de redes.

Os recentes avanços na área de objetos distribuídos utilizando o CORBA (Common Object Request Broker Architeture) e o grande crescimento da Internet e a grande utilização da linguagem de programação Java surgem como poderosos mecanismos para o desenvolvimento de aplicações distribuídas que podem ser utilizadas com grande sucesso neste projeto de gerência de redes.

Este trabalho descreve a utilização dessas tecnologias para o projeto de um ambiente de gerência de redes de computadores distribuída, que é independente de plataforma e utiliza páginas Web para realizar a interface com o usuário final.

ABSTRACT

Corrently Networks Management Systems are powerfull, however maintain centralized and proprietary plataforms, limiting this way, the control in result of the great growth of the networks of computers. This factor generates the necessity to explore them new technologies to me that can be used for the design of an environment of management of networks.

The recents advances in the area of distributed objects using the CORBA and the great growth of the Internet and the great use of the programming language Java appear as powerful mechanisms for the development of distributed applications that can be used with great success in this design of networks management.

This work describes the use of these technologies for the design of an environment of management of networks of computers distributed, that is independent of platform and uses Web pages to carry through the interface with the final user.

1 INTRODUÇÃO

Através dos avanços das tecnologias de interconectividade e dos benefícios proporcionados pelas Redes de Computadores, cada vez mais computadores são interconectados nas organizações. Paralelamente, a diminuição dos custos dos equipamentos permite adquirir e agregar à rede cada vez mais equipamentos, de tipos diversos, tornando essas redes cada vez maiores e mais complexas. Com isso, as redes começaram a ser interconectadas muito rapidamente; redes locais conectadas a redes regionais, as quais, por sua vez, ligadas a backbones nacionais.

Hoje essas redes são extremamente importantes para o dia a dia de muitas empresas em todo o mundo, porque normalmente, junto com sua utilização, vem a eficácia e a competitividade. Essa importância vem crescendo de tal forma que as empresas têm se tornado altamente dependentes destas redes, sentindo imediatamente o impacto quando os seus recursos não estão disponíveis.

Este novo ambiente originou alguns problemas administrativos. As tarefas de configuração, identificação de falhas e controle de dispositivos e recursos da rede passaram a consumir tempo e recursos das organizações.

Cientes desse problema, a solução então passou a ser buscada na atividade de Gerência de Rede. Esta atividade passou a evoluir de forma rápida e concisa, sendo hoje uma das especialidades da área de Redes de Computadores que mais cresce.

De posse destas informações apresentamos este trabalho sobre Gerência de Redes. Num primeiro momento veremos a Gerência de Redes abordando as suas características, onde faremos uma pequena explanação do protocolo de gerenciamento SNMP destacando suas potencialidades, suas limitações e como é o seu funcionamento.

Também será mostrados um estudo feito no padrão CORBA indicando suas características, abordando seus protocolos para comunicação e, destacando a sua utilização na Gerência de Redes de Computadores.

A fim de estarmos atualizados nas principais tecnologias de transmissão de dados veremos o comportamento de uma MIB para redes ATM, destacando suas principais características.

Logo após, será apresentado a proposta para o desenvolvimento de uma Gerência de Redes de Computadores utilizando o padrão CORBA juntamente com a linguagem de programação Java e utilizando a tecnologia da Web através de páginas HTML para termos um ambiente de Gerência distribuído e independente de plataforma. Abordaremos nesta proposta os estudos já realizados sobre este tema, destacando suas características e particularidades e, em cima disto estão as principais motivações para o desenvolvimento deste trabalho.

Por fim será apresentado as conclusões chegadas, bem como perspectivas futuras que foram visualizadas no desenvolvimento deste trabalho.

2 GERÊNCIA DE REDES E O PROTOCOLO SNMP

Gerência de Redes de computadores pode ser conceituada como a coordenação (controle de atividades e monitoração do uso) de recursos materiais (modens, roteadores, switches, etc) e ou lógicos (e.g protocolos), fisicamente distribuídos na rede, assegurando, na medida do possível. confiabilidade, tempos de resposta aceitáveis e segurança nas informações. Portanto, gerenciar uma rede significa dotar este sistema de mecanismos de monitoramento e controle dos elementos de rede de tal sorte a permitir seu funcionamento dentro de uma Qualidade de Serviço (QoS) esperada pelos usuários da mesma [OLI98].

Uma implicação imediata no crescimento das redes de computadores é o aumento substancial de problemas e a conseqüente necessidade de gerência do maior número possível de recursos materiais e lógicos destas redes. É importante notar também que a complexidade desta gerência é provocada, principalmente, pela convivência de sistemas heterogêneos devido a existência de sistemas proprietários (IBM, DEC, CISCO, etc) e suas conseqüentes diferentes soluções de gerência. Assim, a complexidade na atividade de gerenciamento será tão maior quanto mais diversificados forem os equipamentos de rede fornecidos por diferentes fabricantes.

2.1 Etapas no Processo de Gerência

A primeira idéia na solução de um sistema de gerência consiste em se utilizar um computador interagindo com os diversos componentes da rede a serem gerenciados para deles extrair as informações necessárias a sua gerência.

Etapas no processo de gerência de redes:

- **Coleta de dados:** é um processo, em geral automático, que consiste de monitoração sobre os recursos gerenciados.

- **Diagnóstico:** esta etapa consiste no tratamento e análise realizados a partir dos dados coletados. O processo de gerenciamento executa uma série de procedimentos (por intermédio de um operador ou não) com o intuito de determinar a causa do problema representado no recurso gerenciado.
- **Ação:** uma vez diagnosticado o problema, cabe uma ação, ou controle, sobre o recurso, caso o evento não tenha sido passageiro (incidente operacional).

2.2 *Objeto Gerenciado*

Um dos principais conceitos em gerência de redes é o de **Objeto Gerenciado (Managed Object)**, o qual pode ser definido como uma representação abstrata das características relativas à gerência de um recurso real da rede(uma conexão, uma entidade de camada, um dispositivo, etc), participante da comunicação entre sistemas abertos.

Em geral, a definição de um Objeto Gerenciado apresenta dois aspectos:

- Onde ele se situa (localização dentro do sistema sendo gerenciado);
- A natureza (representada por seus atributos)

Assim, a noção de Objeto Gerenciado é utilizada na fase de modelagem de informações, para fazer abstração dos recursos de rede sobre os quais incidem as atividades de gerência. Em sendo o Objeto Gerenciado a representação de um recurso físico ou lógico de rede, qualquer elemento de rede não modelado como um Objeto Gerenciado é invisível ao sistema de gerenciamento.

Dois tipos de interações entre as aplicações de gerência e os objetos gerenciados são identificadas:

- A primeira é relativa às informações estáticas e corresponde ao uso das informações de gerência pelas aplicações como elas estão representadas no interior de uma Base de Informações de Gerenciamento (MIB – Management Information Base), isto é, as aplicações realizam suas funções com base nas informações diretamente recuperadas a partir dos objetos gerenciados.
- A segunda é relativo às informações dinâmicas e corresponde à manutenção da coerência das informações de gerência no interior da MIB. Os recursos reais podem gerar eventos que são transmitidos aos respectivos objetos gerenciados, os quais atualizam os atributos devidos e enviam relatórios de eventos para as aplicações interessadas. No outro sentido, quando uma aplicação envia uma ação para um

objeto gerenciado, este atualiza o atributo referenciado antes de aplicar a referida ação sobre o recurso real. Desse modo, uma visão coerente dos recursos gerenciados é garantida.

2.3 Paradigma Gerente x Agente

Devido a natureza distribuída dos recursos (elementeos de rede) a serem gerenciados, a gerência de redes é uma aplicação distribuída. Os processos usados nas atividades de gerência destes recursos distribuídos são classificados como processo **Gerente** e processo **Agente**.

- processo **Gerente** é a parte de uma aplicação distribuída associada ao usuário da Gerência (processo ou humano). Ele tem a responsabilidade de realizar operações de gerência (através do envio de comandos) sobre os Objetos Gerenciados, através dos processos Agentes. O Gerente recebe também notificações enviadas pelo Agente.

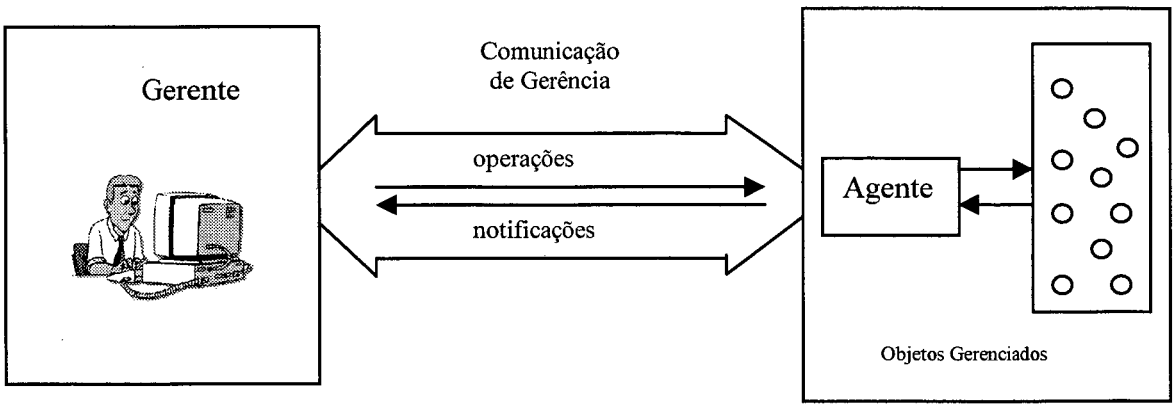

Figura 1- Modelo Gerente x Agente

- processo **Agente** é a parte de uma aplicação distribuída que irá executar sobre o Objeto Gerenciado os comando enviados pelo processo Gerente. Assim, ele passará para o Gerente uma visão dos objetos sendo gerenciados e refletirá o comportamento destes objetos, emitindo notificações sobre os mesmos.

Uma aplicação de gerência pode exercer o papel de Gerente, de Agente ou ambos. Um processo de gerência no papel de Agente atua sobre Objetos em seu ambiente local,

executando ações de gerência sobre estes objetos como conseqüência de operações enviadas pelo Gerente.

O Agente, em geral, não pode determinar o contexto onde se inserem as operações de gerência que ele executa ou das notificações que ele envia ao Gerente. Por exemplo, o Agente não tem como saber se as informações solicitadas sobre um controle de erro serão usadas para a gerência de falhas ou para a gerência de desempenho.

Processos Gerentes monitoram ou modificam as instâncias remotas de Objetos Gerenciados, bastando para tanto se comunicarem (através de um protocolo de gerência) com os Agentes, controlando desta forma os recursos de uma rede.

2.4 Protocolos de Gerência

É clara a necessidade de estabelecer monitoramento e controle sobre todos os componentes da rede, de forma a garantir que esta esteja sempre em funcionamento e que os problemas sejam identificados, isolados e solucionados o mais rápido possível. Entretanto, esta não é uma tarefa fácil. As redes têm assumido grandes proporções, com um grande número de computadores, além da constante adição de novos componentes, oferecendo integração dados/voz, multiplexadores e roteadores, além de tantos outros, o que tem adicionado mais complexidade a esse ambiente[SAM97].

Para atender a esta necessidade de gerenciamento foram desenvolvidos protocolos de gerenciamento. A principal preocupação de um protocolo de gerenciamento é permitir aos gerentes de rede realizar tarefas, tais como: obter dados sobre desempenho e tráfego da rede em tempo real, diagnosticar problemas de comunicação e reconfigurar a rede atendendo às mudanças nas necessidades dos usuários e do ambiente. Porém, vários obstáculos teriam que ser superados, entre eles a heterogeneidade dos equipamentos de rede (computadores, roteadores e dispositivos de meio), dos protocolos de comunicação e das tecnologias de rede. Adicionalmente, era necessário que esse gerenciamento fosse integrado, pois uma solução genérica e integrada auxiliaria os usuários a evitar os altos custos de uma solução específica, além de facilitar a manutenção, o monitoramento, o crescimento e a evolução da rede. Sem um gerenciamento integrado, a rede pode degradar até se tornar completamente ineficiente.

Ciente destas dificuldades, a ISO (*International Organization for Standardization*) vem desenvolvendo padrões para o gerenciamento de redes OSI (*Open Systems Interconnection*), tendo como protocolo de gerência o CMIP (*Common Management Information Protocol*). Do outro lado, existe o IEEE (*Institute of Electrical and Electronics Engineers*) com um conjunto de padrões para o gerenciamento de redes TCP/IP (*Transmission Control Protocol / Internet Protocol*), normalmente referenciados como SNMP (*Simple Network Management Protocol*). Atualmente, quase todas as plataformas de gerenciamento de redes Internet comercialmente disponíveis implementam o protocolo SNMP devido à sua simplicidade de implementação em relação ao CMIP.

Com o objetivo de suprir estas necessidades, a ISO (*International Organization for Standardization*) desenvolve padrões de gerenciamento que permitem a interoperabilidade de múltiplos e diversificados sistemas computacionais e redes de computadores.

Dentro deste contexto, a ISO dividiu a atividade de gerenciamento em cinco áreas funcionais específicas (*SMFA's - Specific Management Functional Areas*): Gerenciamento de Configuração, Gerenciamento de Falhas, Gerenciamento de Contabilização, Gerenciamento de Desempenho e Gerenciamento de Segurança. Dentro de cada Área Funcional, foram desenvolvidos padrões de funções (incluindo requisitos, modelos e serviços) para o gerenciamento das redes. Essas funções são processos de aplicação de gerenciamento que utilizam os serviços oferecidos pela camada de aplicação.

O SNMP, devido a sua natureza como uma solução rápida e reduzida do CMIP da ISO, implementa apenas um pequeno conjunto destas funções definidas pela ISO. Por isso ele é considerado "simples" de implementação e é um protocolo bem mais "leve" que o CMIP.

2.5 Protocolo SNMP

O SNMP foi desenvolvido nos anos 80 como resposta para os problemas de gerenciamento em ambiente TCP/IP Internet, envolvendo redes heterogêneas. Inicialmente foi concebido para ser apenas uma solução provisória até o completo desenvolvimento de um protocolo de gerenciamento mais completo, o CMIP (*Common Management Information Protocol*). Neste contexto, sem um protocolo melhor disponível, o SNMP passou a ser o protocolo mais utilizado.

O *Simple Network Management Protocol* (SNMP) é um protocolo da camada de aplicação, como mostrado na Figura 2, desenvolvido para facilitar a troca de informações de gerenciamento entre dispositivos de rede. Estas informações transportadas pelo SNMP (como pacotes por segundo e taxa de erro na rede), permitem aos administradores da rede gerenciar o desempenho da rede de forma remota, encontrando e solucionando os problemas da rede, e planejar o crescimento da rede.

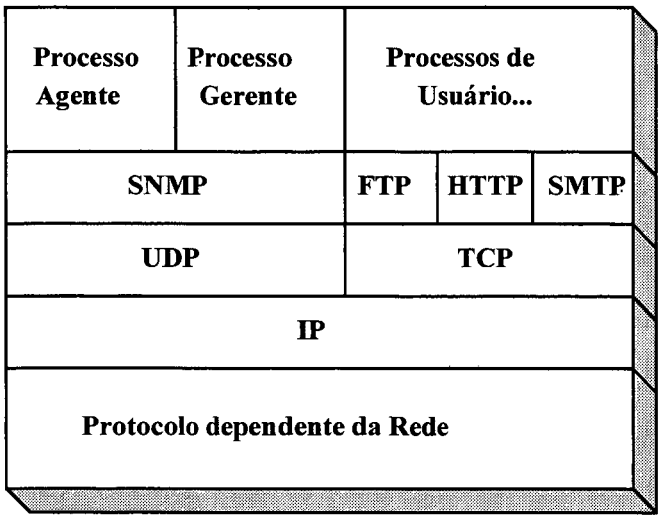

Figura 2 - Protocolo SNMP sobre as camadas do TCP/IP

Assim como o *Transmission Control Protocol* (TCP), o SNMP é um protocolo desenvolvido para a arquitetura Internet. Os padrões que definem a estrutura de gerenciamento de redes Internet, *Network Management Framework* (NMF), são descritos nos documentos: RFC 1155 - *Structure of Management Information (SMI)*; RFC 1156 - *Management Information Base (MIB)*, e RFC 1157 - *Simple Network Management Protocol (SNMP)*. O SNMPv1 NMF está definido nos RFCs 1155, 1157 e 1212; e o SNMPv2 NMF está definido nos RFCs 1441-1452.[RFC1157] [RFC1213] [SAM97] [TAN97] [BRI93]

Hoje o SNMP é o protocolo mais implementado nos produtos comerciais de gerenciamento assim como em universidades e organizações de pesquisa. Isso

A maior vantagem em usar o SNMP é o fato de que sua implementação é relativamente simples. Esta simplicidade torna mais fácil para o usuário programar as variáveis que gostaria de gerenciar. Para o SNMP cada variável consiste das seguintes informações:

- um título que identifica a variável.

- a estrutura de dado da variável (inteiro, string,...)
- o status da variável (*read-only* ou *read-write*)
- o valor da variável.

Outra vantagem do SNMP é a sua vasta utilização atualmente. Esta popularidade deve-se ao fato de que até agora nenhum outro protocolo melhor apareceu para substituir a implementação "provisória" do SNMP. O resultado disso é que a maioria dos vendedores de hardware/software para redes desenvolvem seus produtos para suportar o SNMP como protocolo de gerenciamento padrão.

A grande expansibilidade é outro benefício do SNMP. A relativa facilidade de implementação torna também mais fácil a atualização deste protocolo, visando atender as novas necessidades dos usuários de redes.

2.5.1 Elementos do SNMP

Agentes

No modelo de gerenciamento SNMP [BRI93], hosts, pontes, roteadores e hubs devem ser equipados com agentes SNMP para que possam ser gerenciados pela estação de gerenciamento NMS *(Network Management System)* através do gerente SNMP. O agente responde a requisições da estação de gerenciamento, que pode ser o envio de informações de gerência ou ações sobre as variáveis do dispositivo onde está.

O funcionamento desta estrutura só é possível graças ao acesso direto à MIB que o agente possui, pois todas as informações de gerência encontram-se lá. Ao receber uma mensagem SNMP do gerente, o agente identifica que operação está sendo requisitada e qual(is) a(s) variável(is) relacionadas, e a partir daí executa a operação sobre a MIB ; em seguida, monta uma nova mensagem de resposta, que será enviada ao gerente.

À primeira vista a comunicação do agente com o gerente pode parecer injusta, uma vez que o agente apenas responde ao que lhe é questionado. Mas há momentos em que o agente pode "falar" com o gerente sem que antes seja questionado. Isso ocorre quando o agente detecta, a partir da análise do contexto da MIB, alguma situação inesperada. Neste momento, o agente gera uma mensagem especial, o TRAP, e a envia ao gerente, relatando sobre a situação.

Para o tratamento destas exceções e o envio de TRAPs, é concedido ao agente um certo poder de decisão, cabendo a ele, a partir da análise do contexto da MIB, decidir se é

ou não necessário enviar o TRAP ao gerente. Esse poder de decisão é concedido ao agente para que em certas situações, como quando da inicialização do sistema, TRAPs desnecessários não sejam trafegados pela rede, o que, quando se tratando de dezenas de agentes, poderia interferir no desempenho global da rede. O envio de TRAPs será tratado com mais profundidade posteriormente.

Cabe ao agente um papel fundamental em todo o processo de gerenciamento da rede, acessando e disponibilizando informações de gerência contidas na MIB, além de indicar situações inesperadas de funcionamento do dispositivo ao qual gerencia através do envio de TRAPs ao gerente.

Gerentes

Servindo de interface entre as aplicações de gerência correntes no NMS e os agentes espalhados pelos dispositivos da rede está o gerente. Cabe ao gerente enviar comandos aos agentes, solicitando informações sobre variáveis de um objeto gerenciado, ou modificando o valor de determinada variável.

Os gerentes então processam estas informações colhidas pelos agentes, e as repassam à aplicação que as requisitou. A comunicação entre o gerente e as aplicações é possível através da utilização das APIs do gerente SNMP pelo sistema.

A API (*Application Program Interface*) é um conjunto de funções que intermediam a execução de comandos entre um programa e outro, de forma a simplificar a um programa o acesso a funções do outro programa, e que no caso do SNMP intermediam as execuções entre uma aplicação de gerência e o gerente SNMP.[TAN97]

Quando uma solicitação da aplicação de gerência chega ao gerente, através da API, o gerente mapeia esta solicitação para um ou mais comandos SNMP que, através da troca de mensagens apropriadas, chegarão aos agentes correspondentes. Devemos ressaltar que este comando SNMP montado pelo gerente pode ser enviado a um outro gerente, que pode ou não repassá-lo a um agente. Só que a comunicação gerente-gerente só é possível na versão estendida do SNMP, o SNMPv2, e não faz parte do SNMP padrão. Esta comunicação gerente-gerente será abordada com maiores detalhes mais adiante [TAN97].

Cabe também ao gerente encaminhar à aplicação de gerência os TRAPs que porventura sejam enviados pelos agentes. Assim, o software de gerência terá conhecimento da presença de um novo equipamento na rede ou do mal funcionamento de algum dos dispositivos da rede.

