UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFOMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE GERÊNCIA DE CONFIGURAÇÃO DE REDES LOCAIS

por

Marcelo Augusto Rauh Schmitt

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Professora Liane Margarida Rockenbach Tarouco
Orientadora

Porto Alegre, outubro de 1993.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Schmitt, Marcelo Augusto Rauh

Um modelo de gerência de configuração de redes locais/ Marcelo Augusto Rauh Schmitt. - Porto Alegre: CPGCC da UFRGS, 1993.

132 p.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1993. Orientadora: Tarouco, Liane M. R.

Dissertação: Gerência de configuração, Objetos gerenciados, Padrões de gerência

AGRADECIMENTOS

Agradeço a todas as pessoas que contribuíram para a realização deste trabalho. De forma especial, à professora Liane Margarida Rockenbach Tarouco, minha orientadora, que me despertou o interesse pela área de comunicação de dados desde os tempos de auxiliar de pesquisa e, sem a qual, este trabalho não poderia ser feito.

Também agradeço ao professor Carlos Westphall pelas contribuições e críticas durante a confecção do trabalho e ao Cléber, Ewerton (Mumu), Renata, Spohn, Reges e Dotti pelo apoio e pela companhia.

Por último, agradeço aos meus pais, que me deram a vida, as oportunidades e o estímulo; aos meus irmãos que me incentivaram; à Juju que me ajudou na digitação; e a Deus, que me deu tudo.

SUMÁRIO

LISTA DE ABREVIATURAS

ACSE	Association Control Service Element
API	Application Program Interface
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
CMU	Carnegie Mellon University
EGP	Exterior Gateway Protocol
EQUEL	Embbeded Query Language
IAB	Internet Activities Board
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MIB	Management Information Base
OSI	Open Systems Interconnection
ROSE	Remote Operation Service Element
SDL	Specification and Description Language
SMP	Simple Management Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UFRGS	Universidade Federal do Rio Grande do Sul
X-View	X Window-system-based Visual/Integrated Environment for Workstations

LISTA DE FIGURAS

RESUMO

Uma rede local necessita ter a sua configuração gerenciada para manter a sua utilidade dentro da organização à qual pertence. Esta gerência é complicada pela heterogeneidade da rede e pela sua importância dentro da organização. A tentativa de solucionar este problema passa pela utilização de um padrão de gerência de rede e pelo desenvolvimento de ferramentas que permitam a gerência eficiente da rede pelo administrador. Este trabalho apresenta uma proposta de gerência de configuração de redes locais.

São apresentados os objetos gerenciados necessários à gerência de configuração. Parte destes objetos são previamente definidos pela IAB (Internet Activities Board) em normas; e parte é definida neste trabalho. Os objetos são definidos de acordo com o formato proposto pela IAB. Entretanto, para que os objetos sejam definidos, é necessário, antes, que as funções de gerência de configuração sejam delineadas. Os objetos são definidos a partir das funções que se deseja executar. Portanto, antes da apresentação dos objetos, são definidas funções de gerência de configuração.

Também é apresentada a implementação de um protótipo de gerente que realiza várias das funções definidas no trabalho, utilizando o padrão SNMP (Simple Network Management Protocol). Este protótipo utiliza o pacote de programas CMU (Carnegie Mellon University) SNMP para a implementação das funções. A partir do protótipo pode-se perceber como se realizam as funções de gerência e como deve ser a interface de um gerente de configuração, a fim de que este seja realmente útil nas tarefas do administrador da rede.

Por último, é feita uma modelagem dos objetos gerenciados de acordo com a norma OSI (Open Systems Interconnection). Várias são as modificações decorrentes desta modelagem diferenciada. São apresentadas as diferenças que ocorrem na gerência de configuração quando se utiliza o modelo de dados OSI, em vez do modelo de dados Internet.

PALAVRAS-CHAVE: Gerência de configuração; Objetos gerenciados; Padrões de gerência,

TITLE: "A MODEL OF LOCAL AREA NETWORK CONFIGURATION MANAGEMENT"

ABSTRACT

In order to keep a LAN useful to its organization, its configuration has to be managed. Heterogenity and importance of networks makes management more difficult. An attempt to solve this problem has to consider the management standards and the development of tools which allow an efficient network management. This work presents a proposal of local area network configuration management.

Managed objects necessary to manage configuration are presented. Part of these objects are, previously, define in IAB (Internet Activities Board) standards, and part of them are defined here. Objects are defined according to the format proposed by IAB. However, before the definition of objects, it is necessary to define configuration management functions. Objects are defined according to the functions one wants to implement. Thus, before the presentation of objects, configuration management functions are defined.

It is also presented a manager prototype which executes many of the functions defined in this work, using SNMP (Simple Network Management Protocol). The prototype uses CMU (Carnegie Mellon University) SNMP software package to implement functions. It is possible to notice, from the prototype, how functions are realized and which characteristics the interface of a configuration manager must have to be useful in the tasks performed by human manager.

Finally, modelling of managed objects is done according to OSI (Open Systems Interconnection) standards. Many changes occur because of this diferent modelling. The changes in configuration management which occurs when one uses OSI data model, instead of Internet data model, are presented.

KEYWORDS: Configuration management; Managed objects; Management standards.

1 INTRODUÇÃO

1.1 A necessidade de ferramentas de gerência

A construção de ferramentas para a gerência de redes é uma necessidade para que estas conservem-se úteis aos seus usuários, e possam ser mantidas pelas organizações responsáveis. De acordo com [SLU 89], uma ferramenta integrada de gerência pode permitir que o administrador humano desempenhe o seu papel de forma mais eficiente, uma vez que um melhor acesso às informações deve garantir uma melhor tomada de decisões e, conseqüentemente, um melhor serviço para o usuário. Estas ferramentas têm como objetivos diminuir o tempo de inoperância da rede, e o próprio custo de gerência. Diminuem o tempo de inoperância porque permitem um diagnóstico mais rápido do problema e, mesmo, a previsão deste. Diminuem os gastos com gerência porque, dentre os fatores que compõem estes gastos ([FEH 89]), dois podem se reduzidos com a utilização de ferramentas adequadas: a especialização do profissional e a quantidade de profissionais utilizados na tarefa.

A princípio, prover estes meios parece ser uma simples tarefa de construir programas que implementem os diversos serviços de gerência. Entretanto, existem dois fatores que complicam a solução do problema: a heterogeneidade e a importância da rede.

De acordo com [FEL 89], a evolução das redes começa com um pequeno conjunto de produtos homogêneos e cresce, para incorporar uma grande variedade de tecnologias complexas interconectadas. Ainda de acordo com [FEL 89], essa evolução está, 'geralmente', fora do controle do planejador da rede; afinal, ela é dirigida pelas necessidades dos usuários da corporação em questão, que são variadas. Assim, à medida em que a rede de uma corporação cresce, assume, cada vez mais, um caráter heterogêneo. Heterogêneo porque apresenta componentes desenvolvidos por fabricantes diferentes, com diferentes funções, diferentes tecnologias e diferentes protocolos de comunicação.

O crescimento de uma rede não aumenta a dificuldade de gerenciá-la somente por torná-la heterogênea. Também, quanto maior e mais complexa a rede, maior a sua importância para a corporação à qual pertence; e, portanto, mais crítica

torna-se a sua gerência. De acordo com [WIL 90], num período de cinco anos, os custos operacionais de uma rede podem corresponder a mais do que o dobro do custo inicial de instalação. Se os diversos aspectos de uma rede complexa não puderem ser gerenciados eficientemente, a sua operacionabilidade ficará comprometida, juntamente com os objetivos da corporação, que lhe são dependentes. De acordo com [POT 91], uma rede é valiosa para a organização apenas enquanto puder ser gerenciada para manter a confiabilidade, o baixo custo e aumentar a produtividade

Estas características estão presentes, também, em redes universitárias. A evolução para a heterogeneidade da rede é obvia, se considerarmos as diversas partes de uma universidade. As redes são utilizadas em atividades administrativas e acadêmicas. E, ainda, várias são as atividades acadêmicas, cada uma com suas peculiaridades e necessidades próprias. Assim, por exemplo, os equipamentos, as informações e a tecnologia de interesse do curso de ciência da computação não são os mesmos daqueles do curso de odontologia.

O tráfego é um aspecto da rede muito importante. Em uma rede universitária teremos todo tipo de tráfego, nos diferentes segmentos da rede. Alguns departamentos transmitirão imagens, sobrecarregando a rede, outros não ultrapassarão o tráfego de um correio eletrônico. Há, ainda, uma variedade de categorias de usuários. Se o aluno de computação tende a dominar muito bem a tecnologia, um professor de direito pode ter dificuldades. É importante notar que os dois tipos de usuários trazem dificuldades para a gerência de rede. O último, porque necessita de uma ferramenta fácil de ser utilizada e de aconselhamento; e o primeiro, porque obriga que sejam criados mecanismos de segurança mais sofisticados.

Outra característica importante de uma rede universitária é a estrutura organizacional. As diversas partes que formam a universidade tem uma certa autonomia. Isto faz com que a rede tenha que ser organizada de forma hierárquica e que a gerência também seja feita de forma hierárquica.

1.2 A necessidade de um padrão de gerência

A heterogeneidade da rede é um fator de dificuldade para a gerência de rede porque não basta ter-se ferramentas que gerenciem uma determinada linha de

produtos. São necessárias ferramentas que gerenciem diversos produtos, de diversos fabricantes, e que possam gerenciar quaisquer novos recursos que sejam adicionados à rede. Quanto à importância fundamental da rede dentro da corporação, isto obriga que as ferramentas tenham que gerenciar aspectos realmente importantes para manutenção da operacionabilidade da rede, e não fiquem apenas em funções superficiais. A solução para estes problemas está em eliminar-se as implementações particulares, em adotar-se padrões que permitam a gerência de qualquer recurso e modelos que permitam a definição de funções de gerência complexas o suficiente para serem úteis.

Duas famílias de padrões são utilizadas com este objetivo: Internet e OSI (Open Systems Interconnection). Apesar de apresentarem diferenças, que são melhor detalhadas no capítulo 2 deste trabalho, ambas baseiam-se em uma estrutura semelhante de gerência. Utilizam um esquema cliente-servidor, onde o cliente gerencia a rede consultando e modificando informações mantidas nos servidores. Na nomenclatura usual desta área, o cliente é chamado de gerente e o servidor de nodo gerenciado.

O gerente é responsável pelas atividades de gerência que devem ser realizadas. Apresenta uma interface para o administrador humano, e deve facilitar o seu trabalho, permitindo que a rede cumpra com suas funções na organização em que está situada. O nodo gerenciado é qualquer dispositivo que esteja conectado à rede e que, portanto, necessite ser gerenciado em sua relação com a mesma. Assim , uma ponte, um hub, um multiplexador, uma impressora, um servidor de terminal, um gateway, uma estação de trabalho são nodos gerenciados. A entidade, dentro do nodo gerenciado, que se comunica com o gerente é chamada de agente.

As duas famílias de padrões utilizam o conceito de objetos gerenciados. Todos os recursos da rede que devem ser gerenciados são modelados, e as estruturas de dados resultantes desta modelagem são chamadas de objetos gerenciados. Os gerentes lêem ou modificam instâncias remotas de objetos gerenciados, comunicando-se com os agentes. Cada valor modificado reflete-se no próprio recurso; e cada valor lido representa o estado real do mesmo. Assim, através dos objetos gerenciados, os recursos são controlados. Por exemplo, para gerenciar a tabela de endereços de uma ponte, define-se uma estrutura que a represente adequadamente. Quando o gerente quiser modificar esta tabela, realizará operações sobre o objeto que foi definido.

O conjunto dos objetos gerenciados, o repositório conceitual de todas as informações necessárias para a gerência da rede é denominado MIB (Management Information Base) . É necessário que os agentes apresentem MIBs que representem os recursos a eles associados e que o gerente conheça a semântica e a sintaxe dos objetos gerenciados. Se o gerente não conhecer o esquema do agente, o gerenciamento é impossível. O conhecimento da sintaxe dos objetos não é nenhum problema, uma vez que se utilize um dos padrões. Entretanto não basta que o gerente consiga ter acesso ao objeto gerenciado. É necessário que o gerente conheça o significado do objeto para poder realizar alguma função de gerência, senão, o administrador humano terá que realizar praticamente todas as tarefas. O objetivo do gerente é automatizar ao máximo a gerência da rede, e, para fazer isso, precisa conhecer o significado de cada objeto. Por isso, ambas as famílias de padrões definiram MIBs que apresentam os objetos básicos para a gerência dos seus recursos. Entretanto, isto não elimina as extensões dos fabricantes e, portanto, não resolve completamente o problema.

Para que o gerente possa comunicar-se com o agente, é utilizado um protocolo. Na gerência OSI, utiliza-se o protocolo CMIP (Common Management Information Protocol); na Internet, SNMP (Simple Network Management Protocol). A partir das operações existentes nos protocolos, pode-se alterar e consultar o estado dos diversos recursos da rede. Por exemplo, alterar tabelas de roteamento, analisar a configuração dos equipamentos, detectar falhas, setar parâmetros; enfim controlar os diversos dispositivos da rede. Toda a gerência é baseada na troca de mensagens entre gerentes e agentes.

1.3 Gerência de configuração

Vários aspectos da rede são gerenciados com a troca de informações entre gerentes e agentes. Para atacar o problema de forma organizada, o modelo OSI divide a gerência de redes em cinco categorias básicas: gerência de falhas, gerência de segurança, gerência de contabilidade, gerência de desempenho e gerência de configuração. A gerência de falhas inclui a descoberta e a eliminação de falhas, de componentes avariados ou mal configurados e a monitoração de equipamentos para o descobrimento de indicadores de problemas que possam ser antecipados. A gerência de segurança controla o acesso aos diversos recursos da rede. Aí estão incluídos procedimentos como autorizações, criptografia e gerência de chaves criptográficas. A

gerência de contabilidade é responsável pela organização das informações referentes ao custo de utilização dos diversos recursos da rede. A gerência de desempenho é responsável pela avaliação da eficiência da rede e previsão de futuros problemas relacionados com o desempenho. Medidas como "throughput", número de retentativas, carga média são importantes para uma perfeita compreensão do funcionamento da rede e de seu comportamento futuro.

Neste contexto, está localizada a gerência de configuração. De acordo com [PAU 90], a gerência de configuração inclui tanto a gerência da configuração lógica, quanto da configuração física. A lógica inclui aspectos como a nomeação, os parâmetros das portas e os parâmetros dos protocolos. Trata do controle da localização dos recursos lógicos da rede (programas, dados, servidores) e do controle de parâmetros associados a estes recursos (portas, protocolos). A física trata da localização física dos nodos e da configuração de hardware da rede. De acordo com [POT 91], ter o "hardware" e o "software" apropriados no lugar certo, para o trabalho certo é importantíssimo para a produtividade do ambiente. Ainda para [POT 91], a gerência de configuração trata de tarefas de administração dos sistemas, tais como a monitoração e manutenção diárias dos estados físico e lógico correntes da rede, assim como do reconhecimento e registro de aplicações e serviços da mesma. Para [MAR 90], a configuração da rede e o estado de seus componentes em qualquer momento devem estar prontamente disponíveis. De acordo com [CON 87], a gerência de configuração inclui a criação de novas instâncias de um tipo específico de objeto; a remoção de instâncias de um tipo específico de objeto; a renomeação de objetos; o relato da criação ou remoção de um objeto; a apresentação de uma lista de instâncias de objetos de um tipo específico; a mudança de atributos, estados, e relações de um objeto; o relato sobre as mudanças dos atributos, dos estados e das relações de um objeto e a distribuição de software.

1.4 Conclusão

Qualquer definição de um sistema de gerência de configuração de rede heterogênea deve, em primeiro lugar, basear-se no modelo cliente servidor e escolher, dentre os padrões existentes, aquele que se adequa a suas características. Entretanto, a definição não termina aí. É necessário definir-se outros dois aspectos, sem os quais a gerência não é possível: que informações os nodos gerenciados devem passar ao gerente

e vice-versa; e como o gerente deve tratar estas informações para que estas tenham um valor real para o usuário. Este trabalho tenta realizar exatamente isto.

O capítulo 2 apresenta os padrões de gerência OSI e Internet, suas características e suas diferenças. Desenvolve um pouco o conceito de objetos gerenciados e MIB.

A partir do capítulo 3 é apresentada a solução para a gerência da rede da Universidade Federal do Rio Grande do Sul, que utiliza o padrão TCP/IP (Transmission Control Protocol/Internet Protocol), e que é composta por várias redes locais. São definidas as funções necessárias para a gerência de configuração, bem como os objetos necessários para que estas funções sejam executadas. O modelo de gerência adotado foi o modelo Internet. Alguns objetos apresentados são objetos já definidos na MIB-II da IAB (Internet Activities Board). Outros, são objetos criados para possibilitar a realização das funções definidas.

O capítulo 4 descreve a implementação das funções definidas no capítulo anterior. Várias funções foram implementadas em um protótipo de um gerente de configuração. Juntamente com a implementação das funções, é apresentada a interface de programa com o administrador humano. Esta interface é essencial para que o usuário faça real uso da ferramenta.

Por último, o capítulo 5 apresenta uma modelagem, do mesmo problema, sob o ponto de vista OSI. São apresentadas as árvores de conteúdo e registro dos objetos gerenciados e as implicações que este modelo causa na implementação das funções do gerente. Isto permite uma comparação, não mais geral, entre OSI e Internet, mas uma comparação da utilização dos dois modelos na gerência de configuração.

2 PADRÕES DE GERÊNCIA

2.1 O padrão de gerência Internet

O modelo de gerência definido pela IAB (Internet Activities Board) tem como objetivo gerenciar redes que utilizam os protocolos da família TCP/IP, e segue o esquema básico, apresentado na introdução, *agente-gerente*.

2.1.1 "Management Information Base"

A IAB definiu objetos que devem ser implementados pelos nodos gerenciados em uma rede TCP/IP. Primeiramente foi definida uma MIB com um conjunto restrito de objetos, com a intenção de agilizar as implementações de agentes. Esta MIB recebeu a designação de MIB-I ([McC 90]). Em 1989 foi proposta uma nova MIB (II) ([McC91]) que apresentava os objetos da MIB-I, e acrescentava novos objetos, que permitem uma gerência mais completa da rede. Atualmente a MIB-II e´um padrão da IAB. Diversas empresas estão desenvolvendo seus produtos de acordo com a MIB Internet. Sempre que um produto diz-se compatível com SNMP, ele apresenta uma implementação da MIB definida pela IAB. A MIB-II apresenta objetos organizados em nove grupos funcionais. O primeiro grupo, chamado *system*, apresenta informações genéricas de configuração como: descrição do dispositivo, nome do dispositivo, localização física do dispositivo e serviços oferecidos pelo dispositivo. O segundo grupo, chamado *interfaces*, apresenta informações sobre as diversas interfaces com o meio físico, de cada dispositivo da rede, tais como: descrição da interface, estado da interface, e número de pacotes enviados ou recebidos. O terceiro grupo, chamado *address translation*, contém uma tabela para mapear endereços IP em endereços físicos. Os outros seis grupos apresentam objetos que permitem o controle de cada um dos seguintes protocolos: IP, ICMP (Internet Control Message Protocol), TCP, UDP (User Datagram Protocol), EGP (Exterior Gateway Protocol) e SNMP (Simple Network Management Protocol). Há ainda um último grupo, chamado *transmission*, onde estão sendo definidos objetos para a gerência de diferentes meios de transmissão (IEEE 802.2, IEEE 802.5, FDDI, T1).

Apesar de haver uma MIB padrão, podem-se definir extensões. Assim, cada fabricante, ou grupo de pesquisa, define novos objetos gerenciados, de acordo com as suas necessidades. É logico que os problemas colocados na seção 1.2, referentes ao conhecimento da semântica dos objetos, pelo gerente, não devem ser esquecidos. Neste trabalho, a definição de um objeto gerenciado é feita de acordo com o que é definido por [McC 90]. Esta definição apresenta cinco partes: um nome, acompanhado por um identificador de objeto; a sintaxe abstrata do objeto; uma descrição textual do significado do objeto; o tipo de acesso que pode ser realizado sobre o objeto; e o estado do objeto. A pseudo-descrição a seguir tenta explicar melhor estas cinco partes.

OBJECT

sysDescr {system 1}

Syntax:

DisplayString (SIZE(..255)

Definition: A textual description of the entity. Full name, version number...

Access: read-only.

Status: mandatory.

A primeira parte apresenta o nome do objeto e um identificador de objeto. Identificador de objeto é um conceito para nomenclatura hierárquica de objetos. Nomes são organizados em uma árvore. Um nodo, que representa um objeto, pode ser identificado, pela concatenação de valores associados aos nodos que vão da raiz ao nodo desejado. Assim, um identificador de objeto é uma seqüência de inteiros positivos que percorrem uma árvore, para identificar um objeto qualquer (documento, organização, objeto gerenciado). A figura 1 apresenta esta árvore.

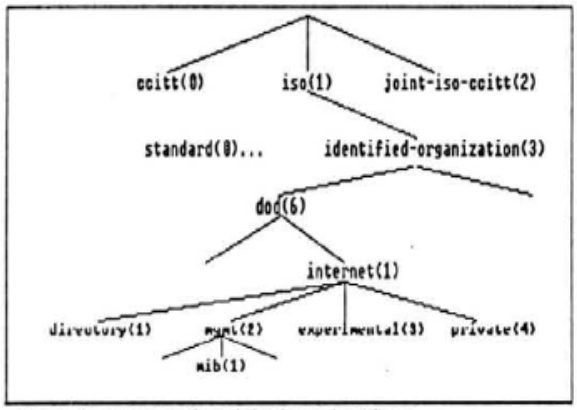

Figura 1: árvore de identificadores de objetos

A definição do identificador de objeto para Internet é feita da seguinte maneira:

internet OBJECT IDENTIFIER ::= {iso(1) org(3) dod(6) 1}

Desta forma, a seqüência de números que identifica o objeto "internet" é 1.3.6.1. O mesmo é feito até se chegar ao objeto "mib", que contém os diversos objetos gerenciados.

mgmt OBJECT IDENTIFIER ::= {internet 2}
mib OBJECT IDENTIFIER ::= {mgmt 1}
system OBJECT IDENTIFIER ::= {mib 1}
experimental OBJECT IDENTIFIER ::= {internet 3}
private OBJECT IDENTIFIER ::= {internet 4}

Todos os objetos gerenciado pertencentes à MIB Internet são definidos dentro da subárvore 1.3.6.1.2.1. Os objetos que estão sendo pesquisados no âmbito da IAB são definidos dentro da subárvore 1.3.6.1.3. Por último, os objetos definidos por outras organizações estão dentro da subárvore 1.3.6.1.4.

