HENRIQUE DENES HILGENBERG FERNANDES

ANEMONA: UMA LINGUAGEM DE CONFIGURAÇÃO PARA APLICAÇÕES DE MONITORAÇÃO DE REDES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, Curso de Mestrado em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Martin Alejandro Musicante
Co-Orientador: Prof. Elias Procópio Duarte Jr.

CURITIBA
2001

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno ***Henrique Denes Hilgenberg Fernandes***, avaliamos o trabalho intitulado ***"ANEMONA: Uma Linguagem de Configuração para Aplicações de Monitoração de Redes"***, cuja defesa foi realizada no dia 21 de setembro de 2001, às treze horas e trinta minutos, no anfiteatro B, do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 21 de setembro de 2001.

Prof. Dr. Martín Alejandro Musicante
DINF/UFPR - Orientador

Prof. Dr. Elias Procópio Duarte Jr.
DINF/UFPR - Co-Orientador

Prof. Dr. José Marcos Silva Nogueira
DCC/UFMG

Prof. Dr. Eduardo Parente Ribeiro
PPGInf/UFPR

Agradecimentos

Primeiramente agradeço a Deus e à minha mãe. Agradeço, em especial ao meu orientador e chefe Prof. Martin Musicante, por ter me aceitado como aluno e que nunca mediu esforços ao cumprir o seu papel de mestre. Muito obrigado por tudo.

Agradeço também ao meu co-orientador Prof. Elias Procópio Duarte Jr. Imensos agradecimentos ao meu amigo Paulo Mauch Neto, que além do apoio e consideração, colocou a minha disposição todos os recursos computacionais da sua divisão.

Ao amigo Luiz Carlos Klug, obrigado pelo apoio e por ter me inserido no magistério superior. Ao colega Aldri Luiz dos Santos, meus agradecimentos pelas dicas que me foram úteis na implementação das MIB's.

Obrigado ao Prof. José Marcos Silva Nogueira, que aceitou participar da banca. Obrigado, também aos Prof. Eduardo Parente Ribeiro e Antônio Urban, pelas cartas de recomendação.

Por fim, agradeço ainda a todos aqueles que confiaram em mim.

Sumário

Lista de Ilustrações

Lista de Abreviaturas e Siglas

ANEMONA	- A NEtwork MONitoring Application
ASCII	- American Standard Code for Information Interchange
ASN.1	- Abstract Syntax Notation One
CCITT	- Comité Consultatif International Téléphonique et Télégraphique
DISMAN	- Distributed Management Working Group
DNS	- Domain Name Server
DOD	- Department of Defense
DOS	- Denial of Service
EGP	- External Gateway Protocol
FTP	- File Transfer Protocol
HEMS	- High-Level Entity Management System
HTTP	- Hipertext Transfer Protocol
ICMP	- Internet Control Message Protocol
IETF	- Internet Engeneering Task Force
IP	- Internet Protocol
ISO	- International Organization for Standardization
ITU	- International Telecomunication Union
MIB	- Management Information Base
MIB2C	- MIB to C
OID	- Object Identifier
OSI	- Open Systems Interconnection
PDU	- Protocol Data Unit
RFC	- Request for Comments
SMI	- Structure of Management Information

SNMP	- Simple Network Management Protocol
TCP	- Transmission Control Protocol
UDP	- User Datagram Protocol
YACC	- Yet Another Compiler Compiler

Resumo

A atividade de gerência de redes consiste no controle e monitoramento dos recursos de hardware e software de uma rede de computadores. O protocolo de gerência de redes padrão da Internet é o SNMP (*Simple Network Management Protocol*). O gerenciamento baseado em SNMP usa bases de informações chamadas MIB's (*Management Information Bases*), que podem ser definidas e atualizadas conforme a aplicação. O gerenciamento de redes é uma tarefa complexa, que pode ser distribuída mediante o uso das MIB's definidas pelo grupo de trabalho DISMAN da IETF, entre as quais estão a *Expression MIB* e a *Event MIB*. No entanto, acreditamos que a utilização destas MIB's vem sendo restringida devido a sua complexidade de operação. Neste trabalho, propomos uma linguagem simples, concisa e precisa, como alternativa para a configuração destas duas MIB's, visando disponibilizar aos administradores de redes uma interface de mais alto nível para facilitar as tarefas de monitoração e gerência de redes de computadores. No âmbito deste trabalho, foram implementados as duas MIB's acima descritas e um tradutor para a linguagem proposta. O conjunto que compõe a ferramenta foi testado e os resultados apurados estão nesta dissertação.

Abstract

Network management consists in the control and monitoring of hardware and software resources from a computer network. The Internet's network management standard protocol is SNMP (Simple Network Management Protocol). Management based on SNMP uses information bases called MIBs (Management Information Bases), that are defined and updated according to the application's needs. Network management is a complex task, that can be simplifyed using some special MIBs as the Expression and Event MIBs. We believe that the utilization of these MIBs is limited because of their complex operation. In this work, we propose a simple, concise and precise language, for the configuration of these two MIBs . Our goal is to help the network administrators in their management and monitoring tasks, by the use of a higher level tool. As part of this work, we implemented the above mentioned MIBs as well as a translator to the proposed language.

Capítulo 1

Introdução

Neste capítulo é dada uma visão geral das idéias que serão mostradas nessa dissertação. Inicialmente são descritos os motivos que levaram a definição da linguagem descrita neste trabalho. A seguir são descritas as atividades realizadas e finalmente é mostrado como estes tópicos estão dispostos no texto.

1.1 Motivação

O resultado da interação da informática e das telecomunicações tem hoje um papel fundamental na economia e no modo de vida da nossa sociedade. Com a popularização das redes de computadores e o requisito de conectividade universal, além da necessidade de funcionamento ininterrupto, estas se tornaram maiores e mais complexas, necessitando de soluções integradas de gerenciamento, que informam e fornecem ferramentas para gerentes humanos monitorarem e controlarem o sistema.

O protocolo de gerência de redes padrão da Internet é o SNMP (*Simple Network Management Protocol*) [1, 2, 3], agora em sua terceira versão. O gerenciamento de redes baseado em SNMP utiliza bases de informação, chamadas MIB's (*Management Information Bases*) [4, 5]. Cada MIB é composta por um grupo de objetos cujo conteúdo pode ser lido e/ou modificado, dependendo da sua concepção. Lendo estes objetos coletamos informações sobre o funcionamento do sistema a ser monitorado/gerenciado e escrevendo neles, podemos atribuir novas configurações aos elementos de rede respectivos.

Como uma possível solução para as necessidades demandadas pelo crescimento das redes de computadores, existe uma corrente identificada como gerência de redes distribuída. Esta metodologia propõe que aplicações de gerência sejam distribuídas a fim de minimizar a interação com usuários, reduzindo o consumo de recursos da rede.

O DISMAN (*Distributed Management Working Group*) faz parte da IETF (*Internet Engeneering Task Force* – Organização de padrões da Internet) e suas atividades são direcionadas para aplicações específicas de gerenciamento distribuído, fazendo uso do protocolo SNMP.

Os sistemas SNMP tanto de domínio público como comerciais já são instalados com algumas MIB's padronizadas. No entanto, é possível estender a funcionalidade do sistema com o acréscimo de bases de informação com propósitos específicos, programando MIB's especialmente para um fim determinado. Existem algumas ferramentas disponíveis que visam facilitar este trabalho, mas de uma maneira geral, implementar uma MIB é uma tarefa árdua para um usuário eventual do SNMP.

O DISMAN propõe, entre outras, duas MIB's: A *Expression MIB* [6] e a *Event MIB* [7]. A *Expression MIB* permite a construção e avaliação de expressões lógicas e aritméticas com valores obtidos de objetos de gerência, enquanto que a *Event MIB* monitora um conjunto de objetos, podendo sinalizar um evento quando uma condição programada ocorrer. A *Event MIB* pode também disparar um evento, programado previamente. Combinando estas duas MIB's o gerente dispõe de uma capacidade de expressão significativa. A principal contribuição do uso conjunto destas MIB's está no fato de que, dependendo da aplicação, não é necessário implementar uma nova MIB, pois o resultado pode ser obtido a partir de MIB's já implementadas, combinando seus objetos, fazendo uso dos recursos da *Event* e da *Expression*.

O problema agora passa a ser atividade de configurar corretamente as duas bases propostas acima a fim de que sejam úteis para as condições de gerenciamento desejadas. Para isso, é proposta neste trabalho, uma linguagem que permite ao usuário abstrair a um nível mais elevado a tarefa de configuração das MIB's *Event* e *Expression*, tendo como única preocupação a lógica das condições a serem verificadas e os eventos relacionados a estas condições.

1.2 A Linguagem Proposta

A linguagem ANEMONA, aqui proposta, tem como requisitos básicos simplicidade, concisão e precisão. Apesar de existirem linguagens que suportem o gerenciamento de redes usando o protocolo SNMP, não foi encontrado na bibliografia nenhum trabalho correlato no domínio específico das duas MIB's descritas na seção anterior para a aplicação à qual a ANEMONA foi proposta, caracterizando assim, a natureza inédita desta abordagem.

Como parte deste estudo, foram implementados subconjuntos das MIB's *Event* e *Expression* e um tradutor para a linguagem ANEMONA. Todas as implementações foram amplamente testadas, ora individualmente, ora acopladas e os resultados dos testes mais importantes são descritos como estudos de caso.

1.3 Organização

Esta dissertação está organizada em oito capítulos. No capítulo 2 serão introduzidos os principais conceitos relacionados ao SNMP, incluindo as MIB's. O capítulo 3 descreve tanto a *Expression MIB* como a *Event MIB*, sua funcionalidade e principais componentes. No capítulo 4 está a definição da linguagem proposta. O capítulo 5 descreve as implementações da *Event MIB* e da *Expression MIB*. O capítulo 6 apresenta o tradutor para ANEMONA, o capítulo 7 mostra resultados experimentais e o capítulo 8 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Gerência de Redes Baseada em SNMP

Este capítulo apresenta conceitos básicos relacionados ao protocolo de gerência de redes padrão da Internet, o SNMP (*Simple Network Management Protocol*), sua arquitetura, suas bases de informações de gerência (MIB's), nomes de objetos e representação de dados em ASN.1.

2.1 Gerência de Redes

Com o crescimento e o aumento da complexidade das redes, que são compostas por equipamentos heterogêneos, com dispositivos fornecidos por diferentes fabricantes [8], passou a ser importante o emprego de sistemas automáticos de gerência de redes de computadores.

Apesar da existência de ferramentas *ad hoc* [9] como o *ping*, que utiliza uma requisição de eco do *Internet Control Message Protocol* (ICMP); o *traceroute*, que exibe todos os roteadores por onde um datagrama enviado passou e, ainda, a utilização de analisadores de protocolos, estas são insuficientes para a realização de todas as tarefas de monitoração e controle de redes.

A *gerência de redes* [9, 10] consiste de atividades de controle e monitoramento dos dispositivos de hardware e software que integram uma rede de computadores. Estes dispositivos podem ser computadores, roteadores, pontes, hubs, modens, terminais, impressoras ou qualquer outro elemento da rede. Também podem ser gerenciadas aplicações sendo executadas em um determinado *host*.

A ISO (*International Organization for Standardization*) dividiu a funcionalidade da gerência de redes em cinco áreas [10, 11]: gerência de falhas, gerência de desempenho, gerência de configuração, gerência de contas e gerência de segurança. Cada uma destas áreas é descrita brevemente a seguir.

Gerência de falhas é o processo de identificar e corrigir, se possível, problemas com a rede. Um exemplo de falha pode ser um *link* inativo e algumas de suas causas podem ser problemas no meio de transmissão, seja ele qual for, ou problemas nos roteadores.

A *gerência de desempenho* deve fornecer métricas do hardware, software e do meio de transmissão de uma rede. São exemplos dessas métricas: vazão total, disponibilidade e utilização de recursos, tempo de resposta e taxas de erros.

A *gerência de configuração* está relacionada com a inicialização e manutenção de dispositivos e aplicações dentro de uma rede. Um exemplo de gerência de configuração é o controle de versão dos softwares instalados em cada máquina.

A *gerência de contas* envolve registrar a utilização dos recursos da rede por cada usuário ou grupo de usuários para garantir que estes disponham de recursos suficientes, através do estabelecimento de cotas, determinação de custos ou contagem de usuários.

A *gerência de segurança* é o processo de controlar o acesso aos dados armazenados ou a recursos, controlando e monitorando dispositivos de segurança.

É importante citar o axioma fundamental da gerência de redes [9], onde o impacto de se adicionar um sistema de gerência em um dado dispositivo gerenciado deve ser o mínimo possível.

O *Simple Network Management Protocol* (SNMP) é o protocolo padrão da Internet para gerência de redes, sendo amplamente difundido e utilizado. Hoje, em sua terceira versão, o SNMP consiste de quatro componentes, que são os *nós gerenciados*, *estações de gerenciamento*, *informações de gerenciamento* e um *protocolo de gerência*. O usuário de um sistema integrado de gerência baseado no protocolo SNMP tem à sua disposição ferramentas para realizar as mais diversas tarefas relacionadas ao monitoramento e controle de uma rede TCP/IP.

2.2 O Modelo SNMP

Nesta seção é apresentado o modelo de gerência de redes baseado no protocolo SNMP.

2.2.1 Arquitetura do SNMP

Como visto na seção anterior, o modelo de gerência de redes do SNMP possui uma arquitetura composta por nós gerenciados, estações de gerenciamento, informações de gerenciamento e um protocolo de gerência. Estes elementos são brevemente descritos a seguir.

Os nós gerenciados podem ser *hosts*, roteadores, pontes, impressoras ou qualquer dispositivo capaz de informar seu estado para uma estação de gerenciamento. Para que um dado dispositivo possa ser gerenciado é necessário que ele execute um processo de gerenciamento SNMP denominado *agente SNMP*. Muitos dispositivos estão sendo fabricados com suporte para o agente SNMP. Cada agente mantém uma base de dados, chamada MIB (*Management Information Base*), com variáveis que refletem o estado atual do dispositivo e que algumas delas, ao serem modificadas, podem alterar configurações ou disparar eventos no dispositivo gerenciado.

Os dispositivos que executam o agente SNMP são gerenciados por estações de gerenciamento, que nada mais são do que computadores tradicionais executando a coleção de ferramentas de gerência. Estas estações doravante serão chamadas *gerentes*. Gerentes e agentes se comunicam usando um protocolo de gerência, onde os gerentes podem modificar e recuperar valores de objetos componentes da MIB de um dado agente.

Quando ocorrem eventos extraordinários, como uma falha ou a interrupção de um enlace, o agente pode notificar as estações de gerenciamento usando um alarme chamado *trap*. De acordo com a programação do agente, na ocorrência de um evento extraordinário, serão geradas *traps* para todos os gerentes de uma lista definida na ocasião da configuração do agente. Ao receber a notificação, o gerente deverá tomar as medidas necessárias para o diagnóstico e possível resolução do problema.

Cada agente mantém objetos que descrevem o estado do dispositivo gerenciado, agrupados em MIB's.

Muitas vezes, um agente pode necessitar de uma informação que não está disponível localmente, pois seu estado atual pode depender desta informação. Nesses casos, o agente executará uma consulta a um objeto de outro agente. Aquele que fez a consulta é chamado de *agente proxy*. Dessa maneira, um gerente consulta o *agente proxy* e este, por sua vez obtém informações em suas respectivas fontes e reporta ao gerente. Ressaltamos que este processo deve ser transparente; como se a informação desejada fosse residente no *agente proxy*. Seu uso pode ser útil em determinados contextos de segurança (onde o gerente não deve acessar diretamente a informação final). Além disso, este agente *proxy* pode atuar como uma ponte entre redes de diferentes protocolos ou ainda, ser um tradutor do protocolo SNMP para dispositivos sem suporte para gerenciamento.

Por fim, é importante citar que o SNMP possui um sistema de segurança e autenticação, de forma que uma requisição somente será respondida se ela for reconhecida como sendo enviada por um gerente autorizado, garantindo a integridade dos dados e o sigilo das informações de gerência.

A figura 1 ilustra dois agentes sendo gerenciados por dois gerentes. Também é mostrado o envio de um alarme (*trap*) de um agente para um dos gerentes e o funcionamento de um *agente proxy*. Cada tipo de interação da figura 1 será detalhado na seção 2.2.3.2.