Quando da inicialização do software de gerência, nada se sabe sobre a configuração ou funcionamento da rede. Essas informações vão sendo conhecidas através de TRAPs que são enviados pelos agentes; a partir daí o gerente realiza *polling* [SAM97] a fim de manter a comunicação com os agentes, possibilitando ao software de gerência mapear, monitorar e controlar a rede.

Management Information Base (MIB)

No modelo SNMP, os recursos de uma rede são representados como objetos. Cada objeto é, essencialmente, uma variável que representa um aspecto do dispositivo gerenciado. Todos os objetos (variáveis) são armazenados na *Management Information Base* (MIB). A MIB do SNMP será abordada com maiores detalhes posteriormente.[BRI93][RFC1157]

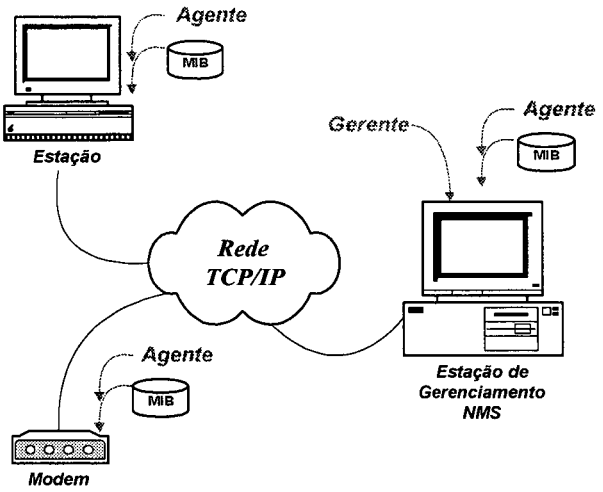

Figura 3 - Elementos básicos do SNMP

Estrutura da MIB

Todos os objetos gerenciados no modelo SNMP estão armazenados e dispostos numa estrutura hierárquica, mais conhecida como **árvore**. As folhas da árvore são os objetos gerenciados atuais, onde cada um representa algum recurso, atividade ou informação relacionada que deve ser gerenciada. A própria estrutura em árvore define o agrupamento de objetos em conjuntos relacionados. Esta estrutura pode ser visualizada na Figura 4 .[SAM97]

Para cada tipo de objeto na MIB é associado um identificador de tipo ASN.1, o OBJECT IDENTIFIER. O identificador serve para nomear o objeto. Como o valor

associado com o tipo OBJECT IDENTIFIER é hierárquico, a nomeação também serve para identificar a estrutura dos tipos de objeto.

O identificador é único para cada tipo de objeto. Seu valor consiste numa seqüência de inteiros. Iniciando pela raiz da árvore, cada parte inteira componente do valor do identificador representa um ramo da árvore. A partir da raiz, encontramos três ramos principais no primeiro nível da árvore: **iso**, **ccitt** e uma junção **iso-ccitt**. Sob o ramo iso uma sub-árvore é utilizada por outras organizações (org), uma das quais é o *U.S. Departament of Defense* (DoD).

A especificação MIB define os grupos de variáveis necessárias à monitoração e controle de vários componentes da rede. Para o SNMP, nem todos os grupos de variáveis definidas pela especificação MIB são obrigatórios. Um dos mais importantes e obrigatório para o SNMP é o grupo Internet. [BRI93]

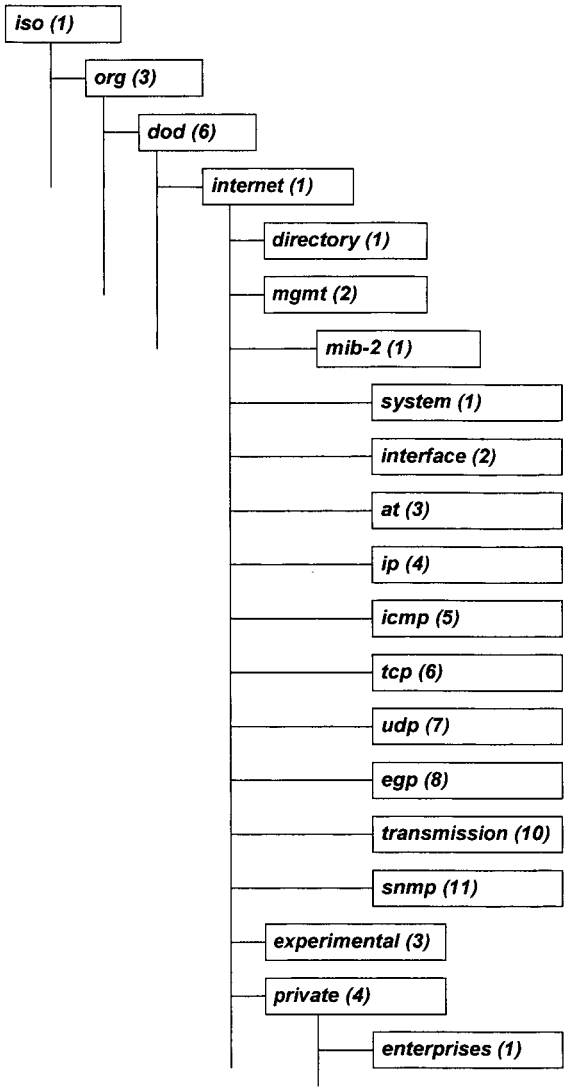

Figura 4-Estrutura em árvore de uma MIB SNMP

2.5.2 Operações SNMP aplicadas às variáveis da MIB

Get

O gerente SNMP envia o comando Get a um determinado agente toda vez que necessita recuperar uma informação de gerenciamento específica do objeto gerenciado pelo agente. Como já foi dito, estas informações encontram-se na forma básica de variáveis, que por sua vez estão na *Management Information Base* (MIB) do elemento de rede gerenciado. Logo, o comando Get, na verdade, solicita ao agente a leitura de determinada(s) variável(is) da MIB em questão. [RFC1157][SAM97]

GetNext

O comando GetNext assemelha-se ao comando Get, no entanto, enquanto o comando Get solicita ao agente a leitura de determinada instância de uma variável, ao receber um comando GetNext, o agente deve ler a próxima instância disponível, na ordem da MIB, da(s) variável(is) associadas.

GetBulk

O propósito desta PDU é minimizar o número de trocas de protocolos requeridos, para retornar uma grande quantidade de informações de gerenciamento. O GetBulkRequest PDU permite um gerente SNMPv2 requerer que a resposta seja tão grande quanto possível

Set

A operação Set requisita a um determinado agente a atribuição/alteração do valor de determinada(s) variável(is) de uma MIB. Alguns desenvolvedores acreditam que este comando não deve retornar um Response. Outros acham que a operação Set deve retornar alguma indicação de que a operação foi efetuada. Porém o mais correto seria que após cada operação Set sobre uma variável, uma operação Get fosse efetuada sobre a mesma variável a fim de assegurar que a operação Set foi efetuada.

Trap

A operação Trap difere de todas as outras. Ela é utilizada por um agente SNMP para notificar de forma assíncrona a um gerente algum evento extraordinário que tenha ocorrido no objeto gerenciado.

Responses

Como já descrito, sempre que um agente recebe um comando Get, GetNext ou Set ele tenta executar a operação associada e, conseguindo ou não, constrói uma outra mensagem que é enviada ao emissor da requisição. Esta mensagem é a GetResponse. Das operações SNMP, apenas o Trap não gera um Response.

2.5.3 O SNMP sobre a camada de Transporte

O protocolo SNMP foi desenvolvido para rodar sobre a camada de transporte, na camada de aplicação da pilha de protocolo TCP/IP. A maioria das implementações do SNMP utilizam o *User Datagram Protocol* (UDP), como protocolo de transporte. O UDP é um protocolo não-confiável, não garantindo a entrega, a ordem ou proteção contra duplicação das mensagens.

O SNMP utiliza o UDP pois foi desenvolvido para funcionar sobre um serviço de transporte sem conexão. A maior razão para isto é o desempenho. Utilizando um serviço orientado à conexão, como o TCP, a execução de cada operação necessitaria de uma prévia conexão, gerando um *overhead* significativo, o que é inaceitável na atividade de gerência onde as informações devem ser trocadas entre as entidades da forma mais rápida possível.

Segmentos UDP são transmitidos em datagramas IP. O cabeçalho UDP inclui os campos de origem e destino, e um *checksum* opcional que protege o cabeçalho UDP e os dados de usuário (em caso do *checksum* ser violado, a PDU é descartada).

Duas portas UDP são associadas às operações SNMP. As operações Get, GetNext, GetBulk, Inform e Set utilizam a porta 161. Por ser acionada em casos de exceção, a operação Trap têm reservada para si a porta 162. [SAM97]

Como o UDP é um protocolo não-confiável, é possível que mensagens SNMP sejam perdidas. O SNMP por si só não fornece garantia sobre a entrega das mensagens. As ações a serem tomadas quando da perda de uma mensagem SNMP não são abordadas pelo padrão. No entanto, algumas considerações podem ser feitas sobre a perda de mensagens SNMP. No caso de uma mensagem de Get, GetNext ou GetBulk, a estação de gerenciamento deve assumir que esta mensagem (ou o GetResponse correspondente) tenha sido perdida se não receber resposta num certo período de tempo. A estação de gerenciamento então pode repetir a mensagem uma ou mais vezes até que obtenha a resposta. Desde que um único *request-id* (que identifica a mensagem) seja utilizado para cada operação distinta, não haverá dificuldade em identificar mensagens duplicadas.

No caso de uma mensagem de Set, a recuperação deve provavelmente envolver o teste no objeto associado à operação através de uma Get sobre este mesmo objeto a fim de determinar se a operação Set foi ou não efetivada.

Uma vez que a operação Trap não gera uma mensagem de resposta, não é fácil identificar a perda de uma mensagem desse tipo.

A Figura 5 ilustra o contexto do protocolo SNMP na pilha de protocolo TCP/IP, utilizando o UDP como protocolo de conexão.

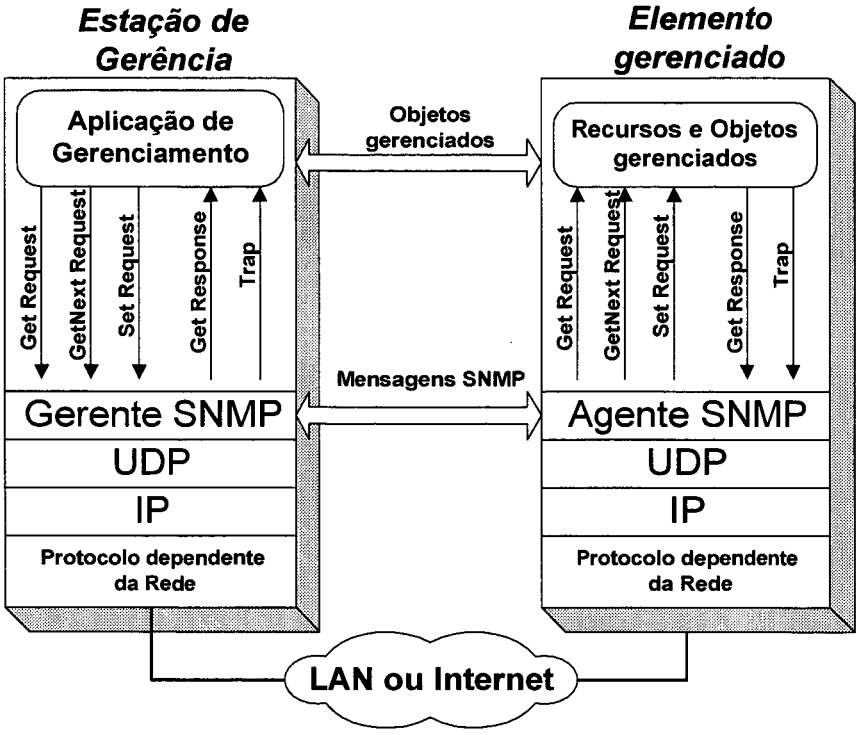

Figura 5-Protocolo SNMP sobre a camada de transporte

O SNMP requer alguns serviços da camada de transporte para que suas mensagens sejam transmitidas entre as entidades de gerenciamento. Isso independe da implementação da camada de transporte.

2.5.4 Formato da mensagem SNMP

No SNMP, as informações são trocadas entre os gerentes e agentes na forma de mensagens. Cada mensagem possui duas partes, um cabeçalho e uma *Protocol Data Unit* (PDU). O cabeçalho inclui um número de versão (*version*) que indica a versão do SNMP e um nome de comunidade (*community*). O nome de comunidade possui duas funções. Primeiro, o nome de comunidade define um dispositivo de acesso para um grupo de NMSs. Segundo, aqueles dispositivos cujo nome de comunidade é desconhecido são excluídos de operações SNMP; os gerentes também podem utilizar o nome de comunidade como uma forma de autenticação. O campo PDU pode conter qualquer um dos cinco tipos de PDUs utilizados pelo SNMP (SNMP PDU). Esta estrutura pode ser visualizada na Figura 6. [RFC1213]

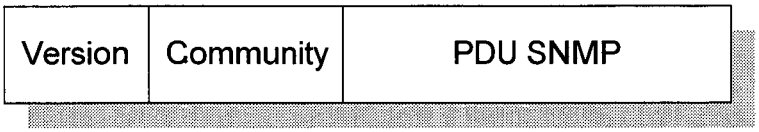

Figura 6-Formato da mensagem SNMPv1

As PDUs GetRequest PDU, GetNextRequest PDU e SetRequest PDU são definidas no SNMP com o mesmo formato da GetResponse PDU com os campos *error-status* e *error-index* com valor zero. Essas PDUs possuem os seguintes campos:

- *version* - Indica a versão do protocolo SNMP utilizado.
- *community* - O nome de comunidade atua como uma senha para autenticar a mensagem SNMP.
- *SNMP PDU* - É a unidade de dados de protocolo (*Protocol Data Unit* - PDU) utilizada pelo SNMP; contém os dados referentes a operação desejada (Get, GetNext, etc).

Para suprir as deficiências apresentadas pelo SNMP original, uma nova versão foi apresentada. O SNMP versão 2 (SNMPv2) provê mecanismos de recuperação de grande quantidade de informações, de gerenciar a gerencia e proporcionando também uma qualidade de segurança.

Uma diferença entre o SNMPv1 e o SNMPv2 é a existência de um mecanismo de comunidade melhorado, que apresenta uma identificação melhorada tanto da origem, como do formato da mensagem SNMPv2, permitindo utilizar métodos de acesso mais convencionais aos objetos gerenciados, além de permitir o uso de métodos de criptografia na troca de mensagens.[SAM97]

O SNMPv2 define propriedades de segurança que não estão disponíveis no SNMPv1, tornando assim incompatíveis os formatos das mensagens.

O SNMPv2 implementa três níveis de segurança, que são eles sem segurança, autenticada mas não privada e autenticada e privada, como vemos na Figura 7.

- **sem segurança** - o SNMPv2 não implementa nenhuma medida de segurança;
- **autenticada mas não privada** - o SNMPv2 utiliza a autenticação para verificar a validade da mensagens mas não utiliza a criptografia;
- **autenticada e privada** - além da autenticação utiliza também a criptografia;

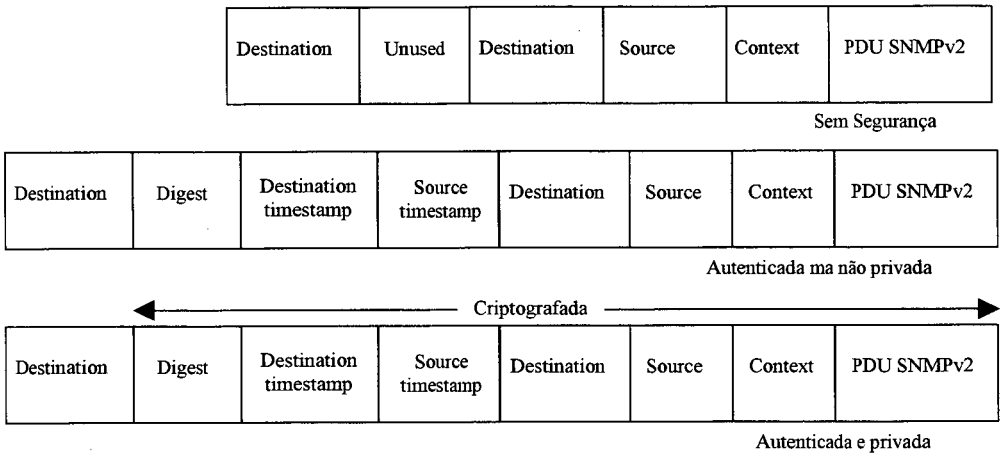

Figura 7-Formato da mensagem SNMPv2

3 CORBA

O padrão CORBA/OMG é um conjunto de padrões e conceitos para objetos distribuídos em ambientes abertos, proposto pela OMG (*Object Management Group*) [OBJ97-2][OBJ91-3]. Segundo essa arquitetura, métodos de objetos remotos podem ser ativados de forma transparente, em ambientes distribuídos e heterogêneos, através de um ORB (*Object Request Broker*). No CORBA, é possível também obter interoperabilidade entre objetos desenvolvidos em linguagens diferentes de programação (C, C++, Ada Cobol, Java, Smaltalk), em que as diferenças de cada um são mascaradas pelo CORBA através do mapeamento adequado da IDL (*Interface Defenition Language*) [ORB], para a linguagem de programação desejada. Cada objeto CORBA tem sua interface especificada em IDL, uma linguagem declarativa, sem nenhuma estrutura algorítmica, com sintaxe e tipos predefenidos, baseados na linguagem C++. Portanto, o uso de uma linguagem de definição de interface (IDL) permite tratar das heterogeneidades do ambiente, tais como os diferentes tipos de máquinas, sistemas operacionais e linguagens de programação.

O ORB 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 esta localizado o objeto servidor, em qual linguagem de progamação 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).

Na arquitetura CORBA, é especificada um adaptador de objeto básico (BOA) que oferece um modo primário para implementação do objeto, acessando os serviços oferecidos pelo ORB. Alguns dos serviços oferecidos pelo ORB, através do BOA são: geração e interpretação de referência de objetos, métodos de invocação, segurança de interações,

ativação e desativação de objeto e implementação, mapeamento de referencia de objeto para implementações e para registro de implementação.

O BOA é um adaptador de objeto padronizado pela OMG. Contudo, pode haver diferentes tipos de adaptadores de objetos, de acordo com as necessidades específicas de uma implementação de objeto que escolha qual irá utilizar, baseando-se no tipo de serviço que se deseja obter do ORB.

A interface do ORB, segundo o padrão CORBA, deve ser idêntica para as diferentes implementações CORBA. Esta interface deve ser independente de qualquer interface de objeto ou adaptador de objeto. Devido a maioria das funcionalidades do ORB serem oferecidas através dos adaptadores de objetos, *stubs, skeleton* ou DII(Interface de Invocação Dinâmica), nesta interface é oferecido apenas um pequeno número de operações que são comuns, tanto para clientes como para implementações de objetos.

O Repositório de interfaces é um serviço que oferece objetos persistentes é que representam a informação IDL em uma forma disponível, em tempo de execução. As informações disponíveis no repositório são geralmente utilizadas pelo ORB para realizar invocações dinâmicas a objetos remotos. Usando a informação disponível no repositório, é possível a um programa encontrar um objeto remoto, mesmo desconhecendo sua interface, seus métodos e parâmetros, e sua forma de ativação. Já, o repositório de implementações contém informações que permitem um ORB localizar e ativar implementações de objetos.

As interações no ambiente CORBA seguem o modelo Cliente/Servidor. Uma requisição cliente a um servidor remoto é transmitida através da rede, usando o ORB que localiza o objeto remoto, transporta os dados e passa o controle para o adaptador de objeto básico (BOA). Neste ponto o BOA prepara a implementação de objeto para receber a requisição e depois ativa a operação requisitada na implementação do objeto servidor, através de *Skeleton* IDL. Quando o processamento da requisição termina, o resultado é passado para o ORB que o entrega ao cliente de forma transparente.