A segunda parte da definição de um objeto gerenciado é a sua estrutura. Esta estrutura é descrita com um subconjunto de ASN.1 (Abstract Syntax Notation One - [ISO 85]). Pode-se utilizar os tipos INTEGER, OCTET STRING, OBJECT

IDENTIFIER, NULL, SEQUENCE e SEQUENCE OF. No exemplo, *sysDescr* é definido como sendo do tipo *DisplayString*. Este é um tipo pré-definido, assim como: *NetworkAddress, IpAddress, Counter, Gauge, TimeTicks* e *Opaque.*

A terceira parte da definição de um objeto é a descrição do que, exatamente, é aquele objeto. É uma definição textual do próprio objeto gerenciado.

A quarta parte da definição de um objeto gerenciado é o tipo de acesso permitido para ele, isto é, que operações um gerente pode fazer sobre uma instância daquele objeto. Ele pode ser de um entre quatro tipos: leitura, escrita, leitura-escrita e sem-acesso.

Por fim, a quinta parte da definição explicita qual a necessidade de implementação daquele objeto para que um produto possa ser considerado dentro do padrão. Existem três estados: obrigatório (*mandatory*), opcional (*optional*) e obsoleto (*obsolet*).

2.1.2 Simple Network Management Protocol

O protocolo de gerência de rede utilizado pela família Internet chama-se SNMP. Este é um protocolo bastante simples, com apenas quatro operações: *get, get-next, set* e *trap*. Preferencialmente, SNMP utiliza um serviço de transporte não orientado a conexão (UDP).

A operação *get* é usada para recuperar uma determinada informação . Por exemplo: *get(sysDescr.0)*. É possível recuperar mais de um objeto em uma operação : *get (sysDescr.0,sysName.0)*. Neste último caso, se uma das instâncias não existe, nenhuma é recuperada, e a resposta indica erro.

A operação *get-next* é usada para recuperar informações na ordem de definição da MIB, isto é, de acordo com a árvore de identificadores de objetos. Desta maneira, é possível percorrer a árvore e descobrir que objetos estão presentes na MIB de um determinado nodo gerenciado.

A operação *set* é usada para modificar instâncias de objetos. E a operação *trap*, para que o agente relate, ao gerente, eventos extraordinários. Existem eventos já definidos e novos eventos podem ser definidos pelas organizações. Entretanto SNMP não foi desenvolvido com o intuito de fazer uma utilização mais efetiva de traps. O esquema preferencial é a consulta iniciada pelo gerente.

As operações somente podem ser executadas sobre objetos elementares, nunca sobre objetos compostos. Não é possível recuperar uma tabela inteira com uma operação, tem-se que recuperar um elemento da tabela de cada vez. Para identificar o objeto gerenciado sobre o qual a operação deve ser executada, usa-se o identificador de objeto acompanhado de um sufixo. Este sufixo varia se o objeto é ou não uma coluna de uma tabela. Se o objeto não for, o sufixo utilizado é 0 (zero). Por exemplo, para identificar o objeto *sysDescr*, basta utilizar o identificador de objeto de *sysDescr* e acrescentar o sufixo 0 (1.3.6.1.2.1.1.1.0).

Se o objeto é uma coluna de uma tabela, o identificador de objeto determina a coluna sobre a qual deseja-se realizar a operação, e o sufixo determina a linha. Este sufixo está relacionado com outras colunas que formam a tabela. O sufixo seleciona uma linha pelo valor de outras colunas. Por exemplo, a tabela a seguir apresenta duas colunas. Uma corresponde a uma descrição e outra a um identificador.

OBJECT

ifTable {interfaces 2}

Syntax:

SEQUENCE OF IfEntry

Definition: Lista de interfaces.

Access: read-only.

Status: mandatoy.

OBJECT

ifEntry {ifTable 1}

Syntax:

IfEntry ::= SEQUENCE {ifIndex INTEGER,
ifDescr DisplayString}

Definition: Número e descrição da interface.

Access: read-only.

Status: mandatory.

OBJECT

ifIndex {ifEntry 1}

Syntax:

INTEGER

Definition: Identificador da interface.

Access: read-only.

Status: mandatory.

OBJECT

ifDescr {ifEntry 2}

Syntax:

DisplayString

Definition: Descrição da interface.

Access: read-only.

Status: mandatory.

Para identificar uma instância de *ifDescr*, basta saber de qual interface é a descrição (*ifIndex*). Utiliza-se o identificador de objeto de *ifDescr* e, como sufixo, o valor de *ifIndex*. Assim, *ifDescr.2* (1.3.6.1.2.1.2.2.1.2.2) identifica a descrição da interface de número 2.

É importante notar que o padrão não apresenta nenhum mecanismo de segurança. Na verdade, existe o conceito de comunidade que define o relacionamento entre um gerente e um agente. De acordo com a comunidade a que um gerente pertence, poderá executar ou não operações sobre objetos. Entretanto, apenas mecanismos triviais de autenticação são utilizados. Se o nome de comunidade enviado é um nome conhecido pela entidade que recebe a mensagem, e está associado ao endereço de transporte que enviou a mensagem, então a entidade que enviou é considerada como pertencente a tal comunidade. Para cada comunidade é definido um nível de acesso à MIB. Isto faz com que SNMP não possa ser utilizado em uma gerência que exige modificações de objetos com grande impacto na rede, já que um gerente sem autorização poderia provocar grandes transtornos. Estão sendo desenvolvidas extensões ao padrão, como SMP (Simple Management Protocol), para solucionar este problema.

objetos. As estruturas resultantes da modelagem dos recursos não são estruturas com as características da orientação a objetos. Na realidade, as estruturas ou são atributos simples, ou são tabelas ("arrays").

2.2 Gerência no contexto OSI

No modelo OSI, a comunicação entre o gerente e o agente é realizada através da entidade de aplicação de gerência de sistemas (SMAE - System Management Application Entity). Esta entidade apresenta diversos elementos de serviço de aplicação OSI (ASEs - Application Service Elements), que trabalham em conjunto: CMISE (Common Management Information Service Element, ROSE (Remote Operation Service Element) e ACSE (Association Control Service Element). A figura 2 mostra o relacionamento destes elementos. O ACSE é utilizado para estabelecer uma associação entre o gerente e o agente, e o ROSE, para executar as operações do protocolo de gerência.

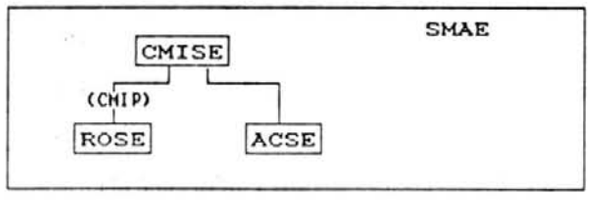

Figura 2: Entidade de aplicação de gerência do sistema

O padrão OSI define três modelos para gerência de redes: o modelo organizacional, o modelo informacional e o modelo funcional. O modelo organizacional descreve a forma pela qual a gerência pode ser distribuída entre domínios e sistemas dentro de um domínio. O modelo funcional descreve as áreas funcionais e seus relacionamentos (configuração, desempenho, falhas, contabilidade e segurança).

O modelo informacional provê a base para a definição de objetos gerenciados e suas relações, classes, atributos, ações e nomes. Aqui é utilizada a

orientação a objetos. Objetos com características semelhantes são agrupados em classes de objetos. Uma classe pode ser uma subclasse de outra, e a primeira herda todas as propriedades da segunda. Uma classe é definida pelos atributos da classe, pelas ações que podem ser invocadas, pelos eventos que podem ser relatados, pela subclasse da qual ela deriva e pela superclasse na qual ela está contida .

Para definição dos objetos gerenciados deve-se considerar três árvores de relacionamento entre os objetos: a árvore de registro, a árvore de herança e a árvore de contensão. A árvore de registro é o mesmo conceito de identificador de objeto utilizado pelo padrão Internet. Está árvore permite a identificação das diversas classes de objetos definidas, através do seqüência de nodos percorridos desde a raiz, até o objeto de interesse.

A árvore de conteúdo define se um objeto está contido em outro. Por exemplo: o objeto fila de impressão está contido dentro de um objeto impressora; uma conexão está contida dentro de entidade de transporte, e assim por diante. Esta árvore serve para identificar as diversas instâncias de objetos. Além disso, pode definir uma dependência de existência dos objetos. Um objeto pode existir, talvez, somente se aquele que o contém também existir; e, dependendo da definição, um objeto só pode ser removido se aqueles que lhe pertencem forem removidos primeiro.

A terceira árvore é a árvore de herança, que tem como objetivo facilitar a modelagem dos objetos, através da utilização do paradigma da orientação a objetos. Desta forma, podem ser definidas classes, superclasses, subclasses. É apenas uma ferramenta para uma definição melhor de classes.

O protocolo utilizado no modelo OSI é o CMIP (Common Management Information Protocol), e os serviços oferecidos pelo CMISE são os seguintes:

M-INITIALIZE - serve para estabelecer uma associação entre dois usuários do serviço;

M-TERMINATE - é usado para terminar a associação;

M-ABORT - é utilizado para terminar abruptamente uma associação;

M-EVENT-REPORT - é utilizado para relatar eventos;

M-GET - permite a recuperação de dados da MIB;

M-SET - permite a modificação de dados da MIB;

M-CREATE - permite a criação de uma representação de uma instância de um objeto gerenciado, com sua identificação e valores;

M-DELETE - permite a remoção de uma representação de uma instância de um objeto;

M-ACTION - permite a invocação de uma ação que não é diretamente relacionada com a manipulação da base de dados.

2.3 OSI x Internet

Existem semelhanças, mas também, várias diferenças entre a gerência OSI e a Internet. A primeira diferença entre eles está na filosofia de acesso aos dados. Enquanto SNMP recupera itens individuais, CMIP é direcionado para a recuperação de dados agregados. Outra diferença é que SNMP trabalha preferencialmente por "polling", ou seja, o gerente fica consultando seguidamente cada agente para monitorá-lo; enquanto no modelo OSI é mais comum que o agente informe ao gerente modificações no recurso enviando notificações sobre eventos.

SNMP é baseado em um serviço de datagramas não confiáveis e pode ser utilizado com Ethernet, IPX da Novell, UDP e outros protocolos de comunicação. CMIP, em contraposição, requer um serviço de transporte orientado a conexão. É mais fácil a implementação de SNMP e o CMIP requer uma capacidade maior de processador e memória. De acordo com [BEN 90], CMIP necessita de mais de 1 Mb de RAM, enquanto SNMP necessita apenas 40 a 50 Kb. Este problema também é apontado por [FIS 91]. SNMP apresenta, ainda, a vantagem de possuir diversas implementações em produtos comerciais (Advanced Computer Communications, Cisco System Inc., Hughes LAN Systems Inc., Proteon Inc., Wellfleet, Novell, 3Com).

Como é de se esperar, todas estas vantagens, do SNMP, em termos de simplicidade, trazem consigo uma série de desvantagens. Em primeiro lugar, o modelo de dados OSI permite a definição de estruturas bem mais complexas, que podem ser fundamentais para a resolução de problemas mais complicados. Objetos que são transientes, isto é, são criados e removidos freqüentemente, podem ser modelados e controlados de uma forma muito mais elegante.

Por último na proposta OSI, exite um modelo organizacional. Este modelo permite que haja uma hierarquia de gerenciamento. Assim, os gerentes podem ser distribuídos de acordo com o nível das tarefas que devem realizar. Esta parece ser uma forma natural de gerenciar redes, sem passar por cima da hierarquia de uma organização. Portanto, apesar de a IAB apresentar um modelo que permite a gerência de dispositivos atualmente; o modelo OSI deve, aos poucos, tomar um lugar muito importante na gerência de redes.

3 OBJETOS PARA GERÊNCIA DE CONFIGURAÇÃO

Como foi assinalado na seção 1.4, é preciso optar por um dos padrões de gerência. Optou-se, neste trabalho, pelo padrão SNMP. As razões para isso são as seguintes: a rede da UFRGS (Universidade Federal do Rio Grande do Sul) utiliza TCP/IP; existem vários agentes SNMP de domínio público disponíveis; existem APIs (Application Program Interfaces) para o desenvolvimento de gerentes e, por ser um protocolo menos complexo, permite a implementação mais rápida de um protótipo.

Também de acordo com a seção 1.4 deste trabalho, para se realizar a gerência de configuração é necessário definir as informações que os nodos gerenciados devem passar ao gerente e vice-versa. É a partir destas informações que poderão ser desenvolvidas ferramentas úteis para o usuário. Primeiramente, precisa-se saber quais as funções que um gerente de configuração deve desempenhar, para então, serem definidos os objetos que permitam a execução destas funções. Não é possível definirem-se objetos gerenciados sem ter em mente um objetivo, porque, mais tarde, estes objetos poderão não ter nenhuma serventia.

Este capítulo apresenta as diversas funções consideradas necessárias para a gerência de configuração e, para cada função, os objetos gerenciados necessários à sua execução. Os objetos são definidos da maneira referida na seção 2.1.1. Várias funções necessitam objetos gerenciados que já foram definidos pela IAB , na MIB-I ou MIB-II. Estes objetos são aproveitados, e suas definições encontram-se no anexo A.

3.1 Controle dos dispositivos existentes na rede

O administrador humano deve saber que equipamentos fazem parte da rede (estações de trabalho, pontes, computadores pessoais, equipamentos de grande porte, etc). O gerente de rede humano deve ter uma visão dos equipamentos existentes na rede, seus endereços e o tipo de cada um, pois podem ocorrer freqüentes mudanças na rede; e o conhecimento das características correntes dos equipamentos é fundamental para a solução de eventuais problemas.

Portanto, duas subfunções são importantes: a descoberta dos dispositivos e a descoberta de suas características. Em relação à anotação de suas características, a IAB define objetos gerenciados suficientes. Estes objetos são os seguintes:

sysDescr

Descrição textual do dispositivo. Contém o nome completo, identificador da versão de "hardware", sistema operacional e "software" de rede.

sysObjectID

Identificação de autorização do fornecedor do subsistema de gerência da entidade.

sysUpTime

Tempo desde que o subsistema de gerência de rede da entidade foi reinicializado.

sysContact

Pessoa responsável por este nodo gerenciado, juntamente com a informação de como contatá-la.

sysName

Nome do nodo. Este nome é dado administrativamente.

sysLocation

Localização física do dispositivo.

sysServices

Conjunto de serviços suportados pelo dispositivo

ipAddrTable

Tabela contendo os endereços IP do dispositivo.

A descoberta dos dispositivos que fazem parte da rede é feita da seguinte maneira. Em primeiro lugar, exite um "trap" chamado *coldStart*, que é enviado ao gerente toda vez que o agente é reinicializado. Assim, sempre que um dispositivo "entra" na rede , envia uma mensagem para o gerente comunicando: "estou ligado". Em

segundo lugar, os diversos dispositivos da rede guardam tabelas de conversão de endereços IP para físicos. Estas tabelas podem ser consultadas. O seu nome é **atTable**. Desta forma, novos endereços vão sendo descobertos através de "traps" e da tabela de conversão.

3.2 Controle do estado dos dispositivos

Não basta descobrir-se quais dispositivos compõem a rede. É necessário saber qual o estado destes dispositivos, qual a sua disponibilidade. Equipamentos podem ser acrescentados e retirados da rede com uma freqüência muito grande.

Existem, basicamente duas formas de se realizar esta função. Ou os agentes enviam mensagens periódicas ao gerente relatando o seu estado, ou o gerente consulta periodicamente os agentes perguntando os seus estados. O modelo de gerência Internet não é orientado para o uso de "traps". Por isso, optou-se pela segunda alternativa. Não são necessários novos objetos gerenciados, basta consultar qualquer um já existente. Se houver resposta, o dispositivo está operacional. Se não houver resposta existem três possibilidades: ou o dispositivo não está operacional, ou foi retirado da rede, ou não "entende" SNMP.

3.3 Controle da ligação dos dispositivos à rede

Os diversos dispositivos da rede, estão ligados a ela de várias maneiras. Um dispositivo pode ter mais de uma interface com o meio físico. O administrador humano deve saber, a qualquer momento, que interfaces são estas, e quais os seus estados. É importante inclusive ter controle sobre estes estados.

Além do conhecimento das interfaces, é importante saber a organização destas interfaces nos diversos segmentos da rede. Em redes TCP/IP, esta função é facilitada, uma vez que se utilize um esquema de subendereçamento. Conhecendo-se a porção do endereço que é utilizada para determinar a rede e as subredes, torna-se fácil descobrir a que subrede pertence cada equipamento. Os objetos utilizados para o controle da ligação dos dispositivos à rede são os seguintes:

ifTable

Tabela de interfaces com o meio do dispositivo. Dentro da tabela os objetos de interesse para a gerência de configuração são os seguintes:

ifDescr - nome do fabricante, nome do produto e a versão de "hardware" do produto;

ifType - tipo da interface de acordo com os protocolos do níves imediatamente abaixo do nível de rede;

ifMTU - tamanho do maior datagrama que pode ser enviado ou recebido pela interface;

ifSpeed - estimativa da velocidade de transmissão;

ifPhysAddress - endereço do nível imediatamente inferior ao nível de rede;

ifAdminStatus - estado desejado para a interface;

ifOperStatus - estado operacional da interface.

ipAddrTable

Contém os endereços IP das várias interfaces do dispositivo e as seguintes informações importantes para o controle da ligação dos dispositivos à rede:

ipAdEntAddr - endereço IP;

ipAdEntIfIndex - número da interface;

ipAdEntNetMask - máscara da subrede para o endereco.

3.4 Controle dos protocolos utilizados pelos nodos

Deve-se conhecer os protocolos utilizados pelos equipamentos da rede, e controlar parâmetros dèstes protocòlos. Por exemplo: o número de retentativas, tempo de vida e tamanho máximo de um pacote. Como o padrão utilizado na rede é o Internet, os objetos gerenciados necessários para esta função já estão todos definidos. Resta apenas identificar aqueles que se enquadram dentro da gerência de configuração. Estes são apresentados a seguir.

ipForwarding

Indica se a entidade age como um "gateway", repassando datagramas IP.

ipDefaultTTL

Tempo de vida do datagrama IP quando não é fornecido pelo nível de transporte.

ipRoutingTable

Rotas conhecidas pelo dispositivo. Esta tabela é fundamental para controlar os "gateways " da rede, para modificação e descobertas de rotas.

tcpRtoAlgorithm

Algoritmo utilizado para determinar o "time-out" de retransmissão de octetos TCP não confirmados.

tcpRtoMin

Valor mínimo permitido para o "time-out" de retransmissão TCP, em milisegundos.

tpRtoMax

Valor máximo permitido para o time-out de retransmissão TCP, em milisegundos.

tcpMaxConn

Limite de conexões que podem ser abertas pela entidade de transporte do dispositivo.

tcpCurrEstab

Número de conexões de transporte correntemente abertas.

tcpConnTable

Tabelas de conexões da entidade de transporte. Apresenta os seguintes campos:

tcpConnState - estado da conexão;
tcpConnLocalAddress - endereço TCP local;
tcpConnLocalPort - endereço IP local;
tcpConnRemAddress - endereço TCP remoto;
tcpConnRemPort - endereço IP remoto.

udpTable

Portas locais que utilizam UDP como ponto de entrada e saída.

3.5 Controle dos usuários de cada estação

O administrador humano também deve conhecer como estão distribuídos os usuários da rede. A qualquer momento deve ser possível descobrir se um usuário em particular tem acesso a uma determinada estação de trabalho; ou, quais os usuários que têm acesso a quais estações. Para tal é necessário definir-se um objeto gerenciado. O objeto definido é uma tabela de usuários com alguns dados básicos e com a capacidade de remoção e adição. Esta capacidade de remover e adicionar usuários em um equipamento já não é mais um problema apenas de configuração, mas, também de segurança. Entretanto, a definição de um objeto deve considerar o seu uso para futuras aplicações, e não apenas para aquela à qual está sendo projetada, por isso os objetos permitem leitura e escrita. Para configuração bastaria a permissão de leitura.

3.5.1 Objetos gerenciados

usuario OBJECT IDENTIFIER {ufrgs 1}

OBJECT

tabelaDeUsuarios {usuario 1}

Syntax:

SEQUENCE-OF DadosDoUsuario

Definition: Tabela contendo dados do usuario do dispositivo.
Access: read-write.
Status: optional.

OBJECT

dadosDoUsuario {tabelaDeUsuarios 1}

Syntax:

```
DadosDoUsuario ::= SEQUENCE    {identificacao DisplayString,
                                nomeDoUsuario DisplayString,
                                tipoDoUsuario DisplayString,
                                estadoDoUsuario INTEGER }
```

Definition: Linha da tabela de usuários. O índice para consulta da tabela é o objeto *identificacao*.

Access: read-write.

Status: mandatory.

OBJECT

identificacao {dadosDoUsuario 1}

Syntax:

DisplayString

Definition:

Identificação do usuário (username).

Access: read-write.

Status: mandatory.

OBJECT

nomeDoUsuario {dadosDoUsuario 2}

Syntax:

DisplayString

Definition: Nome do usuário.

Access: read-write.

Status: mandatory.

OBJECT

tipoDoUsuario {dadosDoUsuario 3}

Syntax:

DisplayString

Definition: Privilégios do usuário (grupo, privilegiado,etc)

Access: read-write.

Status: mandatory.

OBJECT

estadoDoUsuario {dadosDoUsuário 4}

Syntax:

INTEGER {remove(0), permanece(1)}

Definition: Remove a linha do usuário, quando recebe o valor 0.

Acess: read-write.

Status: mandatory.

3.6 Controle do software em execução

Além de conhecer a configuração relativamente estática da rede, isto é, os equipamentos, os usuários, os protocolos dos equipamentos, as interfaces com o meio; o gerente de rede precisa muitas vezes visualizar os programas que estão sendo executados em cada dispositivos. Afinal, esta configuração momentânea pode estar relacionada com algum problema da rede (congestionamento, ou dispositivo com tempo de resposta muito baixo). Assim, a qualquer momento, os programas que estão sendo executados em uma máquina devem estar disponíveis.