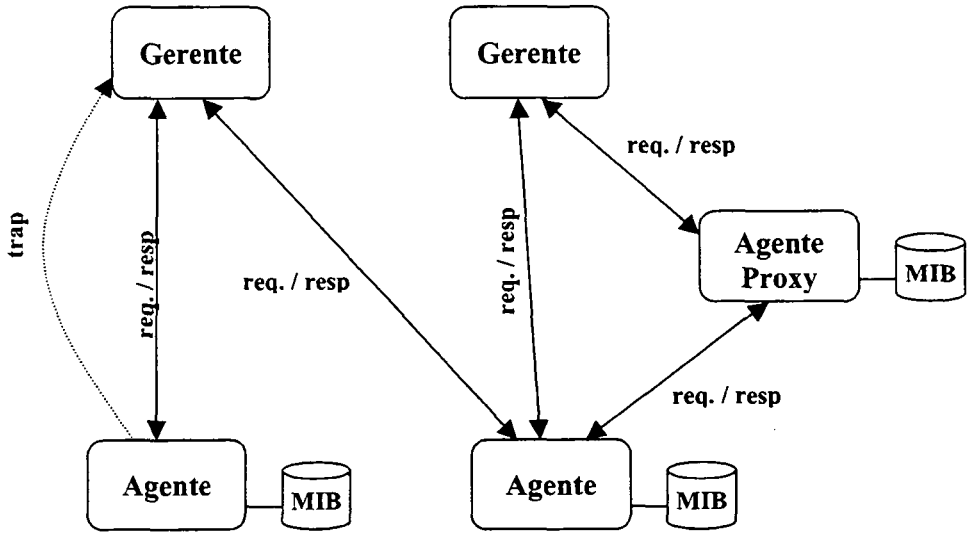

Figura 1: O Modelo SNMP

2.2.2 Informações de Gerência

Conforme abordado na seção anterior, cada item de informação de gerência pode ser visto como um objeto gerenciado. Estes objetos podem ser simples, quando constituem um elemento ou agregados, quando são coleções de registros estruturados sob a forma de tabelas. Os conjuntos de objetos de gerência de um agente são mantidos nas MIB's. Além das definições de objetos, uma MIB pode conter definições de eventos passíveis de serem notificados por uma *trap*. Uma abordagem mais detalhada sobre a definição e o comportamento das MIB's será vista na seção 2.3.

2.2.3 O Protocolo SNMP

A troca de informações de gerência entre as entidades do protocolo SNMP é feita através do envio e recepção de *Protocol Data Units* (PDU's) [4]. O protocolo define a comunicação entre gerentes e agentes.

A seguir são apresentadas as operações do protocolo, tipos de interação entre entidades e o formato dos PDU's.

2.2.3.1 Operações do Protocolo

O SNMP utiliza o paradigma de busca e armazenamento (*fetch-store paradigm)*, originado do protocolo de gerenciamento conhecido por HEMS (*High-level Entity Management System*) [12]. Há, basicamente duas únicas operações primitivas, a primeira (*snmpget*) que permite a recuperação de um valor armazenado em um objeto e a segunda (*snmpset*) que atribui um valor a um objeto. Existem outras operações, as quais são composições das operações primitivas.

Uma relação das operações definidas para o SNMPv3 é mostrada na tabela da Figura 2. A operação *get* solicita o valor de uma variável; *getnext* consulta o valor do objeto

com endereço[1] de valor imediatamente superior ao informado (em ordem lexicográfica); *getbulk* consulta um bloco de objetos de gerência, sendo possível a utilização de *wildcards*, que permitem generalizar endereços truncados; *set* permite ao gerente atualizar valores de variáveis; *inform* permite que um gerente informe a outro gerente quais variáveis está gerenciando; *trap* é uma mensagem enviada de um agente para um gerente quando ocorre um evento extraordinário; *response* é a mensagem de resposta a uma consulta e, por fim, *report* gera notificações internas, dentro da própria entidade, mas entre engenhos diferentes.

Operação	Descrição
Get	Solicita o valor de uma variável
Getnext	Consulta o objeto imediatamente posterior
Getbulk	Consulta um conjunto de objetos de gerência
Set	Armazena um valor em uma variável específica
Inform	Propaga objetos de gerência entre entidades
Response	Resposta às operações descritas acima
Trap	Notifica uma entidade da ocorrência de um evento extraordinário
Report	Notificação interna à entidade

Figura 2: Operações do Protocolo SNMPv3.

2.2.3.2 Interações entre Entidades

Há três tipos de interações entre entidades SNMP, conforme mostra a figura 3 [13].

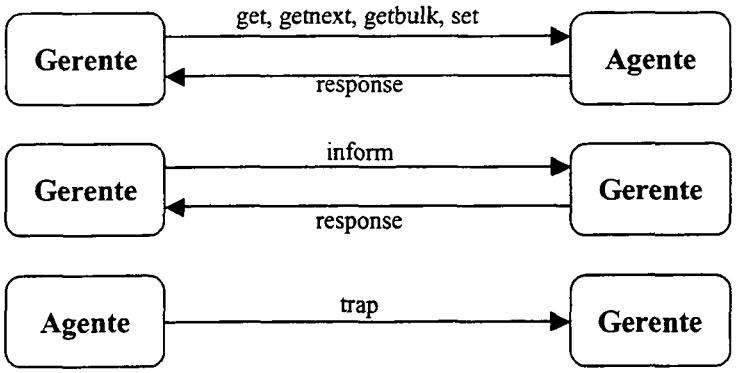

Figura 3: Interações entre entidades.

[1] O endereçamento dos objetos de gerência será descrito na seção 2.3.1.

Na primeira interação, gerentes enviam requisições para agentes e recebem as respostas correspondentes. Na segunda, gerentes enviam notificações para gerentes e recebem confirmações. Finalmente, no terceiro tipo de interação, agentes enviam notificações para gerentes sem esperar qualquer confirmação.

2.2.3.3 Formato dos PDU's

Um PDU pode ser classificado de acordo com suas propriedades funcionais em vários tipos [4]. Entre eles estão os PDU's para leitura e escrita, ilustrados na figura 4.

Tipo de PDU	Identificação da Requisição	NULL	NULL	Interligação de Variáveis (variable-bindings)

Figura 4: PDU para Get, GetNext e Set

O primeiro campo identifica o tipo do PDU, informando a operação do protocolo. O segundo campo é o índice do PDU. Os dois campos seguintes são preenchidos com valores nulos, pois conterão as respostas à consulta após o processamento da requisição. Por fim, há uma lista, denominada *variable-bindings*, que contém os nomes das variáveis envolvidas na requisição e seus respectivos valores. O formato de *variable-bindings* é mostrado na figura 5.

$Nome_1$	$Valor_1$	$Nome_2$	$Valor_2$	...	$Nome_n$	$Valor_n$

Figura 5: Interligação de Variáveis

O PDU de resposta, mostrado na figura 6, contém respostas à requisições e difere dos PDU's de leitura e escrita no terceiro e quarto campos.

Tipo de PDU	Identificação da Requisição	*Status* de erro	Indexador de erro	Interligação de Variáveis (variable-bindings)

Figura 6: PDU para Response

O terceiro campo indica se houve ou não um erro e, caso positivo, informa a sua natureza. O campo indexador de erro informa o índice da variável, em *variable-bindings*,

na qual o erro ocorreu. Maiores informações sobre os outros tipos de PDU's estão disponíveis em [14, 15] e sobre o formato da mensagem SNMP versão 3 em [2].

É importante lembrar que o PDU não é encapsulado diretamente na área de dados do datagrama UDP. O PDU está contido numa mensagem SNMP, que por sua vez está contida na área de dados do datagrama UDP. A mensagem SNMP possui um cabeçalho, onde estão informados a versão do SNMP utilizada e a comunidade em uso. A comunidade faz parte do controle de acesso do SNMP.

2.3 Management Information Bases

Cada item de informação relevante para o gerenciamento, como as informações de gerência referentes ao estado ou estatísticas de utilização de interfaces, tráfego de entrada e saída de pacotes, erros ocorridos, pode ser implementado como um objeto componente da MIB. A definição desses objetos deve ser independente do protocolo de gerenciamento utilizado e deve permitir que uma determinada MIB seja expandida, com a inclusão de novos objetos, a critério do administrador, ou compactada, excluindo objetos desnecessários para uma determinada aplicação.

Diversas MIB's são padronizadas, como a chamada *mib2*, para *hosts* da Internet. Como exemplo de objetos componentes da *mib2* estão o objeto chamado *sysUpTime* que é o tempo aferido desde a última inicialização do agente monitorado, *ipInReceives* que corresponde ao número de datagramas IP recebidos e o objeto chamado *tcpInSegs* que é o número de segmentos TCP recebidos pelo agente. Os objetos mencionados nada mais são do que contadores e podem ser implementados como variáveis simples. Dados que podem ser implementados como variáveis simples (por exemplo, um inteiro, um *string*, um endereço IP...), serão chamados de dados escalares.

Os dados escalares podem ser agrupados de modo a formar estruturas mais complexas como tabelas. O objeto chamado *ipRouteTable*, por exemplo, contém a tabela de roteamento IP e é definido por uma seqüência de agregações de objetos escalares, agregações estas, que constituem as entradas da tabela.

O agente SNMP deve promover a integração entre as variáveis da MIB e o sistema operacional ou outro sistema ou ainda controlador de dispositivo gerenciado, implementando a coleta das informações de gerência e executando ações.

2.3.1 Nomes de Objetos

O espaço de nome dos objetos de gerência é administrado pela ISO e pela *International Telecomunication Union* (ITU – antiga CCITT). A referência a um determinado objeto é feita por identificadores de objetos (*Object Identifier* – OID) que são únicos. Para permitir a existência de nomes absolutos, o espaço de nome do identificador é hierárquico. Sendo assim, a autoridade para atribuição de nomes é dividida por níveis, permitindo subníveis de modo a formar uma árvore.

Cada nó no espaço de nome é identificado por uma *string* e, ao mesmo tempo, por um número inteiro de forma a fazer uma identificação redundante. A *string* serve para facilitar o entendimento do nome por pessoas e o inteiro é a referência utilizada na implementação. Há um mapeamento de *strings* para inteiros, de maneira similar à usada pelo *Domain Name Server* (DNS). Este mapeamento é executado por um software que usa tabelas de equivalência.

A raiz desta árvore não possui nome, mas possui três sub-níveis que são geridos pela ISO, pela ITU e pela ISO e ITU em conjunto, respectivamente. A ISO alocou uma sub-árvore para organizações de padrões e o *U. S. National Institute for Standards and Technology*, por sua vez alocou uma sub-árvore para o Departamento de Defesa dos Estados Unidos (DOD – *Department of Defense*). A estrutura é ilustrada na figura 7.

O nome de um objeto é a seqüência de rótulos nos nós no caminho da raiz para um nó folha. Os componentes do nome são separados por pontos. Por exemplo, *1.3.6.1.2.1.1.1* é um objeto pertencente ao nó *mib2* que contém a descrição do sistema utilizado. É possível fazermos uma referência ao mesmo objeto usando rótulos, obtendo, no caso do exemplo descrito nas linhas acima, o nome:

iso.org.dod.internet.mgmt.mib2.system.sysDescr

Nas mensagens SNMP, os nomes utilizam um sufixo em anexo. No caso das variáveis escalares, "0" é a instância da variável com aquele nome. Assim, para manipularmos uma variável que contém uma descrição do sistema em uso, por exemplo, precisamos nos referir a ela como:

1.3.6.1.2.1.1.1.0

ou ainda *iso.org.dod.internet.mgmt.mib2.system.sysDescr.0*. Já para os dados tabulares, o sufixo serve para endereçar a entrada correspondente na tabela.

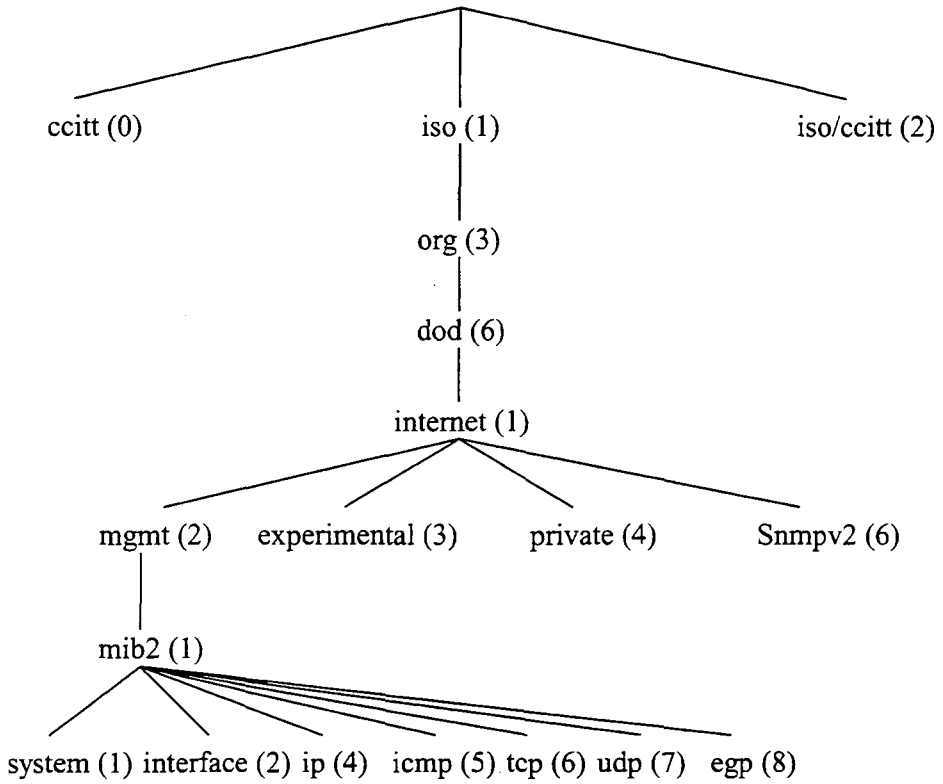

Figura 7: Árvore do espaço de nome

2.3.2 Representação de Dados em ASN.1

Para definir estruturas de dados utilizadas pelo protocolo SNMP de maneira independente da linguagem de programação e da arquitetura da plataforma utilizada, um formalismo chamado *sintaxe abstrata* é usado para representar dados. Para este propósito é usada uma linguagem definida pela ISO, a *Abstract Syntax Notation One* (ASN.1) [16], com algumas extensões introduzidas em [17].

A linguagem ASN.1 fornece uma notação precisa e formal, permitindo uma representação das estruturas de dados compacta e sem ambigüidade. Ela é utilizada com dois objetivos distintos, no SNMP, que consistem em definir o formato dos PDU's trocados pelo protocolo de gerência e definir os objetos gerenciados.

Uma coleção de especificações em ASN.1 é denominada *módulo*. A sintaxe básica de um módulo é mostrada a seguir:

```
<módulo> DEFINITIONS ::= BEGIN
<referências>
<declarações>
END
```

O símbolo *<módulo>* corresponde ao nome do módulo definido. O símbolo *<referências>* permite que o implementador se utilize de declarações feitas em outros módulos localizados em uma biblioteca. Assim, os módulos podem exportar definições para serem utilizadas por outros módulos, os quais devem importar as referidas definições. Finalmente, *<declarações>* contém as definições ASN.1 do módulo corrente.

Três grupos de elementos são definidos usando ASN.1: tipos, valores e macros. Tipos definem tipos de dados básicos, além de novas estruturas de dados. O nome de um tipo começa sempre com uma letra maiúscula (ex. *Gauge*). Um valor é uma instância de um tipo, ou seja, uma variável e seus nomes sempre começam com uma letra minúscula (ex. *sysDescr*). As macros redefinem a gramática usual da ASN.1 e seus nomes são grafados em letras maiúsculas (ex. *OBJECT-TYPE*).

As palavras reservadas da ASN.1 devem estar em maiúsculas.

Para importar definições feitas em outro módulo o comando *IMPORTS* é usado, que identifica o módulo onde foi feita a definição e o rótulo que a identifica, chamado descritor. Este descritor tem como prefixo o nome do módulo ao qual pertence, separado por um ponto, ex.:

módulo.descritor

Os tipos em ASN.1 são definidos usando a seguinte sintaxe:

```
<NomeDoTipo> ::=
        <TIPO>
```

Valores, ou instâncias de um tipo de dado são definidos como:

```
<nomeDoValor> <NomeDoTipo> ::=
        <VALOR>
```

O *framework* de gerência faz uso de quatro grupos de tipos ASN.1, a saber: tipos básicos, tipos agregados, tipos rotulados e subtipos.