Na requisição de um serviço, o cliente pode realizar uma invocação no servidor de duas formas: estática através de *stubs* IDL, ou dinâmica, através de interface de ivocação dinâmica (DII). Em ambas situações, a implementação de objeto (no servidor) não percebe o tipo de invocação utilizado na requisição pelo cliente. Na invocação estática, o cliente faz a invocação de um método no objeto-servidor, utilizando-se de *stubs* IDL apropriados, gerados no processo de compilação de arquivo IDL correspondente. Uma *stub* pode ser entendida como uma representação local do objeto remoto. O processo de definição de serviços, seguindo uma abordagem de invocação estática, é mostrado a seguir:

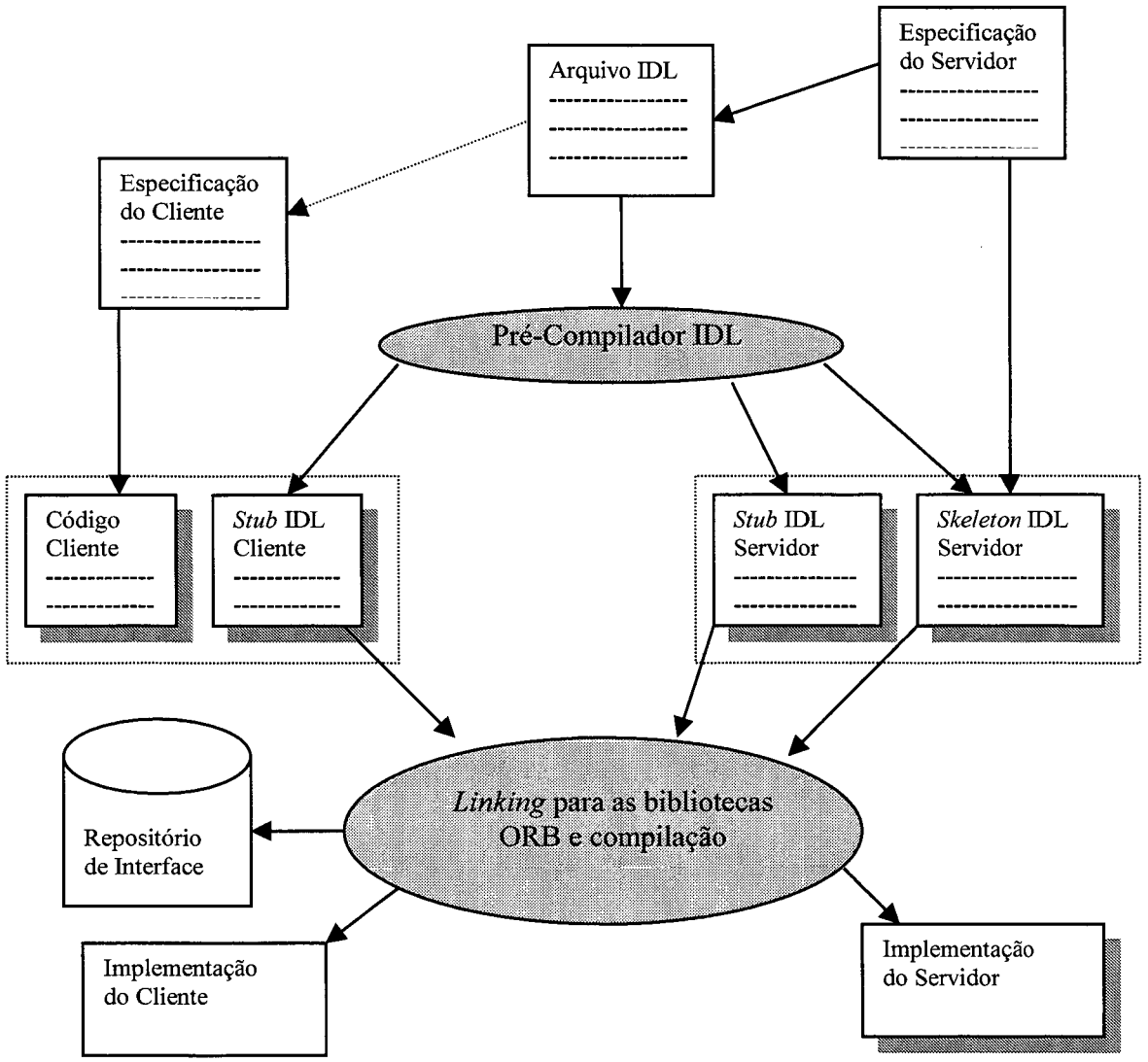

Figura 8-Abordagem de invocação estática utilizando stubs IDL

3.1 Protocolos GIOP E IIOP

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

O GIIOP 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.

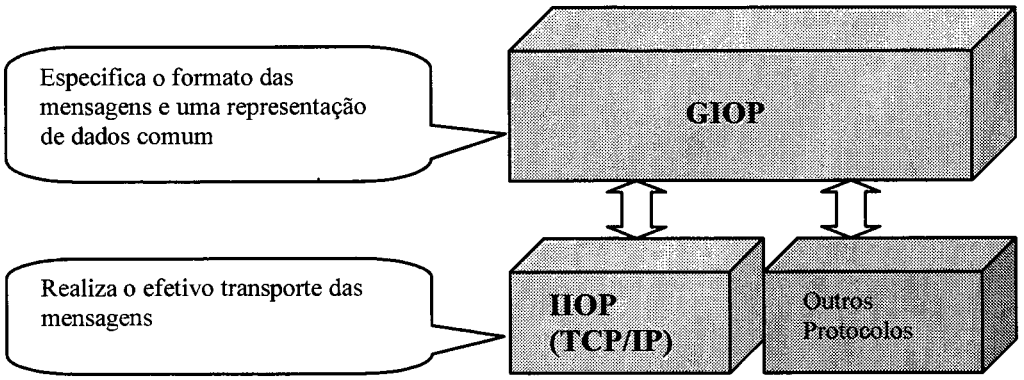

Figura 9-Interoperabilidade entre ORBs utilizando GIOP e IIOP

O IIOP é o protocolo que define como as mensagens GIOP são transmitidas sobre o protocolo TCP/IP (Figura 9). 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.

3.2 A Utilização do CORBA na Gerência de Redes

O CORBA é um padrão cada vez mais utilizado na área de gerência de redes [BAR98]. 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 é 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.

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 pode ser realizada em praticamente qualquer linguagem de programação. 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.

3.3 Gerência de Redes Integrada a Web

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.

Entre as ferramentas estudadas que permitem uma integração com a tecnologia Web e que fazem uso ou do SNMP ou do IIOP/CORBA, podemos destacar as seguintes:

A Advent SNMP API [BAR98] é 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 , 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.

Esta API pode ser utilizada tanto em *applets*, como em *applications* Java. Estas aplicações podem realizar conexões TCP/IP para qualquer equipamento da rede, sem restrições de segurança. No entanto, *applets* (sendo executadas 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. Através deste artifício, é possível utilizar um *applet* Java para realizar consultas SNMP a qualquer equipamento desejado.

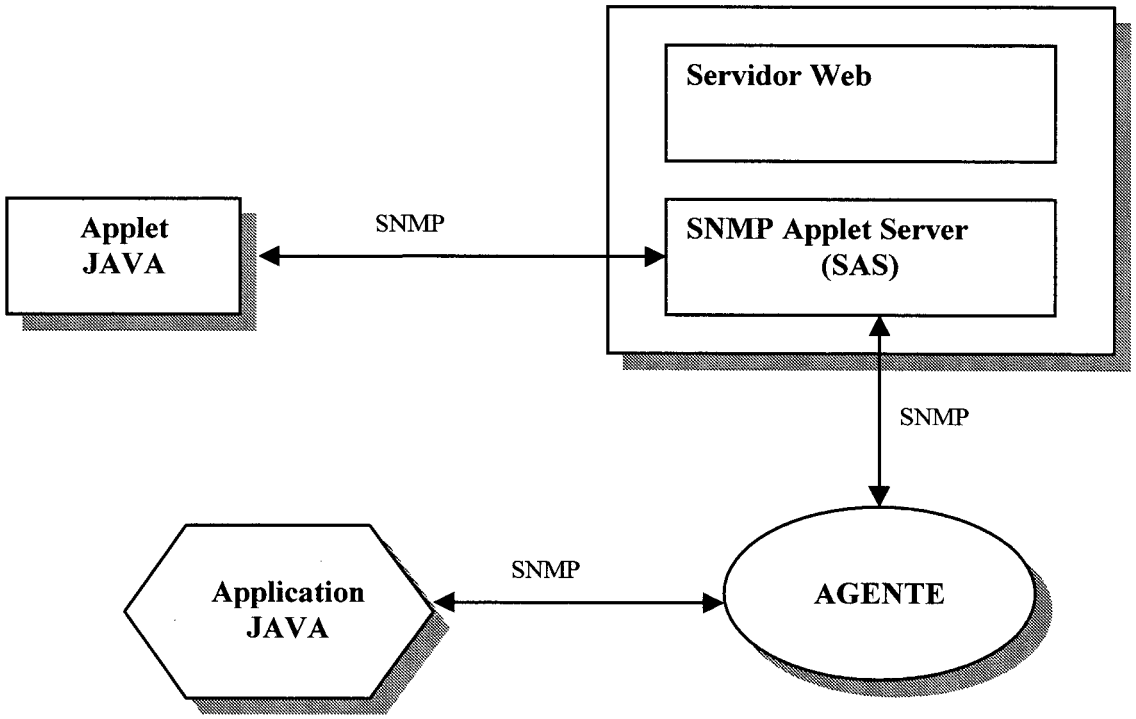

Figura 10-Consultas SNMP ao Agente através de Applets e Applications

Bojangles

Bojangles [BAR98] 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 e JORB. 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, e possui a implementação de um ORB, construído em Java, que pode ser utilizado em qualquer plataforma que possua um interpretador Java instalado.

Outra característica do Bojangles é que ele 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.

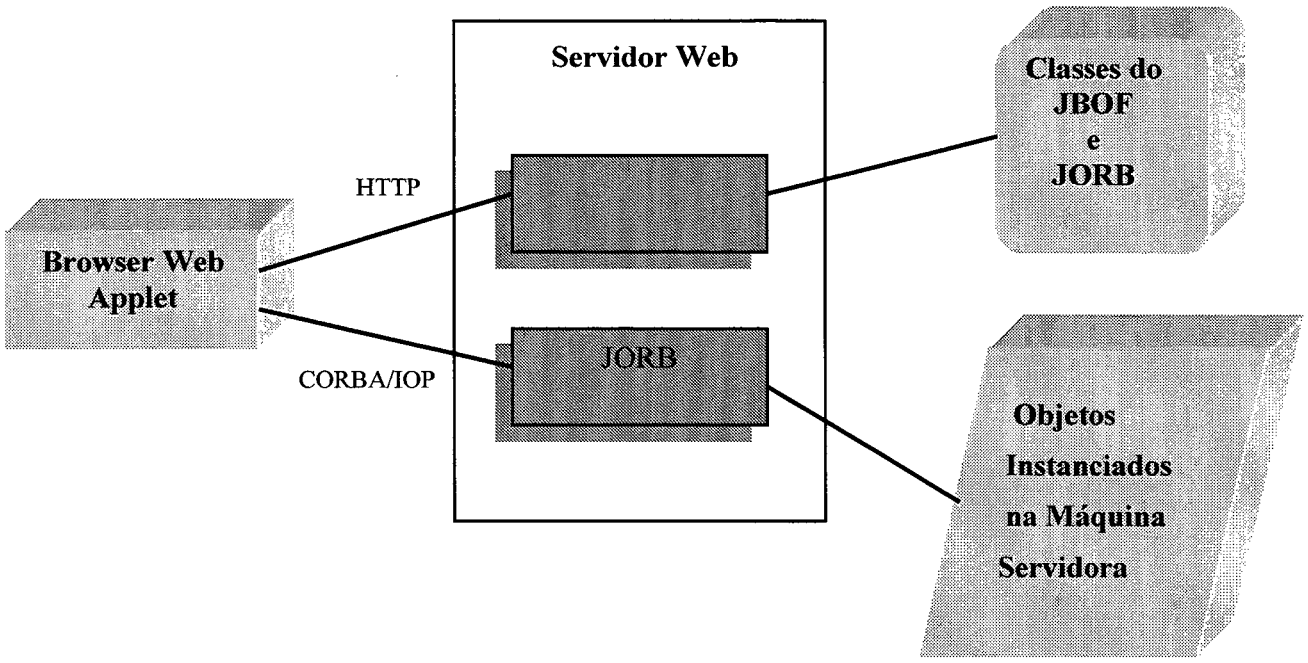

Figura 11-Modelo de utilização do Bojangles

4 GERÊNCIA DE REDES ATM

Serviços multimídia (voz, dados, vídeo, imagem, etc) têm sido cada vez mais requisitados em redes de comunicação e apresentam-se como uma tendência também nas empresas operadoras de telefonia. [SOA95][TAN97]

O grande problema encontrado no tráfego multimídia deve-se ao fato que cada tipo de serviço é adequadamente atendido por um determinado tipo de tecnologia. Por exemplo, comutação de circuitos foi desenvolvida para o serviço de transmissão de voz (rede de telefonia). Já o serviço de transmissão de dados é melhor atendido pela comutação de pacotes (rede de pacotes).

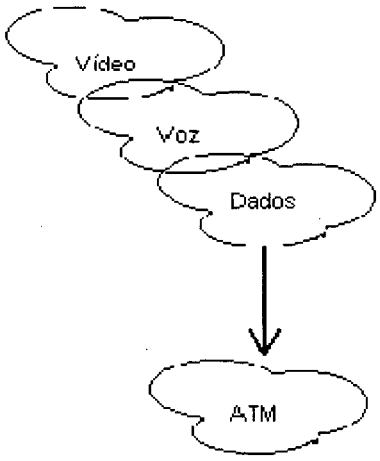

Figura 12-Múltiplas Redes Combinadas em uma só

O advento das chamadas RDSI (Redes Digitais de Serviços Integrados) no contexto das empresas operadoras de telefonia motivou o desenvolvimento de uma tecnologia capaz de atender satisfatoriamente aos requisitos da transmissão de não só dados e voz, mas também outras mídias. Trata-se da tecnologia ATM (*Asynchronous Transfer Mode*).[OLI98]

Embora ATM tenha sido projetada para suportar as RDSI-FL (RDSI de Faixa Larga), ela logo despertou atenção como uma possível solução às redes de computadores de alta

velocidade, permitindo a estas disponibilizarem serviços multimídia, impraticáveis com a tecnologia tradicional baseada em comutação de pacotes.

ATM é uma tecnologia de comunicação que utiliza um protocolo orientado a conexão. Ela oferece uma capacidade de transmissão de dados a altas velocidades e com "*delay*" mínimo e Qualidade de Serviço (QoS) garantida. ATM é caracterizado por pacotes de tamanho fixo de 53 bytes, denominados células.

Trata-se de uma tecnologia de transmissão assíncrona devido à ocorrência de células contendo informações não ser transmitida de forma periódica.

Recursos ATM, tais como, largura de banda e "*buffers*" sã compartilhados entre os usuários, e são alocados somente quando eles tem algo a transmitir. Logo, a rede usa multiplexação estatística para aumentar o "*troughput*" efetivo.

Como o intuito deste trabalho é o de estudar uma nova opção para a Gerência de Redes nada mais justo do que iniciarmos os estudos em uma tecnologia de transmissão de dados atual e de grande crescimento tecnológico como as redes de computadores baseado na arquitetura ATM. Este foi o principal motivo de termos escolhido as redes ATM e com isso vamos, logo a seguir, estudar a MIB que foi desenvolvida exclusivamente para esta arquitetura que é a AToM MIB ([RFC1695]), pois é ela que vamos utilizar para o modelo de Gerência proposto.

4.1 A MIB ATM

Tradicionalmente, dispositivos de rede têm tido propósitos bem definidos. Basicamente, estes dispositivos ou distribuem ou processam os dados sobre links físicos, usando os dados do cabeçalho para determinar o destino. Com o advento do ATM, alguns destes dispositivos (especialmente pontes e roteadores) estão ficando obsoletos, e os comutadores e concentradores (hubs) centras estão começando a realizar todas as funções.

Dentre as MIBs SNMP aplicadas à ATM, a mais importante e a padrão para o gerenciamento ATM é a AtoM MIB. Esta especificação (também chamada de MIB ATM) é de responsabilidade do grupo de trabalho "ATOM MIB" da IETF (*Internet Enginnering Task Force*), que é o encarregado pelas questões de gerenciamento ATM dentro do IETF. Ela se preocupa com a definição de objetos que auxiliam no gerenciamento de interfaces, circuitos virtuais, cruzamento de conexões, entidades AAL5 e conexões AAL5 em um

ambiente ATM. A especificação atual desta MIB é baseada na SMIv2. [RFC1695] [OBJ96-4]

Esta MIB tem como principal propósito o gerenciamento de circuitos virtuais permanentes (PVC – *Permanent Virtual Circuit*) em ambientes ATM. Apesar de oferecer objetos que possuem informações sobre circuito virtuais comutados (SVC – *Switched Virtual Circuit*), este serviço requer a definição de capacidades adicionais não representadas nesta MIB. Com o objetivo de realizar o gerenciamento de interfaces, circuitos virtuais e cruzamento de conexões, outras MIBs são necessárias. Dentre elas temos a MIB II para o gerenciamento de interfaces e do sistema como um todo.

Os objetos de gerenciamento definidos nesta MIB são representados através de tabelas divididas de acordo com suas funções, como descrito a seguir:

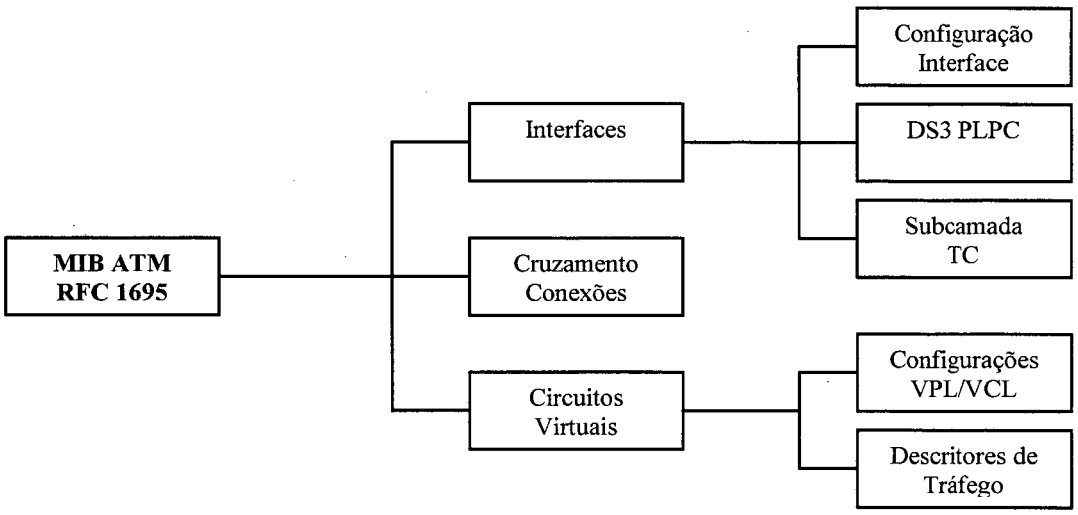

Figura 13-Estrutura da AtoM MIB

4.1.1 Interfaces

Configuração de Interfaces (atmInterfaceConfTable)

Este grupo contém informações de configuração sobre as interfaces ATM, além das encontradas na tabela de interfaces "ifTable" da MIB II.

Cada entrada se relaciona com uma interface ATM presente no dispositivo e é composta de parâmetros de configuração como: endereço ATM da interface (atmInterfaceAdminAddress), número máximo de VPCs e VCCs suportados

(atmInterfaceMaxVpcs e atmInterfaceMaxVccx), número de VPCs e VCCs configurados (atmInterfaceConfVpcs e atmInterfarceConfVccs) (incluindo SVCs e PVCs), VPI e VCI utilizado pelo protocolo ILMI, etc.[OLI98].

Interfaces DS3 PLCP (atmInterfaceDsePlcpTable)

Este grupo é também uma extensão da ifTable (MIB II [RFC1213]) e contem parâmetros de configuração de estado do DS3 PLCP.

Esta tabela possui uma entrada por interface, que se utiliza de DS3 PLCP para transportar células sobre DS3. Cada entrada possui informações sobre o número de eventos de erro (atmInterfaceDsePlcpSEFSs), existência de alarmes (atmInterfaceDs3PlcpAlarmSate) e de períodos de tempo em que a interface não esteve disponível (atmInterfaceDs3PlcpUASs).

Subcamada TC (atmInterfaceTCTable)

Este grupo possui parâmetros de estado e de configuração da subcamada TC. Também é uma extensão da tabela ifTable.

Cada entrada se relaciona com uma interface que utiliza a subcamada TC para transportar células. As entradas possuem dois atributos (atmInterfaceOCDEvents e atmInterfaceTCAlarmState) que contém informações sobre a existência de problemas na delineação de células. Exemplo destas células são aquelas que possuem como camada física SNET ou DS3.

4.1.2 Circuitos Virtuais ATM

Configuração de VPLs e VCLs (atmVplTable e atmVclTable)

Estes grupos contém informações de configuração e de estado de um circuito VPL/VCL bidirecional que são definidos a partir de duas tabelas: atmVplTable e atmVclTable, respectivamente. Como seus parâmetros são similares, iremos detalhá-las juntamente. Estas tabelas são definidas em terminais e comutadores e podem ser utilizadas para se criar, atualizar ou liberar um VPL/VCL.

Os parâmetros comuns às estas tabelas são: identificador de caminho virtual, que é o VPI (atmVplVpi) para conexões VPL e a combinação de VPI e VCI (atmVclVpi e atmVclVci) para VCL; os atributos que caracterizam o estado do circuito, AdminStatus (implementado par aum VPL/VCL final de um VPC/VCC e descreve o estado administrativo do VPL/VCL indicando se o fluxo de tráfego esta habilitado ou não), OperStatus (especifica o estado operacional do VPL/VCL) e LastChange ('Timestamp' que

indica a última alteração do estado do circuito). Os outros parâmetros são índices para a tabela de descrição de tráfego (atmTrafficDescParamTable – um para cada direção de tráfego dentro do circuito), e um índice para cada tabela de cruzamento de conexões (CrossConnectTable).