Os objetos definidos para tal função são simples. Há uma tabela que contém o nome do programa, a identificação do usuário e a hora de início da execução do programa.

3.6.1 Objetos gerenciados

programa OBJECT IDENTIFIER {ufrgs 2}

OBJECT

tabelaDeProgramas {programa 1}

Syntax:

SEQUENCE-OF DadosDoProg

Definition: Tabela de programas em execução.

Acess: read.

Status: optional.

OBJECT

dadosDoProg {tabelaDeProgramas 1}

Syntax:

```
DadosDoProg ::= SEQUENCE        {usuarioDoProg  DisplayString,
                                 nomeDoProg  DisplayString,
                                 horaDoProg  DsplayStrng}
```

Definition: Linha com os dados do programa em execução. O índice para consultar a tabela é formado pelos objetos *usuarioDoProg*, *nomeDoProg* e *horaDoProg*.
Access: read.
Status: mandatory.

OBJECT

usuarioDoProg {dadosDoProg 1}

Syntax:

DisplayString

Definition: Identificação do usuário que disparou o programa. Semelhante ao objeto *identificacao* da tabela de usuários.
Access: read.
Status: mandatory.

OBJECT

nomeDoProg {dadosDoProg 2}

Syntax:

DisplayString

Definition: Nome do programa que está sendo executado.
Access: read.
Status: mandatory.

OBJECT

horaDoProg {dadosDoProg 3}

Syntax

DisplayString

Definition: Hora em que o programa teve sua execução iniciada.
Access: read.
Status: mandatory.

3.7 Controle da localização dos arquivos na rede

A correta distribuição dos servidores de disco em uma rede é importantíssima para que o seu desempenho se mantenha dentro de limites aceitáveis. O

administrador humano deve ter disponível a informação de onde encontra-se determinado arquivo utilizado em uma estação de trabalho.

Os objetos gerenciados definidos para esta função estão agrupados em uma tabela que contém o nome do servidor em que se encontra o arquivo (ou diretório), o nome do arquivo no servidor, o nome do arquivo no dispositivo consultado e o tipo de acesso permitido.

3.7.1 Objetos gerenciados

arquivo OBJECT IDENTIFIER {ufrgs 3}

OBJECT

tabelaDeArquivos {arquivo 1}

Syntax:

SEQUENCE-OF DadosDoArq

Definition: Tabela contendo a distribuição dos arquivos de uma estação de trabalho.
Access: read.
Status: optional.

OBJECT

dadosDoArq {tabelaDeArquivos 1}

Syntax:

DadosDoArq ::= SEQUENCE {nomeDoServ DisplayString,
arquivoServ DisplayString,
arquivoLocal DisplayString,
acessoArq DisplayString }

Definition: Linha com os dados do arquivo. O índice utilizado para consultar a tabela é o objeto *arquivoLocal*.
Access: read.
Status: mandatory.

OBJECT

nomeDoServ {dadosDoArq 1}

Syntax

DisplayString

Definition: Nome do dispositivo onde o arquivo está realmente armazenado.

Access: read.

Status: mandatory.

OBJECT

arquivoServ {dadosDoArq 2}

Syntax

DisplayString

Definition: Nome do arquivo na estação servidora.

Access: read.

Status: mandatory.

OBJECT

arquivoLocal {dadosDoArq 3}

Syntax

DisplayString

Definition: Nome do arquivo na estação consultada.

Access: read.

Status: mandatory.

OBJECT

acessoArq {dadosDoArq 4}

Syntax

DisplayString

Definition: Tipo de acesso permitido ao arquivo.

Access: read.

Status: mandatory.

3.8 Árvore de identificadores de objetos

A figura 3 apresenta a árvore de identificadores de objetos dos novos objetos definidos no âmbito deste trabalho, para a gerência de configuração. A raiz da árvore é "ufrgs", que é um nodo fictício, não registrado. É utilizado apenas para organizar os objetos.

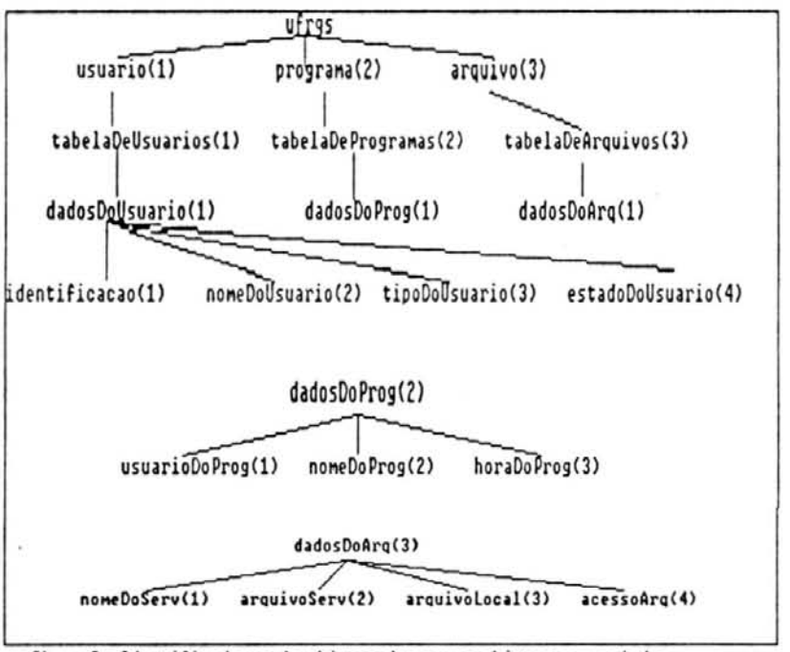

Figura3: Identificadores de objetos dos novos objetos gerenciados

4 SISTEMA GERENTE IMPLEMENTADO

A ferramenta de gerência de configuração é composta de diversas funções que permitem que o usuário gerencie a configuração da rede. Como qualquer ferramenta de gerência, deve facilitar o trabalho do administrador humano. Para tanto é necessário que as funções implementadas sejam úteis e que a interface com o usuário seja definida adequadamente. As várias ferramentas devem interagir com o usuário de maneira semelhante, para que o mesmo não sinta dificuldades ao passar de uma para outra. Além disso, as ferramentas devem permitir que o usuário gerencie a rede sem ter um conhecimento mais aprofundado dos objetos gerenciados. As funções necessárias já foram definidas no capítulo 3. Este capítulo detalha como elas foram implementadas, num protótipo desenvolvido no contexto desta dissertação, e como o administrador humano interage com o protótipo de gerente desenvolvido.

A implementação do protótipo foi feita em C numa plataforma Unix. Para a construção da interface foi utilizado X-View (X Window-System-based Visual/Integrated Enviroment for Workstations), que é uma ferramenta que permite a definição de uma interface para o sistema X-Window . Também é utilizado o banco de dados Ingres, relacional, que utiliza a linguagem EQUEL (Embedded Query Language). Como agente, e para construir as funções do gerente, foi usado o pacote CMU (Carnegie Mellon University) SNMP 1.1, que é um "software" de domínio público.

A seguir são apresentadas as soluções para as diversas funções de gerência de configuração definidas no capítulo 3. As funções são implementadas apenas com objetos da MIB-I, pois o pacote utilizado (CMU) não implementa a MIB-II. Para defnição das funções é utilizada a linguagem SDL(Specification and Description Language - [CCI 88]).

4.1 Controle dos dispositivos existentes na rede

4.1.1 Função

Como foi visto anteriormente, esta função se divide em duas subfunções: a descoberta dos dispositivos e a descoberta de suas características.

A descoberta dos dispositivos é feita a partir da consulta do objeto *atTable*, que é uma tabela de conversão de endereços IP em endereços físicos. Esta tabela apresenta todos os endereços IP conhecidos por um determinado equipamento. Esta função deve ser implementada como um processo paralelo. Isto é necessário, porque é uma função que pode demorar mais tempo, uma vez que todos os dispositivos devem ser consultados, e o gerente não deve permanecer "pendurado" por muito tempo. Enquanto dispositivos são descobertos, outros podem ser consultados. Além desta consulta, também deve-se utilizar os "traps" *coldStart*. Esta função não foi implementada no protótipo.

Além da descoberta automática de dispositivos, é importante que o usuário possa definir dispositivos, e que estas definições sejam validadas. Assim, o gerente permite que novos dispositivos sejam definidos manualmente, e durante esta definição verifica se o dispositivo conhece SNMP. Também verifica a que rede pertence o novo dispositivo descoberto. Esta validação da rede é feita com a utilização do objeto *ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask*. Este objeto guarda a máscara de subrede de um endereço IP. A partir de uma operação de AND, deste objeto com o endereço, é possível identificar a subrede a que pertence uma determinada interface de um dispositivo e se esta rede já foi definida. A figura 4 descreve a função de registro de um dispositivo.

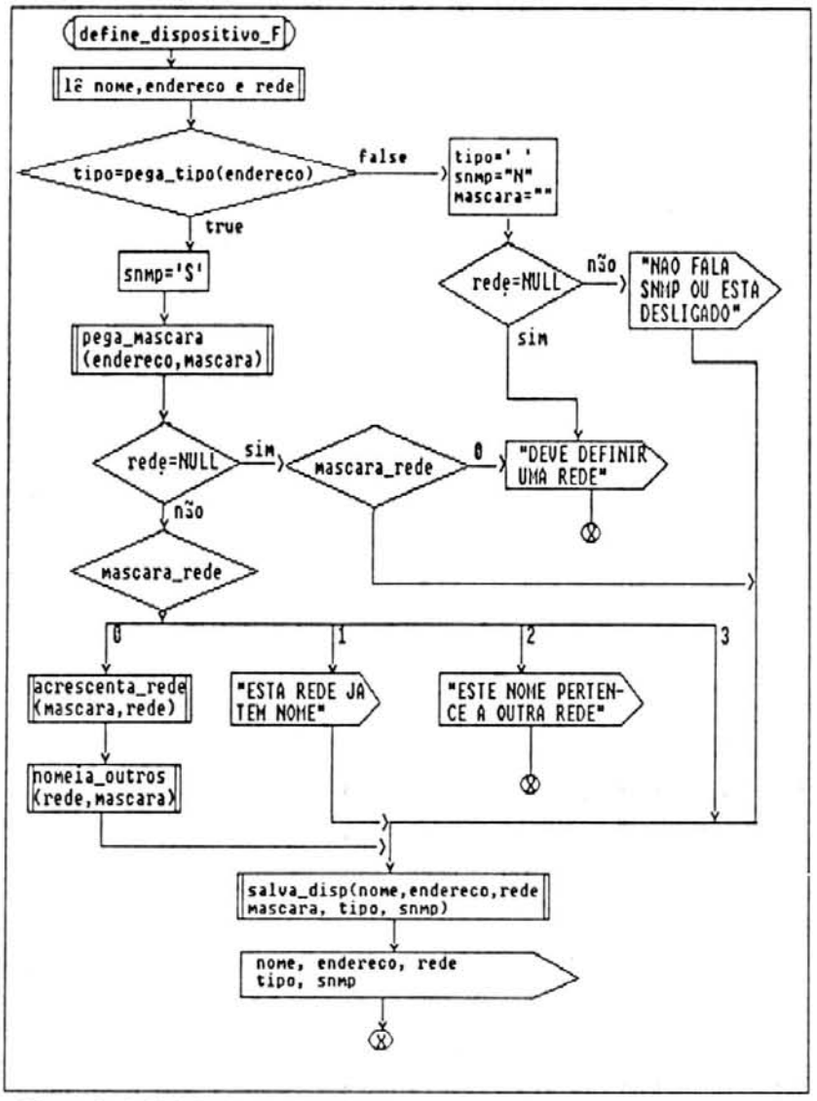

Figura 4: Definição de um dispositivo.

A descoberta das características do dispositivo é feita, simplesmente, consultando-se a MIB do dispositivo. Os objetos pertinentes são definidos na seção 3.1. A figura 5 descreve a função que recupera as características de um dispositivo.

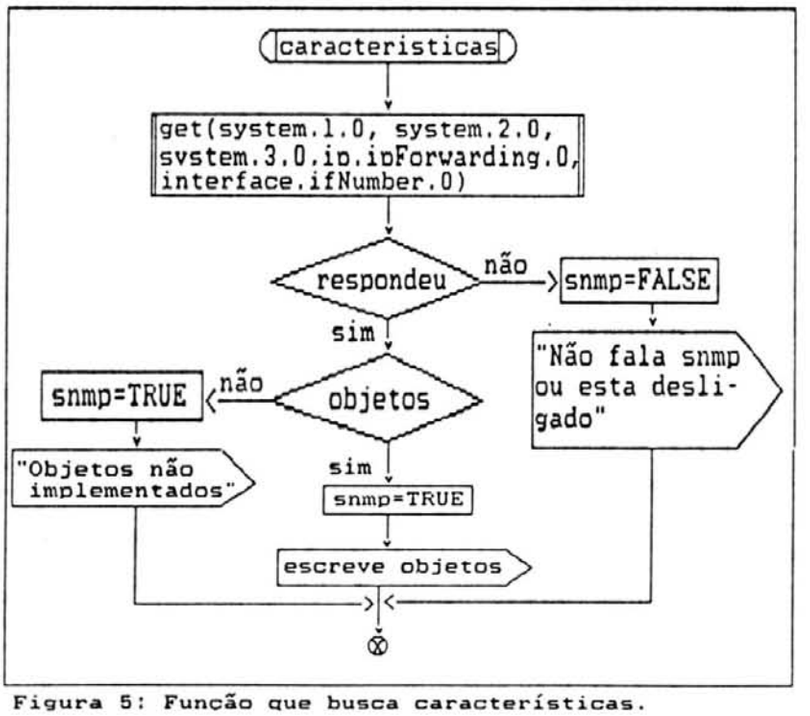

Figura 5: Função que busca características.

4.1.2 Interface

A interface da primeira subfunção consiste de três aspectos: descoberta automática, definição manual e apresentação dos dispositivos existentes na rede. A descoberta automática, apesar de não estar implementada, seria executada a partir do menu chamado *Ferramentas*, selecionado-se o item *Descobre dispositivos*. A definição manual é feita a partir da seleção do item *Define dispostivo*. A apresentação dos dispositivos existentes na rede, porque não se enquadra apenas nesta função, e devido a sua importância dentro do gerente de rede, é apresentada na seção 4.3.2. A figura 6 apresenta o menu *Ferramentas* e a janela para a definição de um dispositivo implementada.

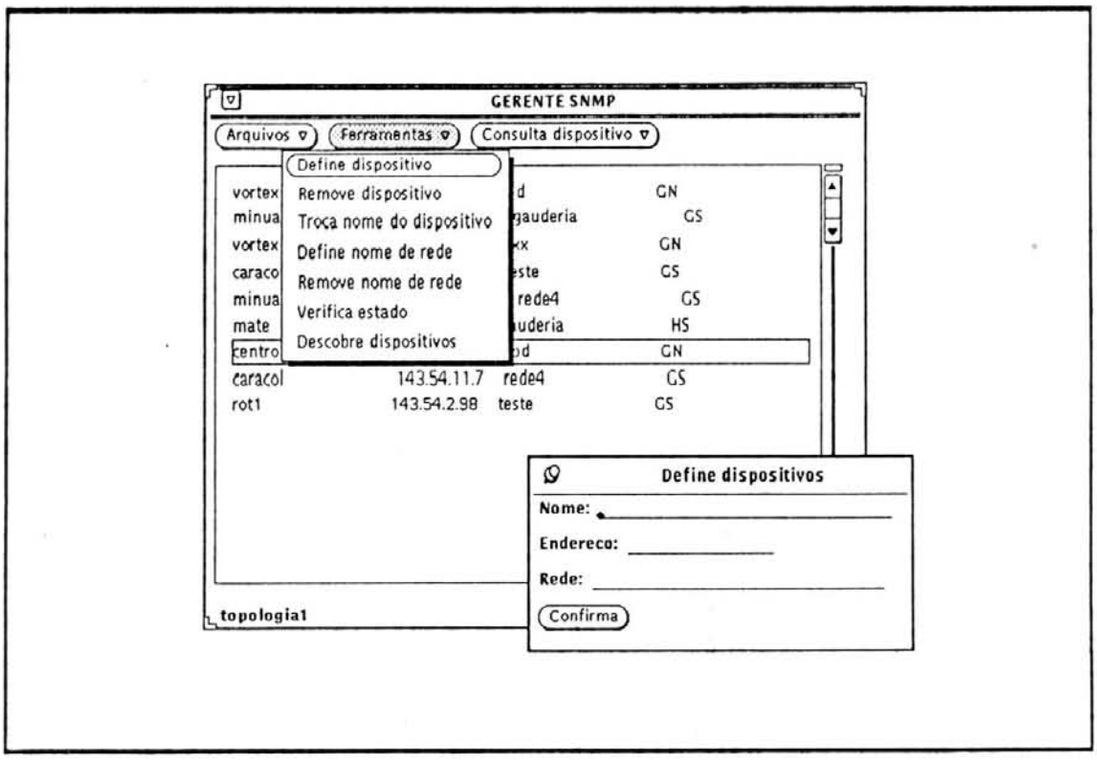

Figura 6: Menu Ferramentas.

A interface da segunda subfunção é simples. Para solicitá-la basta selecionar-se um dispositivo, e escolher, dentro do menu chamado *Consulta dispositivo*, a opção *Características*. O resultado será apresentado em uma nova janela, que poderá permanecer aberta durante toda a utilização do programa, ou poderá ser fechada após a consulta. A figura 7 apresenta o menu e a janela com a reposta à consulta.

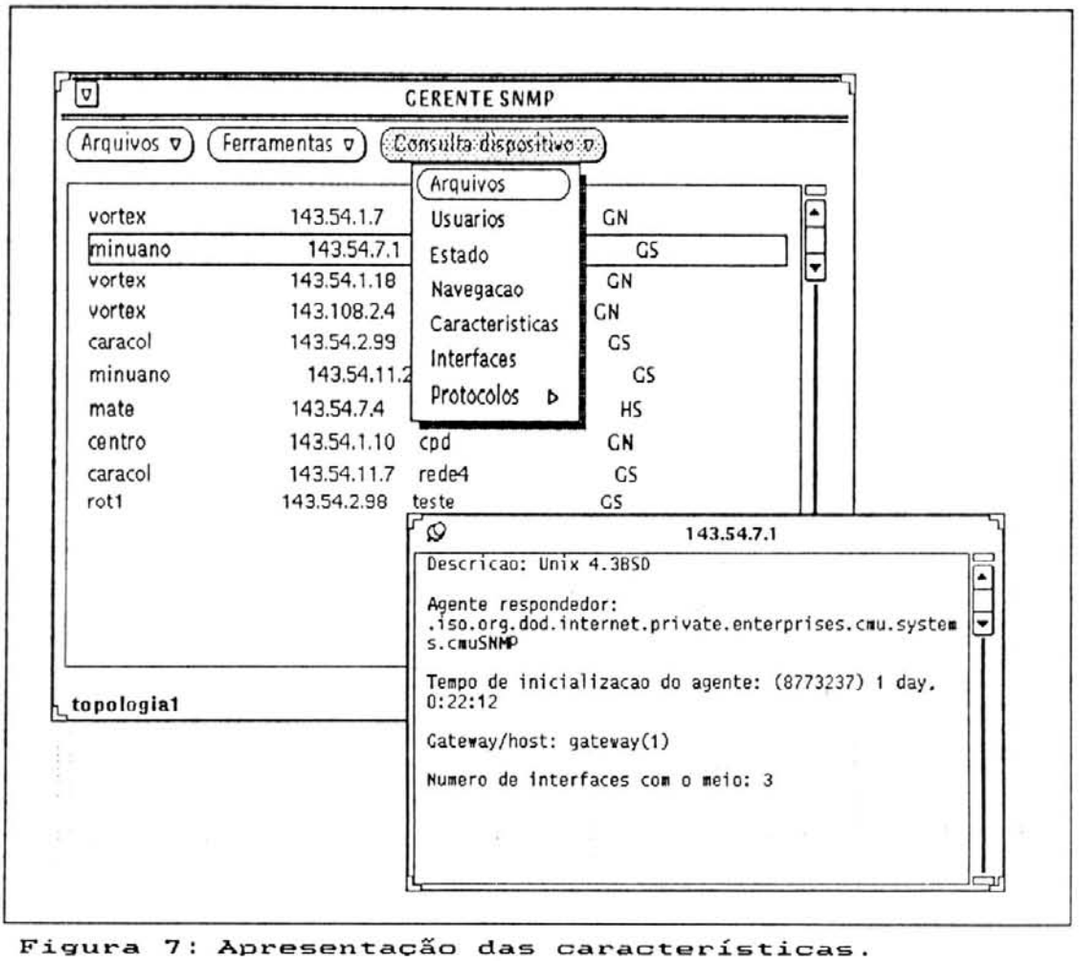

Figura 7: Apresentação das características.

4.2 Controle do estado dos dispositivos

4.2.1 Função

Existem dois momentos em que o estado dos dispositivos são controlados. Em primeiro lugar, ele é controlado sempre que uma consulta é feita a um dispositivo. Neste momento, pode-se saber se o dispositivo respondeu ou não à consulta. Em segundo lugar pode-se fazer uma consulta global, para descobrir o estado de todos os dispositivos. Esta função é implementada como um processo paralelo, pela mesma razão que a função apresentada anteriormente. Afinal, todos os dispositivos devem ser testados. Ela é dispara pelo usuário. Entretanto, é interessante criar-se um mecanismo de disparo automático, com um intervalo de tempo determinável. Isto permitiria que o

administrador humano tivesse sempre a imagem correta da rede, e que determinasse um intervalo de consulta que não sobrecarregasse o tráfego da mesma.

Para se verificar o estado do dispositivo é feita uma consulta a um objeto gerenciado. Se houver resposta, o dispositivo está operacional. Entretanto, se não houver resposta, não é possível dizer nada a respeito dele, pois várias são as alternativas: está desligado, não está mais na rede, ou não possui um agente implementado.