Os tipos básicos são *INTEGER* (inteiro), *BIT STRING, OCTET STRING* e *OBJECT IDENTIFIER. BIT STRING* é um *array* de bits, *OCTET STRING* é um *array* de octetos, com valores de 0 a 255 e *OBJECT IDENTIFIER* é um tipo especialmente concebido para suportar os nomes de objetos mostrados na seção anterior.

Ainda em relação ao tipo *INTEGER*, muitas vezes é conveniente associar nomes a valores, como na enumeração a seguir:

```
Status ::=
        INTEGER { up(1), down(2), testing(3) }

myStatus Status ::= up
```

Os tipos agregados são denominados *SEQUENCE*, que denota uma agregação de zero ou mais elementos, de qualquer tipo ASN.1, que é similar a "*struct*" em algumas linguagens de programação e *SEQUENCE OF <TIPO>*, que constitui um arranjo de zero ou mais elementos de um mesmo tipo ASN.1.

Sub-tipos são definidos a partir de tipos já existentes com restrições, ou seja, sub-tipos são subconjuntos de outros tipos já definidos, ex.

```
IpAddress ::=
        OCTET STRING (SIZE (4))

Counter32 ::=
        INTEGER (0..4294967295)
```

No primeiro exemplo, o subtipo *IpAddress* é representado por tipos *OCTET STRING*, cujo comprimento é exatamente 4. No segundo exemplo, *Counter32* é definido como inteiro não negativo menor que 2^{32}. Os tipos rotulados não serão abordados.

Ainda no que tange a ASN.1, macros são definidas usando a seguinte sintaxe:

```
<nomeDoValor> MACRO
        <cláusulas>
        ::= <VALOR>
```

onde *<nomeDoValor>* é o nome da macro e *<cláusulas>* e *<valor>*, dependerão das definições da macro.

2.3.3 Structure of Management Information

A chamada SMI (*Structure of Management Information*) [18, 19, 20], definida a partir de um subconjunto da ASN.1, define regras para descrever a informação gerenciada, de maneira independente dos detalhes de implementação, definindo um esquema para uma *Management Information Base* (MIB).

Embora as definições de objetos de gerência sejam feitas utilizando a SMI, é relevante citar que em [21] foi apresentada uma abordagem para especificação formal da semântica do objeto, usando o formalismo de Semântica de Ações [22, 23].

2.3.3.1 Módulos de Informação

Um grupo de módulos de informação da ASN.1, definidos usando a SMI, particularmente importante para este estudo são os módulos MIB.

Cada módulo de informações deve começar com a sua identificação e seu histórico de revisões. Para isto, a SMI define uma macro especial denominada *MODULE-IDENTITY*, definida abaixo:

```
MODULE-IDENTITY MACRO ::=
BEGIN
        TYPE NOTATION ::=
                "LAST-UPDATED" value(Update UTCTime)
                "ORGANIZATION" Text
                "CONTACT-INFO" Text
                "DESCRIPTION" Text
                RevisionPart

        VALUE NOTATION ::=
                Value(VALUE OBJECT IDENTIFIER)

        RevisionPart ::=
                Revisions
                | empty
        Revision ::=
                Revision
                | Revisions Revision
        Revision ::=
                "REVISION" value(Update UTCTime)
                "DESCRIPTION" Text
```

```
        Text ::= """" string """"
END
```

Cada macro é definida em três partes: *TYPE NOTATION*, que define a sintaxe das cláusulas, *VALUE NOTATION*, que define o tipo de dado do valor a ser atribuído e uma coleção de produções auxiliares. *TYPE NOTATION* desta macro define uma sintaxe com quatro cláusulas: *LAST-UPDATED*, *ORGANIZATION*, *CONTACT-INFO* e *DESCRIPTION*. A seguir é ilustrada uma chamada da macro *MODULE-IDENTITY*:

```
ufprEventMIB MODULE-IDENTITY
        LAST-UPDATED "200130030000Z"        -- 30 March 2001
        ORGANIZATION "UFPR Universidade Federal do Parana"
        CONTACT-INFO "Henrique Denes Fernandes
                      Universidade Federal do Parana
                      Departamento de Informatica
                      Centro Politecnico s/n
                      Curitiba PR 81.531-970.
                      Phone: +55 41 361 3028
                      Email: denes@cce.ufpr.com.br"
        DESCRIPTION
                      "The MIB module for defining event triggers and actions
                       for network management purposes."
-- Revision History
        REVISION    "200130030000Z"        -- 30 March 2001
        DESCRIPTION " "   ::= { mib-2 99 }
```

No exemplo acima, o módulo *ufprEventMIB* está sendo definido para ter o valor "{ mib-2 99 }". Este valor, um *OBJECT IDENTIFIER*, é usado para identificar o módulo na árvore de espaço de nomes.

A primeira cláusula contém uma variável do tipo *UTCTime*, um *timestamp* definido pelo padrão ASN.1. Neste exemplo, "200130030000Z" refere-se a 30 de março de 2001. As outras três cláusulas são texto simples e descrevem respectivamente a instituição proprietária do módulo, dados do implementador e uma explicação textual do módulo de informação.

Após estas quatro cláusulas, há mais duas descrevendo cada revisão: *REVISION* e *DESCRIPTION*, sendo que a primeira é uma variável do tipo *UTCTime* e a segunda contém texto simples.

2.3.3.2 Definições de Objetos

A SMI define os *OBJECT IDENTIFIER's* que serão empregados no *framework* de gerência. O prefixo do rótulo *internet* na árvore de espaço de nome é definido como:

internet *OBJECT IDENTIFIER ::= { iso 3 6 1 }*

que pode ser interpretado como *1.3.6.1.*

Observando a seqüência de definições abaixo se pode concluir que atribuímos, imperativamente nomes aos ramos da árvore de espaço de nome, desde a raiz até o rótulo *mib2*, conforme podemos comparar com a estrutura mostrada na figura 7:

internet *OBJECT IDENTIFIER ::= { iso 3 6 1 }*
mgmt *OBJECT IDENTIFIER ::= { internet 2 }*
mib2 *OBJECT IDENTIFIER ::= { mgmt 1}*

Sendo assim, o endereço do nó *mib2* pode ser expresso como *1.3.6.1.2.1.*

Quanto aos objetos de gerência, cada um deles é descrito usando a macro *OBJECT-TYPE*, cujas definições são mostradas a seguir:

```
OBJECT-TYPE MACRO ::=
BEGIN
        TYPE NOTATION ::=
                "SYNTAX" type(Syntax)
                UnitsPart
                "MAX-ACCESS" Access
                "STATUS" Status
                "DESCRIPTION" Text
                ReferPart
                IndexPart
                DefValPart

        VALUE NOTATION ::=
                value(VALUE ObjectName)

        UnitsPart ::=
                "UNITS" Text
                | empty
```

```
Access ::=
        "not-accessible"
        | "read-only"
        | "read-write"
        | "read-create"

Status ::=
        "current"
        | "deprecated"
        | "obsolete"

ReferPart ::=
        "REFERENCE" Text
        | empty

IndexPart ::=
        "INDEX"   "{" IndexTypes "}"
        | "AUGMENTS"   "{" Entry "}"
        | empty

IndexTypes ::=
        IndexType
        | IndexTypes "," IndexType

IndexType ::=
        "IMPLIED" Index
        | Index

Index ::=
        value(Indexobject ObjectName)

Entry ::=
        value(Entryobject ObjectName)

DefValPart ::=
        "DEFVAL"   "{" value(Defval Syntax) "}"
        | empty

Text ::= """ string """

END

ObjectName ::=
        OBJECT IDENTIFIER
```

A cláusula *MAX-ACCESS* define níveis de permissões de acesso para o protocolo. Estes níveis podem ser *read-create*, indicando que instâncias de um objeto podem ser lidas, modificadas e criadas; *read-write*, indicando que instâncias de um objeto podem ser lidas ou modificadas, porém não podem ser criadas; *read-only*, indicando que instâncias de um objeto podem ser lidas, mas não podem ser modificadas nem criadas; *not-accessible*, indicando que instâncias de um objeto não podem ser lidas, modificadas nem criadas diretamente.

O *STATUS* de uma definição de objeto pode ser *current*, *deprecated* ou *obsolete*. O *status current* indica que a definição está sendo utilizada atualmente; *deprecated* indica que a definição está em vias de se tornar obsoleta e não está necessariamente implementada; *obsolete* indica que o nó gerenciado não implementa o referido objeto. Definições que se tornaram obsoletas ainda são mantidas por razões de compatibilidade com versões anteriores.

A cláusula *DESCRIPTION* possui uma descrição textual do objeto gerenciado, onde sua semântica será especificada. Esta descrição é informal e deve ser implementada pelo programador na linguagem escolhida. Na abordagem feita em [21], esta descrição passa a ser formal, fazendo uso do formalismo de Semântica de Ações, permitindo uma especificação precisa e não ambígua, que muito contribui para a compreensão e portabilidade de um módulo.

As cláusulas *INDEX* e *AUGMENTS* são mutuamente exclusivas e indicam como instâncias de um objeto são identificadas, como por exemplo, identificar as diversas entradas de uma tabela. Por fim, a cláusula *DEFVAL* permite a utilização de um valor padrão, que será atribuído a uma instância de um objeto criado. A seguir, é mostrado um exemplo de como a macro *OBJECT-TYPE* é utilizada.

```
expExpressionValueType OBJECT-TYPE
        SYNTAX      INTEGER { counter32(1), unsigned32(2), timeTicks(3),
                    integer32(4), ipAddress(5), octetString(6), objectId(7) }
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
                "The type of expression value."
        DEFVAL      { counter32 }
        ::= { expExpressionEntry 3}
```

O exemplo acima define o objeto *expExpressionValueType* e sua sintaxe é um subconjunto dos inteiros de 1 a 7, com permissões de acesso para leitura e modificação, atualmente implementado no módulo que o contém. Sua semântica é definida como sendo o objeto que contém o tipo de valor de retorno de uma expressão e o valor padrão para uma instância criada é "1" ou *counter32*.

2.3.3.3 Sintaxe de Objetos

Os objetos de gerência possuem uma sintaxe definida por tipos de dados ASN.1. Estes tipos podem pertencer a três grupos: *básicos*, que são quatro tipos primitivos da ASN.1; *orientados a aplicação*, que são tipos de dados especiais, definidos pela SMI e *simplesmente agregados*, que são agregações construídas com os tipos *SEQUENCE* e *SEQUENCE OF*.

Os tipos básicos são *INTEGER, OCTET STRING, OBJECT IDENTIFIER* e *BIT STRING*. A SMI redefine o tipo de dados *INTEGER*, de modo que a quantidade referida possa ser representada em 32 bits, definindo o tipo *Integer32*.

Entre os tipos de dados orientados a aplicação definidos pela SMI estão *IpAddress, Counter32, Gauge32, TimeTicks, Counter64* e *Unsigned32*. O primeiro tipo é definido para representar endereços IP. O tipo *Counter32* representa um inteiro não negativo, que será incrementado de um em um, até atingir o valor máximo $(2^{32} - 1)$, quando então retorna ao valor zero. Este tipo implementa contadores, os quais somente serão significativos se forem coletadas duas amostras de valores. Somente a diferença entre estes valores é significativa. Dividindo a diferença pelo intervalo de tempo entre as amostras coletadas, obtemos uma taxa referente ao objeto amostrado. Este tipo de amostragem chama-se *delta*. Para este tipo de objetos, é fundamental que não sejam introduzidas descontinuidades em seus valores, de maneira que o atributo *MAX-ACCESS* destes objetos seja sempre *read-only*.

O tipo *Gauge32* representa um inteiro não negativo, o qual pode ser incrementado ou decrementado, nunca excedendo o valor máximo $(2^{32} - 1)$. Ao atingir o valor máximo, este contador pára. O tipo *TimeTicks* é um inteiro não negativo que computa tempo em centésimos de segundos. O tipo *Counter64* é um contador com precisão de 64 bits e é usado somente quando um contador *Counter32* atingir o valor máximo em menos de uma hora. O tipo *Unsigned32* é um inteiro com 32 bits de precisão e é empregado para valores sempre não-negativos.

Quanto aos tipos simplesmente agregados, a SMI define dois tipos: *linha* e *tabela*.

Linha é um tipo da forma:

```
<linha> ::=
        SEQUENCE {
                <tipo 1>,
                ...
                <tipo N>
        }
```

onde cada *<tipo i>* é um tipo primitivo. Esta agregação é usada para formar uma linha em uma tabela.

Tabela é um tipo da forma:

```
<tabela> ::=
        SEQUENCE OF
                <linha>
```

Uma tabela, quando completamente instanciada consiste de zero ou mais linhas, cada linha contendo um mesmo número de colunas.

2.3.4 A Mib2 da Internet

A fim de permitir um melhor entendimento de como os objetos das MIB's estão distribuídos e como eles podem ser úteis no processo de gerência, a figura 8, extraída de [24], relaciona os grupos de componentes da *mib2*, que é a MIB padrão do SNMP na Internet, em sua segunda versão.

Os grupos de componentes que integram a *mib2* são *System, Interfaces, AT, IP, ICMP, TCP, UDP, EGP, Transmission* e *SNMP*. Cada grupo corresponde a um ramo no espaço de nome, localizado abaixo do nó *iso.org.dod.internet.mgmt.mib2* e agrupam os objetos da MIB2, distribuídos de acordo com suas funções.

Conforme mostrado na tabela da figura 8, os objetos que compõem o grupo *System* armazenam o nome, localização e descrição do equipamento utilizado; o grupo *Interfaces* reúne objetos com informações acerca das interfaces de rede e seu tráfego; o grupo *AT* destinava-se a conversão de endereços e atualmente está em desuso; o grupo *IP* agrega estatísticas de pacotes IP; o grupo *ICMP* possui estatísticas sobre as mensagens ICMP

recebidas; os objetos do grupo *TCP* refletem os algoritmos TCP utilizados, parâmetros e estatísticas deste protocolo; os objetos do grupo *UDP* reúnem estatísticas de tráfego do protocolo UDP; o grupo *EGP* reúne estatísticas de tráfego do protocolo de *gateway* externo (EGP); o grupo *Transmission* foi reservado para MIB's de meios físicos específicos e o grupo *SNMP* contém os objetos com as estatísticas de tráfego do protocolo SNMP.

Grupo	Nº de Objetos	Descrição
System	7	Nome, local e descrição do equipamento
Interfaces	23	Interfaces da rede e seu tráfego
AT	3	Conversão de endereços (obsoleto)
IP	42	Estatísticas de pacotes IP
ICMP	26	Estatísticas sobre as mensagens ICMP recebidas
TCP	19	Algoritmos TCP, parâmetros e estatísticas
UDP	6	Estatísticas de tráfego UDP
EGP	20	Estatísticas de Tráfego de protocolo de gateway externo
Transmission	0	Reservado para MIB's de meios físicos específicos
SNMP	29	Estatísticas de tráfego SNMP

Figura 8: Os objetos da MIB-II da Internet

Como exemplo, é possível citar o objeto *system.sysDescr*, que contém a descrição do sistema utilizado. Pelo seu radical já é possível saber que pertence ao grupo *System*. A sua funcionalidade também permite que este seja inserido somente neste grupo.

2.4 Considerações Finais

Neste capítulo foram introduzidos os conceitos mais importantes do SNMP, incluindo arquitetura, protocolo e informações de gerência, além de representação de dados, módulos de informação e representação de objetos.

Para ter acesso a uma implementação do SNMP, é recomendada ao leitor a distribuição utilizada neste estudo, disponível em [33].

Capítulo 3

Gerência de Redes Distribuída com SNMP: Expressões e Eventos

Neste capítulo, são apresentadas as MIB's do DISMAN (*Distributed Management Working Group*), em particular a *Expression MIB* e a *Event MIB*, sua arquitetura e seu funcionamento. Será ainda abordado o uso combinado destas duas MIB's.

3.1 A Proposta do DISMAN

O DISMAN (*Distributed Management Working Group*) é um grupo de trabalho que faz parte da IETF (*Internet Engeneering Task Force* – Organização responsável por padrões da Internet) e foi formado para definir um conjunto inicial de objetos gerenciados para aplicações específicas de gerenciamento distribuído.