Descritores de Tráfego ATM (atmTrafficDescrParamTable)

Este grupo possui um conjunto de parâmetros que caracterizam o tráfego ATM, incluindo a classe da Qualidade de Serviço (QoS). Esta tabela é definida tanto em terminais como em comutadores.

Cada entrada descreve o tráfego que é transportado sobre um circuito virtual, tanto quantitativamente. Os descritores de tráfego estão de acordo com os resultados da negociação, quando do estabelecimento da conexão. Quando é criada um a nova entrada nesta tabela seus parâmetros são checado para garantir a consistência.

4.1.3 Cruzamento de Conexões

Cruzamento de conexões VP/VC (atmVp(Vc)CrossConnectTable) - PVC

Estes grupos descrevem informações sobre estado e configuração de todos os cruzamentos de conexão VP/VC relacionados com PVC(*Permanent Virtual Circuit*), sejam eles ponto a ponto, ponto a multiponto e multiponto a multiponto. Estas informações são disponibilizadas somente em comutadores, onde temos a funcionalidade de cruzamento de conexões. Os grupos baseiam-se respectivametne nas tabelas atmVpCrossConnectTable e atmVcCrossConnectTable.

5 DESCRIÇÃO DO AMBIENTE DE GERÊNCIA DE REDES UTILIZANDO CORBA, JAVA E HTML

O objetivo deste capítulo é descrever o ambiente de gerência utilizando o CORBA, Java e HTML. Esta gerência deverá atender todos os requisitos básicos oferecidos pelos ambientes já conhecidos de gerência. Para que isso seja possível decidiu-se fazer uso da metodologia de gerência usada pelo padrão SNMP, isto é, o ambiente será baseado em um modelo Gerente-Agente, utilizando-se do método de *Polling* (onde o gerente irá interrogar o agente) para fim de se obter informações sobre seus objetos gerenciados.

Muitos foram os questionamentos de o por quê de se estudar e desenvolver outro ambiente para a gerência se os padrões que hoje estamos utilizando são, de uma forma geral, muito bons e que atendem a todos os requisitos ? Ora, se nós pensarmos nisso, então não precisaríamos mais fazer pesquisas, estudos e projetos para mais nada, pois temos quase de tudo e funcionando razoavelmente bem. Mas o homem desde os seus primardes é assim, sempre querendo inventar, aperfeiçoar tudo o que esta a seu redor.

É com este intúito que se enquadra este trabalho, no espírito da inovação e da busca e estudo de novas tecnologias, ferramentas e etapas necessárias a implementação de um ambiente de gerência que supra as necessidades básicas.

5.1 Ferramentas para o Desenvolvimento do Trabalho

Foram vários os motivos levados em conta para a escolha do SNMP como base para o projeto. Uma das principais foi de que este padrão é largamente utilizado nos dias de hoje para a gerência e, todos os equipamentos que possuem implementados funções de gerência utilizam-se de agentes baseado no SNMP.

Outro motivo que nos levou a esta escolha é de que o SNMP é muito simples no modo de funcionamento e de seu estudo. Seu agente não exige uma grande capacidade de processamento e sua implementação é modesta mas muito eficaz, deixando para o gerente a

maior parte do processamento, o qual geralmente esta instalado em uma estação de trabalho destinada a esta fim.

Por outro lado, o crescimento e o sucesso do modelo integrados de objetos CORBA como solução de interoperabilidade tem motivado o mapeamento das especificações dos padrões OSI/CMIP e SNMP para CORBA IDL, através de adaptadores baseados na possibilidade de o CORBA se tornar um padrão integrado de gerência.

O modelo de informação oferecido por CORBA IDL é comparável ao GDMO (Guidelines For The Definiton of Management Objects) do modelo OSI/CMIP e superior ao do SNMP para definição de MIB, permitindo a clara separação de interface e implementação. O CORBA oferece uma solução muito mais simples e não requer que os desenvolvedores conheçam notações de sintaxe abstrata e esquema de codificação e decodificação.

A linguagem IDL é usada para descrever as interfaces dos objetos CORBA, sendo puramente declarativa. Uma definição IDL especifica cada atributo e parâmetros das operações de uma interface definida. O IDL foi especificado para dar a facilidade de seu mapeamento para linguagens de implementação como C, C++, Smalltalk e Java. Uma outra motivação para a sua utilização é a possibilidade de usar ferramentas de desenvolvimento que suportam a tradução de modelos de objetos sem interfaces IDL com compiladores que geram automaticamente os "stubs" e "skeletons" dos objetos da aplicação. Do ponto de vista de comunicação, CORBA oferece mais transparência e uma arquitetura mais elegante, com os protocolos GIOP e o IIOP, além de termos uma maior segurança nos dados pois utiliza o TCP para o trafego de informações.

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 um browser Web, permite a criação de um mecanismo de transferência de informações de um equipamento cliente a um equipamento Servidor para que seja efetuado a Gerência. Este mecanismo de passagem de informações será discutido mais adiante.

Dentro deste contexto, a linguagem escolhida para as implementações só poderia ser o Java, dado a grande utilização hoje nas aplicações que utilizam a Web. A sua escolha também se apoia na sua característica principal, que é a portabilidade. Um programa Java ao ser compilado, é gerado um código para uma máquina genérica chamado *Byte Code*. Este código pode ser executado tanto por interpretadores como por browser Web, conferindo assim a característica de serem independentes da plataforma que está em uso.

5.2 Desenvolvimento do Projeto Proposto

Como dito anteriormente o CORBA esta sendo muito estudado e utilizado para a Gerência de Redes [BAR98], surgindo como uma alternativa ou como uma nova tecnologia para este fim. Uma característica bastante grande nos estudos e aplicações que já fazem uso deste padrão para fazer a gerência é que utiliza-se somente a sua facilidade de instanciar objetos sem se preocupar com a sua localização e detalhes maiores para que haja uma troca de informações, pois isto é uma das suas principais características. Isto quer dizer que mesmo utilizando o CORBA para fazer a gerência de uma rede ele tem que se basear em algum protocolo de gerenciamento para que possa interagir com o dispositivo a ser gerenciado, isto é, muitas aplicações o utilizam somente para chegar até o dispositivo e depois ainda utilizam um dos padrões para interagir com o agente. Este agente somente aceita comandos de pedido de informações no seu formato original, se este agente é baseado no padrão SNMP ou CMIP, ele somente ira aceitar os comandos nativos destes protocolos.

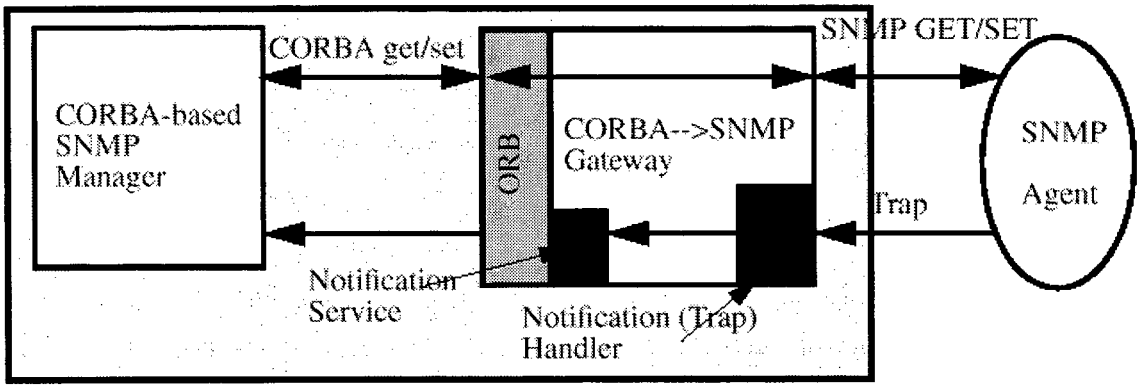

Figura 14-Gerência de Redes utilizando um Gateway CORBA/SNMP

Assim, todos os estudos e aplicações de gerência que utilizam o CORBA em seu projeto ainda necessitam de uma interface que permita que as informações adquiridas do agente (obtidas através da MIB) sejam transformadas da linguagem utilizada por ele (geralmente SNMP – ASN.1) para uma interface que o CORBA possa entender, onde é descrita toda a sua implementação em IDL. Este fato é de fácil entendimento se pensarmos que todos os equipamentos de rede hoje utilizados e que possuem agentes instalados, estão adaptados para que possamos fazer uma gerência utilizando um dos dois protocolos que estão padronizados, que são o SNMP e o CMIP. Desta forma toda vez que quisermos obter informações do agente teremos que interrogá-lo através de um formato que ele entenda,

ficando assim a necessidade de termos uma aplicação que sirva de "gateway" entre o CORBA e o SNMP/CMIP como mostrado na Figura 14. [BAR98][LAP97]

O propósito do trabalho não é este, mas sim de termos um ambiente de gerência, na parte de comunicação com os objetos, totalmente utilizando-se do CORBA, sem utilizarmos nenhum dos outros padrões (SNMP ou CMIP) para o acesso e a troca de informações.

Para que isso possa ser feito, observando que utilizaremos o padrão SNMP como base do projeto, teremos que adaptar todo o comportamento deste padrão para o CORBA. Isto quer dizer, mapear toda a estrutura da MIB que é descrita em ASN.1 para IDL, habilitar um processo que efetue a parte do Agente e um processo que faça o papel do Gerente que tenha suporte a todas operações que serão efetuados por ele em cima da MIB presente nos objetos gerenciados.

5.3 Ambiente Proposto

Para que a gerência possa ser efetuada por qualquer máquina dotada de um navegador Web devemos observar algumas características quanto ao ambiente em que deverá estar empregado.

Sempre quando nós navegamos por páginas HTML não temos a noção precisa de onde estas páginas estão, se estão concentradas em somente um servidor ou estão espalhadas dentro da rede. Por este motivo quando requisitamos alguma informação ou passamos algum dado para uma página HTML não temos noção de onde os processos estão sendo executados e onde estão estes processos, dando a impressão que está tudo num servidor. Mas se analisarmos, veremos que em muitos casos são chamados processos de diferentes máquinas, espalhadas pela rede.

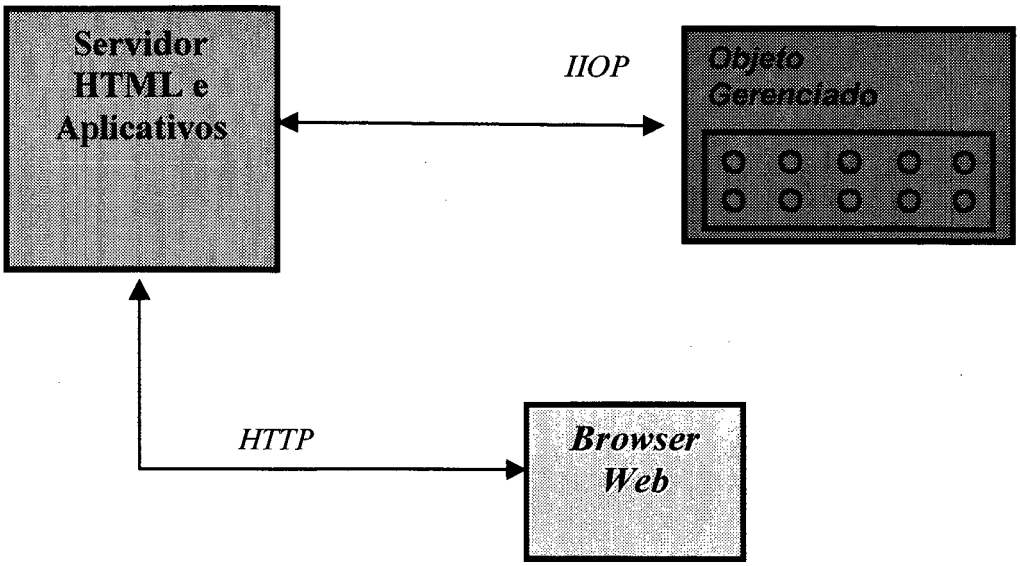

Figura 15-Ambiente Proposto para Gerência

Portanto, para que nós possamos fazer a gerência de redes utilizando a Web, o ambiente deverá ter um servidor onde estarão todas as páginas HTML, ,juntamente com seus applets e aplicativos que farão as invocações de métodos para que seja realizado as operações de gerência desejados nos Objetos Gerenciados. A comunicação com este servidor poderá ser feito por qualquer equipamento que tenha um browser que suporte applets java.

A comunicação entre o Servidor e os Objetos Gerenciados a fim de se executar as operações de gerência, será feita através de aplicativos implementados em Java e utilizando o protocolo IIOP que é parte integrante do padrão CORBA e que roda em cima da plataforma TCP/IP. Esta comunicação de troca de informações se dará somente entre o servidor e o objeto gerenciado e, por este motivo não é necessário termos instalado nas máquinas que irão se conectar com o Servidor para efetuar a gerência uma versão do CORBA.

A não necessidade de termos instalado o CORBA em todas as máquinas que podem ser utilizadas para fazer a Gerência é fundamental para podermos ter uma independência de plataforma e agilidade nas operações. As informações adquiridas pelo Servidor será apresentado ao gerente (que estará em qualquer equipamento) através do Java dentro das páginas HTML.

Por este motivo, a aplicação de gerência torna-se simples, contendo apenas o código necessário para realização da interface com o usuário e também obtendo com isto uma segurança maior, por executar os aplicativos somente no servidor.

5.4 Descrição do Servidor

A estação que servirá de repositório de páginas HTML deverá conter também todos os aplicativos que farão as conexões e as requisições de gerência (leitura e gravação) na MIB. Este servidor também será responsável pela execução destes aplicativos.

De acordo com o projeto, foi definido que a gerência se dará nos mesmos moldes do protocolo SNMP que possui as primitivas de gerenciamento Get, GetNext, GetBulk, Set e Trap e trabalhar no método de *polling*. Estas primitivas são implementadas em java no servidor.

Para que possamos fazer a gerência utilizando o CORBA, precisaremos descrever todas as interfaces das operações que serão executadas, como também os tipos de dados que serão utilizados nas trocas de informações. Estas interfaces serão descritas usando a linguagem IDL pois é com ela que o CORBA utiliza.

```
// Null type
typedef char ASN1_Null;
const ASN1_Null ASN1_NullValue = '\x00';

typedef boolean ASN1_Boolean;

// unsigned integers
 typedef unsigned short   ASN1_Unsigned16;
 typedef unsigned long    ASN1_Unsigned;
typedef unsigned long     ASN1_Unsigned64[2];

// others
ttypedef double ASN1_Real;
ttypedef sequence<octet> ASN1_BitString; // PIDL defined
ttypedef sequence<octet> ASN1_OctetString;
ttypedef string ASN1_ObjectIdentifier;
ttypedef any ASN1_Any;
ttypedef any ASN1_DefinedAny;
```

Figura 16-Descrição dos tipos de dados em IDL

A descrição em IDL dos tipos de dados que serão trocados entre o servidor e os objetos gerenciados observando sempre que os dados se baseiam no protocolo SNMP. Para esta conversão utilizamos o algoritmo desenvolvido pela X/Open – XoJIDM [INT99] que prevê o mapeamento de todos os tipos de dados utilizados pelo SNMP, respeitando sua sintaxe e tamanhos. Na Figura 16 vemos um trecho do código gerado em IDL utilizando o algoritmo.

Como dito anteriormente no Servidor estarão as páginas em HTML (com ou sem applets) ao qual o Gerente se conectará e baixará em sua máquina. Através destas páginas o Gerente irá utilizar as primitivas de gerência para solicitar que seja efetuado determinado comando no objeto gerenciado (ler uma variável, ou gravar um valor para alguma variável). Neste momento esta página deverá enviar uma requisição para o Servidor chamando o método responsável pela operação.

O envio das informações para o Servidor através de páginas HTML poderá se dar de duas formas descritas a seguir.

5.4.1 Ferramentas para conexão com o Servidor

Se utilizarmos somente applets Java para fazer a gerência nós vamos estar estabelecendo a comunicação com o objeto gerenciado diretamente do computador onde está o gerente (que não é o Servidor e sim o equipamento que baixa a página HTML) e, isto pode acarretar vários tipos de problemas, onde destaca-se a questão da segurança e principalmente, teremos que instalar uma versão do CORBA nesta máquina para podermos estabelecer a comunicação.

Problemas como os descritos iriam nos levar a abandonar o projeto deste ambiente, pois precisaríamos que uma gama de equipamentos pré identificados e neles versões dos software que seriam utilizados para executar a Gerência.

Então passou-se a estudar ferramentas que proporcionem a nós enviarmos dados para uma estação servidora e esta, de posse destes dados estabelecer a comunicação com o objeto gerenciado e efetuar as primitivas de gerência (Get, Set, ...). Dentre as que se destacaram estão CGI e Servlets Java.

Common Gateway Interface – CGI

CGI não é realmente uma linguagem ou um protocolo no sentido mais estrito desses termos. É, na realidade, apenas um conjunto de variáveis de nomes e convenções comuns para a passagem de informações do servidor para o cliente e vice-versa. [WEI97]. É uma interface entre o servidor HTTP e os outros recursos – como os seus programas em CGI do Site Web – do computador host no servidor

Programas em CGI podem ser escritos em qualquer linguagem. Os únicos critérios para a linguagem são os seguintes: [WEI97]

- A linguagem deve ser aceita pelo sistema operacional em que o servidor está rodando;
- A linguagem deve ter facilidades suficientes para realizar a tarefa que você precisa realizar com ela;
- O programador deve estar acostumado o bastante com a linguagem para codificar com habilidade.

Os programas em CGI geralmente apanham sua entrada de variáveis de ambiente e enviam sua saída para o stream de saída padrão, geralmente conhecido como saída padrão (Standardt Output, abreviada como stdout). Esta saída é chamada de stream porque é transmitida para o usuário como um stream de bytes ou caracteres. A saída do programa precisa estar em um formato que o browser Web possa exibir. Em geral, este será o formato HTML padrão. [WEI97]

Em CGI a maioria das entradas é passada em *variáveis de ambiente* que são definidas pelo servidor HTTP. Algumas dessas variáveis são padronizadas na especificação CGI, outras são particulares aos browsers, servidores, sites ou outros fatores.

O método *GET* passa dados para o programa em CGI utilizando a variável *QUERY_STRING* que é formatada como um stream de pares *nomes=valor* separados por um caractere *&*. Já o método *POST* passa os mesmos dados, no mesmo formato, usando de arquivo da "entrada padrão" (stdin). A diferença entre os dois métodos é que o Get é inerente limitado pelo tamanho máximo de uma variável de ambiente, que não é garantidamente grande entre as plataformas e provavelmente ficará entre 256 bytes e cerca de 4.096 bytes, limitação que não ocorrerá no método Post pois é passado um arquivo. [WEI97]

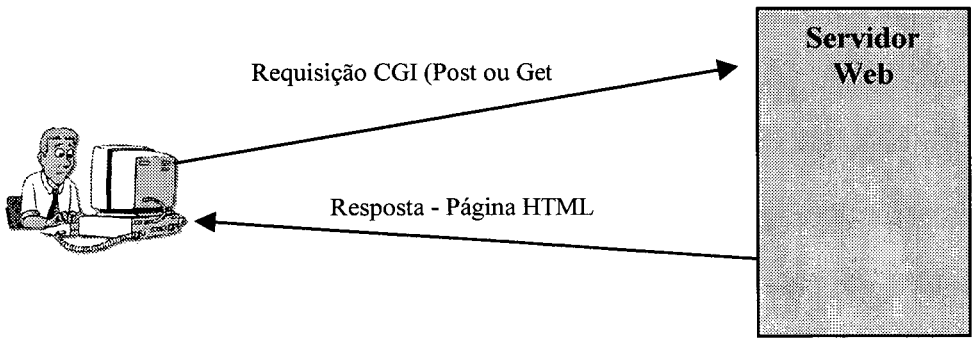

Figura 17 - Funcionamnto do CGI

A Figura 17 nos mostra claramente como é o funcionamento de uma aplicação CGI.

Servlet Java

Servlet é uma tecnologia Java em resposta a programação em CGI. Eles são programas que rodam em um servidor Web, recebem parâmetros via HTTP e retornam páginas Web.

Os objetos Java fornecem algumas vantagens sobre os programas Web orientados a CGI. Eles têm um custo de inicialização menor, permanecem rodando e podem ser usados em qualquer plataforma de software compatível com Java. Sempre que um servidor obtém uma requisição que é manipulada por um programa CGI, ele deve iniciar o programa CGI, que tem uma quantidade fixa de trabalho envolvido. Depois que o programa termina de processar uma requisição, ele finaliza. Os servlets podem também tirar vantagem da estrutura de segurança do Java, permitindo diferentes níveis de segurança para diferentes servlets. [WUT97]

Um dos principais motivos que nos influenciaram ao uso de Servlets e não de CGI é porque em Java o tratamento de variáveis de ambiente fica um pouco mais complicada por causa de sua principal característica, que é a independência de plataforma. Vários estudos foram feitos utilizando CGI e Java e em todos foi conseguido unir estas duas tecnologias, mas o desempenho não foi o mesmo se utilizássemos outra linguagem de programação como C ou Perl. Por este motivo procuramos outras alternativas e depois de estudos e testes, foi escolhido o uso de Servlets neste projeto.