A solução para este problema está em utilizar outro meios para se comunicar com o dispositivo (*ping*, por exemplo) ou delegar a responsabilidade para o administrador humano. A opção escolhida foi a segunda. São apresentados aos usuários os equipamentos que não responderam, e é sua a decisão de removê-los da representação da topologia, ou não.

4.2.2 Interface

Para realizar esta função, basta selecionar, no menu chamado *Ferramentas* , o item *Verifica estado*. A figura 6 apresenta este menu. A resposta é apresentada através de um sinal que distingue se o dispositivo respondeu, ou não à consulta. Na seção 4.3.2, esta interface é melhor comentada.

4.3 Controle da ligação dos dispositivos à rede

4.3.1 Função

O controle da ligação dos dispositivos à rede dá-se em duas situações. A primeira, quando o dispositivo é definido. Sempre que um dispositivo é definido, é solicitado um endereço IP, e uma rede correspondente. Esta rede é testada com uma tabela de redes no banco de dados e com os dado da tabela *ipAddrTable* (máscara de subrede) para verificar se está correta, como está explicado na seção 4.1.1.

A segunda situação ocorre quando o usuário deseja conhecer as diversas interfaces com o meio de um determinado dispositivo. São consultadas as tabelas *ifTable* e *ipAddrTable*, e são descobertas as características da diversas interfaces. As duas tabelas são combinadas. As interfaces descobertas são anotadas no banco de dados que guarda a topologia da rede. A figura 8 descreve esta função.

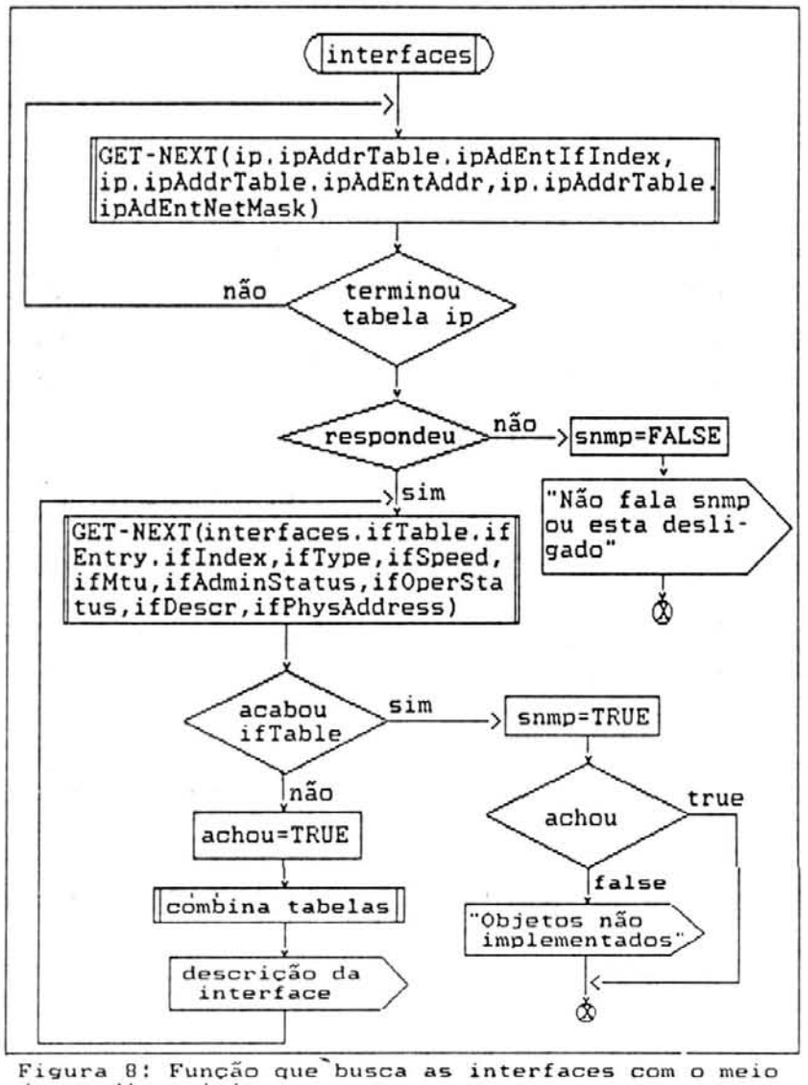

Figura 8: Função que busca as interfaces com o meio de um dispositivo.

O objeto *interfaces.ifTable.ifEntry.ifAdminStatus* permite que se troque o estado de uma interface. Entretanto, esta função não foi implementada no protótipo por

razões de segurança. Como está colocado na seção 2.1.2 ,não existem mecanismos de segurança eficientes em SNMP.

4.3.2 Interface

A interface ideal para esta função, para o controle dos dispositivos existentes na rede (seção 4.1) e para o controle do estado dos dispositivos (seção 4.2) é a utilização de mapas. Os mapas permitem que a interface do programa esteja de acordo com o modelo mental que o administrador humano tem da rede. Ninguém pensa em uma rede como uma tabela de segmentos e equipamentos, mas sim, como máquinas ligadas a segmentos que, por sua vez, são ligados por pontes. Um mapa representa muito melhor uma rede, do que qualquer tabela.

Outra característica importante é a utilização de símbolos que apresentem algum significado. Isto permite que o usuário perceba imediatamente se um dispositivo é uma estação de trabalho ou uma ponte; ou se um dipositivo mudou de estado. Para isso podem ser utilizados símbolos gráficos, cores, ou sinais sonoros.

Entretanto, a contrução deste tipo de interface é, muito mais, um trabalho de computação gráfica e está fora do escopo deste trabalho. Por isso, a interface desenvolvida é baseada em uma tabela de endereços. Como pode ser visto na figura 9, a interface apresenta uma projeção da tabela do banco de dados que guarda a topologia da rede. São apresentados ao usuário o nome do dispositivo, o endereço IP, a rede correspondente àquele endereço, o nome da rede, o tipo de dispositivo (G - age como "gateway", H - age somente como "host"), e o estado do dispositivo (S - apresenta um agente SNMP, N - não apresenta um agente SNMP). Há uma linha para cada endereço IP.

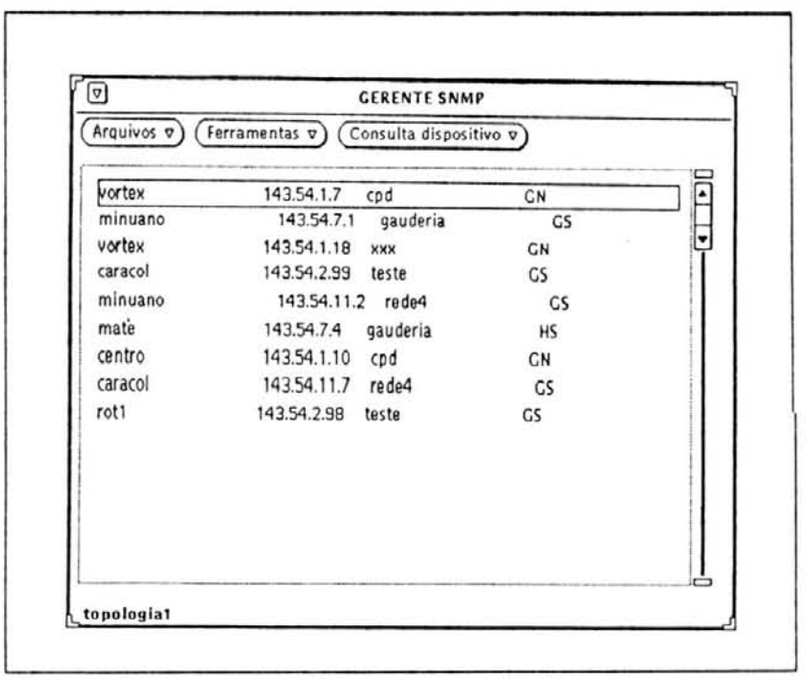

Figura 9: Apresentação dos dispositivos.

Esta interface não é, de maneira alguma, a melhor para um gerente que pretenda ser utilizado eficientemente. Não é sequer razoável. Ela é apenas um meio para a execução das funções definidas no protótipo. Um exemplo de interface útil é apresentado na figura 10. Esta interface pertence ao programa SunNetManager e permite o desenho de mapas, a utilização de símbolos e cores.

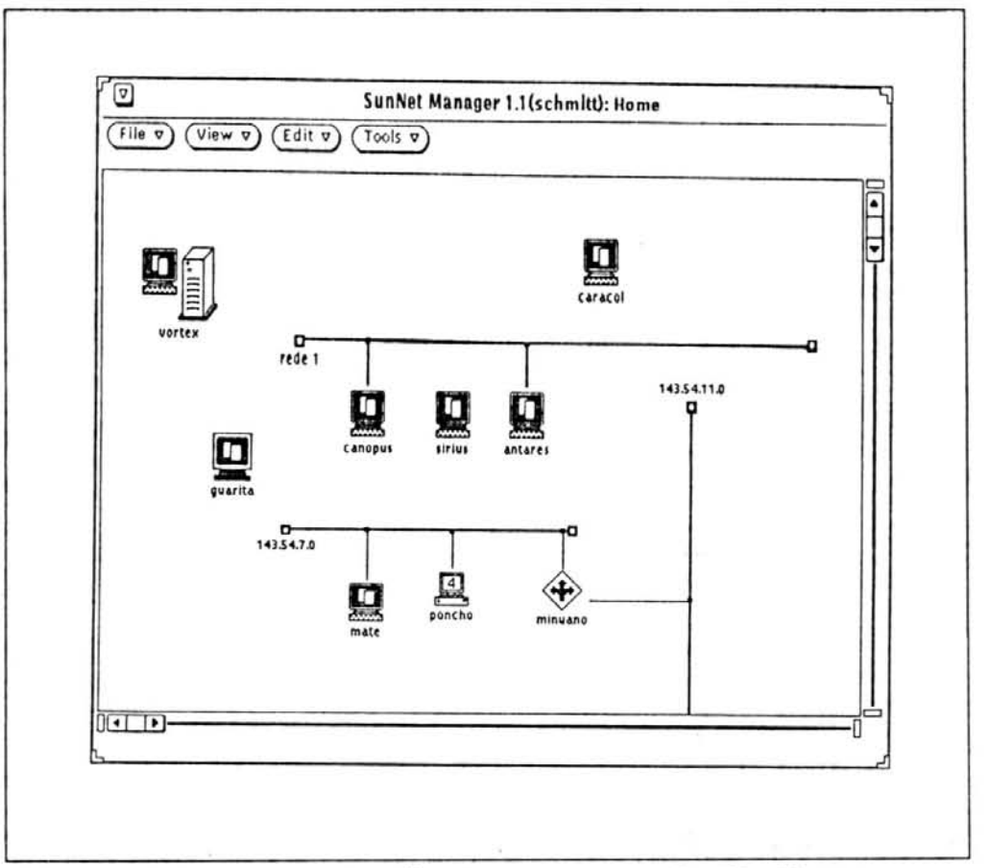

Figura 10: Apresentação do programa SunNetManager.

Além do mapa, o gerente deve apresentar as características de cada interface. Essas características, são apresentadas quando se seleciona o item *Interfaces* do menu chamado *Consulta dispositivo*; e são apresentadas em uma janela com as mesmas características daquela que apresenta as características do dispositivo (seção 4.1.2). A figura 11 apresenta esta janela.

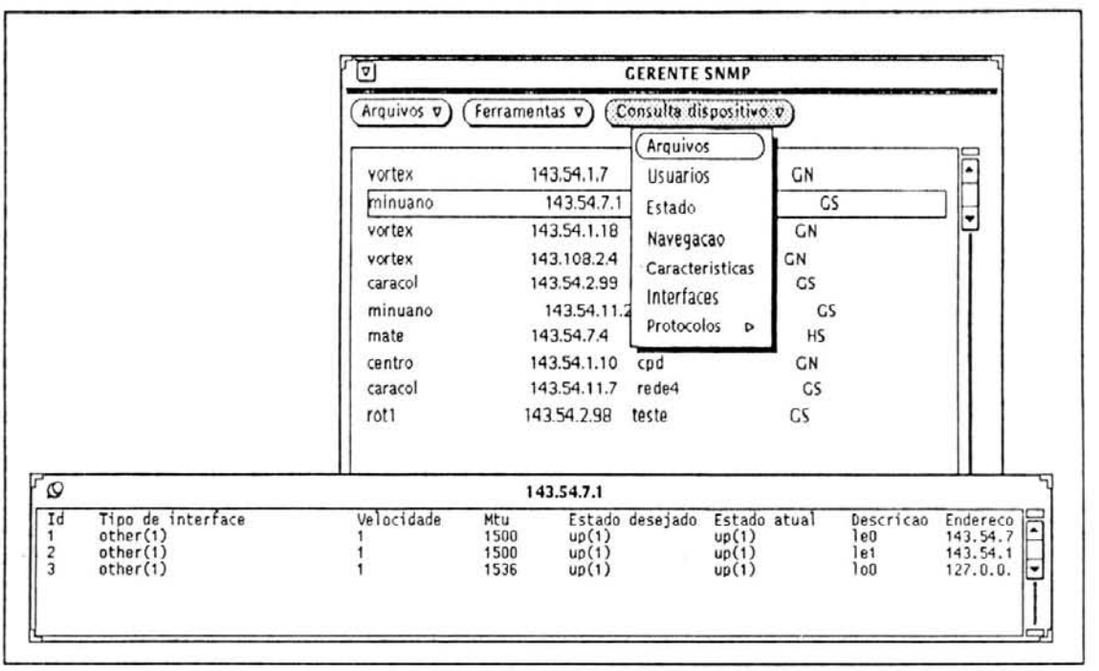

Figura 11: Características das interfaces.

4.4 Controle dos protocolos utilizados pelos nodos

4.4.1 Função

O controle dos protocolos utilizados pelos nodos é, basicamente, feito consultando-se os objetos apresentados na seção 3.4. No protótipo, foram implementadas as funções referentes aos protocolos IP e TCP. Além da consulta, existem objetos que podem ser modificados, como por exemplo: *ipForwarding, ipDefaultTTL, ipRoutingTable* e *ipNetToMediaTable.* Da mesma forma que na alteração do estado de interfaces (seção 4.3.1), a função não foi implementada devido à pouca segurança fornecida pelo SNMP, que torna perigosa a autorização de atualização de objetos. As figuras 12 e 13 descrevem, respectivamente, as funções que buscam os dados relacionados aos protocolos IP e TCP. Elas são muito semelhantes, diferenciando-se, apenas, nos objetos consultados.

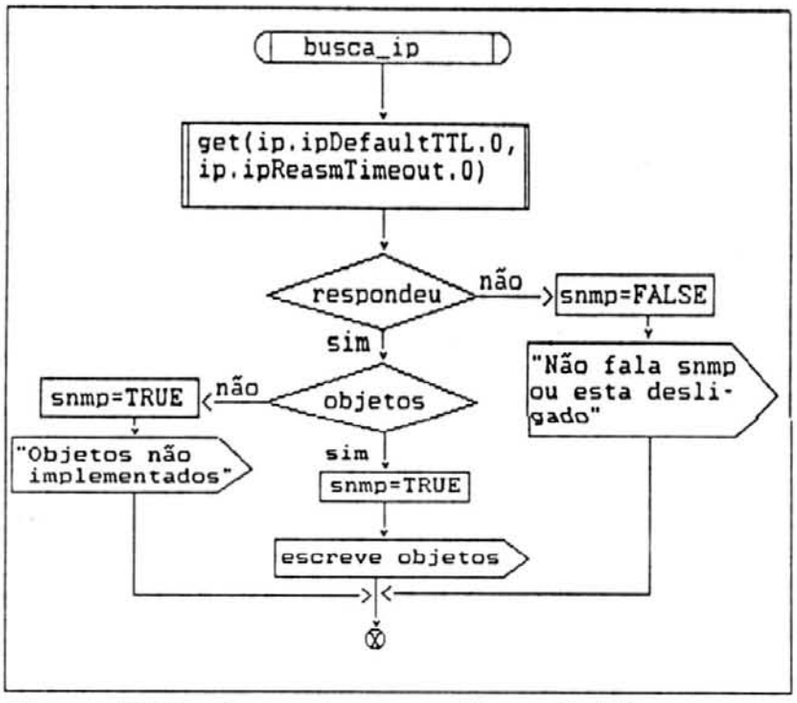

Figura 12:Função que busca configuração IP.

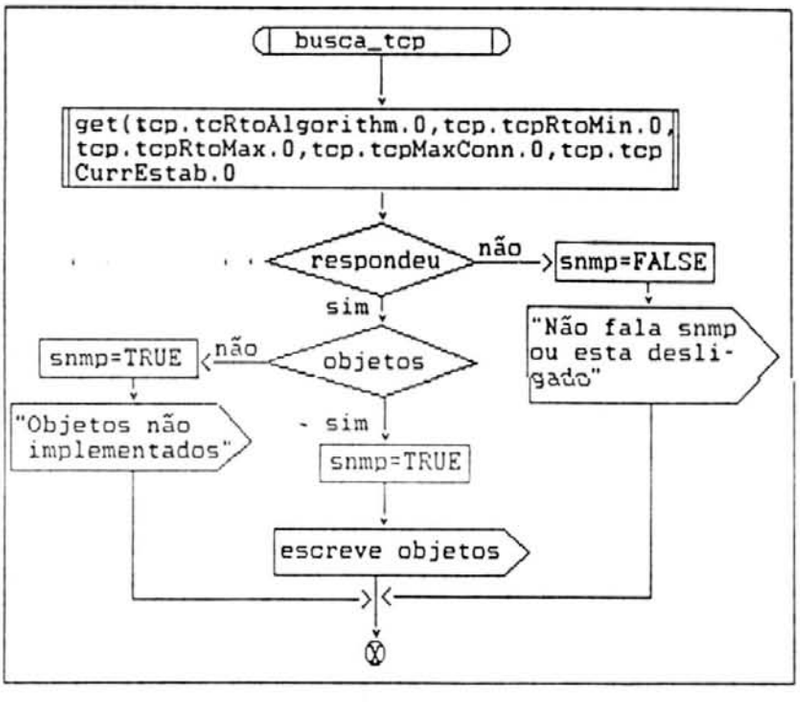

Figura 13:Função que busca configuração TCP.

4.4.2 Interface

Estas funções são realizadas selecionando-se o item *Protocolos* do menu chamado *Consulta dispositivo*. Antes, claro, é necessário selecionar-se o dispositivo em questão. Um outro menu permite escolher entre os protocolos existentes. As janelas que retornam o resultado têm as mesmas características daquelas que retornam as características do dispositivo (seção 4.1.2). A figura 14 apresenta o menu *Protocolos* e as janelas referentes aos protocolos IP e TCP.

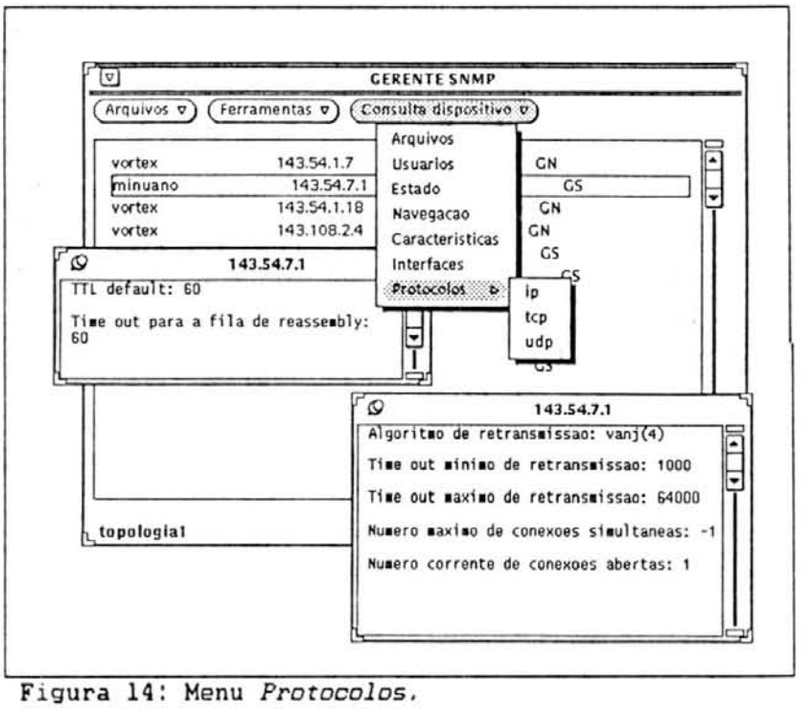

Figura 14: Menu *Protocolos*.

4.5 Controle dos usuários de cada estação

Esta função não foi implementada. Entretanto não é diferente das outras. A consulta aos objetos definidos na seção 3.5 retornará uma tabela de usuários.

4.6 Controle do software em execução

Esta função não foi implementada. Entretanto não é diferente das outras. A consulta aos objetos definidos na seção 3.6 retornará uma tabela de programas em execução.

4.7 Controle da localização dos arquivos na rede

Esta função não foi implementada. Entretanto não é diferente das outras. A consulta aos objetos definidos na seção 3.7 retornará uma tabela de arquivo (ou diretórios) e suas localições. Em um trabalho desenvolvido para o SunNetManager, foi implementado um agente com características semelhantes. O resultado da consulta é apresentado da mesma forma que deve ser feita aqui. A figura 15 mostra como estes resultados devem ser apresentados.

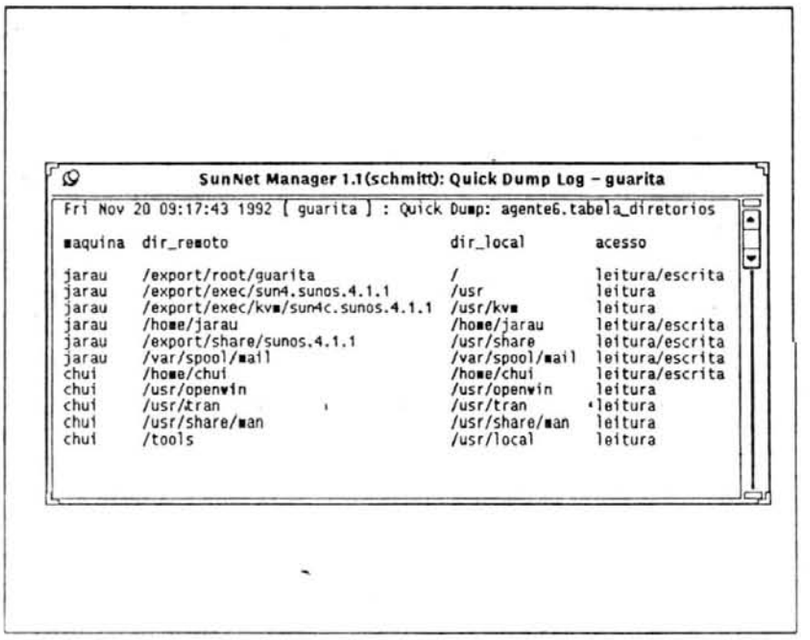

Figura 15: Distribuição dos arquivos.