A gerência de redes distribuída é identificada como uma possível solução para as necessidades demandadas pelo atual ritmo de crescimento das redes de computadores. Esta corrente recomenda fortemente que aplicações caracterizadas como gerentes sejam distribuídas a fim de minimizar a interação com usuários, reduzindo o consumo de recursos da rede.

Até o momento, este grupo de trabalho limitou suas atividades a aplicações de gerência distribuída, cujo protocolo empregado é o SNMP. Como trabalhos do grupo podemos citar, entre outros, as MIB's *Scheduling, Script, Remote Operations, Event,*

Expression, *Notification Log* e *Alarm*, que auxiliam as atividades de gerência. Estas MIB's são sucintamente descritas a seguir:

A *Scheduling MIB* [25] permite o agendamento de ações, que podem ser executadas periodicamente ou em uma data e hora determinadas.

A *Script MIB* [26] é usada para delegar funções de gerência para gerentes distribuídos. As funções de gerência são definidas através de *scripts* que são executados através do sistema de gerência.

A *Remote Operations MIB* [27] permite a monitoração de *hosts* através de operações como *ping* e *traceroute*.

A *Event MIB* [7] permite o monitoramento de objetos SNMP e, quando ocorrem condições programadas, é executada uma ação (evento) também programada.

A *Expression MIB* [6] permite a construção e avaliação de expressões lógicas e aritméticas, que tenham como operandos valores de objetos de gerência locais. Os resultados destas expressões ficam disponíveis como outros objetos de gerência SNMP.

A *Notification Log MIB* [28] registra objetos enviados com notificações SNMP, gerando um arquivo de *Log*.

Por fim, a *Alarm MIB* [29] manipula e armazena alarmes, auxiliando o operador a determinar quais alarmes estão atualmente ativos em um sistema, permitindo que os problemas associados a estes alarmes sejam corrigidos.

Nas próximas seções são descritas a *Expression MIB* e a *Event MIB*.

3.2 A Expression MIB

Usuários do protocolo SNMP muitas vezes precisam fazer uso de objetos de gerência que não estão implementados. Tentando evitar a criação de novos objetos que sejam utilizados poucas vezes, a *Expression MIB* [6] permite a construção e avaliação de expressões com objetos provenientes de MIB's existentes.

Os resultados de uma expressão avaliada ficam disponíveis como objetos MIB, podendo ser usados por uma aplicação de gerência de maneira direta ou serem referenciados por outra MIB, como a *Event MIB*. Estes objetos podem ainda ser usados pela própria *Expression MIB*, formando expressões mais complexas.

3.2.1 Operação da Expression MIB

A *Expression MIB* suporta três tipos de amostragem (coleta de valores) dos objetos que compõem a expressão: *valores absolutos*, *delta* e um *valor booleano* que é verdadeiro quando o valor de um objeto tiver sido modificado.

Um valor absoluto é simplesmente o valor do objeto no instante em que ele foi amostrado. A amostragem através de valores absolutos não pode ser utilizada para a avaliação de expressões que envolvam contadores, sendo que estes devem ser manipulados como um delta (diferença) entre duas amostras. Assim, contadores devem ser amostrados como deltas. A amostragem delta define coletas periódicas dos valores amostrados, além de sempre armazenar o último valor a fim de que este possa ser aplicado no cálculo da diferença entre as duas amostras. A monitoração periódica de objetos gera um constante *overhead* de consultas *snmpget* no sistema, utilizadas na coleta de valores.

Um valor booleano é verdadeiro se o valor da amostra atual for diferente do da amostra anterior, ou seja, se o valor do delta para as duas amostras for diferente de zero. O uso de *wildcards* ou curingas permite a aplicação de uma única expressão para múltiplas instâncias de um dado objeto de gerência. Para definir este tipo de expressão deve-se indicar que estão sendo usados *wildcards* e informar parte de um nome de objeto, com alguns sufixos truncados. A aplicação executa uma operação equivalente a *getnext* para obter os valores dos objetos.

Embora as definições da *Expression MIB* não tratem de nenhum mecanismo para garantir a persistência das informações programadas, uma implementação mais robusta da MIB poderia permitir o salvamento das configurações.

3.2.2 Subconjuntos da Expression MIB

Visando reduzir a complexidade da *Expression MIB* e o uso de recursos do sistema, o próprio RFC que define este padrão [6] sugere que o implementador utilize apenas os subconjuntos da especificação que forem necessários à aplicação.

O fato de não implementar suporte para *wildcards* reduz de maneira significativa o processamento exigido e a complexidade da recuperação de valores para

avaliar as expressões. Dessa maneira, somente são permitidas expressões feitas com objetos individualmente especificados e não nomes truncados que se referem a blocos de objetos.

Sendo assim, o padrão da *Expression MIB* sugere que a implementação seja feita sem *wildcards*. Mesmo sem *wildcards* a *Expression MIB* continua sendo útil para sistemas com tabelas pequenas ou tabelas dinâmicas com poucas instâncias ou ainda, quando a construção de expressões é restrita a um pequeno conjunto de objetos de interesse.

Deixar de implementar deltas reduz significativamente o processamento exigido. Entretanto as expressões que envolvem contadores se tornam inviáveis. Sendo assim, uma implementação sem deltas teria utilidade limitada em sistemas que não utilizam expressões com contadores.

O protótipo da *Expression MIB* implementado neste projeto está descrito na seção 5.2.

3.2.3 Estrutura da Expression MIB

A *Expression MIB* possui as seguintes seções: *Resource, Definition* e *Value.* A seção *Resource* trata do uso dos recursos do sistema. A seção *Definition* é onde as expressões são definidas e a seção *Value* contém os resultados das expressões avaliadas.

A seção *Resource* possui os objetos que gerenciam os recursos utilizados por expressões com *wildcards* ou deltas e é, potencialmente, a maior consumidora de CPU e memória.

A seção *Definition* contém as tabelas que definem as expressões. A tabela de expressões contém, para cada entrada, uma expressão, além de outros parâmetros, como o tipo de dado retornado por esta expressão e o intervalo de amostragem se a expressão contiver deltas ou valores booleanos, que sejam verdadeiros em caso de variação do valor do objeto. A tabela de objetos contém os parâmetros que se aplicam a cada objeto, individualmente, como nome do objeto, tipo de amostragem e um indicador que informa se estão sendo utilizados *wildcards* ou não.

A seção *Value* contém os valores das expressões avaliadas. A tabela de resultados contém uma relação que corresponde aos tipos de resultado, dispostos um tipo por

coluna. Para uma dada expressão, somente uma coluna será instanciada dependendo do tipo de valor de retorno da expressão, configurado na tabela de expressões.

3.2.3.1 A Tabela de Expressões

A expressão a ser avaliada é uma *string*. As variáveis envolvidas na construção de uma expressão são os próprios objetos de gerência e são expressas como um símbolo dólar ('$'), seguido de um inteiro, que serve para indexar o objeto referido na tabela de objetos. Um exemplo de expressão válida é ($1 - $5) * 100, que corresponde a diferença dos valores do primeiro e do quinto objetos da tabela de objetos, multiplicada por 100.

As expressões não podem ser recursivas, porém uma expressão já em uso pode usar resultados de outra expressão, desde que a segunda não possua nenhuma variável que seja o resultado direto ou indireto da sua própria avaliação.

Os únicos operadores permitidos são os delimitadores de escopo '(' e ')', o operador unário indicador de valor negativo '-', adição '+', subtração '-', multiplicação '*', divisão '/', resto da divisão inteira '%', conjunção bit-a-bit '&', disjunção bit-a-bit '|', ou exclusivo bit-a-bit '^', deslocamento para a esquerda '<<', deslocamento bit para a direita '>>', complemento bit-a-bit '~', complemento '!', conjunção '&&', disjunção '||', igualdade '==', desigualdade '!=', maior que '>', maior ou igual '>=', menor que '<' e menor ou igual '<=', cuja semântica é a mesma da linguagem C [39]:

Os únicos tipos permitidos para constantes são inteiro de 32 bits, inteiro de 64 bits, hexadecimal, caractere, *string* e OID. Uma constante *string* deve ser representada entre aspas duplas.

Quanto aos tipos de dados das variáveis, estes são os mesmos tipos suportados pelo SNMP. Os tipos *OCTET STRING* e *OBJECT IDENTIFIER* são tratados como vetores unidimensionais de inteiros de 8 bits sem sinal e inteiros de 32 bits sem sinal, respectivamente. O tipo *IpAddress* é tratado como um inteiro sem sinal de 32 bits. Para exemplificar, a versão hexadecimal de 255.0.0.0 é 0xFF000000, que pode ser representado em 32 bits. Expressões condicionais resultam em um *Unsigned32*, com valor 0 para falso e um valor diferente de 0 para verdadeiro.

A seguir são mostradas as regras para definir o tipo de dado resultante da avaliação de uma expressão, com base no operando:

- Para deslocamentos bit-a-bit à esquerda e à direita o tipo resultante é o mesmo do operando esquerdo.
- Para conjunção e disjunção lógicas, igualdade, desigualdade, menor que, menor ou igual, maior que ou maior ou igual o tipo do resultado será sempre *Unsigned32*.
- Para o operador unário, indicador de valor negativo, o resultado sempre será *Integer32*.
- Para adição, subtração, multiplicação, divisão, resto da divisão inteira, conjunção bit-a-bit, disjunção bit-a-bit e ou exclusivo o resultado é apurado de acordo com as seguintes regras, em ordem de prioridade:
    1. Se o tipo do operando esquerdo e direito for o mesmo, o resultado será deste tipo.
    2. Se qualquer operando for *Counter64*, o resultado será *Counter64*.
    3. Se qualquer operando for *IpAddress*, o resultado será *IpAddress*.
    4. Se qualquer operando for *TimeTicks*, o resultado será *TimeTicks*.
    5. Se qualquer operando for *Counter32*, o resultado será *Counter32*.
    6. Caso contrário, o resultado será *Unsigned32*.

A seguir serão mostrados quais os tipos de dados permitidos para cada operador. Qualquer combinação que não seja explicitamente definida não funcionará. Para todos os operadores são permitidos operandos de tipos *Integer32, Counter32, Unsigned32* e *Counter64*. Os operadores de adição, subtração, multiplicação, divisão, resto da divisão inteira, menor que, menor ou igual, maior que e maior ou igual suportam *TimeTicks*. Conjunção bit-a-bit, disjunção bit-a-bit e ou exclusivo bit-a-bit suportam *IpAddress*. Já os operadores de deslocamento suportam *IpAddress* somente como operando esquerdo. O operador de adição '+' pode executar a concatenação de dois *OCTET STRINGs* ou dois *OBJECT IDENTIFIERs*. Os operadores de conjunção bit-a-bit e disjunção bit-a-bit suportam *OCTET STRINGs*. Os operadores de deslocamento à esquerda e à direita suportam OCTET STRINGs à esquerda.

Entre as funções definidas estão *counter32, counter64, arraySection, average, maximum, minimum* e *exists*.A evocação de uma função se dá pelo seu nome, seguido de uma

lista de parâmetros, os quais podem ser constantes, objetos MIB, funções ou expressões. Abaixo é explicado o comportamento das funções:

- *counter32(*<inteiro>*)* – força um valor inteiro para *Counter32.*
- *counter64(*<inteiro>*)* – força um valor inteiro para *Counter64.*
- *arraySection(<array>*, <integer>, <integer>*)* – retorna uma parte de um *array,* ou seja um *OCTET STRING* ou um *OBJECT IDENTIFIER.* O primeiro argumento inteiro é o índice do início da seção e o segundo argumento inteiro é o índice do final da seção.
- *average(*<inteiro>*)* – calcula o valor médio das amostragens de um objeto com valor inteiro.
- *maximum(*<inteiro>*)* - retorna o maior valor das amostragens de um objeto com valor inteiro.
- *minimum(*<inteiro>*)* - retorna o menor valor das amostragens de um objeto com valor inteiro.
- *exists(<OBJECT IDENTIFIER>)* – Verifica se a instância indicada existe. Retorna um valor booleano. Falso indica não existir o objeto solicitado.

Ainda na tabela de expressões, há um objeto onde deve ser indicado o tipo do valor de retorno da expressão.

3.2.3.2 A Tabela de Objetos

Esta tabela contém os objetos referenciados por cada expressão. Cada expressão, na tabela de expressões, referencia um objeto pelo símbolo dólar '$', seguido do índice do objeto correspondente na tabela de objetos. Esta tabela contém o endereço de cada objeto e o tipo de amostragem a ser tomada que pode ser valores absolutos, booleanos ou deltas.

3.2.3.3 A Tabela de Resultados

Esta tabela contém os resultados das expressões avaliadas. Cada entrada da tabela possui oito campos, um para cada tipo de dado permitido: *Counter32, Unsigned32, TimeTicks, Integer32, IpAddress, OCTET STRING, OBJECT IDENTIFIER* e *Counter64.* Para cada instância da tabela, somente um destes campos será instanciado, que é aquele correspondente ao tipo de valor de retorno da expressão, definido na tabela de expressões. Se, em uma tentativa de avaliar uma expressão, um ou mais objetos necessários não estiverem disponíveis, a entrada correspondente nesta tabela não será instanciada.

3.2.4 Exemplo de Uso da Expression MIB

Como exemplo de uso da *Expression MIB*, é mostrada uma expressão para calcular a taxa de utilização de uma linha de enlace *half-duplex,* adaptada de [11]:

utilização = (ifInOctets + ifOutOctets) * 800 / intervalo / ifSpeed

O resultado da expressão é a taxa percentual de utilização da linha por segundo. O objeto *ifInOctets* contém o número de octetos recebidos pela interface, *ifOutOctets* contém o número de octetos transmitidos pela interface e *ifSpeed* contém o valor da velocidade do enlace dado em bits por segundo. O total de octetos é multiplicado por 8 para obter o total de bits na interface e por 100 para obter um inteiro percentual.

Os objetos acima referidos fazem parte da tabela *interfaces.ifTable,* onde cada instância corresponde a uma interface de rede distinta. Se a implementação da *Expression MIB* suportar *wildcards,* uma única expressão pode calcular a taxa de utilização para cada uma das interfaces. Caso contrário, será necessário uma expressão para cada interface.

Os objetos *ifInOctets* e *IfOutOctets* são do tipo *Counter32* e são contadores. Como tal, devem ser amostrados como deltas para serem significativos. O intervalo da

amostra é de 6 segundos, mas devido a questões de precisão e independência este valor será calculado como um delta de *sysUpTime*.

Para programar esta expressão na *Expression MIB*, é necessário preencher as tabelas de objetos e de expressões. Para preencher a tabela de objetos deve-se:

1. Preencher uma instância para o objeto *ifInOctets* e amostrá-lo como delta.
2. Preencher uma instância para o objeto *ifOutOctets* e amostrá-lo como delta.
3. Preencher uma instância para o objeto *sysUpTime* e amostrá-lo como delta.
4. Preencher uma instância para o objeto *IfSpeed* e amostrá-lo como valor absoluto.

Agora deve-se preencher a tabela de expressões, alocando uma instância e preenchendo-a com a expressão "($1+$2)*800/($4/100)/$3", lembrando que como *sysUpTime* é do tipo *TimeTicks*, o valor do seu delta será dado em centésimos de segundos e deverá ser dividido por 100. Ainda na tabela de expressões, precisamos informar o tipo de valor de retorno da avaliação da expressão, que deve ser inferido a partir das regras descritas na seção 3.2.3.1. Como *ifInOctets, ifOutOctets* e *ifSpeed* são do tipo *Counter32* e *sysUpTime* é do tipo *TimeTicks*, segundo as regras, o resultado da avaliação desta expressão será do tipo *TimeTicks* e deverá ser colocado na tabela de expressões. O valor resultante ficará disponível na tabela de resultados, sendo periodicamente atualizado.

3.3 A Event MIB

A *Event MIB* [7] permite a monitoração de objetos MIB em um sistema local ou remoto e toma ações básicas quando uma condição de disparo for atingida.

A *Event MIB* depende dos serviços da *Management Target MIB* [30] e da *Notification MIB* [30] e ainda pode ser complementada pela *Expression MIB*, a qual fornece os objetos a serem monitorados.

A *Event MIB* possui quatro seções: *triggers*, objetos, eventos e notificações. Os *triggers* definem as condições que acionam os eventos, que por sua vez geram notificações.