A SUN [SUN99] definiu uma API para escrever servlets. O núcleo desta API é a classe Servlet. Os dois métodos mais importantes na classe Servlet são *init* e *service* . O método *init* é chamado quando o servlet é inicialmente criado e é responsável por sua inicialização. O método *service* toma dois parâmetros, um objeto usando a interface *ServletRequest* é responsável por manipular uma requisição HTTP e um objeto usando a interface *ServletResponse* e é responsável por retornar uma resposta.

Ler cabeçalhos vindo de uma página HTML enviado por um formulário é bastante fácil com Servlets. Primeiramente devemos chamar o método *getHeader* da classe *HttpServletRequest*, o qual vai retornar uma *String* se o cabeçalho contiver dados, do contrário vai retornar *null*. De posse desta *String* devemos chamar os métodos e as classes que estão encarregados de fazer a comunicação com os Objetos Gerenciados.

Outra característica muito importante é que o Servidor Web e os aplicativos que farão todo o processamento podem ficar no mesmo equipamento facilitando com isto a agilidade

em executar as suas classes ou, eles podem estar distribuídos dentro da rede e instanciados através de RMI ou CORBA.

De acordo com o objetivo proposto deste trabalho, que é de estabelecer um ambiente para Gerência de Redes utilizando Corba e Java, os servlets Java estão fortemente ligados pois eles são escritos em Java o que nos oferece uma independência de plataforma muito importante. Não precisaremos fazer uma versão do ambiente de Gerência diferente para cada plataforma, pois para rodar este ambiente basta ter uma máquina virtual Java.

5.4.2 Banco de Dados

Um dos objetivos do Servidor é permitir que as informações de gerenciamento possam ser armazenadas em uma base de dados. Isto permite que informações como tráfego nos links, informações sobre traps e outros dados adicionais adquiridos através do *polling* possam ser armazenadas em uma única estrutura de dados. A tecnologia de base de dados escolhida foi a relacional por ter um modo de armazenamento simples e de fácil entendimento.

Quando estas informações estiverem armazenadas em uma base 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.

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 [SUN99]. O JDBC é um pacote padrão que já é incluído no JDK (Java Development Kit) a partir da versão 1.1.

Utilizando o JDBC, é possível realizar a comunicação com os principais servidores de banco de dados relacionais disponíveis comercialmente, através de um 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 conseqüência de que o driver JDBC, assim como as aplicações, são transferidos para a máquina cliente utilizando o protocolo HTTP (como qualquer applet).

```
class Banco{
      Config conf;

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

    public Banco(Config confp)
    {
      conf=confp;
    }
//Realiza conexão 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;
      };
```

Figura 18-Conexão com o Banco de Dados utilizando o Java

5.4.3 Primitivas de Gerência Get, GetNext, GetBulk e Set

Estas primitivas de gerência que são proprietárias do protocolo SNMP também são parte do sistema Servidor. Como visto anteriormente elas tem a função de ler uma variável ou uma lista de variáveis da MIB de um objeto gerenciado.

Para nós conseguirmos ler as variáveis precisamos nos conectar ao objeto gerenciado. Se utilizarmos o SNMP para isso ele irá gerar uma PDU que será enviado através da pilha de protocolos do TCP/IP. Como o nosso ambiente prevê a utilização do padrão CORBA

para este fim, estas primitivas irão utilizar o protocolo IIOP que também roda em cima do TCP/IP.

Um dos motivos para utilizarmos o IIOP para a conexão e transferência dos dados entre os equipamentos é que mesmo ele utilizando os protocolos da Internet ele é muito mais eficiente na entrega das requisições. Enquanto o SNMP utiliza o protocolo UDP para a transmissão e recepção das requisições o CORBA utiliza o protocolo TCP que nos garante a entrega dos pacotes.

```
try
{
// Inicializa o ORB
System.out.println( "Initialising the ORB" );
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, null );
// Use Naming Service
org.omg.CORBA.Object objRef = orb.resolve_initial_references( "NameService" );
```

Figura 19-Exemplo de inicialização do ORB em JAVA

Para podermos executar as primitivas a fim de obtermos dados, precisaremos inicializar o CORBA (se já não estiver inicializado) que é o encarregado de localizar o objeto gerenciado (Figura 19). Os parâmetros a serem passados para a invocação dos métodos onde estão as primitivas que farão a gerência, são os mesmos que passaríamos se utilizassemos o protocolo SNMP, que é o nome do objeto e a comunidade a que ela pertence. Depois que o CORBA localizar o objeto, então é que serão executados os comando de leitura ou gravação nas variáveis da MIB e os valores obtidos serão armazenados no banco de dados e apresentados para o usuário.

Deste modo, estando o usuário em um equipamento com um browser Web ele irá se conectar ao Servidor utilizando o protocolo HTTP, o qual vai retornar uma página HTML contendo um applet (ou um formulário) onde escolheremos qual a primitiva será usado na gerência e, passados os parâmetros necessários para que ela seja efetuado.

5.5 Descrição do Agente

De acordo com o ambiente que estamos propondo, o processo que fará o papel do Agente terá pouca implementação, pois ele está encarregado de estabelecer a comunicação com o Servidor através da instanciação do ORB e repassar ao Servidor os dados

requisitados através das primitivas de gerência. Outra função importante do Agente é a de avisar ao Servidor toda vez que alguma ação extraordinária acontecer em alguma variável, previamente identificada, que esteja sendo monitorada.

Para atender as requisições do Servidor, o Agente, que estará instalado no Objeto Gerenciado, terá acesso direto a MIB do equipamento. É obvio que esta MIB deverá estar adaptada para conversar com o Agente levando-se em conta que ele estará escrito na linguagem Java e utilizará o Corba para a comunicação.

Como o Ambiente que se está propondo tem como base o padrão SNMP, surgiu e eminência de transcrevermos a MIB do ASN.1 para IDL com veremos a seguir.

5.5.1 A transformação da MIB de SNMP/ASN.1 para CORBA/IDL

Num primeiro momento a fim de atender uma expectativa básica do projeto, temos que adaptar a MIB, que é descrita em ASN.1 para que ela seja entendida e suportada pelo novo ambiente onde ela esta sendo submetida. E este novo ambiente, que é o CORBA, nos sugere que tenhamos uma MIB descrita em todas as suas instancias em IDL.

Devemos destacar que esta nova MIB não está implementada em nenhum objeto gerenciado, e por isso ela é somente uma demonstração.

Esta transformação, ou adaptação, da MIB é baseada em estudos realizados por diversos órgãos. Entre os que podemos citar é a recomendação do X/Open – XoJIDM [INT99] onde estuda-se a utilização desta nova tecnologia para a utilização na Gerência de Redes de Computadores.

De acordo com o algoritmo de transformação desenvolvido pela XoJIDM, para podermos fazer esta adaptação da SNMP-MIB para CORBA-IDL necessitamos utilizar dois passos: o primeiro deles é para checarmos a sintaxe e ao mesmo tempo termos o conhecimento básico para a transformação, logo após, vem a segunda fase do processo que é transformar a MIB em interfaces IDL como vemos na Figura 20.

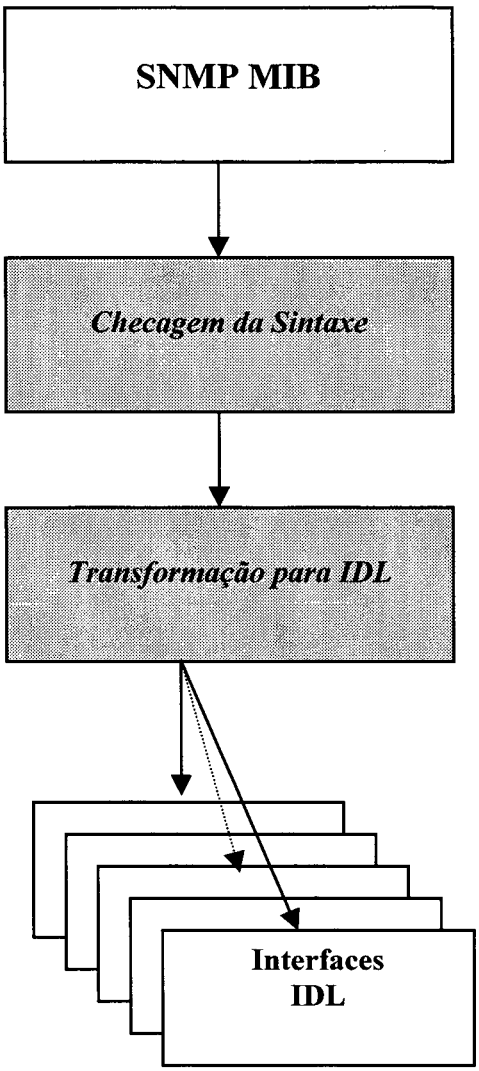

Figura 20-Algoritmo de transformação de MIB/ASN.1 para IDL

O algoritmo de transformação ou adaptação de ASN.1, mapea cada entrada de tabela em ASN.1 para uma interface em IDL onde os componentes de entrada são atributos da interface. Também transforma um instância simples de variável de um grupo em uma interface onde cada variável é mapeada como um atributo nesta interface.

Após aplicarmos este algoritmo, temos como resultado a MIB em IDL com todo o seu comportamento definido. Os tipos de dados utilizados nas definições, que serão usados os mesmos definidos em ASN.1 do SNMP, foram definidos em outra implementação e gerados também como IDL. Estes tipos de dados serão incorporados na definição da MIB e fazem parte do anexo que acompanha este trabalho.

```
atmInterfaceMaxVpcs OBJECT-TYPE
    SYNTAX      INTEGER (0..4096)
    MAX-ACCESS  read-write
    STATUS      current
        ::= { atmInterfaceConfEntry 1}

atmInterfaceMaxVccs OBJECT-TYPE
    SYNTAX      INTEGER (0..65536)
    MAX-ACCESS  read-write
    STATUS      current
    ::= { atmInterfaceConfEntry 2}

atmInterfaceConfVpcs OBJECT-TYPE
    SYNTAX      INTEGER (0..4096)
    MAX-ACCESS  read-only
    STATUS      current
    ::= { atmInterfaceConfEntry 3}

atmInterfaceConfVccs OBJECT-TYPE
    SYNTAX      INTEGER (0..65536)
    MAX-ACCESS  read-only
    STATUS      current
    ::= { atmInterfaceConfEntry 4}
```

Figura 21-Fragmento da MIB em ASN.1

A Figura 21 nos mostra um fragmento da MIB ([RFC1695]) em ASN.1 e a Figura 22 um fragmento do trabalho resultante da transformação da MIB gerado em IDL.

```
typedef ASN1_Integer atmInterfaceMaxVpcsType;
attribute atmInterfaceMaxVpcsType atmInterfaceMaxVpcs;
//--------------------
typedef ASN1_Integer atmInterfaceMaxVccsType;
atribute atmInterfaceMaxVccsType atmInterfaceMaxVccs;
//--------------------
readonly attribute ASN1_Integer atmInterfaceConfVpcs
//--------------------
readonly attribute ASN1_Integer atmInterfaceConfVccs
```

Figura 22-Fragmento da MIB em IDL

6 TESTES COM O AMBIENTE DE GERÊNCIA

Para o desenvolvimento dos testes com o novo Ambiente de Gerência aqui proposto, não podemos contar com todos os equipamentos e ambientes com o qual gostaríamos de ter. Um dos problemas encontrados foi o de não ter um equipamento (como um switch ATM) equipado com um Agente implementado em Java com suporte a comunicação utilizando o CORBA e, consequentemente a sua MIB descrita em IDL.

Para contornar este problema, foi desenvolvido um Agente fictício todo em Java, com acesso ao padrão CORBA para poder estabelecer a comunicação com o Servidor. Também foi descrito uma interface em IDL que simulava uma MIB. Esta implementação utilizada para a simulação baseou-se em trabalhos desenvolvidos na disciplina de Computação Distribuída o qual foi desenvolvido um banco onde recebia parâmetros para consulta e escrita em sua base de dados (dinâmica, não escrita em disco).

Para desenvolver o Servidor foi necessário utilizar um servidor de Web para podermos colocar as páginas em HTML rodando, pois os serviços oferecidos pelos Servlets funcionam somente em servidores (não funciona abrindo um arquivo HTML).O servidor de Web escolhido para este teste foi da Sun Mirosystens [SUN99] chamado *Java Web Server(tm) 2.0* o qual é todo desenvolvido em Java e com suporte a Servlet. Este servidor de Web tem muitas das características que estão sendo usadas neste projeto, destacando-se a sua interação com o Java permitindo com isso uma boa gama de plataformas por ele atendido, como Windows NT, Windows 95 (somente para testes) e Solaris. Foi realizado um estudo para ver se o servidor Apache que é muito utilizado na Internet suportava Servlets e para agrado geral ele também tem uma opção onde podemos disponibilizar esse serviço.

Como este servidor é implementado em Java ele permite que outros processos Java fiquem rodando no mesmo equipamento, com isso podemos deixar todos os aplicativos que farão as buscas nos Objetos Gerenciados no mesmo equipamento. Obviamente conseguiremos uma boa performance no uso dos aplicativos, pois como mencionado anteriormente, os Servlets não encerram os processos depois de efetuado a solicitação.

6.1 Implementação do Servidor

A implementação do ambiente Servidor que será usado para no ambiente de testes foi feito de uma forma muito simples, somente para verificarmos se todos estas tecnologias juntas pudessem nos dar o retorno esperado.

Primeiramente foi feito uma página em HTML (Figura 23) para que pudéssemos nos conectar com o servidor Web e através desta página passar os dados necessários, através do Servlet, para a classe *ServidorRecebe* onde estão as chamadas das primitivas de gerência que irão se conectar com o Objeto Gerenciado. Nesta página não foi usado applets nem nenhum recurso mais avançado no desenvolvimento, para não a deixar muito pesada na hora da leitura. Faz-se uso somente de formulários para podermos digitar os campos e uma opção para enviar os dados. Esta nova idéia de não usar applets talvez seja adotado na totalidade da implementação, pois há um custo razoável de tempo de baixar o arquivo de uma página web que contém uma applet e, sua interpretação pelo browser tornando com isso um ambiente excessivamente lento para uma função tão vital e importante como a gerência.

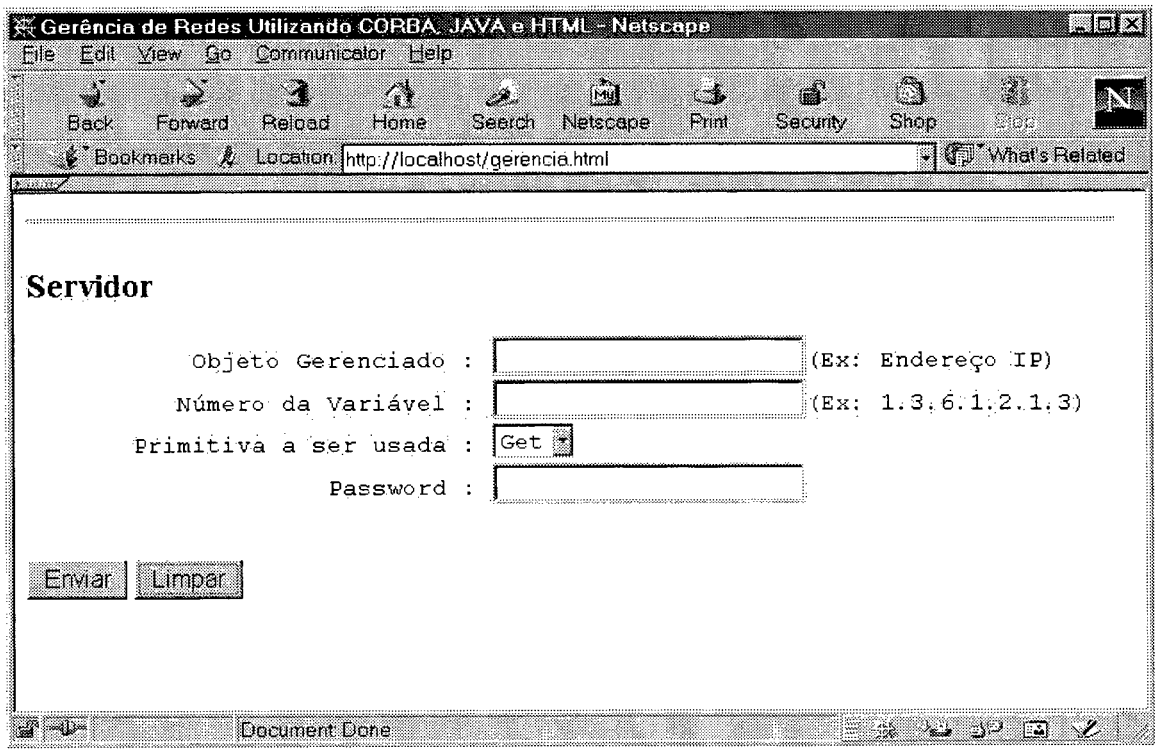

Figura 23-Tela inicial do Ambiente de Gerência

Dentro do código desta página (Figura 24) está a referência ao Servlet que será usado para receber os dados passados através dela e, o modo que estes dados serão transmitidos

(no caso utilizando o método POST). Também veremos no código as variáveis de ambiente onde serão armazenados os dados a serem interpretados pelas classes que efetuarão a conexão e transferência dos dados desde o Objeto Gerenciado.

```
<html><head><title>Gerência de Redes Utilizando CORBA, JAVA e HTML</title></head>
<body bgcolor="#ffffff">
<center><font size=5><b></b></font></center>
<hr>
<h3>Servidor</h3>
<form action="/gerencia/Teste.ServidorRecebe" method="post">
<pre>
 Objeto Gerenciado : <input type=textarea name="objeto" value="">(Ex: Endereço IP)
 Número da Variável : <input type=textarea name="var" value="">(Ex: 1.3.6.1.2.1.3)
 Primitiva a ser usada : <select name="dir" size="1">
                  <option value="get">Get </option>
                  <option value="set">Set </option>
                 </select>
 Password : <input type=password name=password value="">
</pre>
<input type="submit" value="Enviar">
<input type="reset" value="Limpar">
</form></body></html>
```

Figura 24-Código HTML da Página Inicial

Este código seria aconselhável estar em algum diretório dentro do servidor Web e de preferência com restrições de acesso (no ambiente Unix é fácil) para que com isso podemos ter uma segurança maior. No nosso caso especial além de estar em um diretório separado dos demais, ainda utilizamos o encapsulamento de Package que o Java oferece.

A implementação da API Servlet para o Java possui muitas classes e métodos que fazem tudo o que precisamos na hora de ler os dados passados através de páginas HTML. Todos os dados são enviados no formato de pares *nomes=valor* separados por um caractere *&*. Uma das grandes características no uso de Servlets é que toda a leitura de cabeçalhos e análise dos dados postados nos formulários é feito automaticamente simplesmente chamando o método *getParameter* que está na classe *HttpServletRequest*, fornecendo exatamente o nome do parâmetro e o argumento passado. Este método nos retorna uma String correspondendo a primeira ocorrência do parâmetro *name*.

É claro que nós iremos passar vários parâmetros e, para analisarmos todos, devemos chamar o método *getParametrerValues* em vez do *getParameter* que ele nos retornará um vetor de Strings. No nosso exemplo estamos enviando os valores armazenados nas variávies: *objeto* indica a identificação do Objeto Gerenciado (está sendo usado nesta

implementação a identificação através do número IP do Objeto); a variável *var* indica qual a variável que será pesquisada no Objeto Gerenciado; a variável *dir* indica qual diretiva de gerência que será utilizada; e a variável *password* será para verificar a senha do usuário e ver se ele possui direito ou não para utilizar o sistema. Vale salientar que não foi abordado nesta implementação a questão da segurança não foi abordado com a ênfase necessária que ela mereceria.

Depois que os dados são identificados nós iremos instanciar a classe que contém os métodos passando os seus valores por referência. Esta classe então irá conectar com o Objeto Gerenciado que vai retornar com o resultado da pesquisa e devolver ao programa *ServidorRecebe* que irá montar a página Web com os valores retornados e jogar para o computador de onde veio solicitação. Os códigos fontes destes aplicativos estão no anexo que acompanha este trabalho.