4.8 Outras funções

Além das funções definidas, existem outras funções acessórias que fazem com que um gerente seja útil. Por isso foram implementadas funções que permitem carregar topologias, salvar topologias e remover topologias. Isto permite que o administrador humano tenha diversas redes armazenadas. São funções simples, que são executadas selecionando-se o menu *Arquivos*. A figura 16 apresenta este menu.

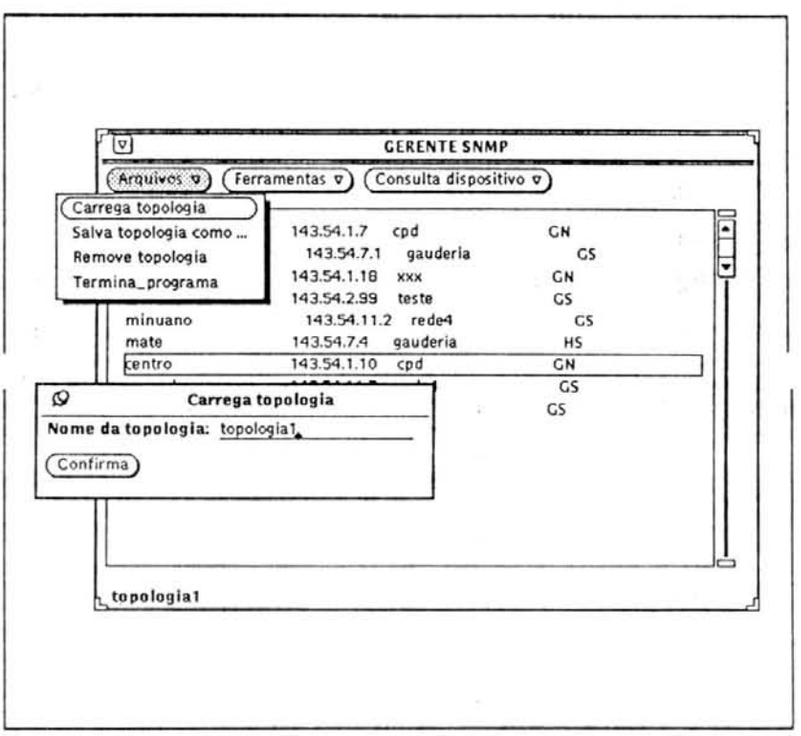

Figura *16:Menu Arquivos.*

Também há funções que permitem que o usuário troque nomes de redes, remova nomes de redes e troque o nome dos dispositivos. Estas funções são selecionadas dentro do menu *Ferramentas*, como pode ser visto na figura 6, apresentada anteriormente.

4.9 O banco de dados de apoio ao sistema de gerência

A utilização de um banco de dados pode facilitar muito a gerência de configuração. Em primeiro lugar, e obviamente, o banco de dados serve para guardar as diversas configurações de redes existentes, as diversas topologias. Isto é, pode-se ter a visão de várias redes independentes. Isto é implementado através das funções explicadas na seção 4.8. O banco de dados facilita a recuperação e a atualização destes dados, que, se mantidos em um arquivo, necessitariam de rotinas mais complexas. É, a partir destes dados, que é possível montar o mapa da rede. Assim, a ferramenta que implementar mapas, utilizará como subsídio, o banco de dados.

No entanto, o banco de dados não serve somente para armazenar topologias, pode ser utilizado para o armazenamento de outros dados. A gerência de configuração, obviamente não trata apenas de descoberta de configuração. A modificação da configuração também faz parte. Apesar de não ter sido implementada nenhuma função de modificação, por razões já apontadas, quando se fala em banco de dados, deve-se levar em consideração tal aspecto. Quando são feitas modificações, deve ser possível armazenar configurações anteriores; ou mesmo planejar configurações futuras. Por exemplo: é possível que se queira guardar a configuração de interfaces de um determinado dispositivo, para uma comparação futura. Ou mesmo, como uma forma de documentação da evolução da rede. Durante testes, é importante retornar a configurações anteriores. Por exemplo: o administrador humano pode querer testar o desempenho da rede com algumas interfaces desligadas. Após o teste, é possível que ele queira voltar ao estado original. Em uma rede com, por exemplo, 300 dispositivos, é necessário que esta configuração anterior seja refeita automaticamente. Para tal, é preciso guardá-la em algum lugar.

O protótipo utiliza o banco de dados somente para a primeira função. O banco de dados utilizado é o banco de dados Ingres. Este banco de dados foi escolhido, por tratar-se de um banco de dados de fácil manipulação (relacional) e por estar disponível para utilização na rede da UFRGS. A opção por um banco de dados orientado à objetos foi considerada. Entretanto, esta solução não foi adotada porque o modelo de dados Internet não utiliza orientação a objetos, utiliza tabelas. Mas o banco de dados orientado a objetos parece ser a escolha mais coerente em uma implementação que utilize o modelo OSI de gerência.

A estrutura para armazenamento da topologia está descrita na figura 17, num diagrama entidade-relacionamento. Identificação é a combinação do endereço IP com a máscara de subrede e identifica uma rede.

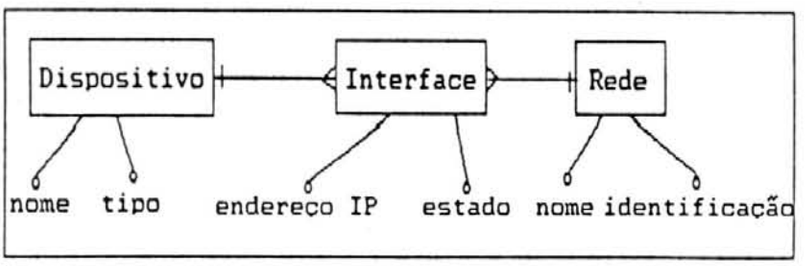

Figura 17: Esquema do armazenamento da topologia.

Apesar da estrutura definida na figura 17, por uma questão de rapidez de implementação, e somente por isso, o protótipo apresenta duas tabelas diferentes, definidas a seguir em EQUEL. A coluna *redeid* corresponde à identificação de rede.

```
create dispositivos
(           nome = c30,
            endereco = c15,
            rede = c30,
            tipo = c1,
            redeid = c15,
            snmp = c1
)

create redes
(           rede = c30,
            redeid = c15
)
```

4.10 Extensões

É importante que o gerente possa ser expandido; que novas funções possam ser definidas. Para tal, é preciso conhecer a estrutura do programa. O programa

é formado por quatro arquivos básicos: snmpget.c, main.c, funcoes.c e db.q. O primeiro apresenta a função para executar uma consulta snmp; o segundo, a definição da interface com o usuário; o terceiro, as funções disponíveis para o usário; e o quarto, as funções executadas sobre o banco de dados. Para extender o programa, deve-se modificar estes arquivos e utilizar as suas funções.

4.10.1 Consulta snmp

Para a recuperação de objetos de um nodo gerenciado, utiliza-se a função snmpget() do arquivo snmpget.c. Este arquivo inclui alguns arquivos *.h e utiliza rotinas da biblioteca do pacote CMU SNMP.

A rotina snmpget() recebe os seguintes parâmetros:

-int operação: pode-se executar as operações *get* (GET_REQ_MSG) e *getnext* (GETNEXT_REQ_MSG);
-char *gateway: endereço, ou nome, do dispositivo a ser consultado;
-char *community: nome da comunidade a que pertence o gerente;
-char *objetos[10][100]: nomes dos objetos a serem consultados;
-int *num_objetos: número de objetos a serem consultados, e, ao final, efetivamente quantos foram consultados;
-char buf1[10][512]: conteúdo das variáveis retornadas;
-char buf2[10][512]: nome das variáveis retornadas;

Ela retorna o status da operação, que pode ser: sucesso (STAT_SUCCESS) ou "time-out" (STAT_TIMEOUT). Mesmo que tenha havido sucesso na operação, pode não ser retornado nenhum objeto. Este é o caso em que os objetos solicitados não existem na MIB consultada.

Nem todos os objetos são necessariamente retornados. Quando vários objetos são pedidos, e alguns não existem na MIB, somente aqueles que existem são retornados. Uma operação SNMP, por definição, não devolve nenhuma variável, quando uma delas não existe. Entretanto a rotina abstrai este aspecto para a aplicação qua a utiliza.

Esta rotina é baseada no programa que implementa a aplicação *snmpget* dentro do pacote CMU, e é muito útil para o desenvolvimento de novas funções, já que permite a recuperação de vários objetos simultaneamente, e abstrai os problemas de "time out" e de objetos desconhecidos.

4.10.2 Interface com o usuário

Para a construção de uma nova função, deve-se definir como ela estará disponível para o usuário. A interface com o usuário está definida no arquivo main.c. Este arquivo também utiliza as definições do arquivo interface.h e dos arquivos utilizados pelo XView.

A interface é definida de acordo com a ferramenta XView. Todos as janela e menus são declarados no arquivo interface.h. A modificação destas janelas é uma tarefa simples; requer, apenas, o conheciento de XView.

4.10.3 Banco de dados

O banco de dados pode ser consultado utilizando-se EQUEL. A definição das tabelas é feita na seção 4.9. A tabela de dispositivos sempre tem o mesmo nome da topologia em uso. E a tabela de redes tem o mesmo nome acrescido da letra "l", no final.

Também podem ser utilizadas algumas rotinas já definidas no arquivo db.q. A seguir são listadas algumas que podem ser úteis para outras funções.

acha_topologia()

Recebe como parâmetro uma string e retorna TRUE se a string corresponde a uma topologia, e FALSE se não corresponde.

le_topologia(Paine_item painel)

Recebe como parâmetro um painel, e escreve nele a topologia corrente.

troca_topologia_corrente(char *nome)

Recebe como parâmetro uma string, e troca topologia corrente.

salva_disp(registro *disp, char *identif)

Salva um dispositivo no banco de dados.

acrescenta_tipo(char *endereco, char *tipo)

Modifica o tipo de um determinado dispositivo. Recebe como parâmetros o endereço e o tipo.

acrescenta_snmp(char *endereco, char *snmp)

Assinala se o dispositivo "entende" snmp. Recebe como parâmetros o endereço e "S" (fala), ou "N" (não fala).

acrescenta_rede(char *identif, char *rede)

Modifica a rede e o identificador de rede de um dispositivo.

nomeia_outros(char *rede, char * idenf)

Modifica o nome de rede de todos os dispositivos que tiverem identificação de rede igual a *identif.*

remove_topologia_bd(char *nome)

Remove a topologia referenciada por *nome.*

salva_topologia_F(char *nome)

Salva topologia corrente como *nome.*

verifica_rede(char *rede)

Verifica se exite rede com nome igual a *rede.*

verifica_mascara(char *endereco)

Verifica se o endereco possui uma identificação.

remove_rede_bd(char *rede)

Remove rede da tabela de redes. Se remover retorna TRUE, senão, retorna FALSE.

troca_nome_bd(char * nome, char *endereco)

Troca o nome do dipositivo.

troca_tudo(char *endereco, char *rede)

Troca o nome de rede do todos os dispositivos que tem uma identificação igual a do dispositivo identificado por *endereco*.

mascara_rede(char *identifi, char *rede)

Recupera a identificação de uma determinada rede. Recebe como parâmetro o nome da rede.

Para utilizar EQUEL, deve-se pré-compilar os arquivos *.q.

4.11 Modificações no CMU SNMP

Para o desenvolvimento e teste do protótipo foi utilizado CMU SNMP 1.1. Este é um pacote, de domínio público, composto por um programa agente, alguns programas aplicativos e uma interface para programas de aplicação. A interface foi utilizada para o desenvolvimento do gerente, e o agente para o teste do primeiro.

Entretanto, algumas modificações tiveram que ser feitas para que o gerente pudesse funcionar de acordo com o que era desejado. A seguir são expostas estas modificações.

4.11.1 Arquivo mib.c

Foram retiradas, de todas as rotinas, a chamada de função que provocava a impressão, na tela, dos valores retornadas em uma consulta. Isto foi feito porque, o protótipo apresenta os resultados de outra maneira. Esta outra impressão apenas diminui a velocidade de execução do programa.

Foi criada uma rotina chamada *sprint_DisplayString()* para o tratamento de variáveis do tipo DisplayString. Estas variáveis eram tratadas como OCTET STRING, o que causava alguns problemas de apresentação. Para que esta nova rotina

fossse chamada adequadamente, a rotina *set_functions()* também foi modificada. Acrescentou-se a opção TYPE_DISPSTR. A rotina *print_variable()* foi substituída por *mar_print_variable()*, para que as variáveis não fossem impressas, e sim retornassem para a função chamadora.

4.11.2 Arquivo parse.c

Neste arquivo estão as funções que analisam a sintaxe dos objetos. Ocorre que o programa desconsiderava o tipo DisplayString. Considerava-o como um OCTET STRING. Entretanto, isto causava um problema de apresentação das variáveis, uma vez que objetos do tipo DisplayString eram apresentados como uma seqüência de números hexadecimais, e não como uma "string". Portanto, teve que ser acrescentado ao "parser", a interpretação do tipo DisplayString.

Foi acrescentada a constante DISPSTR e SIZE. Na estrutura *tokens*, adicionaram-se duas linhas, para que os "tokens" *DisplayString* e *SIZE* fossem aceitos. Na rotina *do_subtree()*, foi acrescentada a opção case DISPSTR. O mesmo ocorreu na rotina *parse_object_type()*.

4.11.3 Arquivo parse.h

Aqui, foi definida a constante TYPE_DISPSTR.

4.11.4 Arquivo snmp_client.c

Como última modificação, a rotina *snmp_synch_response()* teve que ser modificada, pois apresentava um pequeno erro. É função desta rotina, ao final, chamar a função *snmp_free_pdu()*, para liberar uma área de memória que é utilizada para controlar a PDU (Protocol Data Unit). Entretanto, quando uma estação não respondia uma consulta, esta função não era chamada, e a área não era liberada. Isto provocava erro na próxima consulta. Por isso, teve que ser acrescentada uma chamada da função *snmp_free_pdu()*, nesta alternativa.

5 ANÁLISE DO SISTEMA DE GERÊNCIA NO CONTEXTO OSI

Assim como pode-se fazer uma comparação, sob um aspecto mais geral, entre o modelo OSI e o Internet; deve-se compará-los especificamente em termos de gerência de configuração. Para isto, neste capítulo é feita a modelagem dos dados de acordo com o modelo OSI e são apontadas as conseqüências desta abordagem na gerência de configuração.

5.1 Objetos gerenciados

Na definição dos objetos, de acordo com o modelo OSI, é muito importante a definição da árvore de conteúdo. A árvore de conteúdo apresenta que objetos estão contidos em outros. Esta árvore é apresentada na figura 18. Esta figura apresenta somente os objetos utilizados na gerência de configuração, e tenta conservar os nomes utilizados no capítulo 3. Note-se que esta árvore não é igual à árvore definida por [WAR 89a], porque este tenta apenas adaptar os objetos OSI à árvore de conteúdo, para permitir consultas em CMIP. Enquanto a árvore aqui apresentada é uma tentativa de seguir o modelo de dados OSI.

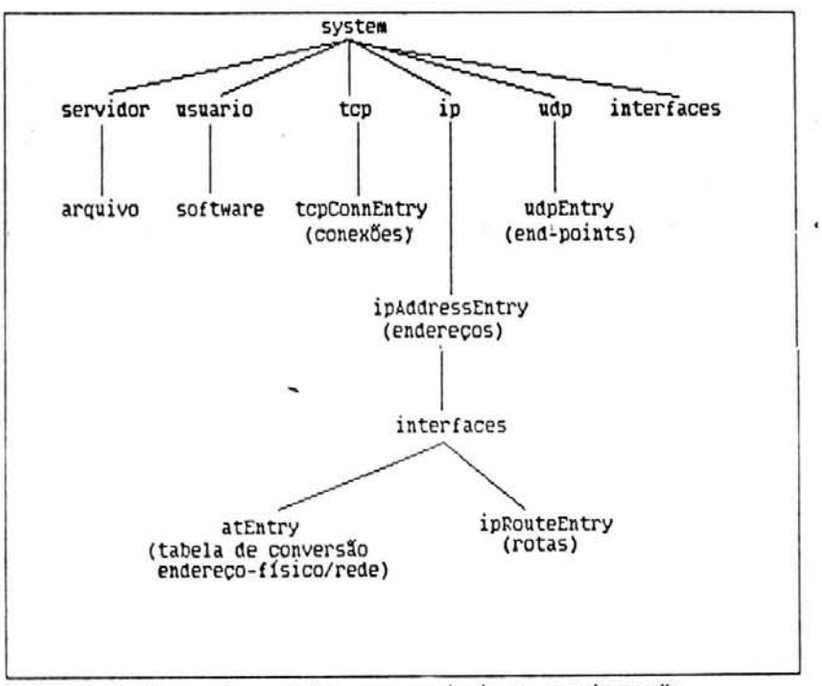

Figura 18: Árvore de conteúdo para gerência de configuração.

A árvore da figura 18 apresenta apenas os objetos e não os atributos de cada um deles. Estes são definidos mais adiante, nesta seção.

A árvore de registro, em princípio não precisaria ser modificada em relação à utilizada no capítulo 3 e no anexo A. Mas, como pode ser percebido, com a modelagem OSI, deixa de ser necessária a utilização da estrutura **Table-Entry** para definição de tabelas. Por isso, a figura 19 apresenta uma variação da árvore de registro. Esta figura é apenas parte da árvore de registro. Ela só quer mostrar a retirada dos objetos *ipAddrTable*, *ipRouteTable*, *tcpConnTable*, *udpTable* e *atTable*. Estes objetos fazem sentido apenas quando se utiliza SNMP. A árvore de conteúdo permite uma modelagem abstraindo-se o conceito de tabela e faz com que tais objetos não necessitassem mais ser identificados.

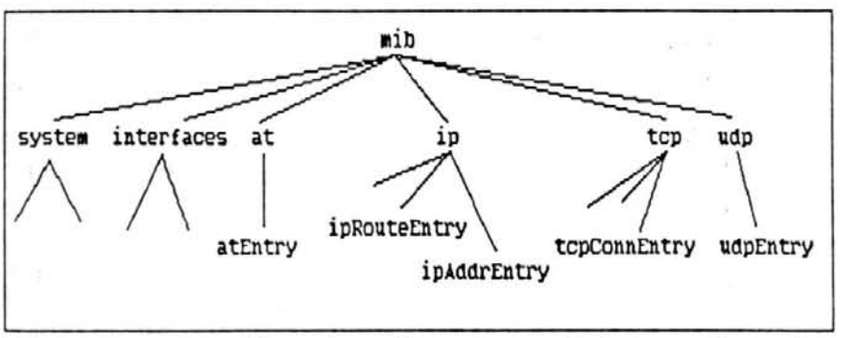

Figura 19: Parte da árvore de registro para gerência de configuração.

Em [WAR 89a] é utilizada uma estrutura definição simplificada de classe, que não permite a definição de operações e notificações, mas que permite anotar-se as árvores de conteúdo e de registro. A definição é a seguinte:

```
OBJECT-CLASS MACRO ::=
BEGIN
TYPENOTATION := SubClassOf Superiors Names Attributes
VALUE NOTATION := value(VALUE OBJECT IDENTIFIER)

SubClassOf ::= "SUBCLASS OF" value(OBJECT-CLASS)
            | empty
Superiors := "SUPERIORS" "{" SuperiorList "}"
```

```
            | empty
Names := "NAMES" "{" AttributeList "}"
            | empty
Attributes := "CONTAINS" "{" AttributeList "}"
            | empty

SuperiorList ::= Superior | Superior "," SuperiorList
Superior := value(OBJECT-CLASS)

AttributeList := Attribute | Attribute "," AttributeList
Attribute ::= value(OBJECT-TYPE)
```

NAMES é utilizado para determinar o atributo que identifica o objeto; SUPERIORS, para identificar onde está contido o objeto; CONTAINS, para anotar os atributos do objeto; e SUBCLASS-OF, para descrever a árvore de hierarquia. A árvore de registro é determinada pelo VALUE OBJECT IDENTIFIER. Os objetos gerenciados para a função de configuração estão descritos a seguir, neste formato. Os atributos de cada objeto são apresentados. A sintaxe dos atributos não é descrita , pois é a mesma do capítulo 3 e do anexo A.

```
system OBJECT CLASS
   NAMES {sysName}
  CONTAINS {
            sysDescr,
            sysDescr,
            sysObjectID,
            sysUpTime,
            sysContact,
            sysName,
            sysLocation,
            sysServices}
::= {mib 1}

ip OBJECT-CLASS
  SUPERIORS {system}
```

```
  CONTAINS {
            ipForwarding,
            ipDefaultTTL}

ipAddressEntry OBJECT-CLASS
  SUPERIORS {ip}
  NAMES {ipAdEntAddr}
  CONTAINS {
            ipAdEntAddr,
            ipAdEntNetMask,
            ipAdEntBcastAddr,
            ipAdEntReasmMaxSize}
::= {mib 4}

interfaces OBJECT-CLASS
  SUPERIORS {system, ipAddressEntry}
  NAMES {ifIndex}
  CONTAINS {
            ifIndex,
            ifDescr,
            ifType,
            ifMtu,
            ifSpeed,
            iPhysAddress,
            ifAdminStatus,
            ifOperStatus}
::= {mib 2}

atEntry OBJECT-CLASS
  SUPERIORS {interfaces}
  NAMES {atPhysAddress, atNetAddress}
  CONTAINS {
            atPhysAddress,
            atNetAddress}
::= {mib 3}
```

```
ipRouteEntry OBJECT-CLASS
  SUPEIORS {interfaces}
  NAMES {ipRouteDest}
  CONTAINS {
            ipRouteDest,
            ipRouteMetric1,
            ipRouteMetric2,
            ipRouteMetric3,
            ipRouteMetric4,
            ipRouteNextHop,
            ipRouteType,
            ipRouteProto,
            ipRouteAge,
            ipRouteMask}

tcp OBJECT-CLASS
  SUPERIORS {system}
  CONTANS {
            tcpRoutingTable,
            tcpRtoAlgorithm,
            tcpRtoMin,
            tcpRtoMax,
            tcpMaxConn,
            tcpCurrEstab}
::= {mib 5}

tcpConnEntry OBJECT-CLASS
  SUPERIORS {tcp}
  CONTAINS {
            tcpConnState,
            tcpConnLocalAddress,
            tcpConnLocalPort,
            tcpConnRemAddress,
            tcpConnRemPort}

udp OBJECT-CLASS
```

```
  SUPERIORS {system}

udpEntry OBJECT-CLASS
  SUPERIORS {udp}
  CONTAINS {
            udpLocalAddress,
            udpLocalPort}

usuario OBJECT-CLASS
  SUPERIORS {system}
  NAMES {identificacao}
  CONTAINS {
            identificacao,
            nome_Do_Usuario,
            tipo_Do_Usuario}
::= {ufrgs 1}

software OBJECT-CLASS
  SUPERIORS { usuario}
  NAMES {nomeDoProg}
  CONTAINS {
            nome_Do_Prog,
            hora_Do_Prog}
::= {ufrgs 2}

servidor OBJECT-CLASS
  SUPERIORS {system}
  NAMES {nomeDoServ}
  CONTAINS {
            nomeDoServ}
::= {ufrgs 3}

arquivo OBJECT-CLASS
  SUPERIORS {servidor}
  NAMES {arquivoServ}
  CONTAINS {
```

arquivoServ,
arquivoLocal,
acessoArq}
::= {servidor 2}

5.2 Modificações necessárias para a gerência no contexto OSI

Esta estrutura diferente leva a modificações importantes em termos de operações de gerência.