Uma tabela de *triggers* define os objetos a serem monitorados e define como relacionar cada *trigger* a um evento. A seção de objetos define alguns objetos que podem ser enviados junto com uma notificação. Uma tabela de eventos determina o que deverá acontecer quando um evento for disparado. O disparo de um evento pode gerar o envio de uma

notificação, a alteração do valor de um objeto de uma MIB, ou ambos. A seção de *notificações* define um conjunto de notificações genéricas a serem enviadas.

3.3.1 Operação da Event MIB

Em [7] é proposto que a implementação mais simples da *Event MIB* monitore objetos individualmente, num sistema local. Uma implementação mais robusta poderia monitorar objetos remotamente além de suportar *wildcards*, que permitem a especificação de conjuntos a serem monitorados.

O monitoramento remoto faz uso do serviço de "*tags*" da *Management Target MIB* para acessar objetos em outros sistemas. Um sistema para gerenciamento local não precisa suportar monitoração remota.

O uso de *wildcards* permite que a aplicação se utilize de operações como *GetNext* para recuperar zero ou mais instâncias de um nome de objeto truncado. Cada instância de um *wildcard* é tratada como se fosse uma entrada separada, o que quer dizer que são independentes umas das outras. Por exemplo, um ou mais objetos pertencentes a um bloco (tratados como *wildcards*), satisfazem uma condição que resulta no disparo de um evento, o qual pode alterar os valores de todos os objetos de um outro bloco (que também são tratados como *wildcards*).

Como a maioria das MIB's, a *Event MIB* não apresenta mecanismos que permitam a persistência das instruções programadas, porém uma implementação mais robusta poderia incluir esta funcionalidade.

3.3.2 A Estrutura da Event MIB

A seção *triggers* é composta pela tabelas *Trigger, Trigger Delta Table, Trigger Existence Table, Trigger Boolean Table* e *Trigger Threshold Table.* Estas tabelas são descritas a seguir.

A tabela *Trigger* contém as informações sobre os *triggers* programados. Para cada *trigger,* é possível especificar um tipo de teste a ser executado, que pode ser *boolean,*

threshold ou *existence.* Para os testes *boolean* e *threshold* o objeto monitorado deve possuir um valor inteiro. Os três tipos de teste são descritos abaixo.

O teste do tipo *existence* pode disparar eventos quando a instância do objeto monitorado se tornar presente, se tornar ausente ou ainda mudar de valor. Se um evento for acionado quando uma instância de um objeto se tornar presente, esta instância deverá deixar de existir antes que o evento possa ser evocado novamente e *vice versa.* Se um evento for acionado devido a uma mudança de valor do objeto, o mesmo evento será acionado no caso de futuras mudanças de valor, sem maiores restrições.

O teste do tipo *boolean* exige que um teste específico seja selecionado na *Trigger Boolean Table.* Se o resultado for verdadeiro, um evento será disparado. Este mesmo evento só será disparado novamente quando o valor do teste passar a ser falso e tornar a ser verdadeiro.

O teste do tipo *threshold* permite que sejam definidos alguns valores de limiares para objetos monitorados. Estes limiares podem ser de subida ou descida e a amostragem é feita em intervalos periódicos de tempo. Se o valor do objeto estiver aumentando de uma amostragem para a seguinte e o atingir o limiar de subida, será disparado um evento para o limiar de subida programado. Se o valor deste objeto estiver diminuindo de uma amostragem para a seguinte e atingir o limiar de descida, será disparado um outro evento para o limiar de descida programado. Este tipo de teste ainda permite que os valores monitorados sejam amostrados como valores absolutos ou deltas.

As demais tabelas da seção *triggers* incluem a *Trigger Delta Table,* que contém informações referentes à amostragem delta de alguns objetos monitorados; a *Trigger Existence Table,* que contém informações sobre os *triggers* que realizam teste do tipo *existence* com os objetos; a *Trigger Boolean Table,* que contém informações sobre os *triggers* que realizam testes booleanos (do tipo *boolean*) com objetos, e a *Trigger Threshold Table,* que define informações sobre os *triggers* que realizam testes do tipo *threshold..*

A seção de objetos possui apenas uma tabela que relaciona objetos a serem enviados juntos com notificações, através da *Notification MIB.* Este objeto poderá ser relacionado com o *trigger,* com o teste executado pelo *trigger* ou ainda com o próprio evento disparado.

A seção de eventos possui as tabelas *Event, Event Notification Table* e *Event Set Table.*

A tabela *Event* contém informações sobre as ações a serem executadas na ocorrência de um evento, que podem ser a geração de uma notificação do tipo *trap* ou *inform* ou ainda a alteração do valor de um objeto.

A *Event Notification Table* contém informações sobre notificações a serem geradas e a *Event Set Table* contém informações sobre eventos do tipo *set*, ou seja, que procedem com a alteração do valor de um objeto.

Por fim, a seção de notificações define um conjunto de notificações genéricas para serem acionadas por eventos, para a manipulação de erros da *Event MIB* e ainda define um conjunto de objetos para serem enviados com as notificações.

3.3.2.1 A Seção de Triggers

Na seção de *triggers*, a tabela *Trigger* define o objeto a ser monitorado, pelo seu identificador; o tipo de teste a ser executado com os valores deste objeto, que pode ser *existence, threshold* ou *boolean*; o tipo de amostragem, que pode ser através de valores absolutos ou valores delta; indicando também se o nome do objeto está completamente especificado ou se está truncado e, neste caso, devem ser utilizados *wildcards*; uma *tag*, no caso do objeto monitorado não estar disponível localmente, utilizando o serviço da *Management Target MIB,* se implementado; o intervalo entre as amostragens, dado em segundos (o intervalo definido aqui não é usado para o cálculo do delta, e sim a freqüência de execução do teste para cada *trigger*) e um controle para permitir que um *trigger* seja configurado, mas não usado.

Quando o teste escolhido for do tipo *existence*, deverá ser indicado, em *Trigger Existence Table,* se o disparo deverá ocorrer quando a instância do objeto especificado se tornar presente, ausente ou mudar de valor e o índice do evento correspondente na tabela *Event.*

Se o teste escolhido for do tipo *boolean,* deverá ser indicado, em *Trigger Boolean Table,* um valor de referência, que obrigatoriamente será um inteiro, para ser comparado com o valor do objeto monitorado; a comparação que será executada entre o valor amostrado e o valor de referência, que poderá ser diferente, igual, menor, menor ou igual, maior ou maior ou igual e o índice do evento correspondente na tabela *Event.*

Quando o teste escolhido for do tipo *threshold,* as informações necessárias ao funcionamento do *trigger* deverão estar na tabela *Trigger Threshold Table.* Maiores detalhes sobre esta tabela estão disponíveis em [7].

O disparo de um evento acontece sempre que a condição de teste especificada para o *trigger* é verificada. Cada *trigger* contém um índice que aponta o evento, na tabela *Event,* que deverá ser acionado.

3.3.2.2 A Seção de Eventos

Na seção de eventos, a tabela *Event* contém informações sobre os eventos programados. Para cada evento deve ser definido se a ação a ser tomada por ocasião do acionamento do evento será uma notificação ou a alteração do valor de um objeto (*set*). Há ainda um controle para permitir que um evento seja programado, porém não usado.

Se a ação evocada pelo evento programado for uma notificação, a *Event Notification Table*, deverá possuir informações sobre a notificação correspondente àquela entrada da tabela *Event.*

Quando a ação evocada pelo evento programado for uma alteração de valor de um objeto (*set*), a *Event Set Table*, deverá conter informações como o nome do objeto cujo valor será alterado; indicar se o nome do objeto está completamente especificado ou se está truncado e assim, faz uso de *wildcards*; o novo valor a ser atribuído ao objeto, que necessariamente deverá ser um inteiro e uma *tag,* caso o objeto não esteja disponível localmente, fazendo uso do serviço da *Management Target MIB.*

3.3.3 Exemplo

Como exemplo, é mostrado como a *Event MIB* pode ser configurada para detectar uma possível invasão do tipo *TCP Syn Flooding* [38].

Este tipo de invasão consiste em inundar um determinado *host* com segmentos TCP solicitando abertura de conexão, provocando o travamento de um ou mais processos

servidores. Quando temos uma elevação súbita no número de solicitações de abertura de conexões TCP em um computador é provável que este esteja sofrendo uma invasão.

O objeto *tcpPassiveOpens* é um contador que registra o número de vezes que a máquina de estados TCP passa do estado *LISTEN* para *SYNRCVD* [8 - página 242] ou seja, indica o número de solicitações de abertura de conexão TCP recebidas. Como *tcpPassiveOpens* é um contador, ele deve ser amostrado como delta para ser significativo. Supondo *n* um valor limite para o delta deste objeto, a *Event MIB* deve monitorá-lo periodicamente e, quando o valor do seu delta for superior a *n*, será gerada uma notificação para um gerente e o administrador deverá então executar as consultas necessárias e tomar as devidas providências.

Para programar esta aplicação na *Event MIB* é necessário seguir as seguintes etapas, setando os objetos adequados em cada passo:

1. Na tabela *Trigger*:
   1.1. Alocar uma entrada para o objeto *tcpPassiveOpens*.
   1.2. Escolher o tipo de teste *boolean*.
   1.3. Escolher o tipo de amostragem delta.
   1.4. Informar que o nome do objeto está completamente definido e não se trata de um *wildcard*.
   1.5. Informar a freqüência da amostragem em segundos.
2. Na tabela *Trigger Boolean Table*:
   2.1. Escolher a comparação do tipo *maior*.
   2.2. Colocar *n* como valor de referência.
3. Na tabela *Event:*
   3.1. Alocar uma entrada.
   3.2. Configurar a ação como notificação.
4. Em *Event Notification Table* configurar a notificação a ser gerada.
5. Retornar a *Trigger Boolean Table*, e associar o evento a ser evocado com o índice da entrada alocada no passo 3.1.
6. Retornar a tabela *Event* e habilitar o evento.
7. Retornar a tabela *Trigger* e habilitar o *trigger*.

3.4 O Uso Combinado da Expression MIB e da Event MIB

Conforme exposto anteriormente, a *Event MIB* pode ser usada em conjunto e complementada pela *Expression MIB*, onde os objetos com resultados das expressões definidas através da *Expression MIB* são monitorados pela *Event MIB*. Com este conjunto dispomos de uma ferramenta poderosa para a monitoração de objetos. No entanto, devido a algumas dificuldades de ordem prática, estas ferramentas têm sido pouco utilizadas pelos administradores de redes. A maioria destas dificuldades consiste no complicado processo de configurar as expressões, os *triggers* e os eventos, exigindo uma série de operações *snmpset* e conhecimento detalhado acerca da arquitetura das MIB's envolvidas. Também é bastante difícil gerenciar todas as diversas tabelas das duas MIB's, exercendo controle sobre suas entradas e relacionamentos entre tabelas. Além de este processo ser trabalhoso, ele está muito sujeito à incidência de erros.

Como exemplo de uma aplicação que envolva o uso dessas duas MIB's, é citado o caso em que a taxa de utilização de um enlace é monitorada e cada vez que esta taxa ultrapassar um dado percentual *p*, será gerada uma notificação para um gerente.

A fórmula para o cálculo da taxa de utilização é a mesma mostrada na seção 3.2.4:

utilização = (ifInOctets + ifOutOctets) * 800 / intervalo / ifSpeed

A programação desta aplicação se dá da seguinte maneira:

1. Na tabela de objetos da *Expression MIB*:
   1.1. Preencher uma instância para o objeto *ifInOctets* e amostrá-lo como delta.
   1.2. Preencher uma instância para o objeto *ifOutOctets* e amostrá-lo como delta.
   1.3. Preencher uma instância para o objeto *sysUpTime* e amostrá-lo como delta.
   1.4. Preencher uma instância para o objeto *ifSpeed* e amostrá-lo como valor absoluto.
2. Na tabela de expressões da *Expression MIB*:
   2.1. Alocar uma entrada.
   2.2. Programar a expressão "($1+$2)*800/($3/100)/$4".

2.3. Informar o tipo *TimeTicks* como valor de retorno, com base nas regras para inferência de tipos.

3. Na tabela *Trigger* da *Event MIB*:
   3.1. Alocar uma entrada.
   3.2. Amostrar a entrada da tabela de valores da *Expression MIB,* que contenha os resultados da avaliação da expressão definida no item 2.2.
   3.3. Escolher o tipo de teste *boolean*.
   3.4. Escolher o tipo de amostragem por valores absolutos.
   3.5. Informar que o nome do objeto está completamente definido e não se trata de um *wildcard.*
   3.6. Informar a freqüência da amostragem em segundos.
4. Na tabela *Trigger Boolean Table* da *Event MIB*:
   4.1. Escolher a comparação do tipo *maior.*
   4.2. Colocar *p* como valor de referência.
5. Na tabela *Event* da *Event MIB:*
   5.1. Alocar uma entrada.
   5.2. Configurar a ação como notificação.
6. Em *Event Notification Table,* da *Event MIB* configurar a notificação a ser gerada.
7. Retornar a *Trigger Boolean Table,* da *Event MIB* e associar o evento a ser evocado com o índice da entrada alocada no passo 5.1.
8. Retornar a tabela *Event,* da *Event MIB* e habilitar o evento.
9. Retornar a tabela *Trigger,* da *Event MIB* e habilitar o *trigger.*

Como pode ser visto no exemplo acima, as tarefas relativas a esta operação são muitas, porém simples de serem automatizadas. Este é o objetivo da ferramenta apresentada neste trabalho, que executa estas tarefas a partir de uma especificação de alto nível.

Capítulo 4

Uma Linguagem para a Monitoração Distribuída de Objetos SNMP

Este trabalho descreve uma alternativa para facilitar a configuração das MIB's *Event* e *Expression*, elevando o nível de abstração desta atividade. Essa proposta consiste numa linguagem de programação chamada ANEMONA (*A NEtwork MONitoring Application*) que, ao ser interpretada, gera ações para configurar as MIB's.

Um programa da linguagem ANEMONA primeiramente define as entidades de gerência a serem monitoradas. Os *agentes monitorados* são quaisquer dispositivos que executem o software agente do SNMP e possuam em sua base de informações de gerência as MIB's *Event* e *Expression*.

O tradutor da linguagem ANEMONA pode ser executado em qualquer computador que possua acesso físico à rede, execute o software gerente SNMP e tenha permissões de escrita nos objetos de gerência dos agentes monitorados. Na figura 9, é apresentada uma máquina como a descrita no parágrafo anterior, rodando o Gerente B, a qual executa o tradutor da linguagem. O administrador especifica as condições de monitoração no programa. Quando o tradutor é executado, uma seqüência de operações *snmpget* e *snmpset* é gerada a fim de configurar as MIB's *Event* e *Expression*, residentes no agente monitorado especificado no programa. Os agentes monitorados, com suas MIB's devidamente programadas, passam a monitorar objetos ou expressões de objetos. Na ocorrência de um evento programado, o agente monitorado gera uma notificação (*trap*) para um gerente especificado, no caso o Gerente A ou, opcionalmente, pode executar uma operação *snmpset*,

previamente definida sobre um objeto local ou sobre um objeto remoto, fazendo uso do serviço da *Management Target MIB*.

É importante lembrar que uma mesma máquina pode implementar mais de uma entidade, de acordo com a arquitetura definida pelo administrador.