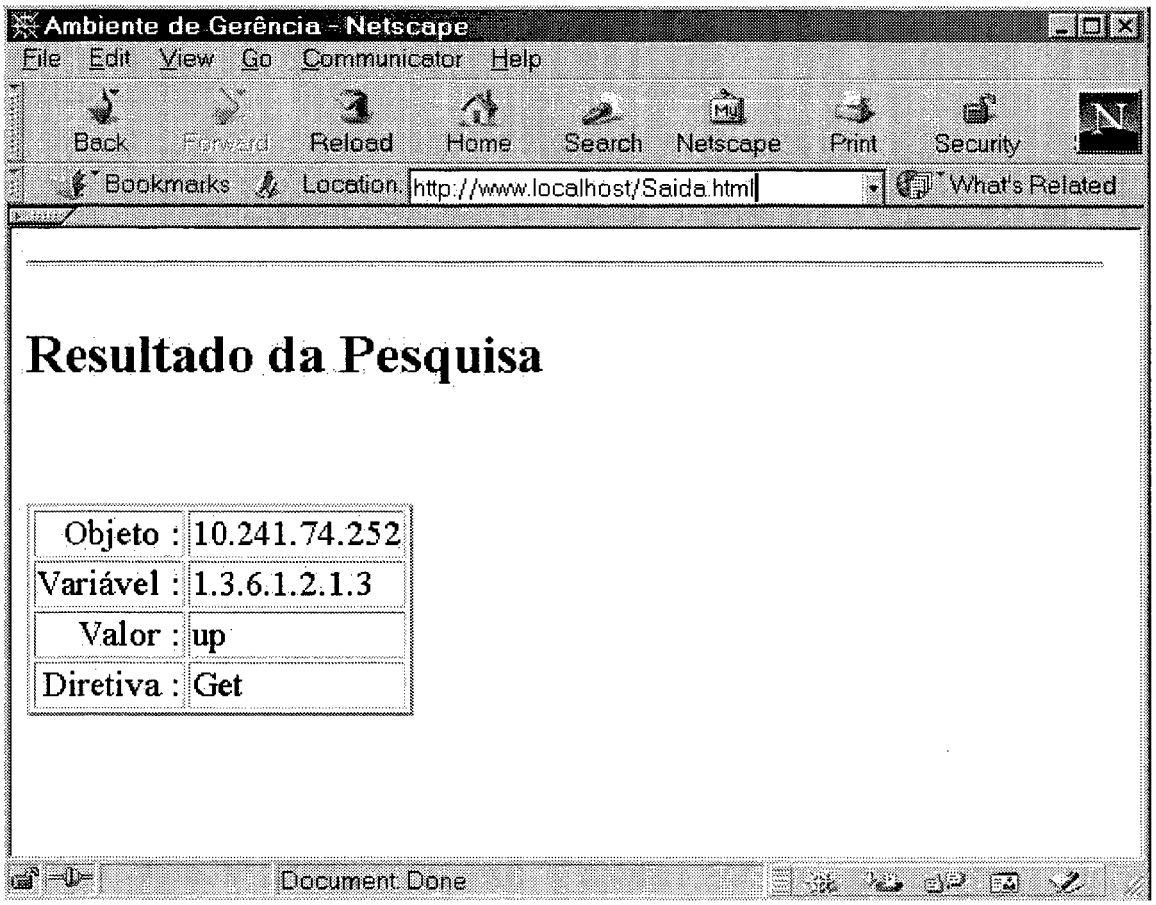

Figura 25-Página Web com o resultado da pesquisa

A página Web que teremos quando utilizarmos estes métodos será parecida com a apresentada na Figura 25. Está página é gerada dentro do Servlet que recebe os dados obtidos

através do CORBA quando feito a consulta ao objeto gerenciado. Estes dados são do formato *String* , e jogados na página com simples comandos de escrita gerados pelo Java, como vemos na Figura 26. Os campos do cabeçalho onde contém o endereço do computador que fez a solicitação ao Servlet estão armazenados dentro da classe *HttpServletResponse,* que se encarrega de armazenar todas as informações necessárias para podermos retornar a página.

```
private void printPageHeader(PrintWriter out) {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Ambiente de Gerência</title>");
            out.println("</head>");
            out.println("<body bgcolor=\"#ffffff\">");
            out.println("<hr><h3>Resultado da Pesquisa</h3>");
            out.println("<pre>");
            out.println("<hr>");
            out.println("<table border=3>");
           out.println("<tr>");
            out.println("<tr><td>  Objeto : </td>");
            out.println("<td>"+ objeto"</td>");
            out.println("<tr><td>   Variavel : </td>");
            out.println("<td>"+ var"</td>");
            out.println("<tr><td>     Valor : </td>");
            out.println("<td>"+ ler.pega()"</td>");
            out.println("<tr><td>  Diretiva : </td>");
            out.println("<td>"+ dir"</td>");
            out.println("</pre>"); */
            printPageFooter(out);
        }
    }
 private void printPageFooter(PrintWriter out) {
    out.println("</body>");
        out.println("</html>");
        out.flush();
```

Figura 26-Código da Página Web Gerada de Retorno

7 CONCLUSÕES

O objetivo principal traçado para este trabalho foi a de pesquisar e estudar as potencialidades de termos um ambiente de Gerência de Redes de Alta Velocidade com uma tecnologia totalmente nova, utilizando para isto uma arquitetura de objetos distribuídos, no caso o CORBA, e utilizando da linguagem de programação Java para instanciar os Objetos Gerenciados. Também foi proposto que esta gerência teria que utilizar o protocolo Http para fazer interface com o gerente que poderá utilizar qualquer equipamento dotado de um browser Web e ligado a Internet.

Num primeiro momento foi estudo a Gerência de Redes como um todo, observando os seus paradigmas e suas peculiaridades, estendendo logo depois para o protocolo de Gerência mais utilizado hoje em dia, que é o protocolo SNMP, o qual verificamos suas facilidades e simplicidade para o desenvolvimento do projeto, levando-se em conta o funcionamento e a estrutura que ele utiliza nos processos de leitura e escrita da MIB.

Logo após estudou-se o padrão CORBA para sistemas distribuído, onde vimos como podemos utilizar suas facilidade de localizar objetos na rede e sua facilidade de operação.

Também estudamos as Redes ATM que é uma das mais novas tecnologia de redes de alta velocidade em particular observamos como é executado a sua Gerência para o SNMP. O estudo da MIB-1695 foi aprofundado e em muito utilizado no restante do desenvolvimento do trabalho com base para a sua transformação para IDL.

Logo após é apresentado o Ambiente Proposto para a Gerência de Redes utilizando-se de todos as teorias estudadas e observando sempre o que há de mais novo em termos de comunicação no mundo da Internet, destacando quais as tecnologias que iríamos utilizar e o porque.

Por último, foi feito um pequeno teste do Ambiente aqui proposto. É claro que não nos foi possível implementar todo o projeto por ele ser muito extenso e complexo, mas para validação das plataformas que iremos utilizar já é o suficiente.

7.1 Resultados Obtidos

Não foi objetivo deste trabalho simular fielmente as características, o comportamento e a gerência dos equipamentos reais do ambiente. Como a proposta do ambiente é apenas uma simulação simples e abstraída sem compromisso com o mundo real, e também objetivando uma clareza acadêmica, a implementação conseguiu atingir seus objetivos.

Com o objetivo de pesquisarmos e estudarmos novas tecnologias que pudéssemos utilizar para fazer uma Gerência de Redes, usando como pré-requisitos para este projeto que deveria utilizar o CORBA como plataforma encarregada de localizar e se comunicação entre os equipamentos envolvidos no ambiente e, que deveremos utilizar a Web como instrumento de entrada e saída dos dados do ambiente, podemos concluir que este objetivo foi alcançado.

Este projeto baseou-se em um ambiente totalmente novo e, apesar de não implementar na totalidade o ambiente desejado (não foi possível desenvolver uma plataforma completa) os objetivos traçados inicialmente foram todos atingidos. Objetivos estes que tiverem principal interesse em juntar diversas tecnologias diferentes e utilizá-las para o desenvolvimento de um novo ambiente de gerência.

O uso do Java nos possibilitou um amplo conhecimento nesta linguagem e também podemos conhecer todas as tecnologias que esta linguagem nos proporciona e as ferramentas desenvolvidas para ela ser usada, ferramentas como o Servlet que foi desenvolvida exclusivamente para prover um ambiente de passagem de parâmetros entre o browser HTML e uma aplicação que fica no servidor de HTTP. Em cima desta tecnologia foi desenvolvido o nosso projeto.

7.2 Perspectivas Futuras

Neste trabalho foram efetuadas pesquisas visando a realização de um ambiente de gerência de redes de alta velocidade utilizando o CORBA, Java e HTML. Os objetivos inicialmente propostos para este trabalho foram alcançados. No entanto, existem ainda muitas pesquisas a serem realizadas nesta área.

Uma das principais perspectivas para a continuidade deste trabalho é de que ele é um tanto que inovador, se levarmos em conta os ambientes hoje utilizados e mais ainda se olharmos que a Internet está sendo usada para fazermos tudo o que antes não poderíamos.

Há uma grande esperança na finalização das implementações, para que possamos mostrar os resultados obtidos e tentarmos atrair mais pessoas para o estudo deste ambiente.

Outra perspectiva é em termos de segurança que que aqui não foi estudado, mas é de primordial importância se quisermos que este projeto tenha continuidade. Um exemplo que podemos citar é que todo ambiente gerenciado pelo SNMP, para nós termos uma gerência completa, teremos que abrir as portas dos equipamentos e, com isso atrair cada vez mais invasores.

7.3 Dificuldades Encontradas

Uma das principais dificuldades encontradas no desenvolvimento do projeto foi o de não poder dispor do tempo necessário para um estudo mais amplo e aprofundado. Este problema se deve principalmente ao motivo de não contar com uma bolsa de estudos que proporcionasse uma tranqüilidade a respeito de despesas oriundas da moradia e Florianópolis e estudando dentro da UFSC. Por este motivo tive que deixar a Universidade e trabalhar para me sustentar e, como todo trabalhador, encontrei várias dificuldades na questão de horários disponíveis para destinar exclusivamente aos estudos. Esta mudança também acarretou num outro empecilho muito grande, que foi a de não dispor de equipamentos vitais no desenvolvimento e aplicação das metodologias aqui expostas, destacando-se a não disposição de estações de trabalho como Sun, Risc, Switches ATM e sim somente microcomputadores do tipo PC.

Além dos problemas oriundos dos equipamentos, tenho que destacar que o não estar no ambiente de Pós-Graduação, onde há muitas cabeças pensantes e o convívio com professores que possam nos orientar mais diretamente também dificultou um pouco as coisas.

BIBLIOGRAFIA

. [ANER] ANEROUSIS, Nikolaos. **Scalable Managemente Services Using Java and the World Wide Web**. AT&T Labs Reserch, http://www.research.att.com/~nilkos/.

. [ASU98] ASUMAN, Dogac, Et al.**Distributed Object Computing Platforms. Communication of the ACM**- 1998/Vol.41, N° 9.

. [BAR98] BAROTO, André Mello. **Realização da Gerência Distribuída de Redes Utilizando SNMP, JAVA , WWW e CORBA**. Dissertação de Mestrado. UFSC. Florianópiolis – SC.1998.

. [BRI93] BRISA. **Gerenciamento de Redes: Uma Abordagem de Sistemas Abertos.** São Paulo: Editora Makron Books, 1993.

. [HAG98] HAGGERTY, Paul, SEETHARAMAN, Drishnan. **The Benefits of CORBA – Based Network Management. Communication of the ACM-** 1998/Vol.41, N° 10

. [INP00] INPRISE CORPORATION, Visibroker for Java – Version 3.3, Scotts Valley, CA.2000.

. [INT99] INTER-DOMAIN MANAGEMENT: SPECIFICATION TRANSLATION. http://www.opengroup.org/onlinepubs/8349099/front.htm 1999.

. [LAP97] LAPPINEM, Mika; PULKKINEN, Pekka; RAUTIAINEN, Aapo. **Java and *CORBA* based network management.** IEEE Computer. Junho de 1997. http://dlib.computer.org/

. [HAL99] HALL, Marty. **Tutorial on Servlets na JSP.** Johns Hopkins University – Applied Phisics Lab. 1999. www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

. [OBJ94-1] OBJECT MANAGEMENT GROUP. **IDL C++ Language Mapping Specification**. OMG Document, Setembro de 1994.

. [OBJ97-2] OBJECT MANAGEMENT GROUP. **Java Language Mapping RFP**. OMG Document, Março de 1997.

. [OBJ91-3] OBJECT MANAGEMENT GROUP. **The Common Object Request Brocker 2.0/IIOP specification Revision 1.1**. OMG Document, Dezembro de 1991.

. [OBJ96-4] OBJECT MANAGEMENT GROUP. **The Common Object Request Brocker 2.0/IIOP specification Revision 2.0**. OMG Document, Agosto de 1996.

. [OLI98] OLIVEIRA, Mauro, Et al. **Introdução a Gerência de Redes ATM**. XVI Simpósio Brasileiro de Redes de Computadores. Rio de Janeiro, 1998.

. [ORB] ORBYCOM, **SNMP MIB to CORBA-IDL translator**. França. www.orbycom.fr.

. [RFC1157] RFC 1157 - NETWORK WORKING GROUP **A Simple Network Management Protocol**. Network Working Group. EUA. 1990.

. [RFC1213] RFC 1213 - **Management Information Base for Network Management of TCP/IP-based internets: MIB-II**. Network Working Group. Hughes LAN Systems, Mellon University. Abril de 1993.

. [RFC1695] RFC 1695 – BELL COMMUNICATIONS RESEARCH. **Definitions of Managed Objects for ATM Management Version8.0 using SMIv2.** Nework Working Group. Red Bank. EUA 1994.

. [RIX] RIXON, Jeremy . **ATM Management Using HP DM CORBA and Java.** Artigo.

. [ROG99] ROGÉRIO, Ketter Ohnes. **Aplicação do Modelo TMN na Gerência de Redes de Alta Velocidade.** Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis, SC.1999.

. [SAM97] SAMPAIO, Sílvio. **SNMP – Uma introdução a Gerência de Redes.** Trabalho de Específico. Universidade de Salvador. Bahia. 1997.

. [SOA95] SOARES, L. F. G et alli – **Das LAN's MAN's e WAN's às Redes ATM.**

Segunda Edição. Rio de Janeiro. Editora Campus, 1995.

[SUN99] SUN MICROSSYSTEMS INC. The Java 2 SDK – Standard Edition Version 1.2.2. Palo Alto – USA. http://java.sun.com.1999.

[TAN97] TANEMBAUM, Andrew S. **Redes de Computadorer.** Tradução da 3° ed. Rio de Janeiro. Editora Campus, 1997.

[VAN] VANDERMEULEN, Filip, Et al. **On the Desing of an Integrated Management Platform for an ADSL/ATM basead Access Network using CORBA**. Departament of Information Techmology, University of Ghent – Belgium.

[WEI97] WEINMAN, William E. **Manual de CGI.** Trad. Daniel Vieira; São Paulo. Makron Books, 1997.

[WUT97] WUTKA, Mark. Et al. **Java – Técnicas Avançadas.** Trad. Luís Maian. São Paulo, Berkeley, 1997.