A primeira modificação importante é a possibilidade de recuperar dados agregados. Não é necessário recuperar atributo por atributo, pode-se buscar um objeto complexo com uma só operação. Assim, todas as funções que apresentam uma repetição de operações poderiam ser simplificadas. Por exemplo: na recuperação da tabela *ifTable*, não é mais necessário realizar-se uma seqüência de operações *get-next*. Basta recuperar todas as intâncias do objeto interfaces de um determinado *system* ou de um determinado *ipAddressEntry*.

Em segundo lugar a organização dos objetos simplifica o gerente, pois tira deste algumas funções de organização dos dados. Por exemplo: para a função de controle da ligação dos dispositivos à rede é necesário concatenar as tabelas *ifTable* e *ipAddrTable*, quando se utiliza SNMP. Com OSI, isto não é mais necessário, pois estes objetos já estão relacionados na árvore de conteúdo, e uma consulta traz embutida, em sua estrutura, este relacionamento.

Por último, o modelo OSI facilita muito a gerência de objetos que devem ser criados e removidos. Apesar de estas funções não terem sido implementadas no protótipo, elas são muito importantes na gerência de configuração, já que a configuração deve ser alterada. A remoção de um usuário através da utilização de um atributo de uma tabela não é uma solução natural. As operações DELETE e CREATE são muito mais elegantes. Além disso, a possibilidade de controlar o estado operacional e administrativo de um objeto já está embutida no modelo OSI, eliminando a necessidade de declarar-se variáveis como *ifOperStatus* e *ifAdmStatus*

Sem dúvida nenhuma, o modelo OSI é uma ferramenta muito mais potente para gerência de redes do que o Internet. Facilita a criação de funções muito mais complexas, com o uso, inclusive, de filtros para a realização de consultas. A gerência de configuração tende a ser facilitada com o seu uso. Assim, a migração para o modelo OSI tornar-se-á uma necessidade, à medida em que houver uma demanda por funções mais complexas.

6 CONCLUSÕES

Este trabalho mostra que, já tendo definidos os modelo organizacional e funcional, o passo principal para o planejamento e a execução da gerência de configuração, e de outros aspectos, de uma rede é a definição dos objetos gerenciados. E esta definição não é uma atividade simples, porque dela depende todo o sucesso da gerência. É necessário que se utilize uma metodologia para que os objetos gerenciados sejam definidos, e, depois de um tempo, não se chegue a conclusão de que faltam objetos ou que existem objetos sem função. Neste trabalho os objetos foram definidos somente após a definição das funções de gerência de configuração pretendidas.

Também percebe-se que os objetos definidos no padrão Internet são insuficientes. Logicamente estes objetos foram definidos para gerenciar recursos que estão no âmbito dos protocolos Internet, mas se não for possível gerenciar funções de mais alto nível, dentro de uma rede, como usuários, programas e arquivos, o administrador de rede continuará a ter que fazer boa parte das suas funções manualmente.

Mas, não basta um conjunto de objetos bem definidos. É imprescindível que haja ferramentas que tornem estes objetos úteis ao administrador humano. Não é suficiente que este percorra os diversos objetos da MIB. Ele deve poder desconhecer nomes e sintaxe dos objetos. É a interface que torna a ferramenta útil, ou não, para o usuário. Navegadores, que apenas permitem que o usuário percorra os objetos da MIB, são ferramentas ineficientes, pois exigem que o usuário conheça o significado de cada objeto e que os relacione mentalmente. Neste trabalho, a solução proposta é a definição de funções. O administrador não consulta uma variável, ele escolhe uma função.

Também é possível perceber que o modelo OSI é muito mais poderoso, pelo menos em termos de gerência de configuração. A modelagem dos dados permite a definição de estruturas mais próximas do mundo real e facilitam a implementação das funções de gerência de configuração. O grande problema que surge é a complexidade para se implementar agentes OSI em um dispositivo de menor porte. O futuro talvez seja uma combinação entre os dois esquemas; ou, pelo menos, a utilização de um padrão com uma versão reduzida para estes dispositivos.

O protótipo apresenta alguns aspectos da construção da gerência de configuração em uma rede, mais especificamente em uma rede local de uma Universidade. O protótipo pode servir como uma base para o desenvolvimento de uma ferramenta realmente eficiente. Existem quatro áreas que podem ser desenvolvidas a partir dele, em termos de gerência de configuração: funções não implementadas, "traps", banco de dados e gráficos. As funções que ficaram pendentes são muito úteis para o usuário, e devem merecer atenção. Principalmente a descoberta dos dispositivos da rede, que parece ser uma função muito desejada por usuários. Em relação aos "traps", é importante que o gerente consiga interpretá-los, pois eles podem diminuir o número de consultas que o gerente faz aos agentes e diminuir a carga da rede. Novos "traps" devem poder ser definidos, para novas funções, sem necessidade de grandes alterações. O esquema possivelmente deve basear-se em uma função geral de tratamento de traps, que invoque as funções específicas. Cabe pesquisar qual a melhor maneira de relacionar a primeira função com as outras, de forma que o usuário possa definir novos traps , e novas funções, sem dificuldade. Quanto ao banco de dados, o trabalho de definição dos outros dados que usuário tem necessidade de guardar é necessário. Estes dados devem ser modelados e implementados. Ainda, é fundamental, para que o protótipo se transforme em ferramenta útil, que seja construída uma interface com mapas, alarmes, sinais, etc.

Por último, este trabalho permite que se perceba as possibilidades e as dificuldades existentes em um projeto de gerência de redes. Especificamente, introduz o problema da gerência de configuração. Pode servir como um exemplo, ou mesmo uma base, para o desenvolvimento de sistemas mais complexos que solucionem o problema da gerência de rede.