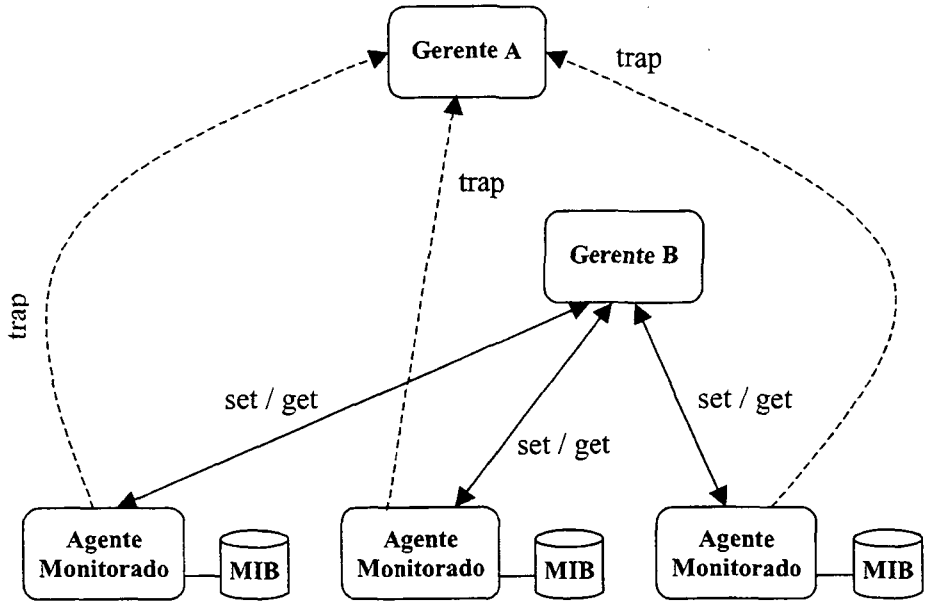

Figura 9: Uma alternativa para a utilização das MIB's Event e Expression

4.1 A Linguagem Proposta

A proposta original deste trabalho consistia em definir o comportamento das MIB's *Event* e *Expression* através da Semântica de Ações [21, 22], uma vez que existe a proposta para a definição formal da semântica de objetos MIB, usando tal formalismo [24]. A idéia inicial consistia em definir um subconjunto da Semântica de Ações como linguagem que serviria para o propósito definido no escopo deste trabalho. Entretanto, muitas adaptações seriam necessárias, os programas seriam bem mais complexos do que se fossem feitos em ANEMONA e isto exigiria que o programador dominasse a notação de ações.

Concluiu-se que a melhor alternativa seria propor uma linguagem nova, que permitisse a elaboração de programas mais simples e legíveis e cuja definição é apresentada a seguir.

4.1.1 Corpo do Programa

Todo programa na linguagem proposta possui três seções obrigatórias. A primeira delas é o comando *watch*, que indica o *host* que executa o agente a ser configurado. A especificação da comunidade a ser utilizada é obrigatória. A segunda seção contém as declarações dos objetos referidos no corpo do programa. A última seção contém o corpo do programa propriamente dito, obrigatoriamente precedido de *begin* e terminado por *end.* A estrutura básica de um programa é ilustrada abaixo.

```
watch <monitor> using <comunidade>
<declarações>
begin
        <código>
end
```

4.1.2 Declarações

É necessário declarar os objetos para que a ferramenta possa administrar os seus tipos e ainda determinar como eles são amostrados: valores absolutos (*absolute*), valores delta (*delta*) ou valor booleano verdadeiro se modificado (*modified).* A declaração de um objeto é ilustrada abaixo.

```
<OID> is <tipo>: <tipo de amostragem>
```

Sempre que um objeto for referenciado, a referência a ele se dará pelo seu nome (*OBJECT IDENTIFIER*).

4.1.3 Tipos de Dados

Os tipos de dados suportados são *Integer*, *OctetString*, *OID* (*OBJECT IDENTIFIER*), *IPAddress*, *Counter32*, *Unsigned*, *TimeTicks* e *Counter64*. Os tipos escolhidos

foram definidos a partir de um subconjunto dos tipos definidos pela SMI, onde foram aproveitados os tipos que são usados com maior freqüência nas aplicações de gerência. A tabela da figura 10 descreve estes tipos.

No caso do tipo OID, sempre que este não vier precedido de um ponto, ele é procurado na árvore do espaço de nome a partir do ramo *mib-2* (1.3.6.1.2.2). Caso o OID venha precedido de um ponto, ele será procurado a partir da raiz da árvore do espaço de nome.

Tipo	Descrição	Exemplos
Integer	Inteiros decimais negativos ou não negativos.	354 -354
OctetString	Strings de caracteres ASCII.	"Braque"
OID	Nome de objeto.	system.sysDescr.0 1.1.0 .1.3.6.1.2.1.1.1.0
IPAddress	Endereço IP.	200.17.210.163 denes.cce.ufpr.br
Counter32	Naturais de 0 a 2^{32}.	657454
Unsigned	Inteiros em módulo.	354
TimeTicks	Tempo desde a inicialização do computador, dado em centésimos de segundos.	2199
Counter64	Naturais de 0 a 2^{64}.	454687

Figura 10: Os tipos suportados pela linguagem

4.1.4 Operadores

A seguir são descritos os operadores aritméticos, relacionais, lógicos booleanos, lógicos bit a bit, de concatenação e condicionais suportados pela linguagem. Toda expressão, seja qual for a sua natureza, deverá ser limitada por um ponto e vírgula no final ';'.

4.1.4.1 Operadores Aritméticos

A linguagem dispõe dos seguintes operadores aritméticos binários: adição '+', subtração '-', multiplicação '*', divisão '/' e resto da divisão inteira '%'. Também dispõe do operador unário indicador de número negativo '-'.

Para operações aritméticas os tipos de operandos suportados são *Integer32*, *Counter32*, *Counter64*, *Unsigned* e *TimeTicks*.

O tradutor ANEMONA deve decidir o tipo de retorno de cada operação de acordo com as seguintes regras, em ordem de precedência:

1. Para o operador unário '-', indicador de valor negativo, o resultado será sempre do tipo *Integer32*.
2. Se os operadores esquerdo e direito forem de tipos iguais, o retorno será deste tipo.
3. Se um dos operadores for *Counter64*, o retorno será *Counter64*.
4. Se um dos operadores for *IPAdress*, o retorno será *IPAdress*.
5. Se um dos operadores for *TimeTicks*, o retorno será *TimeTicks*.
6. Se um dos operadores for *Counter32*, o retorno será *Counter32*.
7. Caso contrário, o retorno será *Unsigned*.

4.1.4.2 Operadores Relacionais

São suportados os seguintes operadores relacionais comparativos: igualdade '==', desigualdade '!=', maior que '>', menor que '<', maior ou igual '>=' e menor ou igual '<='. Os operandos podem ser de qualquer tipo e os resultados sempre serão do tipo *Unsigned*, com valor igual a zero caso seja falso e diferente de zero, sendo verdadeiro.

4.1.4.3 Operadores Lógicos Booleanos

Os operadores lógicos booleanos são a conjunção '*and*', disjunção '*or*' e negação '*not*', sempre grafados em caracteres minúsculos. Os operandos serão expressões que resultem um valor *Unsigned* e o retorno sempre será *Unsigned*.

4.1.4.4 Operadores Lógicos Bit a Bit

Os operadores lógicos bit a bit são conjunção bit a bit '*AND*', disjunção bit a bit '*OR*', negação bit a bit '*NOT*' e ou exclusivo bit a bit '*XOR*'. O que os diferencia sintaticamente dos operadores lógicos booleanos é o fato de serem escritos com caracteres maiúsculos. Os operandos, ambos do mesmo tipo, podem ser *IPAddress* ou OID e o tipo de valor de retorno será igual ao tipo dos operandos.

4.1.4.5 Concatenação

Para os tipos *OctetString* e OID é possível a concatenação, através do operador '+'. Ambos os operandos devem ser do mesmo tipo e o tipo do resultado será o mesmo dos operandos.

4.1.4.6 Condicionais

É suportado um operador condicional, que atribui valores a objetos. Este operador avalia constantemente uma expressão com valor de retorno booleano e, enquanto esta for verdadeira, o objeto acima referido terá um primeiro valor. Se o resultado da expressão for falso e enquanto permanecer falso, será atribuído um segundo valor ao mesmo objeto.

if <expressão> then <valor 1> else <valor 2> rec by <OID / binding>

4.1.4.7 Delimitadores de Escopo

Como delimitadores de escopo na avaliação de expressões são suportados os operadores '(' e ')'.

4.1.5 Macros

Nessa linguagem, as macros são usadas para atribuir a um nome de objeto ou ao resultado de uma expressão um identificador. Esse identificador reduz o trabalho do programador, que não precisará repetir referências longas a objetos, porém a sua função mais importante é identificar entradas da tabela de resultados da *Expression MIB*, para expressões que tenham sido programadas pela ferramenta.

Ao encerrar a tradução de um programa, a ferramenta reportará o nome de objeto que está vinculado a cada identificador de macro, a fim de que o administrador possa coletar resultados de expressões intermediárias.

bind <identificador> to <expressão>

4.1.6 Comandos Básicos

Os comandos que correspondem às ações básicas que serão tomadas na ocorrência de eventos são *set* e *notify*. *Set* atribui um valor a um determinado objeto. Neste caso, é obrigatório declarar a comunidade que será utilizada e o valor a ser atribuído deverá ser um inteiro, não sendo suportadas expressões, devido a restrições operacionais da *Event MIB* [7].

O comando *notify* enviará uma *trap* a um dado gerente. Neste caso a comunidade utilizada também deverá ser explicitada além de um objeto que identifica a natureza da *trap*.

```
set <OID> at <IP> using <comunidade> to <valor inteiro>
notify <IP> <comunidade> <OID>
```

4.1.7 Triggers

Nesta linguagem, *triggers* são implementados pelo comando *when*. Ele recebe uma expressão com resultado booleano e executa uma lista com comandos básicos dentre aqueles definidos em 4.1.6. A expressão será programada na *Expression MIB* e para cada comando da lista, será configurado, na *Event MIB*, um *trigger* para monitorar o resultado da expressão e um evento para a cada ação a ser tomada. Quando o resultado da expressão for verdadeiro, as ações correspondentes serão executadas.

```
when <expressão>
do
        <lista de ações>
end
```

4.1.8 Funções

Para serem avaliadas junto às expressões, da mesma maneira que os operadores definidos anteriormente, são suportadas as funções pré-definidas *counter32*, *counter64*, *arraysection* e *exists*.

A função *counter32*(<inteiro>) converte qualquer valor inteiro para *Counter32* e a função *counter64*(<inteiro>) converte qualquer valor inteiro para *Counter64*. A função *arraysection*(<*OctetString*>, <inteiro>, <inteiro>) retorna uma seção de um string, com início no primeiro inteiro e final no segundo.

Por fim, a função *exists*(<OID>) verifica se a instância do objeto informada existe e tem retorno do tipo *Unsigned*. Caso a instância exista, retornará um valor diferente de zero e retornará zero caso não exista.

As funções *maximum, minimum* e *average*, definidas na seção 3.2.3.1, são suportadas pela *Expression MIB* e futuramente poderão ser incorporadas à linguagem.

4.2 Exemplos de Programas

Os exemplos a seguir ilustram programas que configuram as MIB's *Event* e *Expression* para os propósitos abordados em seções anteriores, além de alguns programas que servirão para mostrar algumas das funcionalidades da ANEMONA.

O programa a seguir configura a *Expression MIB* para calcular a taxa de utilização de uma linha, com enlace *half-duplex*, conforme descrito na seção 3.2.4.

```
watch: denes.cce.ufpr.br using private
interfaces.ifTable.ifEntry.ifInOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifOutOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifSpeed.1 is Counter32: absolute
system.sysUpTime.0 is TimeTicks: delta
begin
        bind line_utilization to (interfaces.ifTable.ifEntry.ifInOctets.1 +
        interfaces.ifTable.ifEntry.ifOutOctets.1) * 800 / (system.sysUpTime.0 / 100)
        / interfaces.ifTable.ifEntry.ifSpeed.1
end
```

Quando traduzido, este programa irá configurar as MIB's localizadas na máquina "denes.cce.ufpr.br", usando a comunidade "*private*". O tipos dos objetos são declarados e *ifInOctets, ifOutOctets* e *sysUpTime* serão amostrados como delta. *IfSpeed* que contém a velocidade da interface, deverá ser amostrado como valor absoluto.

No corpo do programa está a definição da expressão, associada à macro *line_utilization*. Depois da tradução, a ferramenta irá imprimir o endereço da entrada correspondente à avaliação da expressão, na tabela de valores da *Expression MIB*, a fim de que o administrador possa coletar os seus valores.

O segundo programa configura a *Event MIB* para monitorar o objeto *tcpPassiveOpens* a fim de detectar uma invasão iminente, descrito na seção 3.3.3.

```
watch: denes.cce.ufpr.br using private
tcp.tcpPassiveOpens.0 is Counter32: delta
begin
        when tcp.tcpPassiveOpens.0 > 5
        do
                notify denes.cce.ufpr.br private tcp.tcpPassiveOpens.0
        end
end
```

Este programa irá configurar as MIB's da máquina "denes.cce.ufpr.br", utilizando a comunidade "*private*". O objeto *tcpPassiveOpens* é um contador de 32 bits e está amostrado como delta. O intervalo de amostragem para o cálculo do delta é configurado diretamente nas MIB's e, neste caso foi configurado para 6 segundos.

Se, durante o intervalo amostrado forem solicitadas mais de cinco aberturas de conexão TCP, será gerado uma *trap* para a máquina "denes.cce.ufpr.br", usando a comunidade *"private"*.

O terceiro exemplo implementa o procedimento abordado na seção 3.4, onde cada vez que a taxa de utilização da linha for superior a um dado limiar *p*, será gerada uma *trap* para um gerente especificado. Para programar esta aplicação assume-se que o limiar *p* é igual a 25 %.

```
watch: denes.cce.ufpr.br using private
interfaces.ifTable.ifEntry.ifInOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifOutOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifSpeed.1 is Counter32: absolute
system.sysUpTime.0 is TimeTicks: delta
begin
        bind line_utilization to (interfaces.ifTable.ifEntry.ifInOctets.1 +
        interfaces.ifTable.ifEntry.ifOutOctets.1) * 800 / (system.sysUpTime.0 / 100)
        / interfaces.ifTable.ifEntry.ifSpeed.1
        when line_utilization > 25
        do
                notify denes.cce.ufpr.br private line_utilization
        end
end
```

A expressão que calcula a taxa de utilização da linha é vinculada à macro *line_utilization*. O objeto associado à *line_utilization* será monitorado pelo *trigger* e, quando o seu valor for superior a 25, será gerada uma *trap* para o gerente "denes.cce.ufpr.br" contendo o objeto vinculado à macro *line_utilization*.

A fim de ilustrar algumas funcionalidades oferecidas pela linguagem ANEMONA, o quarto programa implementa a mesma aplicação do primeiro, mostrando o comportamento de macros como operandos de expressões.

São definidas quatro macros: *total_octets, seconds, den* e *line_utilization.* A macro *total_octets* define a expressão '*ifInOctets* + *ifOutOctets*', que contém o total de octetos enviados e recebidos pelo *host*; *seconds* contém o valor do intervalo de amostragem em função de *sysUpTime*, dado em segundos; *den* é o denominador da expressão e *line_utilization* contém o resultado final da taxa de utilização.

```
watch: denes.cce.ufpr.br using private
interfaces.ifTable.ifEntry.ifInOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifOutOctets.1 is Counter32: delta
interfaces.ifTable.ifEntry.ifSpeed.1 is Counter32: absolute
system.sysUpTime.0 is TimeTicks: delta
begin
        bind total_octets to interfaces.ifTable.ifEntry.ifInOctets.1 +
        interfaces.ifTable.ifEntry.ifOutOctets.1;
        bind seconds to system.sysUpTime.0 / 100;
        bind den to seconds * interfaces.ifTable.ifEntry.ifSpeed.1;
        bind line_utilization to total_octets * 800 / den;
end
```

Por fim, como exemplo de operador condicional, o quinto programa mantém o valor do objeto 98.1.1.1.1.3.9 como 3 enquanto o sistema estiver ativo por um tempo inferior ou igual a 24 horas. Quando o sistema estiver ativo por mais de 24 horas o valor de 98.1.1.1.1.3.9 será alterado para 2. Como pode ser notado, o valor de *sysUpTime* é dividido por 100, para obter o tempo de atividade do sistema em segundos e por 3600, para obter este tempo em horas.

```
watch: denes.cce.ufpr.br using private
system.sysUpTime.0 is TimeTicks: absolute
begin
        if system.sysUpTime.0 / 3600 / 100 > 24 then 2 else 3 rec by
        98.1.1.1.1.3.9 at denes.cce.ufpr.br
end
```

Capítulo 5

Implementação das MIB's Expression e Event

Com a finalidade de criar um ambiente de testes para a obtenção de resultados experimentais decorrentes do uso da linguagem proposta, foram implementadas as MIB's *Event* e *Expression*, uma vez que não foi encontrada nenhuma implementação disponível destas MIB's. Devido ao grande esforço que seria demandado na implementação completa, isto é de todos os objetos previstos nestas MIB's e considerando o escopo deste trabalho, os protótipos implementados constituem subconjuntos da *Event MIB* e da *Expression MIB*, com restrições aos modelos propostos em [6, 7], visando compatibilizar a complexidade da implementação com os resultados desejados.