ANEXOS

Anexo – I

AToM MIB

Definitions

```
ATM-MIB DEFINITIONS ::= BEGIN
IMPORTS
MODULE-IDENTITY, OBJECT-TYPE, OBJECT-IDENTITY, Counter32, Integer32, IpAddress
FROM SNMPv2-SMI
TEXTUAL-CONVENTION, DisplayString,
TimeStamp, RowStatus
FROM SNMPv2-TC
MODULE-COMPLIANCE, OBJECT-GROUP
FROM SNMPv2-CONF
ifIndex, mib-2
FROM RFC1213-MIB;
atmMIB MODULE-IDENTITY
LAST-UPDATED "9406072245Z"
ORGANIZATION "IETF AToM MIB Working Group" CONTACT-INFO
                "       Masuma Ahmed
                 Postal: Bellcore
                         331 Newman Springs Road
                         Red Bank, NJ 07701
                         US
                  Tel:   +1 908 758 2515
                  Fax:   +1 908 758 4131
                  E-mail: mxa@mail.bellcore.com
Kaj Tesink
Postal: Bellcore
331 Newman Springs Road
Red Bank, NJ 07701
US
                  Tel:   +1 908 758 5254
                  Fax:   +1 908 758 4196
                  E-mail: kaj@cc.bellcore.com"

          atmMIBObjects  OBJECT IDENTIFIER ::= {atmMIB 1}

          -- This ATM MIB Module consists of the following groups:

          -- (1) ATM Interface configuration group
          -- (2) ATM Interface DS3 PLCP group
          -- (3) ATM Interface TC Sublayer group
          -- (4) ATM Interface VPL configuration group
          -- (5) ATM Interface VCL configuration group
          -- (6) ATM VP Cross Connect group
          -- (7) ATM VC Cross Connect group
          -- (8) ATM Interface AAL5 VCC performance statistics
          --    group

          IfIndex ::= TEXTUAL-CONVENTION
             STATUS    current
             SYNTAX    Integer32

          AtmTrafficDescrParamIndex ::= TEXTUAL-CONVENTION
             STATUS    current
             SYNTAX    Integer32
```

```
atmTrafficDescriptorTypes  OBJECT IDENTIFIER ::=
                        {atmMIBObjects 1}

-- The following values are defined for use as
-- possible values of the ATM traffic descriptor type.
-- ATM Forum specified seven types of ATM traffic
-- descriptors.
    atmNoTrafficDescriptor OBJECT-IDENTITY
    STATUS    current

    ::= { atmTrafficDescriptorTypes 1}
    atmNoClpNoScr OBJECT-IDENTITY
    STATUS    current
    ::= { atmTrafficDescriptorTypes 2}
    atmClpNoTaggingNoScr OBJECT-IDENTITY
    STATUS    current
    ::= { atmTrafficDescriptorTypes 3}
    atmClpTaggingNoScr OBJECT-IDENTITY
   STATUS    current

   ::= { atmTrafficDescriptorTypes 4}
    atmNoClpScr OBJECT-IDENTITY
    STATUS    current


   ::= { atmTrafficDescriptorTypes 5}
   atmClpNoTaggingScr OBJECT-IDENTITY
    STATUS    current

   ::= { atmTrafficDescriptorTypes 6}
   atmClpTaggingScr OBJECT-IDENTITY
    STATUS    current

    ::= { atmTrafficDescriptorTypes 7}

--   ATM Interface Configuration Parameters Group

-- This group contains ATM specific
-- configuration information associated with
-- an ATM interface beyond those
-- supported using the ifTable.

atmInterfaceConfTable   OBJECT-TYPE
    SYNTAX     SEQUENCE OF AtmInterfaceConfEntry
    MAX-ACCESS  not-accessible
    STATUS      current

    ::= { atmMIBObjects 2 }

atmInterfaceConfEntry   OBJECT-TYPE
    SYNTAX       AtmInterfaceConfEntry
    MAX-ACCESS    not-accessible
    STATUS       current

    INDEX { ifIndex }
    ::= { atmInterfaceConfTable  1}

AtmInterfaceConfEntry   ::= SEQUENCE  {
    atmInterfaceMaxVpcs          INTEGER,
    atmInterfaceMaxVccs          INTEGER,
    atmInterfaceConfVpcs         INTEGER,
    atmInterfaceConfVccs         INTEGER,
    atmInterfaceMaxActiveVpiBits   INTEGER,
    atmInterfaceMaxActiveVciBits   INTEGER,
    atmInterfaceImiVpi           INTEGER,
```

```
    atmInterfaceIlmiVci          INTEGER,
    atmInterfaceAddressType      INTEGER,
    atmInterfaceAdminAddress     OCTET STRING,
    atmInterfaceMyNeighborIpAddress IpAddress,
    atmInterfaceMyNeighborIfName   DisplayString
       }

atmInterfaceMaxVpcs   OBJECT-TYPE
    SYNTAX     INTEGER (0..4096)
    MAX-ACCESS   read-write
    STATUS       current

    ::= { atmInterfaceConfEntry 1}

atmInterfaceMaxVccs   OBJECT-TYPE
    SYNTAX     INTEGER (0..65536)
    MAX-ACCESS    read-write
    STATUS       current

    ::= { atmInterfaceConfEntry 2}

atmInterfaceConfVpcs   OBJECT-TYPE
    SYNTAX     INTEGER (0..4096)
    MAX-ACCESS    read-only
    STATUS       current

    ::= { atmInterfaceConfEntry 3}

atmInterfaceConfVccs   OBJECT-TYPE
    SYNTAX     INTEGER (0..65536)
    MAX-ACCESS   read-only
    STATUS       current

    ::= { atmInterfaceConfEntry 4}

atmInterfaceMaxActiveVpiBits   OBJECT-TYPE
    SYNTAX         INTEGER (0..12)
    MAX-ACCESS        read-write
    STATUS          current

    ::= { atmInterfaceConfEntry 5}

atmInterfaceMaxActiveVciBits   OBJECT-TYPE
    SYNTAX          INTEGER (0..16)
    MAX-ACCESS        read-write
    STATUS          current

    ::= { atmInterfaceConfEntry 6}

atmInterfaceIlmiVpi       OBJECT-TYPE
    SYNTAX        INTEGER (0..255)
    MAX-ACCESS      read-write
    STATUS         current

    DEFVAL { 0 }
    ::= { atmInterfaceConfEntry 7}

atmIn INTEGER (0..65535)
    MAX-AC terfaceIlmiVci       OBJECT-TYPE
    SYNTAX CESS      read-write
    STATUS         current

    DEFVAL { 16 }
    ::= { atmInterfaceConfEntry 8}
```

```
atmInterfaceAddressType   OBJECT-TYPE
   SYNTAX     INTEGER {
                private(1),
                nsapE164(2),
                nativeE164(3),
                other(4)
               }
   MAX-ACCESS   read-only
   STATUS       current

   ::= { atmInterfaceConfEntry 9 }

atmInterfaceAdminAddress   OBJECT-TYPE
   SYNTAX     OCTET STRING (SIZE(0..255))
   MAX-ACCESS   read-only
   STATUS       current

   ::= { atmInterfaceConfEntry 10 }
   atmInterfaceMyNeighborIpAddress OBJECT-TYPE
   SYNTAX     IpAddress
   MAX-ACCESS  read-write
   STATUS      current

   ::= { atmInterfaceConfEntry 11 }

atmInterfaceMyNeighborIfName   OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS  read-write
   STATUS      current


   ::= { atmInterfaceConfEntry 12 }

-- The ATM Interface DS3 PLCP Group

-- This group contains the DS3 PLCP configuration and
-- state parameters of those ATM interfaces
-- which use DS3 PLCP for carrying ATM cells over DS3.

atmInterfaceDs3PlcpTable   OBJECT-TYPE
   SYNTAX     SEQUENCE OF AtmInterfaceDs3PlcpEntry
   MAX-ACCESS  not-accessible
   STATUS      current

   ::= { atmMIBObjects 3}

atmInterfaceDs3PlcpEntry   OBJECT-TYPE
   SYNTAX     AtmInterfaceDs3PlcpEntry
   MAX-ACCESS  not-accessible
   STATUS      current

   INDEX   {ifIndex }
   ::= { atmInterfaceDs3PlcpTable 1}

AtmInterfaceDs3PlcpEntry   ::= SEQUENCE  {
   atmInterfaceDs3PlcpSEFSs       Counter32,
   atmInterfaceDs3PlcpAlarmState   INTEGER,
   atmInterfaceDs3PlcpUASs        Counter32
    }

atmInterfaceDs3PlcpSEFSs   OBJECT-TYPE
   SYNTAX     Counter32
   MAX-ACCESS  read-only
   STATUS      current
```

```
    ::= { atmInterfaceDs3PlcpEntry 1}

atmInterfaceDs3PlcpAlarmState   OBJECT-TYPE
    SYNTAX      INTEGER {
                noAlarm(1),
                receivedFarEndAlarm(2),
                incomingLOF(3)
                    }
    MAX-ACCESS   read-only
    STATUS       current

    ::= { atmInterfaceDs3PlcpEntry 2}

atmInterfaceDs3PlcpUASs   OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current

    ::= { atmInterfaceDs3PlcpEntry 3}

-- The ATM Interface TC Sublayer Group

-- This group contains TC sublayer configuration and
-- state parameters of those ATM interfaces
-- which use TC sublayer for carrying ATM cells over
-- SONET or DS3.

atmInterfaceTCTable   OBJECT-TYPE
    SYNTAX     SEQUENCE OF AtmInterfaceTCEntry
    MAX-ACCESS  not-accessible
    STATUS      current

    ::= { atmMIBObjects 4}

atmInterfaceTCEntry   OBJECT-TYPE
    SYNTAX        AtmInterfaceTCEntry
    MAX-ACCESS    not-accessible
    STATUS        current

    INDEX  {ifIndex }
    ::= { atmInterfaceTCTable 1}

AtmInterfaceTCEntry   ::= SEQUENCE  {
    atmInterfaceOCDEvents    Counter32,
    atmInterfaceTCAlarmState  INTEGER
     }

atmInterfaceOCDEvents   OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current

    ::= { atmInterfaceTCEntry 1}

atmInterfaceTCAlarmState   OBJECT-TYPE
    SYNTAX      INTEGER {
                noAlarm(1),
                lcdFailure(2)
                }
    MAX-ACCESS  read-only
    STATUS      current

     ::= { atmInterfaceTCEntry 2}

-- ATM Traffic Descriptor Parameter Group
```

```
-- The ATM Traffic Descriptor Parameter Table

atmTrafficDescrParamTable   OBJECT-TYPE
   SYNTAX     SEQUENCE OF AtmTrafficDescrParamEntry
   MAX-ACCESS  not-accessible
   STATUS      current

   ::= { atmMIBObjects 5}

atmTrafficDescrParamEntry   OBJECT-TYPE
   SYNTAX         AtmTrafficDescrParamEntry
   MAX-ACCESS      not-accessible
   STATUS         current

   INDEX {atmTrafficDescrParamIndex}
   ::= { atmTrafficDescrParamTable 1}

AtmTrafficDescrParamEntry   ::= SEQUENCE {
 atmTrafficDescrParamIndex  AtmTrafficDescrParamIndex,
 atmTrafficDescrType        OBJECT IDENTIFIER,
 atmTrafficDescrParam1      Integer32,
 atmTrafficDescrParam2      Integer32,
 atmTrafficDescrParam3      Integer32,
 atmTrafficDescrParam4      Integer32,
 atmTrafficDescrParam5      Integer32,
 atmTrafficQoSClass         INTEGER,
 atmTrafficDescrRowStatus   RowStatus
         }

atmTrafficDescrParamIndex   OBJECT-TYPE
   SYNTAX         AtmTrafficDescrParamIndex
   MAX-ACCESS      not-accessible
   STATUS      current

   ::= { atmTrafficDescrParamEntry 1}

atmTrafficDescrType   OBJECT-TYPE
   SYNTAX          OBJECT IDENTIFIER
   MAX-ACCESS       read-create
   STATUS          current

    DEFVAL { atmNoTrafficDescriptor }
   ::= { atmTrafficDescrParamEntry 2}

atmTrafficDescrParam1   OBJECT-TYPE
   SYNTAX          Integer32
   MAX-ACCESS       read-create
   STATUS          current

   DEFVAL { 0 }
   ::= { atmTrafficDescrParamEntry 3}

atmTrafficDescrParam2   OBJECT-TYPE
   SYNTAX          Integer32
   MAX-ACCESS       read-create
   STATUS          current

   DEFVAL { 0 }
   ::= { atmTrafficDescrParamEntry 4}

atmTrafficDescrParam3   OBJECT-TYPE
   SYNTAX          Integer32
   MAX-ACCESS       read-create
```

```
        STATUS          current

        DEFVAL { 0 }
        ::= { atmTrafficDescrParamEntry 5}

    atmTrafficDescrParam4   OBJECT-TYPE
        SYNTAX          Integer32
        MAX-ACCESS      read-create
        STATUS          current

        DEFVAL { 0 }
        ::= { atmTrafficDescrParamEntry 6}

    atmTrafficDescrParam5   OBJECT-TYPE
        SYNTAX          Integer32
        MAX-ACCESS      read-create
        STATUS          current

        DEFVAL { 0 }
        ::= { atmTrafficDescrParamEntry 7}

    atmTrafficQoSClass  OBJECT-TYPE
        SYNTAX      INTEGER (0..255)
        MAX-ACCESS  read-create
        STATUS      current

        DEFVAL { 0 }
        ::= { atmTrafficDescrParamEntry 8}

    atmTrafficDescrRowStatus    OBJECT-TYPE
        SYNTAX  RowStatus
        MAX-ACCESS  read-create
        STATUS      current

        DEFVAL { active }
        ::= {atmTrafficDescrParamEntry 9}

    -- ATM Interface Virtual Path Link (VPL) Group


    -- The ATM Interface VPL Table

    atmVplTable   OBJECT-TYPE
        SYNTAX    SEQUENCE OF AtmVplEntry
        MAX-ACCESS  not-accessible
        STATUS    current

        ::= { atmMIBObjects 6}

    atmVplEntry   OBJECT-TYPE
        SYNTAX      AtmVplEntry
        MAX-ACCESS  not-accessible
        STATUS      current

INDEX {ifIndex, atmVplVpi }
        ::= { atmVplTable  1}

    AtmVplEntry   ::= SEQUENCE  {
        atmVplVpi               INTEGER,
        atmVplAdminStatus       INTEGER,
        atmVplOperStatus        INTEGER,
        atmVplLastChange        TimeStamp,
        atmVplReceiveTrafficDescrIndex
                AtmTrafficDescrParamIndex,
        atmVplTransmitTrafficDescrIndex
```

```
            AtmTrafficDescrParamIndex,
    atmVplCrossConnectIdentifier   INTEGER,
    atmVplRowStatus              RowStatus
      }

atmVplVpi   OBJECT-TYPE
    SYNTAX   INTEGER (1..4095)
    MAX-ACCESS    not-accessible
    STATUS       current

    ::= { atmVplEntry 1}

atmVplAdminStatus  OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),
                down(2)
                  }
    MAX-ACCESS  read-create
    STATUS      current

    DEFVAL { down }
    ::= { atmVplEntry 2}

atmVplOperStatus  OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),
                down(2),
                unknown(3)
                  }
    MAX-ACCESS  read-only
    STATUS      current

    ::= { atmVplEntry 3}

atmVplLastChange    OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current

    ::= { atmVplEntry 4 }

atmVplReceiveTrafficDescrIndex   OBJECT-TYPE
    SYNTAX          AtmTrafficDescrParamIndex
    MAX-ACCESS        read-create
    STATUS          current

     ::= { atmVplEntry 5}

atmVplTransmitTrafficDescrIndex   OBJECT-TYPE
    SYNTAX          AtmTrafficDescrParamIndex
    MAX-ACCESS        read-create
    STATUS          current

     ::= { atmVplEntry 6}

atmVplCrossConnectIdentifier  OBJECT-TYPE
    SYNTAX       INTEGER (0..2147483647)
    MAX-ACCESS     read-only
    STATUS        current

    ::= {atmVplEntry 7}

atmVplRowStatus      OBJECT-TYPE
    SYNTAX   RowStatus
    MAX-ACCESS  read-create
```

```
    STATUS        current

    DEFVAL { active }
    ::= {atmVplEntry 8}

-- ATM Interface Virtual Channel Link (VCL) Group


-- The ATM Interface VCL Table

atmVclTable    OBJECT-TYPE
    SYNTAX     SEQUENCE OF AtmVclEntry
    MAX-ACCESS  not-accessible
    STATUS      current

    ::= { atmMIBObjects 7}

atmVclEntry    OBJECT-TYPE
    SYNTAX         AtmVclEntry
    MAX-ACCESS     not-accessible
    STATUS         current

  INDEX {ifIndex, atmVclVpi, atmVclVci }
    ::= { atmVclTable  1}

AtmVclEntry    ::= SEQUENCE  {
    atmVclVpi                   INTEGER,
    atmVclVci                   INTEGER,
    atmVclAdminStatus           INTEGER,
    atmVclOperStatus            INTEGER,
    atmVclLastChange            TimeStamp,
    atmVclReceiveTrafficDescrIndex
                AtmTrafficDescrParamIndex,
    atmVclTransmitTrafficDescrIndex
                AtmTrafficDescrParamIndex,
    atmVccAalType               INTEGER,
    atmVccAal5CpcsTransmitSduSize   INTEGER,
    atmVccAal5CpcsReceiveSduSize    INTEGER,
    atmVccAal5EncapsType        INTEGER,
    atmVclCrossConnectIdentifier    INTEGER,
    atmVclRowStatus             RowStatus
        }

atmVclVpi    OBJECT-TYPE
    SYNTAX    INTEGER (0..4095)
    MAX-ACCESS    not-accessible
    STATUS        current

    ::= { atmVclEntry 1}

atmVclVci    OBJECT-TYPE
    SYNTAX    INTEGER (0..65535)
    MAX-ACCESS    not-accessible
    STATUS        current

    ::= { atmVclEntry 2}

atmVclAdminStatus  OBJECT-TYPE
    SYNTAX   INTEGER {
                up(1),
                down(2)
                  }
    MAX-ACCESS  read-create
    STATUS      current
```

```
    ::= { atmVclEntry 3}

atmVclOperStatus  OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),
                down(2),
                unknown(3)
                  }
    MAX-ACCESS  read-only
    STATUS      current

    ::= { atmVclEntry 4}

atmVclLastChange    OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current

    ::= { atmVclEntry 5 }

atmVclReceiveTrafficDescrIndex   OBJECT-TYPE
    SYNTAX          AtmTrafficDescrParamIndex
    MAX-ACCESS      read-create
    STATUS          current

    ::= { atmVclEntry 6}

atmVclTransmitTrafficDescrIndex   OBJECT-TYPE
    SYNTAX          AtmTrafficDescrParamIndex
    MAX-ACCESS      read-create
    STATUS          current

    ::= { atmVclEntry 7}

atmVccAalType   OBJECT-TYPE
    SYNTAX      INTEGER {
                aal1(1),
                aal34(2),
                aal5(3),
                other(4),
                unknown(5)
                }
    MAX-ACCESS   read-create
    STATUS       current

    ::= { atmVclEntry 8 }

    atmVccAal5CpcsTransmitSduSize OBJECT-TYPE
    SYNTAX     INTEGER (1..65535)
    MAX-ACCESS  read-create
    STATUS      current

    DEFVAL { 9188 }
        ::= { atmVclEntry 9 }
    atmVccAal5CpcsReceiveSduSize OBJECT-TYPE
    SYNTAX     INTEGER (1..65535)
    MAX-ACCESS  read-create
    STATUS      current

    DEFVAL { 9188 }
        ::= { atmVclEntry 10 }
    atmVccAal5EncapsType OBJECT-TYPE
    SYNTAX  INTEGER {
              vcMultiplexRoutedProtocol(1),
```

```
            vcMultiplexBridgedProtocol8023(2),
            vcMultiplexBridgedProtocol8025(3),
            vcMultiplexBridgedProtocol8026(4),
            vcMultiplexLANemulation8023(5),
            vcMultiplexLANemulation8025(6),
            llcEncapsulation(7),
            multiprotocolFrameRelaySscs(8),
            other(9),
            unknown(10)
              }
    MAX-ACCESS  read-create
    STATUS      current

     DEFVAL { llcEncapsulation }
        ::= { atmVclEntry 11 }

atmVclCrossConnectIdentifier  OBJECT-TYPE
    SYNTAX       INTEGER (0..2147483647)
    MAX-ACCESS     read-only
    STATUS       current

    ::= {atmVclEntry 12}

atmVclRowStatus       OBJECT-TYPE
    SYNTAX     RowStatus
    MAX-ACCESS  read-create
    STATUS     current

    DEFVAL { active }
        ::= {atmVclEntry 13}

--    ATM Virtual Path (VP) Cross Connect Group


atmVpCrossConnectIndexNext   OBJECT-TYPE
    SYNTAX  INTEGER (0..2147483647)
    MAX-ACCESS  read-only
    STATUS  current

    ::= { atmMIBObjects 8 }

 -- The ATM VP Cross Connect Table
atmVpCrossConnectTable OBJECT-TYPE
SYNTAX SEQUENCE OF AtmVpCrossConnectEntry MAX-ACCESS not-accessible
STATUS current

            ::= { atmMIBObjects 9 }
atmVpCrossConnectEntry OBJECT-TYPE
SYNTAX AtmVpCrossConnectEntry
MAX-ACCESS not-accessible
STATUS current

INDEX { atmVpCrossConnectIndex,
atmVpCrossConnectLowIfIndex,
atmVpCrossConnectLowVpi,
atmVpCrossConnectHighIfIndex,
atmVpCrossConnectHighVpi }
    ::= { atmVpCrossConnectTable 1 }

 AtmVpCrossConnectEntry ::= SEQUENCE {
    atmVpCrossConnectIndex        INTEGER,
    atmVpCrossConnectLowIfIndex    IfIndex,
    atmVpCrossConnectLowVpi       INTEGER,
    atmVpCrossConnectHighIfIndex   IfIndex,
```

```
    atmVpCrossConnectHighVpi       INTEGER,
    atmVpCrossConnectAdminStatus   INTEGER,
    atmVpCrossConnectL2HOperStatus INTEGER,
    atmVpCrossConnectH2LOperStatus INTEGER,
    atmVpCrossConnectL2HLastChange TimeStamp,
    atmVpCrossConnectH2LLastChange TimeStamp,
    atmVpCrossConnectRowStatus     RowStatus
        }
atmVpCrossConnectIndex OBJECT-TYPE
SYNTAX INTEGER (1..2147483647)
MAX-ACCESS not-accessible
STATUS current

    ::= { atmVpCrossConnectEntry 1 }
atmVpCrossConnectLowIfIndex OBJECT-TYPE
SYNTAX IfIndex
MAX-ACCESS not-accessible
STATUS current

    ::= { atmVpCrossConnectEntry 3 }
atmVpCrossConnectHighIfIndex OBJECT-TYPE
SYNTAX IfIndex
MAX-ACCESS not-accessible
STATUS current

    ::= { atmVpCrossConnectEntry 4 }
 atmVpCrossConnectHighVpi OBJECT-TYPE
    SYNTAX  INTEGER  (1..4095)
    MAX-ACCESS  not-accessible
    STATUS  current

    ::= { atmVpCrossConnectEntry 5 }
atmVpCrossConnectAdminStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2)
}
MAX-ACCESS read-create
STATUS current

DEFVAL { down }
    ::= { atmVpCrossConnectEntry 6 }
atmVpCrossConnectL2HOperStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2),
unknown(3)
}
MAX-ACCESS read-only
STATUS current

    ::= { atmVpCrossConnectEntry 7 }
atmVpCrossConnectH2LOperStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2),
unknown(3)
}
MAX-ACCESS read-only
STATUS current

            ::= { atmVpCrossConnectEntry 8 }
atmVpCrossConnectL2HLastChange OBJECT-TYPE
SYNTAX TimeStamp
MAX-ACCESS read-only
```

```
STATUS current

    ::= { atmVpCrossConnectEntry 9 }
atmVpCrossConnectH2LLastChange OBJECT-TYPE
SYNTAX TimeStamp
MAX-ACCESS read-only
STATUS current

    ::= { atmVpCrossConnectEntry 10 }
atmVpCrossConnectRowStatus OBJECT-TYPE
SYNTAX RowStatus
MAX-ACCESS read-create
STATUS current

DEFVAL { active }
    ::= { atmVpCrossConnectEntry 11 }

--   ATM Virtual Channel (VC) Cross Connect Group


atmVcCrossConnectIndexNext   OBJECT-TYPE
    SYNTAX  INTEGER (0..2147483647)
    MAX-ACCESS  read-only
    STATUS  current

    ::= { atmMIBObjects 10 }

-- The ATM VC Cross Connect Table

  atmVcCrossConnectTable OBJECT-TYPE
  SYNTAX SEQUENCE OF AtmVcCrossConnectEntry MAX-ACCESS not-accessible
  STATUS current

    ::= { atmMIBObjects 11 }
     atmVcCrossConnectEntry OBJECT-TYPE
     SYNTAX AtmVcCrossConnectEntry
     MAX-ACCESS not-accessible
     STATUS current

     INDEX { atmVcCrossConnectIndex,
     atmVcCrossConnectLowIfIndex,
     atmVcCrossConnectLowVpi,
     atmVcCrossConnectLowVci,
     atmVcCrossConnectHighIfIndex,
     atmVcCrossConnectHighVpi,
     atmVcCrossConnectHighVci }
     ::= { atmVcCrossConnectTable 1 }
    AtmVcCrossConnectEntry ::= SEQUENCE {
     atmVcCrossConnectIndex         INTEGER,
     atmVcCrossConnectLowIfIndex    IfIndex,
     atmVcCrossConnectLowVpi        INTEGER,
     atmVcCrossConnectLowVci        INTEGER,
     atmVcCrossConnectHighIfIndex   IfIndex,
     atmVcCrossConnectHighVpi       INTEGER,
     atmVcCrossConnectHighVci       INTEGER,
     atmVcCrossConnectAdminStatus   INTEGER,
     atmVcCrossConnectL2HOperStatus  INTEGER,
     atmVcCrossConnectH2LOperStatus  INTEGER,
     atmVcCrossConnectL2HLastChange  TimeStamp,
     atmVcCrossConnectH2LLastChange  TimeStamp,
     atmVcCrossConnectRowStatus     RowStatus
          }
     atmVcCrossConnectIndex OBJECT-TYPE
     SYNTAX INTEGER (1..2147483647)
```

```
MAX-ACCESS not-accessible
STATUS current

::= { atmVcCrossConnectEntry 1 }
atmVcCrossConnectLowIfIndex OBJECT-TYPE
SYNTAX IfIndex
MAX-ACCESS not-accessible
STATUS current

::= { atmVcCrossConnectEntry 2 }
atmVcCrossConnectLowVpi OBJECT-TYPE
SYNTAX INTEGER (0..4095)
MAX-ACCESS not-accessible
STATUS current

::= { atmVcCrossConnectEntry 3 }
atmVcCrossConnectLowVci OBJECT-TYPE
SYNTAX INTEGER (0..65535)
MAX-ACCESS not-accessible
STATUS current

::= { atmVcCrossConnectEntry 4 }
atmVcCrossConnectHighIfIndex OBJECT-TYPE
SYNTAX IfIndex
MAX-ACCESS not-accessible
STATUS current

::= { atmVcCrossConnectEntry 5 }
atmVcCrossConnectHighVpi OBJECT-TYPE
SYNTAX  INTEGER  (0..4095)
MAX-ACCESS  not-accessible
STATUS  current

::= { atmVcCrossConnectEntry 6 }
atmVcCrossConnectHighVci OBJECT-TYPE
SYNTAX  INTEGER  (0..65535)
MAX-ACCESS  not-accessible
STATUS current

::= { atmVcCrossConnectEntry 7 }
atmVcCrossConnectAdminStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2)
}
MAX-ACCESS read-create
STATUS current
DEFVAL { down }
::= { atmVcCrossConnectEntry 8 }
atmVcCrossConnectL2HOperStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2),
unknown(3)
}
MAX-ACCESS read-only
STATUS current

::= { atmVcCrossConnectEntry 9 }
atmVcCrossConnectH2LOperStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1),
down(2),
unknown(3)
}
```

```
    MAX-ACCESS read-only
    STATUS current

    ::= { atmVcCrossConnectEntry 10 }
    atmVcCrossConnectL2HLastChange OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current

    ::= { atmVcCrossConnectEntry 11 }
    atmVcCrossConnectH2LLastChange OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current

    ::= { atmVcCrossConnectEntry 12 }
    atmVcCrossConnectRowStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current

    ::= { atmVcCrossConnectEntry 13 }

-- AAL5 Virtual Channel Connection Performance Statistics
-- Group

-- This group contains the AAL5
-- performance statistics of a VCC at the
-- interface associated with an AAL5 entity in an ATM
-- host or ATM switch.

aal5VccTable   OBJECT-TYPE
   SYNTAX     SEQUENCE OF Aal5VccEntry
   MAX-ACCESS  not-accessible
   STATUS      current

   ::= { atmMIBObjects 12 }

aal5VccEntry   OBJECT-TYPE
   SYNTAX       Aal5VccEntry
   MAX-ACCESS    not-accessible
   STATUS       current

   INDEX { ifIndex, aal5VccVpi, aal5VccVci }
   ::= { aal5VccTable 1 }

  Aal5VccEntry   ::= SEQUENCE  {
   aal5VccVpi                  INTEGER,
   aal5VccVci                  INTEGER,
   aal5VccCrcErrors              Counter32,
   aal5VccSarTimeOuts             Counter32,
   aal5VccOverSizedSDUs           Counter32
      }

aal5VccVpi            OBJECT-TYPE
   SYNTAX     INTEGER (0..4095)
   MAX-ACCESS   not-accessible
   STATUS       current

   ::= { aal5VccEntry 1 }

aal5VccVci            OBJECT-TYPE
   SYNTAX     INTEGER (0..65535)
   MAX-ACCESS   not-accessible
   STATUS       current
```

```
    ::= { aal5VccEntry 2 }

aal5VccCrcErrors   OBJECT-TYPE
    SYNTAX  Counter32
    MAX-ACCESS  read-only
    STATUS        current

    ::= { aal5VccEntry 3 }

aal5VccSarTimeOuts   OBJECT-TYPE
    SYNTAX  Counter32
    MAX-ACCESS  read-only
    STATUS        current

    ::= { aal5VccEntry 4 }

aal5VccOverSizedSDUs   OBJECT-TYPE
    SYNTAX  Counter32
     MAX-ACCESS  read-only
    STATUS        current

    ::= { aal5VccEntry 5 }

-- Conformance Information

atmMIBConformance   OBJECT IDENTIFIER ::= { atmMIB 2 }

atmMIBGroups       OBJECT IDENTIFIER
                   ::= { atmMIBConformance 1 }
atmMIBCompliances  OBJECT IDENTIFIER
                   ::= { atmMIBConformance 2 }

-- Compliance Statements

atmMIBCompliance      MODULE-COMPLIANCE
    STATUS     current

       MODULE -- this module
       MANDATORY-GROUPS {atmInterfaceConfGroup, atmTrafficDescrGroup}
      OBJECT   atmInterfaceMaxVpcs
      MIN-ACCESS read-only

      OBJECT   atmInterfaceMaxVccs
      MIN-ACCESS read-only
      OBJECT   atmInterfaceMaxActiveVpiBits
      MIN-ACCESS read-only

      OBJECT   atmInterfaceMaxActiveVciBits
      MIN-ACCESS read-only

      OBJECT   atmInterfaceImiVpi
      MIN-ACCESS read-only
     OBJECT   atmInterfaceImiVci
      MIN-ACCESS read-only

      OBJECT   atmInterfaceMyNeighborIpAddress
      MIN-ACCESS read-only

      OBJECT   atmInterfaceMyNeighborIfName
      MIN-ACCESS read-only

      OBJECT   atmTrafficDescrType
      MIN-ACCESS read-only
```

```
OBJECT  atmTrafficDescrParam1
MIN-ACCESS read-only

OBJECT  atmTrafficDescrParam2
MIN-ACCESS read-only

OBJECT  atmTrafficDescrParam3
MIN-ACCESS read-only
OBJECT  atmTrafficDescrParam4
MIN-ACCESS read-only

OBJECT  atmTrafficDescrParam5
MIN-ACCESS read-only

OBJECT  atmTrafficQoSClass
MIN-ACCESS read-only

OBJECT  atmTrafficDescrRowStatus
SYNTAX  INTEGER {active(1)}
        -- subset of RowStatus
MIN-ACCESS read-only


GROUP   atmInterfaceDs3PlcpGroup


GROUP   atmInterfaceTCGroup

GROUP   atmVpcTerminationGroup

GROUP   atmVpCrossConnectGroup
OBJECT  atmVplVpi
SYNTAX  INTEGER (1..255)

OBJECT  atmVplAdminStatus
MIN-ACCESS read-only

OBJECT  atmVplReceiveTrafficDescrIndex
MIN-ACCESS read-only

OBJECT  atmVplTransmitTrafficDescrIndex
MIN-ACCESS read-only

OBJECT  atmVplRowStatus
SYNTAX  INTEGER {active(1)}
        -- subset of RowStatus
MIN-ACCESS read-only

OBJECT  atmVpCrossConnectLowVpi
SYNTAX  INTEGER (1..255)

OBJECT  atmVpCrossConnectHighVpi
SYNTAX  INTEGER (1..255)
OBJECT  atmVpCrossConnectAdminStatus
MIN-ACCESS read-only
DESCRIPTION
 "Write access is not required."

OBJECT  atmVpCrossConnectRowStatus
SYNTAX  INTEGER {active(1)}
        -- subset of RowStatus
MIN-ACCESS read-only

GROUP   atmVccTerminationGroup
```

```
    GROUP   atmVcCrossConnectGroup

    OBJECT   atmVclVpi
    SYNTAX   INTEGER (0..255)

    OBJECT   atmVclAdminStatus
    MIN-ACCESS read-only

    OBJECT   atmVclReceiveTrafficDescrIndex
    MIN-ACCESS read-only

    OBJECT   atmVclTransmitTrafficDescrIndex
     MIN-ACCESS read-only

    OBJECT     atmVccAalType
    MIN-ACCESS read-only

    OBJECT   atmVclRowStatus
    SYNTAX   INTEGER {active(1)}
             -- subset of RowStatus
    MIN-ACCESS read-only

    OBJECT   atmVcCrossConnectLowVpi
    SYNTAX   INTEGER (0..255)

    OBJECT   atmVcCrossConnectHighVpi
    SYNTAX   INTEGER (0..255)

    OBJECT   atmVcCrossConnectAdminStatus
    MIN-ACCESS read-only

    OBJECT   atmVcCrossConnectRowStatus
    SYNTAX   INTEGER { active(1)}
             -- subset of RowStatus
    MIN-ACCESS read-only

    GROUP   aal5VccGroup

    OBJECT   aal5VccVpi
    SYNTAX   INTEGER (0..255)

    OBJECT     atmVccAal5CpcsTransmitSduSize
    MIN-ACCESS read-only

    OBJECT     atmVccAal5CpcsReceiveSduSize
    MIN-ACCESS read-only

    OBJECT     atmVccAal5EncapsType
    MIN-ACCESS read-only
    ::= { atmMIBCompliances 1 }

-- Units of Conformance

atmInterfaceConfGroup   OBJECT-GROUP
    OBJECTS {
        atmInterfaceMaxVpcs, atmInterfaceMaxVccs,
        atmInterfaceConfVpcs, atmInterfaceConfVccs,
        atmInterfaceMaxActiveVpiBits,
        atmInterfaceMaxActiveVciBits,
        atmInterfaceImiVpi,
        atmInterfaceImiVci,
        atmInterfaceAddressType,
        atmInterfaceAdminAddress,
        atmInterfaceMyNeighborIpAddress,
        atmInterfaceMyNeighborIfName}
```

```
    STATUS    current
      ::= { atmMIBGroups 1 }

atmTrafficDescrGroup   OBJECT-GROUP
    OBJECTS {
      atmTrafficDescrType, atmTrafficDescrParam1,
      atmTrafficDescrParam2, atmTrafficDescrParam3,
      atmTrafficDescrParam4, atmTrafficDescrParam5,
      atmTrafficQoSClass, atmTrafficDescrRowStatus}
    STATUS    current
 ::= { atmMIBGroups 2 }

atmInterfaceDs3PlcpGroup   OBJECT-GROUP
    OBJECTS {atmInterfaceDs3PlcpSEFSs,
      atmInterfaceDs3PlcpAlarmState,
      atmInterfaceDs3PlcpUASs}
    STATUS    current
    ::= { atmMIBGroups 3 }

atmInterfaceTCGroup   OBJECT-GROUP
    OBJECTS { atmInterfaceOCDEvents,
      atmInterfaceTCAlarmState }
    STATUS    current
     ::= { atmMIBGroups 4 }

atmVpcTerminationGroup   OBJECT-GROUP
    OBJECTS  {atmVplOperStatus, atmVplAdminStatus,
      atmVplLastChange,
      atmVplReceiveTrafficDescrIndex,
      atmVplTransmitTrafficDescrIndex,
      atmVplRowStatus }
    STATUS    current
     ::= { atmMIBGroups 5 }

atmVccTerminationGroup   OBJECT-GROUP

OBJECTS {atmVclOperStatus, atmVclAdminStatus, atmVclLastChange,
 atmVclReceiveTrafficDescrIndex,
atmVclTransmitTrafficDescrIndex,
atmVccAalType, atmVclRowStatus }
    STATUS    current
      ::= { atmMIBGroups 6 }

atmVpCrossConnectGroup   OBJECT-GROUP
    OBJECTS { atmVplReceiveTrafficDescrIndex,
      atmVplTransmitTrafficDescrIndex,
      atmVplOperStatus, atmVplRowStatus,
      atmVpCrossConnectAdminStatus,
      atmVpCrossConnectL2HOperStatus,
      atmVpCrossConnectH2LOperStatus,
      atmVpCrossConnectL2HLastChange,
      atmVpCrossConnectH2LLastChange,
      atmVpCrossConnectRowStatus,
      atmVplCrossConnectIdentifier,
      atmVpCrossConnectIndexNext }
    STATUS    current
      ::= { atmMIBGroups 7 }

atmVcCrossConnectGroup   OBJECT-GROUP
    OBJECTS { atmVclReceiveTrafficDescrIndex,
      atmVclTransmitTrafficDescrIndex,
      atmVclOperStatus, atmVclRowStatus,
      atmVcCrossConnectAdminStatus,
      atmVcCrossConnectL2HOperStatus,
      atmVcCrossConnectH2LOperStatus,
```

```
            atmVcCrossConnectL2HLastChange,
            atmVcCrossConnectH2LLastChange,
            atmVcCrossConnectRowStatus,
            atmVclCrossConnectIdentifier,
            atmVcCrossConnectIndexNext }
        STATUS    current
            ::= { atmMIBGroups 8 }

    aal5VccGroup   OBJECT-GROUP
        OBJECTS {atmVccAal5CpcsTransmitSduSize,
            atmVccAal5CpcsReceiveSduSize,
            atmVccAal5EncapsType,
            aal5VccCrcErrors, aal5VccSarTimeOuts,
            aal5VccOverSizedSDUs }
        STATUS    current
        DESCRIPTION
            "A collection of objects providing
            AAL5 configuration and performance statistics
            of a VCC."
        ::= { atmMIBGroups 9 }
END
```