ANEXO A

```
RFC1213-MIB DEFINITIONS ::= BEGIN

IMPORTS
      mgmt, NetworkAddress, IpAddress, Counter, Gauge,
          TimeTicks
        FROM RFC1155-SMI
      OBJECT-TYPE
          FROM RFC-1212;

-- This MIB module uses the extended OBJECT-TYPE macro as
-- defined in [14];


-- MIB-II (same prefix as MIB-I)

mib-2     OBJECT IDENTIFIER ::= { mgmt 1 }

-- textual conventions

DisplayString ::=
   OCTET STRING
-- This data type is used to model textual information taken
-- from the NVT ASCII character set.  By convention, objects
-- with this syntax are declared as having
--
--     SIZE (0..255)

PhysAddress ::=
   OCTET STRING
-- This data type is used to model media addresses.  For many
-- types of media, this will be in a binary representation.
-- For example, an ethernet address would be represented as
-- a string of 6 octets.


-- groups in MIB-II

system      OBJECT IDENTIFIER ::= { mib-2 1 }

interfaces  OBJECT IDENTIFIER ::= { mib-2 2 }

at          OBJECT IDENTIFIER ::= { mib-2 3 }

ip          OBJECT IDENTIFIER ::= { mib-2 4 }
```

```
icmp       OBJECT IDENTIFIER ::= { mib-2 5 }

tcp        OBJECT IDENTIFIER ::= { mib-2 6 }

udp        OBJECT IDENTIFIER ::= { mib-2 7 }

egp        OBJECT IDENTIFIER ::= { mib-2 8 }

-- historical (some say hysterical)
-- cmot     OBJECT IDENTIFIER ::= { mib-2 9 }

transmission OBJECT IDENTIFIER ::= { mib-2 10 }

snmp       OBJECT IDENTIFIER ::= { mib-2 11 }


-- the System group

-- Implementation of the System group is mandatory for all
-- systems. If an agent is not configured to have a value
-- for any of these variables, a string of length 0 is
-- returned.

sysDescr OBJECT-TYPE
   SYNTAX  DisplayString (SIZE (0..255))
   ACCESS  read-only
   STATUS  mandatory
   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."
   ::= { system 1 }

sysObjectID OBJECT-TYPE
   SYNTAX  OBJECT IDENTIFIER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The vendor's authoritative identification of the
         network management subsystem contained in the
         entity.  This value is allocated within the SMI
         enterprises subtree (1.3.6.1.4.1) and provides an
         easy and unambiguous means for determining `what
```

```
            kind of box' is being managed.  For example, if
            vendor `Flintstones, Inc.' was assigned the
            subtree 1.3.6.1.4.1.4242, it could assign the
            identifier 1.3.6.1.4.1.4242.1.1 to its `Fred
            Router'."
        ::= { system 2 }

    sysUpTime OBJECT-TYPE
        SYNTAX  TimeTicks
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "The time (in hundredths of a second) since the
            network management portion of the system was last
            re-initialized."
        ::= { system 3 }

    sysContact OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
            "The textual identification of the contact person
            for this managed node, together with information
            on how to contact this person."
        ::= { system 4 }

    sysName OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
            "An administratively-assigned name for this
            managed node.  By convention, this is the node's
            fully-qualified domain name."
        ::= { system 5 }

    sysLocation OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
            "The physical location of this node (e.g.,
            `telephone closet, 3rd floor')."
        ::= { system 6 }

    sysServices OBJECT-TYPE
```

```
    SYNTAX  INTEGER (0..127)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A value which indicates the set of services that
            this entity primarily offers.

            The value is a sum.  This sum initially takes the
            value zero, Then, for each layer, L, in the range
            1 through 7, that this node performs transactions
            for, 2 raised to (L - 1) is added to the sum.  For
            example, a node which performs primarily routing
            functions would have a value of 4 (2^(3-1)).  In
            contrast, a node which is a host offering
            application services would have a value of 72
            (2^(4-1) + 2^(7-1)).  Note that in the context of
            the Internet suite of protocols, values should be
            calculated accordingly:

                layer  functionality
                  1  physical (e.g., repeaters)
                  2  datalink/subnetwork (e.g., bridges)
                  3  internet (e.g., IP gateways)
                  4  end-to-end  (e.g., IP hosts)
                  7  applications (e.g., mail relays)

            For systems including OSI protocols, layers 5 and
            6 may also be counted."
    ::= { system 7 }

-- the Interfaces group

-- Implementation of the Interfaces group is mandatory for
-- all systems.

ifNumber OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of network interfaces (regardless of
            their current state) present on this system."
    ::= { interfaces 1 }


-- the Interfaces table
```

```
-- The Interfaces table contains information on the entity's
-- interfaces.  Each interface is thought of as being
-- attached to a `subnetwork'.  Note that this term should
-- not be confused with `subnet' which refers to an
-- addressing partitioning scheme used in the Internet suite
-- of protocols.

ifTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF IfEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "A list of interface entries.  The number of
         entries is given by the value of ifNumber."
   ::= { interfaces 2 }

ifEntry OBJECT-TYPE
   SYNTAX  IfEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "An interface entry containing objects at the
         subnetwork layer and below for a particular
         interface."
   INDEX   { ifIndex }
   ::= { ifTable 1 }

IfEntry ::=
   SEQUENCE {
      ifIndex
         INTEGER,
      ifDescr
         DisplayString,
      ifType
         INTEGER,
      ifMtu
         INTEGER,
      ifSpeed
         Gauge,
      ifPhysAddress
         PhysAddress,
      ifAdminStatus
         INTEGER,
      ifOperStatus
         INTEGER,
      ifLastChange
         TimeTicks,
```

```
    ifInOctets
      Counter,
    ifInUcastPkts
      Counter,
    ifInNUcastPkts
      Counter,
    ifInDiscards
      Counter,
    ifInErrors
      Counter,
    ifInUnknownProtos
      Counter,
    ifOutOctets
      Counter,
    ifOutUcastPkts
      Counter,
    ifOutNUcastPkts
      Counter,
    ifOutDiscards
      Counter,
    ifOutErrors
      Counter,
    ifOutQLen
      Gauge,
    ifSpecific
      OBJECT IDENTIFIER
  }

ifIndex OBJECT-TYPE
  SYNTAX  INTEGER
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
      "A unique value for each interface.  Its value
      ranges between 1 and the value of ifNumber.  The
      value for each interface must remain constant at
      least from one re-initialization of the entity's
      network management system to the next re-
      initialization."
  ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
  SYNTAX  DisplayString (SIZE (0..255))
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
      "A textual string containing information about the
```

```
        interface.  This string should include the name of
        the manufacturer, the product name and the version
        of the hardware interface."
    ::= { ifEntry 2 }

ifType OBJECT-TYPE
    SYNTAX  INTEGER {
                other(1),       -- none of the following
                regular1822(2),
                hdh1822(3),
                ddn-x25(4),
                rfc877-x25(5),
                ethernet-csmacd(6),
                iso88023-csmacd(7),
                iso88024-tokenBus(8),
                iso88025-tokenRing(9),
                iso88026-man(10),
                starLan(11),
                proteon-10Mbit(12),
                proteon-80Mbit(13),
                hyperchannel(14),
                fddi(15),
                lapb(16),
                sdlc(17),
                ds1(18),            -- T-1
                e1(19),             -- european equiv. of T-1
                basicISDN(20),
                primaryISDN(21),    -- proprietary serial
                propPointToPointSerial(22),
                ppp(23),
                softwareLoopback(24),
                eon(25),            -- CLNP over IP [11]
                ethernet-3Mbit(26),
                nsip(27),           -- XNS over IP
                slip(28),           -- generic SLIP
                ultra(29),          -- ULTRA technologies
                ds3(30),            -- T-3
                sip(31),            -- SMDS
                frame-relay(32)
            }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The type of interface, distinguished according to
        the physical/link protocol(s) immediately `below'
        the network layer in the protocol stack."
    ::= { ifEntry 3 }
```

```
ifMtu OBJECT-TYPE
  SYNTAX INTEGER
  ACCESS read-only
  STATUS mandatory
  DESCRIPTION
      "The size of the largest datagram which can be
      sent/received on the interface, specified in
      octets. For interfaces that are used for
      transmitting network datagrams, this is the size
      of the largest network datagram that can be sent
      on the interface."
  ::= { ifEntry 4 }

ifSpeed OBJECT-TYPE
  SYNTAX Gauge
  ACCESS read-only
  STATUS mandatory
  DESCRIPTION
      "An estimate of the interface's current bandwidth
      in bits per second. For interfaces which do not
      vary in bandwidth or for those where no accurate
      estimation can be made, this object should contain
      the nominal bandwidth."
  ::= { ifEntry 5 }

ifPhysAddress OBJECT-TYPE
  SYNTAX PhysAddress
  ACCESS read-only
  STATUS mandatory
  DESCRIPTION
      "The interface's address at the protocol layer
      immediately `below' the network layer in the
      protocol stack. For interfaces which do not have
      such an address (e.g., a serial line), this object
      should contain an octet string of zero length."
  ::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
  SYNTAX INTEGER {
          up(1),      -- ready to pass packets
          down(2),
          testing(3)  -- in some test mode
        }
  ACCESS read-write
  STATUS mandatory
  DESCRIPTION
```

```
        "The desired state of the interface.  The
        testing(3) state indicates that no operational
        packets can be passed."
    ::= { ifEntry 7 }

ifOperStatus OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),      -- ready to pass packets
                down(2),
                testing(3)  -- in some test mode
            }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The current operational state of the interface.
        The testing(3) state indicates that no operational
        packets can be passed."
    ::= { ifEntry 8 }

ifLastChange OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The value of sysUpTime at the time the interface
        entered its current operational state.  If the
        current state was entered prior to the last re-
        initialization of the local network management
        subsystem, then this object contains a zero
        value."
    ::= { ifEntry 9 }

ifInOctets OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of octets received on the
        interface, including framing characters."
    ::= { ifEntry 10 }

ifInUcastPkts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of subnetwork-unicast packets
```

```
        delivered to a higher-layer protocol."
    ::= { ifEntry 11 }

ifInNUcastPkts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of non-unicast (i.e., subnetwork-
        broadcast or subnetwork-multicast) packets
        delivered to a higher-layer protocol."
    ::= { ifEntry 12 }

ifInDiscards OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of inbound packets which were chosen
        to be discarded even though no errors had been
        detected to prevent their being deliverable to a
        higher-layer protocol.  One possible reason for
        discarding such a packet could be to free up
        buffer space."
    ::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of inbound packets that contained
        errors preventing them from being deliverable to a
        higher-layer protocol."
    ::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of packets received via the interface
        which were discarded because of an unknown or
        unsupported protocol."
    ::= { ifEntry 15 }

ifOutOctets OBJECT-TYPE
```

```
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of octets transmitted out of the
        interface, including framing characters."
    ::= { ifEntry 16 }

ifOutUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of packets that higher-level
        protocols requested be transmitted to a
        subnetwork-unicast address, including those that
        were discarded or not sent."
    ::= { ifEntry 17 }

ifOutNUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of packets that higher-level
        protocols requested be transmitted to a non-
        unicast (i.e., a subnetwork-broadcast or
        subnetwork-multicast) address, including those
        that were discarded or not sent."
    ::= { ifEntry 18 }

ifOutDiscards OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of outbound packets which were chosen
        to be discarded even though no errors had been
        detected to prevent their being transmitted. One
        possible reason for discarding such a packet could
        be to free up buffer space."
    ::= { ifEntry 19 }

ifOutErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
```

```
    DESCRIPTION
        "The number of outbound packets that could not be
        transmitted because of errors."
    ::= { ifEntry 20 }

ifOutQLen OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The length of the output packet queue (in
        packets)."
    ::= { ifEntry 21 }

ifSpecific OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "A reference to MIB definitions specific to the
        particular media being used to realize the
        interface. For example, if the interface is
        realized by an ethernet, then the value of this
        object refers to a document defining objects
        specific to ethernet. If this information is not
        present, its value should be set to the OBJECT
        IDENTIFIER { 0 0 }, which is a syntatically valid
        object identifier, and any conformant
        implementation of ASN.1 and BER must be able to
        generate and recognize this value."
    ::= { ifEntry 22 }


-- the Address Translation group

-- Implementation of the Address Translation group is
-- mandatory for all systems. Note however that this group
-- is deprecated by MIB-II. That is, it is being included
-- solely for compatibility with MIB-I nodes, and will most
-- likely be excluded from MIB-III nodes. From MIB-II and
-- onwards, each network protocol group contains its own
-- address translation tables.

-- The Address Translation group contains one table which is
-- the union across all interfaces of the translation tables
-- for converting a NetworkAddress (e.g., an IP address) into
-- a subnetwork-specific address. For lack of a better term,
```

```
-- this document refers to such a subnetwork-specific address
-- as a `physical' address.

-- Examples of such translation tables are: for broadcast
-- media where ARP is in use, the translation table is
-- equivalent to the ARP cache; or, on an X.25 network where
-- non-algorithmic translation to X.121 addresses is
-- required, the translation table contains the
-- NetworkAddress to X.121 address equivalences.

atTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF AtEntry
   ACCESS  not-accessible
   STATUS  deprecated
   DESCRIPTION
         "The Address Translation tables contain the
         NetworkAddress to `physical' address equivalences.
         Some interfaces do not use translation tables for
         determining address equivalences (e.g., DDN-X.25
         has an algorithmic method); if all interfaces are
         of this type, then the Address Translation table
         is empty, i.e., has zero entries."
   ::= { at 1 }

atEntry OBJECT-TYPE
   SYNTAX  AtEntry
   ACCESS  not-accessible
   STATUS  deprecated
   DESCRIPTION
         "Each entry contains one NetworkAddress to
         `physical' address equivalence."
   INDEX   { atIfIndex,
             atNetAddress }
   ::= { atTable 1 }

AtEntry ::=
   SEQUENCE {
      atIfIndex
         INTEGER,
      atPhysAddress
         PhysAddress,
      atNetAddress
         NetworkAddress
   }

atIfIndex OBJECT-TYPE
   SYNTAX  INTEGER
```

```
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
            "The interface on which this entry's equivalence
            is effective.  The interface identified by a
            particular value of this index is the same
            interface as identified by the same value of
            ifIndex."
    ::= { atEntry 1 }

atPhysAddress OBJECT-TYPE
    SYNTAX  PhysAddress
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
            "The media-dependent `physical' address.

            Setting this object to a null string (one of zero
            length) has the effect of invaliding the
            corresponding entry in the atTable object.  That
            is, it effectively dissasociates the interface
            identified with said entry from the mapping
            identified with said entry.  It is an
            implementation-specific matter as to whether the
            agent removes an invalidated entry from the table.
            Accordingly, management stations must be prepared
            to receive tabular information from agents that
            corresponds to entries not currently in use.
            Proper interpretation of such entries requires
            examination of the relevant atPhysAddress object."
    ::= { atEntry 2 }

atNetAddress OBJECT-TYPE
    SYNTAX  NetworkAddress
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
            "The NetworkAddress (e.g., the IP address)
            corresponding to the media-dependent `physical'
            address."
    ::= { atEntry 3 }


-- the IP group

-- Implementation of the IP group is mandatory for all
-- systems.
```

```
ipForwarding OBJECT-TYPE
  SYNTAX  INTEGER {
          forwarding(1),   -- acting as a gateway
          not-forwarding(2) -- NOT acting as a gateway
        }
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
        "The indication of whether this entity is acting
        as an IP gateway in respect to the forwarding of
        datagrams received by, but not addressed to, this
        entity.  IP gateways forward datagrams.  IP hosts
        do not (except those source-routed via the host).

        Note that for some managed nodes, this object may
        take on only a subset of the values possible.
        Accordingly, it is appropriate for an agent to
        return a `badValue' response if a management
        station attempts to change this object to an
        inappropriate value."
  ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
  SYNTAX  INTEGER
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
        "The default value inserted into the Time-To-Live
        field of the IP header of datagrams originated at
        this entity, whenever a TTL value is not supplied
        by the transport layer protocol."
  ::= { ip 2 }

ipInReceives OBJECT-TYPE
  SYNTAX  Counter
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
        "The total number of input datagrams received from
        interfaces, including those received in error."
  ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
  SYNTAX  Counter
  ACCESS  read-only
  STATUS  mandatory
```

```
    DESCRIPTION
        "The number of input datagrams discarded due to
        errors in their IP headers, including bad
        checksums, version number mismatch, other format
        errors, time-to-live exceeded, errors discovered
        in processing their IP options, etc."
    ::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of input datagrams discarded because
        the IP address in their IP header's destination
        field was not a valid address to be received at
        this entity. This count includes invalid
        addresses (e.g., 0.0.0.0) and addresses of
        unsupported Classes (e.g., Class E). For entities
        which are not IP Gateways and therefore do not
        forward datagrams, this counter includes datagrams
        discarded because the destination address was not
        a local address."
    ::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of input datagrams for which this
        entity was not their final IP destination, as a
        result of which an attempt was made to find a
        route to forward them to that final destination.
        In entities which do not act as IP Gateways, this
        counter will include only those packets which were
        Source-Routed via this entity, and the Source-
        Route option processing was successful."
    ::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of locally-addressed datagrams
        received successfully but discarded because of an
```

```
        unknown or unsupported protocol."
    ::= { ip 7 }

ipInDiscards OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of input IP datagrams for which no
        problems were encountered to prevent their
        continued processing, but which were discarded
        (e.g., for lack of buffer space).  Note that this
        counter does not include any datagrams discarded
        while awaiting re-assembly."
    ::= { ip 8 }

ipInDelivers OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of input datagrams successfully
        delivered to IP user-protocols (including ICMP)."
    ::= { ip 9 }

ipOutRequests OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of IP datagrams which local IP
        user-protocols (including ICMP) supplied to IP in
        requests for transmission.  Note that this counter
        does not include any datagrams counted in
        ipForwDatagrams."
    ::= { ip 10 }

ipOutDiscards OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of output IP datagrams for which no
        problem was encountered to prevent their
        transmission to their destination, but which were
        discarded (e.g., for lack of buffer space).  Note
        that this counter would include datagrams counted
```

```
        in ipForwDatagrams if any such packets met this
        (discretionary) discard criterion."
    ::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of IP datagrams discarded because no
        route could be found to transmit them to their
        destination.  Note that this counter includes any
        packets counted in ipForwDatagrams which meet this
        `no-route' criterion.  Note that this includes any
        datagarms which a host cannot route because all of
        its default gateways are down."
    ::= { ip 12 }

ipReasmTimeout OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The maximum number of seconds which received
        fragments are held while they are awaiting
        reassembly at this entity."
    ::= { ip 13 }

ipReasmReqds OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of IP fragments received which needed
        to be reassembled at this entity."
    ::= { ip 14 }

ipReasmOKs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of IP datagrams successfully re-
        assembled."
    ::= { ip 15 }

ipReasmFails OBJECT-TYPE
```

```
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of failures detected by the IP re-
        assembly algorithm (for whatever reason: timed
        out, errors, etc). Note that this is not
        necessarily a count of discarded IP fragments
        since some algorithms (notably the algorithm in
        RFC 815) can lose track of the number of fragments
        by combining them as they are received."
    ::= { ip 16 }

ipFragOKs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of IP datagrams that have been
        successfully fragmented at this entity."
    ::= { ip 17 }

ipFragFails OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of IP datagrams that have been
        discarded because they needed to be fragmented at
        this entity but could not be, e.g., because their
        Don't Fragment flag was set."
    ::= { ip 18 }

ipFragCreates OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of IP datagram fragments that have
        been generated as a result of fragmentation at
        this entity."
    ::= { ip 19 }


-- the IP address table

-- The IP address table contains this entity's IP addressing
```

```
-- information.

ipAddrTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF IpAddrEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "The table of addressing information relevant to
         this entity's IP addresses."
   ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
   SYNTAX  IpAddrEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "The addressing information for one of this
         entity's IP addresses."
   INDEX   { ipAdEntAddr }
   ::= { ipAddrTable 1 }

IpAddrEntry ::=
   SEQUENCE {
      ipAdEntAddr
         IpAddress,
      ipAdEntIfIndex
         INTEGER,
      ipAdEntNetMask
         IpAddress,
      ipAdEntBcastAddr
         INTEGER,
      ipAdEntReasmMaxSize
         INTEGER (0..65535)
   }

ipAdEntAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The IP address to which this entry's addressing
         information pertains."
   ::= { ipAddrEntry 1 }

ipAdEntIfIndex OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-only
```

```
    STATUS  mandatory
    DESCRIPTION
            "The index value which uniquely identifies the
            interface to which this entry is applicable. The
            interface identified by a particular value of this
            index is the same interface as identified by the
            same value of ifIndex."
    ::= { ipAddrEntry 2 }

ipAdEntNetMask OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The subnet mask associated with the IP address of
            this entry.  The value of the mask is an IP
            address with all the network bits set to 1 and all
            the hosts bits set to 0."
    ::= { ipAddrEntry 3 }

ipAdEntBcastAddr OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The value of the least-significant bit in the IP
            broadcast address used for sending datagrams on
            the (logical) interface associated with the IP
            address of this entry.  For example, when the
            Internet standard all-ones broadcast address is
            used, the value will be 1.  This value applies to
            both the subnet and network broadcasts addresses
            used by the entity on this (logical) interface."
    ::= { ipAddrEntry 4 }

ipAdEntReasmMaxSize OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The size of the largest IP datagram which this
            entity can re-assemble from incoming IP fragmented
            datagrams received on this interface."
    ::= { ipAddrEntry 5 }
```

```
-- the IP routing table

-- The IP routing table contains an entry for each route
-- presently known to this entity.

ipRouteTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF IpRouteEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "This entity's IP Routing table."
   ::= { ip 21 }

ipRouteEntry OBJECT-TYPE
   SYNTAX  IpRouteEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "A route to a particular destination."
   INDEX   { ipRouteDest }
   ::= { ipRouteTable 1 }

IpRouteEntry ::=
   SEQUENCE {
      ipRouteDest
         IpAddress,
      ipRouteIfIndex
         INTEGER,
      ipRouteMetric1
         INTEGER,
      ipRouteMetric2
         INTEGER,
      ipRouteMetric3
         INTEGER,
      ipRouteMetric4
         INTEGER,
      ipRouteNextHop
         IpAddress,
      ipRouteType
         INTEGER,
      ipRouteProto
         INTEGER,
      ipRouteAge
         INTEGER,
      ipRouteMask
         IpAddress,
      ipRouteMetric5
```

```
        INTEGER,
      ipRouteInfo
        OBJECT IDENTIFIER
    }

ipRouteDest OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The destination IP address of this route.  An
        entry with a value of 0.0.0.0 is considered a
        default route.  Multiple routes to a single
        destination can appear in the table, but access to
        such multiple entries is dependent on the table-
        access mechanisms defined by the network
        management protocol in use."
    ::= { ipRouteEntry 1 }

ipRouteIfIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The index value which uniquely identifies the
        local interface through which the next hop of this
        route should be reached.  The interface identified
        by a particular value of this index is the same
        interface as identified by the same value of
        ifIndex."
    ::= { ipRouteEntry 2 }

ipRouteMetric1 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The primary routing metric for this route.  The
        semantics of this metric are determined by the
        routing-protocol specified in the route's
        ipRouteProto value.  If this metric is not used,
        its value should be set to -1."
    ::= { ipRouteEntry 3 }

ipRouteMetric2 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
```

```
    STATUS  mandatory
    DESCRIPTION
            "An alternate routing metric for this route.  The
            semantics of this metric are determined by the
            routing-protocol specified in the route's
            ipRouteProto value.  If this metric is not used,
            its value should be set to -1."
    ::= { ipRouteEntry 4 }

ipRouteMetric3 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "An alternate routing metric for this route.  The
            semantics of this metric are determined by the
            routing-protocol specified in the route's
            ipRouteProto value.  If this metric is not used,
            its value should be set to -1."
    ::= { ipRouteEntry 5 }

ipRouteMetric4 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "An alternate routing metric for this route.  The
            semantics of this metric are determined by the
            routing-protocol specified in the route's
            ipRouteProto value.  If this metric is not used,
            its value should be set to -1."
    ::= { ipRouteEntry 6 }

ipRouteNextHop OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The IP address of the next hop of this route.
            (In the case of a route bound to an interface
            which is realized via a broadcast media, the value
            of this field is the agent's IP address on that
            interface.)"
    ::= { ipRouteEntry 7 }

ipRouteType OBJECT-TYPE
    SYNTAX  INTEGER {
```

```
            other(1),       -- none of the following

            invalid(2),     -- an invalidated route
                            -- route to directly
            direct(3),      -- connected (sub-)network
                            -- route to a non-local
            indirect(4)     -- host/network/sub-network
        }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The type of route.  Note that the values
        direct(3) and indirect(4) refer to the notion of
        direct and indirect routing in the IP
        architecture.

        Setting this object to the value invalid(2) has
        the effect of invalidating the corresponding entry
        in the ipRouteTable object.  That is, it
        effectively dissasociates the destination
        identified with said entry from the route
        identified with said entry.  It is an
        implementation-specific matter as to whether the
        agent removes an invalidated entry from the table.
        Accordingly, management stations must be prepared
        to receive tabular information from agents that
        corresponds to entries not currently in use.
        Proper interpretation of such entries requires
        examination of the relevant ipRouteType object."
    ::= { ipRouteEntry 8 }

ipRouteProto OBJECT-TYPE
    SYNTAX  INTEGER {
            other(1),       -- none of the following

                            -- non-protocol information,
                            -- e.g., manually configured
            local(2),       -- entries

                            -- set via a network
            netmgmt(3),     -- management protocol

                            -- obtained via ICMP,
            icmp(4),        -- e.g., Redirect

                            -- the remaining values are
                            -- all gateway routing
```

```
                -- protocols
            egp(5),
            ggp(6),
            hello(7),
            rip(8),
            is-is(9),
            es-is(10),
            ciscoIgrp(11),
            bbnSpfIgp(12),
            ospf(13),
            bgp(14)
        }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The routing mechanism via which this route was
        learned.  Inclusion of values for gateway routing
        protocols is not intended to imply that hosts
        should support those protocols."
    ::= { ipRouteEntry 9 }

ipRouteAge OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "The number of seconds since this route was last
        updated or otherwise determined to be correct.
        Note that no semantics of `too old' can be implied
        except through knowledge of the routing protocol
        by which the route was learned."
    ::= { ipRouteEntry 10 }

ipRouteMask OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "Indicate the mask to be logical-ANDed with the
        destination address before being compared to the
        value in the ipRouteDest field.  For those systems
        that do not support arbitrary subnet masks, an
        agent constructs the value of the ipRouteMask by
        determining whether the value of the correspondent
        ipRouteDest field belong to a class-A, B, or C
        network, and then using one of:
```

```
            mask          network
            255.0.0.0     class-A
            255.255.0.0   class-B
            255.255.255.0 class-C
        If the value of the ipRouteDest is 0.0.0.0 (a
        default route), then the mask value is also
        0.0.0.0. It should be noted that all IP routing
        subsystems implicitly use this mechanism."
    ::= { ipRouteEntry 11 }

ipRouteMetric5 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "An alternate routing metric for this route. The
        semantics of this metric are determined by the
        routing-protocol specified in the route's
        ipRouteProto value. If this metric is not used,
        its value should be set to -1."
    ::= { ipRouteEntry 12 }

ipRouteInfo OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A reference to MIB definitions specific to the
        particular routing protocol which is responsible
        for this route, as determined by the value
        specified in the route's ipRouteProto value. If
        this information is not present, its value should
        be set to the OBJECT IDENTIFIER { 0 0 }, which is
        a syntatically valid object identifier, and any
        conformant implementation of ASN.1 and BER must be
        able to generate and recognize this value."
    ::= { ipRouteEntry 13 }


-- the IP Address Translation table

-- The IP address translation table contain the IpAddress to
-- `physical' address equivalences. Some interfaces do not
-- use translation tables for determining address
-- equivalences (e.g., DDN-X.25 has an algorithmic method);
-- if all interfaces are of this type, then the Address
-- Translation table is empty, i.e., has zero entries.
```

```
ipNetToMediaTable OBJECT-TYPE
  SYNTAX  SEQUENCE OF IpNetToMediaEntry
  ACCESS  not-accessible
  STATUS  mandatory
  DESCRIPTION
      "The IP Address Translation table used for mapping
      from IP addresses to physical addresses."
  ::= { ip 22 }

ipNetToMediaEntry OBJECT-TYPE
  SYNTAX  IpNetToMediaEntry
  ACCESS  not-accessible
  STATUS  mandatory
  DESCRIPTION
      "Each entry contains one IpAddress to `physical'
      address equivalence."
  INDEX   { ipNetToMediaIfIndex,
        ipNetToMediaNetAddress }
  ::= { ipNetToMediaTable 1 }

IpNetToMediaEntry ::=
  SEQUENCE {
    ipNetToMediaIfIndex
      INTEGER,
    ipNetToMediaPhysAddress
      PhysAddress,
    ipNetToMediaNetAddress
      IpAddress,
    ipNetToMediaType
      INTEGER
  }

ipNetToMediaIfIndex OBJECT-TYPE
  SYNTAX  INTEGER
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
      "The interface on which this entry's equivalence
      is effective.  The interface identified by a
      particular value of this index is the same
      interface as identified by the same value of
      ifIndex."
  ::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
  SYNTAX  PhysAddress
```

```
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The media-dependent `physical' address."
    ::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The IpAddress corresponding to the media-
            dependent `physical' address."
    ::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
    SYNTAX  INTEGER {
                other(1),       -- none of the following
                invalid(2),     -- an invalidated mapping
                dynamic(3),
                static(4)
            }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The type of mapping.

            Setting this object to the value invalid(2) has
            the effect of invalidating the corresponding entry
            in the ipNetToMediaTable.  That is, it effectively
            dissasociates the interface identified with said
            entry from the mapping identified with said entry.
            It is an implementation-specific matter as to
            whether the agent removes an invalidated entry
            from the table.  Accordingly, management stations
            must be prepared to receive tabular information
            from agents that corresponds to entries not
            currently in use.  Proper interpretation of such
            entries requires examination of the relevant
            ipNetToMediaType object."
    ::= { ipNetToMediaEntry 4 }


-- additional IP objects

ipRoutingDiscards OBJECT-TYPE
    SYNTAX  Counter
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of routing entries which were chosen
        to be discarded even though they are valid. One
        possible reason for discarding such an entry could
        be to free-up buffer space for other routing
        entries."
    ::= { ip 23 }


-- the ICMP group

-- Implementation of the ICMP group is mandatory for all
-- systems.

icmpInMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of ICMP messages which the
        entity received. Note that this counter includes
        all those counted by icmpInErrors."
    ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP messages which the entity
        received but determined as having ICMP-specific
        errors (bad ICMP checksums, bad length, etc.)."
    ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Destination Unreachable
        messages received."
    ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
    SYNTAX Counter
```

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Time Exceeded messages
        received."
    ::= { icmp 4 }

icmpInParmProbs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Parameter Problem messages
        received."
    ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Source Quench messages
        received."
    ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Redirect messages received."
    ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Echo (request) messages
        received."
    ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
```

```
        "The number of ICMP Echo Reply messages received."
    ::= { icmp 9 }

icmpInTimestamps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Timestamp (request) messages
        received."
    ::= { icmp 10 }

icmpInTimestampReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Timestamp Reply messages
        received."
    ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Address Mask Request messages
        received."
    ::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of ICMP Address Mask Reply messages
        received."
    ::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of ICMP messages which this
        entity attempted to send. Note that this counter
        includes all those counted by icmpOutErrors."
```

```
    ::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of ICMP messages which this entity did
            not send due to problems discovered within ICMP
            such as a lack of buffers.  This value should not
            include errors discovered outside the ICMP layer
            such as the inability of IP to route the resultant
            datagram.  In some implementations there may be no
            types of error which contribute to this counter's
            value."
    ::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of ICMP Destination Unreachable
            messages sent."
    ::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of ICMP Time Exceeded messages sent."
    ::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of ICMP Parameter Problem messages
            sent."
    ::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
```

```
    DESCRIPTION
        "The number of ICMP Source Quench messages sent."
    ::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of ICMP Redirect messages sent.  For a
        host, this object will always be zero, since hosts
        do not send redirects."
    ::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of ICMP Echo (request) messages sent."
    ::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of ICMP Echo Reply messages sent."
    ::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of ICMP Timestamp (request) messages
        sent."
    ::= { icmp 23 }

icmpOutTimestampReps OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of ICMP Timestamp Reply messages
        sent."
    ::= { icmp 24 }
```

```
icmpOutAddrMasks OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The number of ICMP Address Mask Request messages
         sent."
   ::= { icmp 25 }

icmpOutAddrMaskReps OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The number of ICMP Address Mask Reply messages
         sent."
   ::= { icmp 26 }


-- the TCP group

-- Implementation of the TCP group is mandatory for all
-- systems that implement the TCP.

-- Note that instances of object types that represent
-- information about a particular TCP connection are
-- transient; they persist only as long as the connection
-- in question.

tcpRtoAlgorithm OBJECT-TYPE
   SYNTAX  INTEGER {
            other(1),   -- none of the following

            constant(2), -- a constant rto
            rsre(3),    -- MIL-STD-1778, Appendix B
            vanj(4)     -- Van Jacobson's algorithm [10]
           }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The algorithm used to determine the timeout value
         used for retransmitting unacknowledged octets."
   ::= { tcp 1 }

tcpRtoMin OBJECT-TYPE
   SYNTAX  INTEGER
```

```
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The minimum value permitted by a TCP
            implementation for the retransmission timeout,
            measured in milliseconds.  More refined semantics
            for objects of this type depend upon the algorithm
            used to determine the retransmission timeout.  In
            particular, when the timeout algorithm is rsre(3),
            an object of this type has the semantics of the
            LBOUND quantity described in RFC 793."
    ::= { tcp 2 }


tcpRtoMax OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The maximum value permitted by a TCP
            implementation for the retransmission timeout,
            measured in milliseconds.  More refined semantics
            for objects of this type depend upon the algorithm
            used to determine the retransmission timeout.  In
            particular, when the timeout algorithm is rsre(3),
            an object of this type has the semantics of the
            UBOUND quantity described in RFC 793."
    ::= { tcp 3 }

tcpMaxConn OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The limit on the total number of TCP connections
            the entity can support.  In entities where the
            maximum number of connections is dynamic, this
            object should contain the value -1."
    ::= { tcp 4 }

tcpActiveOpens OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of times TCP connections have made a
            direct transition to the SYN-SENT state from the
```

```
        CLOSED state."
    ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of times TCP connections have made a
        direct transition to the SYN-RCVD state from the
        LISTEN state."
    ::= { tcp 6 }

tcpAttemptFails OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of times TCP connections have made a
        direct transition to the CLOSED state from either
        the SYN-SENT state or the SYN-RCVD state, plus the
        number of times TCP connections have made a direct
        transition to the LISTEN state from the SYN-RCVD
        state."
    ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of times TCP connections have made a
        direct transition to the CLOSED state from either
        the ESTABLISHED state or the CLOSE-WAIT state."
    ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
    SYNTAX  Gauge
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of TCP connections for which the
        current state is either ESTABLISHED or CLOSE-
        WAIT."
    ::= { tcp 9 }

tcpInSegs OBJECT-TYPE
```

```
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of segments received, including
            those received in error.  This count includes
            segments received on currently established
            connections."
    ::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of segments sent, including
            those on current connections but excluding those
            containing only retransmitted octets."
    ::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of segments retransmitted - that
            is, the number of TCP segments transmitted
            containing one or more previously transmitted
            octets."
    ::= { tcp 12 }


-- the TCP Connection table

-- The TCP connection table contains information about this
-- entity's existing TCP connections.

tcpConnTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "A table containing TCP connection-specific
            information."
    ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
```

```
    SYNTAX TcpConnEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "Information about a particular current TCP
        connection.  An object of this type is transient,
        in that it ceases to exist when (or soon after)
        the connection makes the transition to the CLOSED
        state."
    INDEX  { tcpConnLocalAddress,
             tcpConnLocalPort,
             tcpConnRemAddress,
             tcpConnRemPort }
    ::= { tcpConnTable 1 }

TcpConnEntry ::=
    SEQUENCE {
      tcpConnState
        INTEGER,
      tcpConnLocalAddress
        IpAddress,
      tcpConnLocalPort
        INTEGER (0..65535),
      tcpConnRemAddress
        IpAddress,
      tcpConnRemPort
        INTEGER (0..65535)
    }

tcpConnState OBJECT-TYPE
    SYNTAX  INTEGER {
              closed(1),
              listen(2),
              synSent(3),
              synReceived(4),
              established(5),
              finWait1(6),
              finWait2(7),
              closeWait(8),
              lastAck(9),
              closing(10),
              timeWait(11),
              deleteTCB(12)
            }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
```

```
        "The state of this TCP connection.