No início deste estudo, ainda não haviam sido publicados os RFC's que padronizavam as MIB's *Event* e *Expression.* Os documentos consultados até então foram os *Internet Drafts* "*Distributed Management Expression MIB*" [31] e "*Distributed Management Event MIB*" [32].

Os protótipos foram, então, definidos como subconjuntos das MIB's propostas em [31, 32], e cujas definições continuam coerentes com aquelas dos RFC's 2982 (*Distributed Management Expression MIB*) [6] e 2981 (*Event MIB*) [7]. É importante mencionar que a implementação SNMP utilizada durante todo este estudo foi o NET-SNMP [33] versão 4.1.2 sob sistema operacional Linux, em plataforma *Pentium.* Para a implementação, os subconjuntos das MIB's foram definidos em ASN.1 e, a partir dessas definições, foi utilizada a ferramenta MIB2C [33] para auxiliar na geração do código.

A ferramenta MIB2C, distribuída junto com o NET-SNMP [33], é um *script* PERL que recebe definições de MIB's especificadas através do formalismo ASN.1 e gera um

código em linguagem C. O código C gerado pode ser usado como um modelo (*template*) para auxiliar o programador a implementar a MIB. O programador deve, então, editar o código C gerado, completando as funções e recompilando o agente NET-SNMP, que suportará a nova MIB.

A ferramenta MIB2C recebe como argumento um OID, que é o endereço do módulo ASN.1 a ser traduzido. O MIB2C percorre, então, a árvore de espaço de nomes até encontrar o módulo ASN.1 e gerar o código C referente ao modelo. Este modelo gerado ainda não é funcional contendo apenas a estrutura básica do código e, embora diminua de maneira significativa o trabalho de desenvolvimento, a tarefa de implementar uma MIB continua sendo bastante árdua, exigindo conhecimento e experiência do programador.

O código gerado apenas contém definições das estruturas de dados e constantes necessárias, além dos protótipos das funções para leitura e escrita. Cabe ao programador implementar todo o comportamento do objeto, declarar e inicializar variáveis utilizadas e programar as funções de leitura e escrita de valores de objetos.

Na seção 5.2 está a descrição do protótipo da *Expression MIB* e na seção 5.3 é descrito o protótipo da *Event MIB*.

5.1 Subconjunto da Expression MIB

Nesta seção, é apresentado o subconjunto de objetos da *Expression MIB* que foi implementado com o objetivo de testar a ferramenta proposta e cujas definições, na linguagem ASN.1, estão disponíveis no Anexo 1. O protótipo construído suporta os três tipos de amostragem definidos em [6]: valores absolutos, delta e valores booleanos que indicam se houve alteração do valor. Não são suportados *wildcards*.

O protótipo da *Expression MIB* é composto por três tabelas: uma tabela de expressões, uma tabela de objetos e uma tabela de resultados. Todas as tabelas possuem tamanho fixo e são indexadas por inteiros. Estas tabelas serão descritas nesta seção.

Para definir uma nova expressão, é necessário primeiramente definir todos os objetos referenciados pela expressão na tabela de objetos e depois incluir a expressão, na tabela de expressões, referenciando variáveis com o símbolo dólar '$', seguido pelo índice correspondente ao objeto na tabela de objetos.

A tabela de expressões possui quatro campos, a saber:

- *expExpressionNumber*
- *expExpression*
- *expExpressionValueType*
- *expExpressionComment*

O campo *expExpressionNumber* é o índice da tabela. As definições da expressão devem ser informadas no campo *expExpression* como *strings*. Para definir a expressão, os operadores suportados pela implementação são os delimitadores de escopo, operadores aritméticos, lógicos booleanos e relacionais, não sendo suportados os operadores bit a bit, nem concatenação.

Os tipos de dados válidos para objetos amostrados são *Integer32, Counter32, Unsigned32* e *TimeTicks*. Os tipos *OCTET STRING, OID* e *IpAddress* foram excluídos em virtude da supressão das operações bit a bit e de concatenação. Já o tipo *Counter64* foi excluído devido ao fato de poucas aplicações usarem este tipo de dado. As funções pré-definidas, mencionadas na seção 3.2.3.1 também não foram implementadas como mais uma medida que visa reduzir a complexidade deste módulo.

O campo *expExpressionValueType* define o tipo de valor de retorno da avaliação da expressão, determinando assim o campo, na tabela de resultados, onde o resultado da expressão deverá ser instanciado. O campo *expExpressionComment* contém uma *string* e é um comentário geral, descrevendo a expressão.

A segunda tabela implementada, isto é, a tabela de objetos possui três campos:

- *expObjectIndex*
- *expObjectID*
- *expObjectSampleType*

O campo *expObjectIndex* é o índice da tabela. O campo *expObjectID* contém o OID do objeto a ser amostrado e *expObjectSampleType* define a forma de amostragem. Para a amostragem delta e a amostragem através de valores booleanos que indicam modificações no valor, é executada em paralelo com o agente SNMP uma *thread* que coleta amostras do objeto especificado no campo *expObjectID* periodicamente, avalia as expressões com os valores amostrados e instancia a tabela de resultados. O valor do intervalo entre uma amostra e outra é fixo.

Por fim, a tabela de resultados possui os seguintes campos:

- *expValueIndex*
- *expValueCounter32Val*
- *expValueUnsigned32Val*
- *expValueTimeTicksVal*
- *expValueInteger32Val.*

O campo *expValueIndex* é o índice da tabela e, dos campos restantes, somente um será instanciado: aquele que corresponder ao tipo de retorno da avaliação da expressão.

5.1.1 Exemplo de Utilização

Para permitir um melhor entendimento acerca da operação e utilização do protótipo da *Expression MIB,* é mostrado a seguir um exemplo de programação deste protótipo para permitir o cálculo da taxa de utilização de uma linha, conforme descrito na seção 3.2.4. A fórmula a ser utilizada é:

utilização = (*ifInOctets* + *ifOutOctets*) * 800 / intervalo / *ifSpeed*

Primeiramente deve-se preencher, na tabela de objetos, uma entrada para cada objeto referenciado. Torna-se necessário encontrar uma entrada vazia na tabela, o que é permitido pelo comando:

```
snmptable localhost public 98.1.1.2
```

O comando acima exibe a tabela permitindo ao administrador encontrar uma entrada vazia. Os nomes de objetos serão indicados como números e os rótulos correspondentes se encontram nas definições do subconjunto da MIB, no Anexo 1.

Os objetos referenciados pela expressão deverão ser colocados na tabela, através dos procedimentos da seção 3.2.4. Supondo que a primeira entrada da tabela esteja livre, os comandos utilizados pelo administrador serão os seguintes:

```
snmpset localhost private 98.1.1.2.1.2.1 o interfaces.ifTable.ifEntry.ifInOctets.1
snmpset localhost private 98.1.1.2.1.3.1 i deltaValue
snmpset localhost private 98.1.1.2.1.2.2 o interfaces.ifTable.ifEntry.ifOutOctets.1
snmpset localhost private 98.1.1.2.1.3.2 i deltaValue
snmpset localhost private 98.1.1.2.1.2.3 o system.sysUpTime.0
snmpset localhost private 98.1.1.2.1.3.3 i deltaValue
snmpset localhost private 98.1.1.2.1.2.4 o interfaces.ifTable.ifEntry.ifSpeed.1
```

snmpset localhost private 98.1.1.2.1.3.4 i absolute

O próximo passo é configurar a tabela de expressões. Antes de tudo é necessário obter uma entrada livre nesta tabela:

snmptable localhost public 98.1.1.1

Supondo que a primeira entrada da tabela esteja vazia, será necessário inferir o tipo de retorno da expressão, com base nas regras da seção 3.2.3.1 e, então fazer uso de mais dois comandos para programar a expressão e o tipo de retorno:

snmpset localhost private 98.1.1.1.1.2.1 s "($1+$2)*800/($3/100)/$4"

snmpset localhost private 98.1.1.1.1.3.1 i timeTicks

Como a expressão foi configurada na primeira entrada da tabela de expressões e o seu resultado será do tipo TimeTicks, o resultado da avaliação estará disponível no objeto 98.1.2.1.1.4.1 que também pode ser lido como 98.1.2.1.1.expValueTimeTicksVal.1.

5.2 Subconjunto da Event MIB

O subconjunto implementado da *Event MIB* é capaz de monitorar objetos em um sistema local e possui duas tabelas que implementam as duas seções principais da *Event MIB*. As tabelas implementadas são a tabela de *triggers* e a tabela de eventos. Ambas possuem um tamanho fixo e são indexadas por inteiros. A tabela de *triggers* define os objetos a serem monitorados, bem como as condições de teste e disparo, associando cada entrada a um evento. A tabela de eventos define o que deverá acontecer quando um evento for acionado: gerar uma notificação ou alterar o valor de um dado objeto de gerência.

As definições do subconjunto da *Event MIB* em ASN.1 podem ser vistas no Anexo 2.

A tabela de *triggers* possui os seguintes campos, descritos a seguir:

- *mteTriggerIndex*
- *mteTriggerComment*
- *mteTriggerTest*
- *mteTriggerSampleType*
- *mteTriggerValueID*
- *mteTriggerEnabled*
- *mteTriggerExistenceTest*
- *mteTriggerEvent*

- *mteTriggerBooleanComparison*
- *mteTriggerBooleanValue*

O campo *mteTriggerIndex* é o índice da tabela e *mteTriggerComment* é um comentário sobre a natureza do *trigger*, no formato de *string*. O campo *mteTriggerValueID* deve conter o nome do objeto a ser amostrado e o campo *mteTriggerEnabled* permite que um *trigger* seja configurado mas não utilizado até que o valor deste campo seja verdadeiro. São suportados dois tipos de amostragem: por valores absolutos e delta. O tipo de amostragem deve ser informado no campo *mteTriggerSampleType*. No caso da amostragem delta, o valor dos intervalos é fixo.

Os tipos de teste para o valor de um dado objeto de gerência suportados pela implementação são *boolean* e *existence*. O tipo de teste ao qual o valor de um objeto é submetido deve ser informado no campo *mteTriggerTest*. Caso o tipo de teste especificado para um objeto seja *existence,* o campo *mteTriggerExistenceTest* deverá conter um dos seguintes valores: *present, absent* ou *changed*, valores estes que determinarão se o evento relacionado será disparado quando a instância do objeto informada se tornar presente, ausente ou mudar de valor.

Caso o teste especificado seja do tipo *boolean,* o campo *mteTriggerBooleanValue* deverá conter um inteiro, que será usado como valor de referência na comparação informada no campo *mteTriggerBooleanComparison* que poderá ser igual, desigual, maior, menor, maior ou igual e menor ou igual. Uma *thread* permanece coletando amostras de valores dos objetos e executando os testes especificados periodicamente, com um intervalo de tempo fixo. Finalmente, o campo *mteTriggerEvent* contém o índice do evento, na tabela de eventos, que deverá ser ativado quando a condição de disparo programada for verificada.

A tabela de eventos possui os campos a seguir:

- *mteEventIndex*
- *mteEventComment*
- *mteEventActions*
- *mteEventSetObject*
- *mteEventSetValue*
- *mteEventNotification*
- *mteEventEnabled*

O campo *mteEventIndex* é o índice da tabela e *mteEventComment* é um comentário geral sobre o evento a ser executado. No campo *mteEventActions* é informada a natureza do evento que poderá ser *notification* ou *set*. A natureza do evento determina se é gerada uma notificação ou se é alterado o valor de um objeto de gerência através da operação *snmpset*.

Caso a natureza do evento seja *notification*, o nome do objeto que é enviado com a notificação deverá estar no campo *mteEventNotification*. Caso a natureza do evento seja *set*, deverá ser informado o nome do objeto que terá o seu valor alterado, no campo *mteEventSetObject* e um valor inteiro, para ser atribuído a este objeto, que será informado no campo *mteEventSetValue*. O campo *mteEventEnabled* permite que um evento seja configurado e não usado até que o valor deste campo seja verdadeiro.

5.2.1 Exemplo de Utilização

Visando permitir um melhor entendimento acerca da operação e funcionamento deste protótipo da *Event MIB*, é demonstrada a programação da aplicação que detecta possíveis invasões do tipo *TCP Syn Flooding*, descrita na seção 3.3.3. A primeira tabela a ser preenchida é a tabela de *triggers*. É necessário encontrar uma entrada livre na tabela, utilizando o comando:

```
snmptable localhost public 99.1.1.1
```

Supondo que a primeira entrada esteja livre, o primeiro *trigger* deverá amostrar o objeto *tcp.tcpPassiveOpens.0* como delta, executar um teste do tipo *boolean* com uma comparação "maior que". Ainda deve ser suposto que o número máximo de solicitações de abertura de conexão recebidas no período de amostragem é 5, uma vez que o dado intervalo foi definido como 6 segundos, diretamente no módulo MIB. Sendo assim, o valor de referência deve ser 5. A seqüência de comandos seguinte deve ser executada:

```
snmpset localhost private 99.1.1.1.1.3.1 i boolean
snmpset localhost private 99.1.1.1.1.4.1 i deltaValue
snmpset localhost private 99.1.1.1.1.5.1 o tcp.tcpPassiveOpens.0
snmpset localhost private 99.1.1.1.1.9.1 i greater
snmpset localhost private 99.1.1.1.1.10.1 i 5
```

Os nomes dos objetos informados nos comandos são fornecidos através de números. Os valores textuais dos rótulos podem ser vistos no Anexo 2, nas definições do subconjunto da *Event MIB*. Agora a tabela de eventos deve ser configurada. É possível ler o índice de uma entrada livre através do comando:

```
snmptable localhost public 99.1.2.1
```

Supondo que a primeira entrada da tabela esteja livre, esta entrada deve ser configurada como uma notificação, que envia *tcp.tcpPassiveOpens.0* junto com a notificação. Os dois comandos abaixo executam esta tarefa.

```
snmpset localhost private 99.1.2.1.1.3.1 i notification
snmpset localhost private 99.1.2.1.1.6.1 o tcp.tcpPassiveOpens.0
```

Como o evento foi configurado na primeira entrada da tabela de eventos, o seu índice deve ser preenchido no *trigger* anteriormente configurado.

```
snmpset localhost private 99.1.1.1.1.8.1 i 1
```

Finalizando, é necessário habilitar o evento e o *trigger* configurados, o que pode ser feito pelos comandos:

```
snmpset localhost private 99.1.1.1.1.6.1 i true
snmpset localhost private 99.1.2.1.1.7.1 i true
```

5.3 Exemplo de Uso Combinado das Duas MIB's

Concluindo este capítulo, é ilustrado o exemplo em que um alarme é gerado sempre que a taxa de utilização da linha exceder um dado percentual *p*, descrito na seção 3.4. Supondo que *p* seja igual a 25 %, a primeira etapa consiste em configurar a *Expression MIB*. Para isto é suficiente repetir todos os procedimentos descritos na seção 5.1.1. O resultado da avaliação da expressão estará disponível em 98.1.2.1.1.4.1.

A segunda etapa consiste em configurar a *Event MIB*. Para preencher a tabela de *triggers,* deve-se, antes de tudo, encontrar uma entrada livre:

```
snmptable localhost public 99.1.1.1
```

Supondo que a primeira entrada da tabela esteja livre, deve-se configurar um *trigger* para amostrar o valor do objeto 98.1.2.1.1.4.1, como valor absoluto, uma vez que é o resultado da avaliação de uma expressão e é dado em valor percentual. O tipo de teste é *boolean*, acionando o evento quando o resultado da expressão for maior que 25.

```
snmpset localhost private 99.1.1.1.1.3.1 i boolean
snmpset localhost private 99.1.1.1.1.4.1 i absoluteValue
```

```
snmpset localhost private 99.1.1.1.1.5.1 o 98.1.2.1.1.4.1
snmpset localhost private 99.1.1.1.1.9.1 i greater
snmpset localhost private 99.1.1.1.1.10.1 i 25
```

Para preencher a tabela de eventos é necessário encontrar uma entrada livre e configurar uma notificação, com o objeto 98.1.2.1.1.4.1. Para isso é necessário:

```
snmptable localhost public 99.1.2.1
```

Supondo que a primeira entrada da tabela de eventos esteja livre:

```
snmpset localhost private 99.1.2.1.1.3.1 i notification
snmpset localhost private 99.1.2.1.1.6.1 o 98.1.2.1.1.4.1
```

Voltando a tabela de *triggers*, é necessário fornecer o índice do evento a ser ativado:

```
snmpset localhost private 99.1.1.1.1.8.1 i 1
```

Por fim, para ativar a aplicação, deve-se habilitar o *trigger* e o evento configurados:

```
snmpset localhost private 99.1.2.1.1.6.1 i true
snmpset localhost private 99.1.2.1.1.7.1 i true
```

Após a descrição da configuração das MIB's nesta aplicação é possível concluir que, uma vez que o tradutor ANEMONA pode gerar todas as ações envolvidas neste processo automaticamente, é muito mais trabalhoso configurar as MIB's diretamente, do que fazer uso da ferramenta.