Anexo – II

AToM MIB definida em IDL

```
#ifndef RFC1695_MIB_idl
#define RFC1695_MIB_idl
#include "ASN1Types.idl"
#include "ManagedObject.idl"
#include "SNMPMgmt.idl"
#include "SNMPv2-TC.idl"
#include "SNMPv2-CONF.idl"
#include "rfc1155.idl"

module rfc1695 {
typedef rfc1155::TimeTicksType TimeTicksType;
typedef rfc1155::GaugeType GaugeType;
typedef rfc1155::CounterType CounterType;

#define mgmt rfc1155::mgmt;
const ASN1_ObjectIdentifier mib_2 = "mgmt.1";
const ASN1_ObjectIdentifier system = "mib_2 .1";
const ASN1_ObjectIdentifier interfaces = "mib_2.2";
//_______ Type: DisplayString _____________________
typedef ASN1_OctetString DisplayStringType;
//___________________ Type: PhysAddress ____________________
typedef ASN1_OctetString PhysAddressType;
//================ interface ifEntry ======================
interface IfEntryObject:SNMPv2_SMI::SmiEntry{

//#ifIndex
/
//------------------------------
readonly attribute ASN1_Integer ifIndex;
//------------------------------
typedef sequence < DisplayStringType,255> IfDescrType;
readonly attribute IfDescrType ifDescr;
//------------------------------
typedef ASN1_Integer IfTypeType;
const IfTypeType other=1;
const IfTypeType regular1822=2;
const IfTypeType hdh1822=3;
const IfTypeType ddn_x25=4;
const IfTypeType rfc877_x25=5;
const IfTypeType ethernet_csmacd=6;
const IfTypeType iso88023_csmacd=7;
const IfTypeType iso88024_tokenBus=8;
const IfTypeType iso88025_tokenRing=9;
const IfTypeType iso88026_man=10;
const IfTypeType starLan=11;
const IfTypeType proteon_10Mbit=12;
const IfTypeType proteon_80Mbit=13;
const IfTypeType hyperchannel=14;
const IfTypeType fddi=15;
const IfTypeType lapb=16;
const IfTypeType sdlc=17;
const IfTypeType ds1=18;
```

```
const IfTypeType e1=19;
const IfTypeType basicISDN=20;
const IfTypeType primaryISDN=21;
const IfTypeType propPointToPointSerial=22;
const IfTypeType ppp=23;
const IfTypeType softwareLoopback=24;
const IfTypeType eon=25;
const IfTypeType ethernet_3Mbit=26;
const IfTypeType nsip=27;
const IfTypeType slip=28;
const IfTypeType ultra=29;
const IfTypeType ds3=30;
const IfTypeType sip=31;
const IfTypeType frame_relay=32;
readonly attribute IfTypeType ifType;
//---------------------------------
readonly attribute ASN1_Integer ifMtu;
//---------------------------------
readonly attribute GaugeType ifSpeed;
//---------------------------------
readonly attribute PhysAddressType ifPhysAddress;
//---------------------------------
typedef ASN1_Integer IfAdminStatusType;
const IfAdminStatusType up=1;
const IfAdminStatusType down=2;
const IfAdminStatusType testing=3;
attribute IfAdminStatusType ifAdminStatus;
//---------------------------------
typedef ASN1_Integer IfOperStatusType;
const IfOperStatusType up__1=1;
const IfOperStatusType down__1=2;
const IfOperStatusType testing__1=3;
readonly attribute IfOperStatusType ifOperStatus;
//---------------------------------
readonly attribute TimeTicksType ifLastChange;
//---------------------------------
readonly attribute CounterType ifInOctets;
//---------------------------------
readonly attribute CounterType ifInUcastPkts;
//---------------------------------
readonly attribute CounterType ifInNUcastPkts;
//---------------------------------
readonly attribute CounterType ifInDiscards;
//---------------------------------
readonly attribute CounterType ifInErrors;
//---------------------------------
readonly attribute CounterType ifInUnknownProtos;
//---------------------------------
readonly attribute CounterType ifOutOctets;
//---------------------------------
readonly attribute CounterType ifOutUcastPkts;
//---------------------------------
readonly attribute CounterType ifOutNUcastPkts;
//---------------------------------
readonly attribute CounterType ifOutDiscards;
//---------------------------------
readonly attribute CounterType ifOutErrors;
//---------------------------------
```

```
readonly attribute GaugeType ifOutQLen;
//-------------------------------
readonly attribute ASN1_ObjectIdentifier ifSpecific;
};


//================== interface system ======================
interface SystemGroup:SNMPv2_SMI::SmiEntry{
//-------------------------------
/* DESCRIPTION:
"A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters."*/
typedef sequence < DisplayStringType,255> SysDescrType;
readonly attribute SysDescrType sysDescr;
//-------------------------------
readonly attribute ASN1_ObjectIdentifier sysObjectID;
//-------------------------------
readonly attribute TimeTicksType sysUpTime;
//-------------------------------
typedef sequence < DisplayStringType,255> SysContactType;
attribute SysContactType sysContact;
//-------------------------------
typedef sequence < DisplayStringType,255> SysNameType;
attribute SysNameType sysName;
//-------------------------------
typedef sequence < DisplayStringType,255> SysLocationType;
attribute SysLocationType sysLocation;
//-------------------------------
typedef unsigned short SysServicesType;
readonly attribute SysServicesType sysServices;
};
//================ interface ATM ======================

interface ATM Group:SNMPv2_SMI::SmiEntry{
//-------------------------------
/*This group contains ATM specific configuration information associated with an ATM interface beyond those supported using the ifTable.*/
//-------------------------------

typedef ASN1_Integer atmInterfaceMaxVpcsType;
attribute atmInterfaceMaxVpcsType atmInterfaceMaxVpcs;
//-------------------------------
typedef ASN1_Integer atmInterfaceMaxVccsType;
atribute atmInterfaceMaxVccsType atmInterfaceMaxVccs;
//-------------------------------
readonly attribute ASN1_Integer atmInterfaceConfVpcs
//-------------------------------
readonly attribute ASN1_Integer atmInterfaceConfVccs
//-------------------------------
typedef ASN1_Integer atmInterfaceMaxActiveVpiBitsType;
attribute atmInterfaceMaxActiveVpiBitsType atmInterfaceMaxActiveVpiBits;
//-------------------------------
typedef ASN1_Integer atmInterfaceMaxActiveVciBitsType;
attibute atmInterfaceMaxActiveVciBitsType atmInterfaceMaxActiveVciBits;
//-------------------------------
typedef ASN1_Integer atmInterfaceImiVpiType;
attibute atmInterfaceImiVpiType atmInterfaceImiVpi;
```

```
//--------------------------------
typedef ASN1_Integer atmInterfaceIlmiVciType;
attribute atmInterfaceIlmiVciType atmInterfaceIlmiVci;
//--------------------------------
const atmInterfaceAddress private=1;
const atmInterfaceAddress  nsapE164=2;
const atmInterfaceAddress nativeE164=3;
const atmInterfaceAddress other=4;
readonly attribute atmInterfaceAddressType atmInterfaceAddress;
//--------------------------------
readonly attribute ASN1_OctetString atmInterfaceAdminAddress;
//--------------------------------
typedef sequence <DisplayStringType, 225> atmInterfaceMyNeighborIfNameType;
attribute atmInterfaceMyNeighborIfNameType atmInterfaceMyNeighborIfName;

}
}
```

Anexo – II

ASN1Types Definida em IDL

```
//
// ASN1Types.idl
//

#ifndef _ASN1TYPES_IDL_
#define _ASN1TYPES_IDL_

// ASN.1 base types

// Null type
typedef char        ASN1_Null;
const ASN1_Null     ASN1_NullValue = '\x00';

typedef boolean     ASN1_Boolean;

// unsigned integers
typedef unsigned short      ASN1_Unsigned16;
typedef unsigned long       ASN1_Unsigned;
typedef unsigned long       ASN1_Unsigned64[2];

// integers
typedef short       ASN1_Integer16;
typedef long        ASN1_Integer;
typedef long        ASN1_Integer64[2];

// others
typedef double      ASN1_Real;
typedef sequence<octet>     ASN1_BitString; // PIDL defined
typedef sequence<octet>     ASN1_OctetString;
typedef string      ASN1_ObjectIdentifier;
typedef any         ASN1_Any;
typedef any         ASN1_DefinedAny;

struct ASN1_External {
    ASN1_ObjectIdentifier  syntax;
    ASN1_DefinedAny        data_value;  // by syntax
};

// ASN.1 strings (which may not contain binary zeros)

typedef string      ASN1_IA5String;
typedef string      ASN1_NumericString;
typedef string      ASN1_PrintableString;
typedef string      ASN1_TeletexString;
typedef string      ASN1_T61String;
typedef string      ASN1_VideotexString;
typedef string      ASN1_VisibleString;
typedef string      ASN1_ISO646String;
```

```
// PIDL defined
typedef ASN1_VisibleString  ASN1_GeneralizedTime;
typedef ASN1_VisibleString  ASN1_UTCTime;

// ASN.1 strings of octets (which may contain binary zeros)

typedef sequence<octet>     ASN1_GeneralString;
typedef sequence<octet>     ASN1_GraphicString;

// ASN.1 strings of wide characters (which may contain binary zeros)

typedef sequence<unsigned short>     ASN1_BMPString;
typedef sequence<unsigned long>      ASN1_UniversaiString;

typedef ASN1_GraphicString ASN1_ObjectDescriptor;

// define constants for ASN.1 Real infinity

#include<ASN1Limits.idl>

#endif /* _ASN1TYPES_IDL_ */
```