        The only value which may be set by a management
        station is deleteTCB(12).  Accordingly, it is
        appropriate for an agent to return a `badValue'
        response if a management station attempts to set
        this object to any other value.

        If a management station sets this object to the
        value deleteTCB(12), then this has the effect of
        deleting the TCB (as defined in RFC 793) of the
        corresponding connection on the managed node,
        resulting in immediate termination of the
        connection.

        As an implementation-specific option, a RST
        segment may be sent from the managed node to the
        other TCP endpoint (note however that RST segments
        are not sent reliably)."
    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local IP address for this TCP connection.  In
        the case of a connection in the listen state which
        is willing to accept connections for any IP
        interface associated with the node, the value
        0.0.0.0 is used."
    ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local port number for this TCP connection."
    ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The remote IP address for this TCP connection."
```

```
    ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The remote port number for this TCP connection."
    ::= { tcpConnEntry 5 }


-- additional TCP objects

tcpInErrs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of segments received in error
            (e.g., bad TCP checksums)."
    ::= { tcp 14 }

tcpOutRsts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of TCP segments sent containing the
            RST flag."
    ::= { tcp 15 }


-- the UDP group

-- Implementation of the UDP group is mandatory for all
-- systems which implement the UDP.

udpInDatagrams OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of UDP datagrams delivered to
            UDP users."
    ::= { udp 1 }

udpNoPorts OBJECT-TYPE
```

```
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of received UDP datagrams for
            which there was no application at the destination
            port."
    ::= { udp 2 }

udpInErrors OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of received UDP datagrams that could
            not be delivered for reasons other than the lack
            of an application at the destination port."
    ::= { udp 3 }

udpOutDatagrams OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of UDP datagrams sent from this
            entity."
    ::= { udp 4 }


-- the UDP Listener table

-- The UDP listener table contains information about this
-- entity's UDP end-points on which a local application is
-- currently accepting datagrams.

udpTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF UdpEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "A table containing UDP listener information."
    ::= { udp 5 }

udpEntry OBJECT-TYPE
    SYNTAX  UdpEntry
    ACCESS  not-accessible
    STATUS  mandatory
```

```
    DESCRIPTION
        "Information about a particular current UDP
        listener."
    INDEX   { udpLocalAddress, udpLocalPort }
    ::= { udpTable 1 }

UdpEntry ::=
    SEQUENCE {
      udpLocalAddress
        IpAddress,
      udpLocalPort
        INTEGER (0..65535)
    }

udpLocalAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local IP address for this UDP listener.  In
        the case of a UDP listener which is willing to
        accept datagrams for any IP interface associated
        with the node, the value 0.0.0.0 is used."
    ::= { udpEntry 1 }

udpLocalPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local port number for this UDP listener."
    ::= { udpEntry 2 }


-- the EGP group

-- Implementation of the EGP group is mandatory for all
-- systems which implement the EGP.

egpInMsgs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of EGP messages received without
        error."
    ::= { egp 1 }
```

```
egpInErrors OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The number of EGP messages received that proved
         to be in error."
   ::= { egp 2 }

egpOutMsgs OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The total number of locally generated EGP
         messages."
   ::= { egp 3 }

egpOutErrors OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "The number of locally generated EGP messages not
         sent due to resource limitations within an EGP
         entity."
   ::= { egp 4 }


-- the EGP Neighbor table

-- The EGP neighbor table contains information about this
-- entity's EGP neighbors.

egpNeighTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF EgpNeighEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
         "The EGP neighbor table."
   ::= { egp 5 }

egpNeighEntry OBJECT-TYPE
   SYNTAX  EgpNeighEntry
   ACCESS  not-accessible
   STATUS  mandatory
```

```
    DESCRIPTION
        "Information about this entity's relationship with
        a particular EGP neighbor."
    INDEX   { egpNeighAddr }
    ::= { egpNeighTable 1 }

EgpNeighEntry ::=
    SEQUENCE {
        egpNeighState
            INTEGER,
        egpNeighAddr
            IpAddress,
        egpNeighAs
            INTEGER,
        egpNeighInMsgs
            Counter,
        egpNeighInErrs
            Counter,
        egpNeighOutMsgs
            Counter,
        egpNeighOutErrs
            Counter,
        egpNeighInErrMsgs
            Counter,
        egpNeighOutErrMsgs
            Counter,
        egpNeighStateUps
            Counter,
        egpNeighStateDowns
            Counter,
        egpNeighIntervalHello
            INTEGER,
        egpNeighIntervalPoll
            INTEGER,
        egpNeighMode
            INTEGER,
        egpNeighEventTrigger
            INTEGER
    }

egpNeighState OBJECT-TYPE
    SYNTAX  INTEGER {
                idle(1),
                acquisition(2),
                down(3),
                up(4),
                cease(5)
```

```
        }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The EGP state of the local system with respect to
        this entry's EGP neighbor.  Each EGP state is
        represented by a value that is one greater than
        the numerical value associated with said state in
        RFC 904."
    ::= { egpNeighEntry 1 }

egpNeighAddr OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The IP address of this entry's EGP neighbor."
    ::= { egpNeighEntry 2 }

egpNeighAs OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The autonomous system of this EGP peer.  Zero
        should be specified if the autonomous system
        number of the neighbor is not yet known."
    ::= { egpNeighEntry 3 }

egpNeighInMsgs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of EGP messages received without error
        from this EGP peer."
    ::= { egpNeighEntry 4 }

egpNeighInErrs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The number of EGP messages received from this EGP
        peer that proved to be in error (e.g., bad EGP
        checksum)."
    ::= { egpNeighEntry 5 }
```

```
egpNeighOutMsgs OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of locally generated EGP messages to
        this EGP peer."
   ::= { egpNeighEntry 6 }

egpNeighOutErrs OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of locally generated EGP messages not
        sent to this EGP peer due to resource limitations
        within an EGP entity."
   ::= { egpNeighEntry 7 }

egpNeighInErrMsgs OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of EGP-defined error messages received
        from this EGP peer."
   ::= { egpNeighEntry 8 }

egpNeighOutErrMsgs OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of EGP-defined error messages sent to
        this EGP peer."
   ::= { egpNeighEntry 9 }

egpNeighStateUps OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of EGP state transitions to the UP
        state with this EGP peer."
   ::= { egpNeighEntry 10 }
```

```
egpNeighStateDowns OBJECT-TYPE
   SYNTAX Counter
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The number of EGP state transitions from the UP
        state to any other state with this EGP peer."
   ::= { egpNeighEntry 11 }

egpNeighIntervalHello OBJECT-TYPE
   SYNTAX INTEGER
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The interval between EGP Hello command
        retransmissions (in hundredths of a second). This
        represents the t1 timer as defined in RFC 904."
   ::= { egpNeighEntry 12 }

egpNeighIntervalPoll OBJECT-TYPE
   SYNTAX INTEGER
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The interval between EGP poll command
        retransmissions (in hundredths of a second). This
        represents the t3 timer as defined in RFC 904."
   ::= { egpNeighEntry 13 }

egpNeighMode OBJECT-TYPE
   SYNTAX INTEGER { active(1), passive(2) }
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
        "The polling mode of this EGP entity, either
        passive or active."
   ::= { egpNeighEntry 14 }

egpNeighEventTrigger OBJECT-TYPE
   SYNTAX INTEGER { start(1), stop(2) }
   ACCESS read-write
   STATUS mandatory
   DESCRIPTION
        "A control variable used to trigger operator-
        initiated Start and Stop events. When read, this
        variable always returns the most recent value that
        egpNeighEventTrigger was set to. If it has not
```

```
            been set since the last initialization of the
            network management subsystem on the node, it
            returns a value of `stop'.

            When set, this variable causes a Start or Stop
            event on the specified neighbor, as specified on
            pages 8-10 of RFC 904.  Briefly, a Start event
            causes an Idle peer to begin neighbor acquisition
            and a non-Idle peer to reinitiate neighbor
            acquisition.  A stop event causes a non-Idle peer
            to return to the Idle state until a Start event
            occurs, either via egpNeighEventTrigger or
            otherwise."
    ::= { egpNeighEntry 15 }


-- additional EGP objects

egpAs OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The autonomous system number of this EGP entity."
    ::= { egp 6 }


-- the Transmission group

-- Based on the transmission media underlying each interface
-- on a system, the corresponding portion of the Transmission
-- group is mandatory for that system.

-- When Internet-standard definitions for managing
-- transmission media are defined, the transmission group is
-- used to provide a prefix for the names of those objects.

-- Typically, such definitions reside in the experimental
-- portion of the MIB until they are "proven", then as a
-- part of the Internet standardization process, the
-- definitions are accordingly elevated and a new object
-- identifier, under the transmission group is defined. By
-- convention, the name assigned is:
--
--     type OBJECT IDENTIFIER    ::= { transmission number }
--
-- where "type" is the symbolic value used for the media in
```

```
-- the ifType column of the ifTable object, and "number" is
-- the actual integer value corresponding to the symbol.


-- the SNMP group

-- Implementation of the SNMP group is mandatory for all
-- systems which support an SNMP protocol entity.  Some of
-- the objects defined below will be zero-valued in those
-- SNMP implementations that are optimized to support only
-- those functions specific to either a management agent or
-- a management station.  In particular, it should be
-- observed that the objects below refer to an SNMP entity,
-- and there may be several SNMP entities residing on a
-- managed node (e.g., if the node is hosting acting as
-- a management station).

snmpInPkts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of Messages delivered to the
            SNMP entity from the transport service."
    ::= { snmp 1 }

snmpOutPkts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Messages which were
            passed from the SNMP protocol entity to the
            transport service."
    ::= { snmp 2 }

snmpInBadVersions OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Messages which were
            delivered to the SNMP protocol entity and were for
            an unsupported SNMP version."
    ::= { snmp 3 }

snmpInBadCommunityNames OBJECT-TYPE
```

```
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of SNMP Messages delivered to
        the SNMP protocol entity which used a SNMP
        community name not known to said entity."
    ::= { snmp 4 }

snmpInBadCommunityUses OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of SNMP Messages delivered to
        the SNMP protocol entity which represented an SNMP
        operation which was not allowed by the SNMP
        community named in the Message."
    ::= { snmp 5 }

snmpInASNParseErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of ASN.1 or BER errors
        encountered by the SNMP protocol entity when
        decoding received SNMP Messages."
    ::= { snmp 6 }


-- { snmp 7 } is not used

snmpInTooBigs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of SNMP PDUs which were
        delivered to the SNMP protocol entity and for
        which the value of the error-status field is
        `tooBig'."
    ::= { snmp 8 }

snmpInNoSuchNames OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
```

```
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP PDUs which were
            delivered to the SNMP protocol entity and for
            which the value of the error-status field is
            `noSuchName'."
    ::= { snmp 9 }

snmpInBadValues OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP PDUs which were
            delivered to the SNMP protocol entity and for
            which the value of the error-status field is
            `badValue'."
    ::= { snmp 10 }

snmpInReadOnlys OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number valid SNMP PDUs which were
            delivered to the SNMP protocol entity and for
            which the value of the error-status field is
            `readOnly'.  It should be noted that it is a
            protocol error to generate an SNMP PDU which
            contains the value `readOnly' in the error-status
            field, as such this object is provided as a means
            of detecting incorrect implementations of the
            SNMP."
    ::= { snmp 11 }

snmpInGenErrs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP PDUs which were
            delivered to the SNMP protocol entity and for
            which the value of the error-status field is
            `genErr'."
    ::= { snmp 12 }

snmpInTotalReqVars OBJECT-TYPE
```

```
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of MIB objects which have been
            retrieved successfully by the SNMP protocol entity
            as the result of receiving valid SNMP Get-Request
            and Get-Next PDUs."
    ::= { snmp 13 }

snmpInTotalSetVars OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of MIB objects which have been
            altered successfully by the SNMP protocol entity
            as the result of receiving valid SNMP Set-Request
            PDUs."
    ::= { snmp 14 }

snmpInGetRequests OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Get-Request PDUs which
            have been accepted and processed by the SNMP
            protocol entity."
    ::= { snmp 15 }

snmpInGetNexts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Get-Next PDUs which have
            been accepted and processed by the SNMP protocol
            entity."
    ::= { snmp 16 }

snmpInSetRequests OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Set-Request PDUs which
```

```
        have been accepted and processed by the SNMP
        protocol entity."
    ::= { snmp 17 }

snmpInGetResponses OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP Get-Response PDUs which
        have been accepted and processed by the SNMP
        protocol entity."
    ::= { snmp 18 }

snmpInTraps OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP Trap PDUs which have
        been accepted and processed by the SNMP protocol
        entity."
    ::= { snmp 19 }

snmpOutTooBigs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP PDUs which were
        generated by the SNMP protocol entity and for
        which the value of the error-status field is
        `tooBig.'"
    ::= { snmp 20 }

snmpOutNoSuchNames OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP PDUs which were
        generated by the SNMP protocol entity and for
        which the value of the error-status is
        `noSuchName'."
    ::= { snmp 21 }

snmpOutBadValues OBJECT-TYPE
```

```
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP PDUs which were
            generated by the SNMP protocol entity and for
            which the value of the error-status field is
            `badValue'."
    ::= { snmp 22 }

-- { snmp 23 } is not used

snmpOutGenErrs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP PDUs which were
            generated by the SNMP protocol entity and for
            which the value of the error-status field is
            `genErr'."
    ::= { snmp 24 }

snmpOutGetRequests OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Get-Request PDUs which
            have been generated by the SNMP protocol entity."
    ::= { snmp 25 }

snmpOutGetNexts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Get-Next PDUs which have
            been generated by the SNMP protocol entity."
    ::= { snmp 26 }

snmpOutSetRequests OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of SNMP Set-Request PDUs which
```

```
        have been generated by the SNMP protocol entity."
    ::= { snmp 27 }

snmpOutGetResponses OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP Get-Response PDUs which
        have been generated by the SNMP protocol entity."
    ::= { snmp 28 }

snmpOutTraps OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The total number of SNMP Trap PDUs which have
        been generated by the SNMP protocol entity."
    ::= { snmp 29 }

snmpEnableAuthenTraps OBJECT-TYPE
    SYNTAX  INTEGER { enabled(1), disabled(2) }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "Indicates whether the SNMP agent process is
        permitted to generate authentication-failure
        traps.  The value of this object overrides any
        configuration information; as such, it provides a
        means whereby all authentication-failure traps may
        be disabled.

        Note that it is strongly recommended that this
        object be stored in non-volatile memory so that it
        remains constant between re-initializations of the
        network management system."
    ::= { snmp 30 }

END
```

ANEXO B

COMO COMPILAR E EXECUTAR O PROTÓTIPO

Os arquivos *Makefile*, *interface.h*, *main.c*, *funcoes.c*, *snmpget.c* e *db.q* , que foram desenvolvidos encontram-se no diretório */marcelo/snmp/trab1*. Os arquivos **.h* do pacote CMU estão localizados no diretório */marcelo/snmp/include*. Os arquivos **.h* necessários para a construção da interface com o usuário está no diretório */usr/openwin/include*.

Para gerar o código executável, o programa deve ser ligado às bibliotecas do pacote SNMP- CMU, do banco de dados Ingres e do XView. A primeira está localizada no diretório */snmplib* e é referenciada por *libsnmp.a*. A segunda está localizada no diretório/*home/espora/ingres/lib* e *é referenciada por libq.a*. Por último, para a utilização do Xview, deve-se ligar ao programa as bibliotecas *lxview*, *lolgx* e *lX11*, localizadas no diretório */usr/openwin/lib*.

O arquivo *Makefile* gera o programa corretamente. Entretanto, antes de utilizar o comando *make*, deve-se pré-compilar o arquivo *db.q*, através do comando *equel db.q*. Este comando gera o arquivo *db.c*.

O programa deve ser compilado e executado numa estação que tiver instalado o banco de dados Ingres. O nome do programa gerado é *gerente*.

BIBLIOGRAFIA

Bibliografia citada

[BEN 90] BENNET, Geoff. The simple network management protocol. **Telecommunications**, Norwood, USA, v.24, n.2, p.21-26, Feb. 1990.

[CCI 88] CCITT. **Recomendation Z100**: specification and description language (SDL). Genebra: CCITT, 1988. 180p. (Blue Book, v.10, n.2)

[CON 87] CONFIGURATION management service definition. [S.l.] : European Computer Manufecturers Association (ECMA), 1987. 49p. (Technical Report) Final draft.

[FEH 89] FEHSKENS, Len. An architectural Strategy for Enterprise Management. In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management**. Amsterdam:Elsevier, 1989. v.1. p.41-55.

[FEL 89] FELDKKHUN, Lev. Integrated network management systems: a global perspective on the issue. In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management**. Amsterdam:Elsevier, 1989. v.1. p.279-301.

[FIS 91] FISHER, Sharon. Dueling Protocols. **Byte**, Hightstown, USA. v.10, n.3, p.183-190, Feb. 1991.

[ISO 85] ISO. **Information processing systems - open systems interconnection - specification of abstract syntax notation one (ASN.1)**. Switzerland: [s.n.] 1985. (Draft international standard ISO/DIS 8824)

[MAR 90] MARQUIE, A.; BENZERKI, D.; RAYNAUD, Y. An implementation support to a local system OSI manager. In: INTERNATIONAL SYMPOSIUM ON LOCAL COMMUNICATIONS SYSTEM MANAGEMENT, Sept. 18-19, 1990. Canterbury, UK. **Proceedings...** Canterbury:[s.n.], 1990. p.31-50.

[McC 90] McCLOGHRIE, Keith; ROSE, Marshall. **Management information base for network management of TCP/IP-based internets.** [s.l.]:[s.n.], 1990. 91p. (RFC, 1156).

[McC 91] McCLOGHRIE, Keith; ROSE, Marshall. **Management information ase for network management of TCP/IP-based internets: MIB-II.** [s.l.]:[s.n.], 1991. 70p. (RFC, 1213)

[PAU 90] PAULISCH, Sephan. Configuration and performance management of LANS. In: INTERNATIONAL SYMPOSIUM ON LOCAL COMMUNICATIONS SYSTEM MANAGEMENT, Sept. 18-19, 1990, Canterbury, UK. **Proceedings...** Canterbury: [s.n.], 1990. p.237-253.

[POT 91] POTTER, Ducan. The Need for network management. **Computer Communications**, Aldershot, UK, v.14, n.2, p.121-126, Mar. 1991.

[SLU 89] SLUMAN, Chris. A Tutorial on OSI management.**Computer Networks and ISDN Systems**, Atlanta, USA, v.17, n.4-5, p.270-278, Oct. 1989.

[WAR 89a] WARRIER, U. S.; BESAW, L. **The common management information services and protocol over TCP/IP (CMOT).** [s.l.]:[s.n.],1989. (RFC, 1095).

[WIL 90] WILLETS, Keith. Concert - an initiative towards open integrated network management. **Computer Communications**, Aldershot, UK, v.13, n.9, p.527-532, Nov. 1990.

Bibliografia consultada

[BRU 91] BRUSIL, J. Paul. Components of OSI: systems manager. **Connexions-The interoperability report**, Mountain View, USA, v.5, n.4, p.2-15, Apr. 1991.

[CLA 90] CLAUDÉ, J. P. et al. Management of open networks in heterogeneous context. In: INTERNATIONAL SYMPOSIUM ON LOCAL COMMUNICATIONS SYSTEM MANAGEMENT, 18-19 de setembro de 1990, Canterbury, UK. **Proceedings...** Canterbury:[s.n.], 1990. p.65-84.

[COL 89] COLLINS, Will. OSI management service elements, protocols and application layer structure (ALS). In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management**. Amsterdam: Elsevier, 1989. v.1. p.119-131.

[CRA 89] CRAWFORD, John L. Graphics for network management: an interactive approach. In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management**. Amsterdam: Elsevier, 1989. v.1. p.197-208.

[KLE 88] KLERER, Mark S. The OSI management architecture: an overview. **IEEE Network**, v.2, n.2, p.20-29, Mar. 1988.

[McC 89] McCLOGHRIE, Keith; ROSE, Marshall T. Defining a protocol-independent management information base. In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management**. Amsterdam:Elsevier, 1989. v.1. p.185-195.

[PRE 90] PRESUHN, Randy. Considering CMIP. **Data Communications**, v.19, n.3, p.55-59, Mar. 1990.

[ROS 91] ROSE, Marshall. **The Simple Book:** an introduction to management of TCP/IP-based internets. Englewood Cliffs: Prentice-Hall, 1991. 347p.

[TAR 90] TAROUCO, L. M. R. Support to network management. In: CONFERENCE ON INFORMATION NETWORK AND DATA COMMUNICATION, Mar. 26-29, 1990, Lillehammer, Norway. **Proceedings...** Lillehammer:[s.n.], 1990. 1v.

[VAN 90] VANDENBERG, Chris. MIB II extends SNMP interoperability. **Data Communications**, New York, USA, v.19, n.13, p.97-102, Oct. 1990.

[WAR 89b] WARRIER, U. S.; SUNSHINE C. A. A platform for heterogeneous interconnection network managment. In: MEANDZIJA, B.; WESTCOTT, J. (eds.) **Integrated network management.** Amsterdam:Elsevier, 1989. v.1. p.13-24.

[WES 90] WESTPHALL, C. B.;SEKKAKI, A. Heterogeneous LANs management. In: INTERNATIONAL SYMPOSIUM ON LOCAL COMMUNICATIONS SYSTEM MANAGEMENT, 18-19 de setembro de 1990, Canterbury, UK. **Proceedings...** Canterbury:[s.n.], 1990. p.17-29.

[WES 91] WESTPHALL, C. B. **Conception et developpement de l'architecture d'administration d'un reseau metropolitain.** Toulouse: Institut de Recherche en Informatique de Toulouse / Université Paul Sabatier, 1991. 123p. (Tese de Doutorado)

Informática
UFRGS

Um Modelo de Gerência de Configuração de Redes Locais.

Dissertação apresentada aos Senhores:

Prof. Dr. Carlos Becker Westphall (UFSC)

Profa. Dra. Liane Margarida Rockenbach Tarouco

Prof. Dr. Maurício Ferreira Magalhães (UNICAMP)

Prof. Dr. Philippe Olivier Alexandre Navaux

Vista e permitida a impressão.
Porto Alegre, 20/04/94.

Profa. Dra. Liane M. R. Tarouco,
Orientador.

Prof. Dr. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.