Capítulo 6

Um Tradutor para ANEMONA

Neste capítulo é descrita a implementação de um tradutor para a linguagem ANEMONA. A ferramenta descrita recebe como entrada um programa em ANEMONA e, a partir daí, gera todas as ações necessárias para configurar e monitorar expressões e eventos na rede através da *Event* MIB e da *Expression* MIB.

Para desenvolver o tradutor foram utilizadas técnicas padronizadas para construção de compiladores, descritas em [34], incluindo as ferramentas *Lex* e *Bison*. A ferramenta *Lex* [34] gera um analisador léxico a partir de definições dos *tokens* da linguagem, feitas utilizando o formalismo de *expressões regulares* [35]. A ferramenta *Bison* [36] gera um analisador sintático (*parser*) a partir de definições da sintaxe da linguagem (regras gramaticais). Nas seções seguintes são abordados as ferramentas *Lex* e *Bison* e o protótipo do tradutor ANEMONA construído.

6.1 A Ferramenta Lex

A ferramenta *Lex* permite a geração de analisadores léxicos a partir de expressões regulares, sendo também amplamente disponível e popular. A ferramenta *Lex* permite que a especificação de padrões com expressões regulares seja combinada com ações, que permitem, por exemplo, instanciar entradas em uma tabela de símbolos, o que é exigido de um analisador léxico. O uso da ferramenta *Lex* exige que primeiramente a especificação do analisador seja feita na linguagem *Lex*. A seguir, a especificação é processada pelo compilador *Lex* a fim de gerar um programa em linguagem C. Este programa em C será

utilizado como biblioteca para o analisador sintático, para o qual fornecerá os *tokens* que compõem as sentenças. Finalmente os analisadores sintático e léxico, ambos em C, são compilados em conjunto produzindo um programa objeto.

Um programa *Lex* é constituído de três seções: declarações, regras de tradução e procedimentos auxiliares. A primeira seção contém declarações de variáveis, constantes e definições regulares. Uma definição regular é definida através de um identificador associado a uma expressão regular. Estes identificadores são usados nas regras de tradução.

A segunda seção contém as regras de tradução, que são enunciadas da forma:

p {ação}

Onde *p* é o identificador de uma expressão regular e *ação* é um fragmento de programa, em linguagem C, implementando as ações que o analisador léxico deverá executar quando reconhecer um lexema pertencente ao padrão descrito por *p*.

A terceira seção contém uma coleção de procedimentos auxiliares (em linguagem C) que são evocados pelas ações.

Um analisador léxico criado por *Lex* pode trabalhar em conjunto com um analisador sintático, sendo um procedimento do *parser*. Quando ativado pelo *parser*, o analisador léxico começa a ler a sua entrada do início do arquivo ou do ponto onde parou, um caractere de cada vez até que tenha encontrado o mais longo prefixo que seja reconhecido por uma das expressões regulares definidas. Em seguida o analisador léxico executa a ação que foi associada a esta expressão, na seção de regras de tradução. Tipicamente esta ação devolve o controle ao *parser*, mas se não o fizer, o analisador léxico prossegue, encontrando mais lexemas até que uma ação devolva o controle ao *parser*.

6.2 A Ferramenta Bison

O gerador de analisadores sintáticos mais popular é o Yacc (*Yet Another Compiler-Compiler*) [34, 37]. O Bison [36] é um gerador de *parser* projetado pela GNU, para substituir o Yacc, sendo compatível com os arquivos de entrada do Yacc. Um tradutor pode ser construído a partir de uma especificação em Bison num arquivo texto, o qual será processado pelo compilador Bison que por sua vez gera um programa C. Este programa C gerado implementa um analisador sintático ascendente [34], além de outras rotinas que o

programador pode definir. Compilando este programa em C, será gerado o código executável do *parser*.

Um programa fonte Bison é composto por três seções: declarações, regras de tradução e rotinas de suporte C. Na parte de declarações estão declaradas as variáveis que as rotinas em C usam para implementar as regras de tradução, inclusões de bibliotecas C e declarações dos *tokens* da gramática. Na seção de regras de tradução, cada regra consiste em uma produção gramatical e uma ação semântica associada. Um conjunto de produções que era escrita como:

<lado esquerdo> → <alt 1> | <alt 2> | ... | <alt n>

é escrita em Bison como:

```
<lado esquerdo>     : <alt 1> {ação semântica 1}
                    | <alt 2> {ação semântica 2}
                    ...
                    | <alt n> {ação semântica n}
                    ;
```

Lados direitos alternativos, representados acima por <alt 1>, <alt 2> ... <alt n> são separados por barras verticais e cada regra de tradução é terminada por um ponto e vírgula. O lado esquerdo da primeira regra é considerado símbolo de partida.

As ações semânticas são trechos de código em linguagem C que serão executados após o reconhecimento de um lado direito de uma produção. Se o programador fizer uso de gramáticas que contenham ambigüidades, o algoritmo de análise ascendente irá gerar conflitos de ações semânticas. A ferramenta Bison reporta ao usuário todos os conflitos ocorridos. Para estes casos, o Bison dispõe de operadores que estabelecem uma ordem de precedência para determinadas regras. Se estes operadores não forem utilizados, o Bison irá resolver um conflito do tipo empilhar/reduzir em favor de empilhar e um conflito do tipo reduzir/reduzir através da opção pelas produções conflitantes listadas por primeiro na especificação.

A terceira parte de uma especificação em Bison contém as rotinas de suporte em linguagem C. Entre estas rotinas estão procedimentos de recuperação de erros, outras rotinas auxiliares e um analisador léxico *yylex( )*. O analisador léxico retorna ao *parser* pares consistindo de um *token* associado a um valor de atributo. O *token* retornado deverá ter sido previamente declarado na seção de declarações. Este analisador léxico pode ser programado diretamente em C pelo programador ou pode ser gerado pela ferramenta *Lex*.

Para permitir que o Bison utilize analisadores léxicos gerados pelo *Lex*, é necessário, primeiramente, fazer uma especificação *Lex* dos padrões a serem reconhecidos, num arquivo de texto. Este arquivo será processado pelo compilador *Lex* e, então será gerado um código C para o analisador léxico. Depois é feita a especificação em Bison, a qual deverá ter em sua seção de rotinas de suporte C, o enunciado que inclui o arquivo C gerado pelo *Lex* como biblioteca. A especificação Bison será processada pelo compilador Bison gerando um outro arquivo com um programa C, que depois de compilado, em conjunto com o arquivo gerado pelo *Lex*, constitui um analisador léxico e sintático.

O Bison permite, ainda, a definição de regras de produção de erros. Quando uma entrada não satisfaz nenhuma regra de produção, a regra de erro será considerada e uma ação semântica para tratamento de erros será executada.

6.3 O Tradutor da Linguagem ANEMONA

As definições regulares dos *tokens* da linguagem ANEMONA, que foram utilizadas para a geração do analisador léxico, podem ser vistas no Anexo 3. A gramática ANEMONA utilizada para a geração do analisador sintático do protótipo também está disponível no Anexo 3. O tradutor é composto por módulos analisadores léxico e sintático, como pode ser visto na figura 11. Na figura, um fluxo de caracteres de entrada no primeiro módulo é convertido em uma seqüência de *tokens* associados a atributos, sempre que um dos padrões definidos no Anexo 3 for reconhecido. Ao ser processada pelo segundo módulo, esta seqüência de *tokens*, resulta na geração de ações que criam PDU's *snmpget* e *snmpset* os quais irão programar a *Event MIB* e a *Expression MIB.*

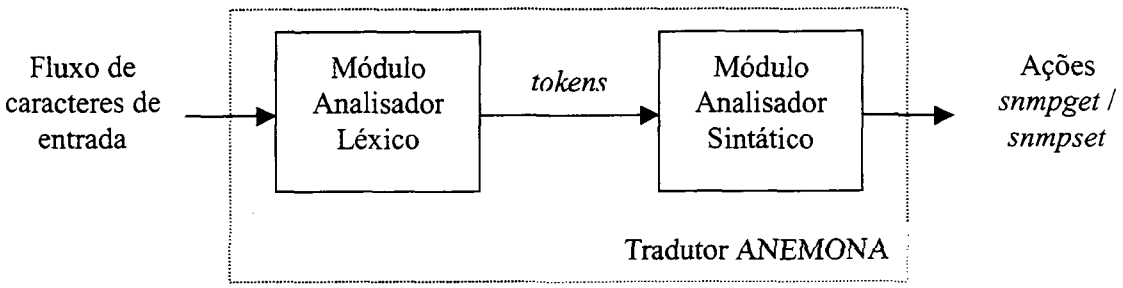

Figura 11: Estrutura do Tradutor ANEMONA com Lex e Bison

O analisador sintático recebe esta seqüência de *tokens* e, através das regras gramaticais definidas, valida sentenças e executa ações semânticas, usando como parâmetros os atributos associados aos *tokens*. No que concerne à execução das ações semânticas, existem três estruturas de dados que orientam os procedimentos: tabela de objetos, tabela de macros e uma pilha de ações.

Como foi visto em seções anteriores, existem diversas maneiras de designar um mesmo objeto, por exemplo, uma instância do objeto *sysDescr* pode ser referenciado por:

.1.3.6.1.2.1.1.1.0,

.iso.org.dod.internet.mgmt.mib2.system.sysDescr.0,

1.1.0 ou ainda

system.sysDescr.0

Para permitir que nomes diferentes se refiram a um mesmo objeto e ainda para testar se um objeto foi ou não declarado, existe uma tabela de objetos, com os campos índice, nome do objeto, tamanho do nome e tipo do objeto. Quando o tradutor lê a seção de declarações de um programa ANEMONA, para cada instância de objeto declarada, o tradutor pesquisa se não há múltiplas declarações da mesma instância, converte o nome do objeto para um tipo especial definido em linguagem C como *oid,* que é uma seqüência de índices dos nós na árvore do espaço de nome, insere este *oid* na tabela de objetos (no campo nome do objeto) e preenche uma entrada na tabela de objetos da *Expression MIB*.

Embora possa parecer redundante haver uma tabela de objetos interna ao tradutor, uma vez que existe uma tabela semelhante na *Expression MIB*, a tabela interna contribui positivamente para o desempenho da ferramenta, pois funciona como um *cache* do compilador, evitando o envio de consultas externas redundantes para recuperar valores de objetos. Quando há uma referência a uma instância de um objeto, o tradutor consulta esta tabela para confirmar se o objeto referenciado existe. Para executar esta consulta, o tradutor deve, primeiramente, converter o nome da instância do objeto para uma variável do tipo *oid*, que padroniza as referências, permitindo que em partes diferentes de um mesmo programa sejam feitas referências por nomes distintos de um mesmo objeto.

A tabela de macros é composta pelos campos nome da macro, nome do objeto, tipo do objeto e índice. Esta tabela associa um identificador de uma macro a um objeto de gerência. Suas entradas são preenchidas quando uma macro é declarada e, sempre que houver um identificador fazendo referência a um objeto, esta tabela será consultada para permitir que o nome do objeto associado seja recuperado.

A pilha de ações é utilizada pelo tradutor, sempre que este precisar interpretar um comando *when*. A pilha se faz necessária pelo fato da análise ascendente processar a gramática das ações dentro do laço *when* antes da condição de ativação deste *trigger*. Por isso, estas ações devem ser empilhadas e somente poderão ser configuradas na *Event MIB* quando todo o laço *when* for analisado.

Ao traduzir um comando *when*, as ações *set* e *notify* que estiverem definidas dentro do laço serão empilhadas, juntamente com seus parâmetros. Quando a última ação do laço *when* for processada, a condição de ativação do *trigger* será avaliada. Para cada ação empilhada será programada uma entrada na tabela de eventos da *Event MIB*, correspondente à ação *set* ou *notify* no topo da pilha (que agora será removida) e uma entrada na tabela de *triggers* da *Event MIB*, correspondente à condição de disparo. Dessa maneira, há um *trigger* programado para cada ação *set* ou *notify*, uma vez que um *trigger* somente pode acionar um único evento na *Event MIB*.

Para tradução dos comandos condicionais, são programados dois eventos *set* na *Event MIB*: um para o caso em que a condição testada é verdadeira e outro para o caso em que esta condição é falsa. A cada evento é associado um *trigger* da *Event MIB*. O primeiro *trigger* acionará um evento, que atribui um primeiro valor, quando a condição passar de falsa para verdadeira e um outro *trigger* acionará um evento, que atribui um segundo valor, quando a condição passar de verdadeira para falsa.

6.3.1 Tratamento de Erros

No que concerne ao tratamento de erros no tradutor implementado, há mensagens para erros de sintaxe, macros e objetos referenciados e não declarados, erros de comunicação com as MIB's, erros de protocolo e quando não houverem entradas livres suficiente em qualquer uma das tabelas das MIB's.

A implementação utilizou o recurso de detecção de erros de sintaxe do *Bison*, que consiste em uma regra gramatical cujo lado direito é um erro. Este recurso permite que sempre que não for possível concluir a árvore sintática até o símbolo de partida (lembrando que esta análise é ascendente) a ação associada à produção de erro será executada.

As ações tomadas na incidência de erros sintáticos são imprimir uma mensagem para o usuário, indicando a ocorrência de erro de sintaxe, o número da linha que contém o erro e o conteúdo desta linha, além de cancelar o processo de análise sintática.

Uma vez que a linguagem exige que todo o objeto referenciado seja declarado, ao avaliar uma expressão que referencie um objeto, a tabela de objetos do tradutor será consultada a fim de verificar se este está sendo amostrado e de inferir o tipo de valor resultante da expressão. Se o objeto referenciado não possuir uma entrada nesta tabela, a execução do tradutor será cancelada e o usuário será avisado que o referido objeto não foi declarado.

Ao avaliar uma expressão que referencie uma macro, o tradutor busca as suas definições na tabela de macros. Se não existir uma entrada para o identificador da macro referenciada, o usuário será informado que a macro associada aquele identificador não foi definida e a execução do tradutor é finalizada.

Quanto aos erros de protocolo, há dois tipos diferentes. Quando o tradutor envia um PDU com uma ação para o agente a ser configurado e não obtém resposta num intervalo de tempo, o usuário é informado que o agente especificado não está disponível. Quando o tradutor recebe um PDU com indicação de erro, o usuário é informado que o agente está ativo, mas que houve um erro em uma variável. Por fim, se a ferramenta não conseguir alocar entradas em qualquer uma das tabelas das MIB's *Event* e *Expression*, o usuário é avisado de que não há entradas livres disponíveis na tabela correspondente.

Capítulo 7

Estudos de Caso

Este capítulo aborda estudos de caso envolvendo situações reais de gerência de redes de computadores nas quais a ferramenta desenvolvida foi aplicada. Os estudos de caso tratam de programas ANEMONA que geram uma notificação ao administrador quando detectam um possível ataque ou a saturação da capacidade de transmissão do enlace, com base no número de pacotes IP transmitidos e recebidos.

7.1 Detecção de Ataques

Este estudo de caso consiste na utilização de programas ANEMONA que visam detectar prováveis ataques do tipo *Denial of Service* (DOS), que causam a indisponibilidade de serviços do servidor atacado. Os ataques abordados foram o *ICMP Flooding* e o *TCP Syn Flooding* [38], ambos baseados em inundação. O uso da ferramenta permitiu a detecção de ambos os ataques com sucesso.

7.1.1 Detecção de um Ataque ICMP Flooding

O ataque conhecido como *ICMP Flooding* consiste em inundar uma determinada máquina com requisições de eco do protocolo ICMP [8]. O computador atacado irá tentar responder a todas as requisições, consumindo recursos do sistema e, posteriormente tornando alguns serviços indisponíveis.

O objeto de gerência *icmp.InEchos* contém o número de requisições de eco ICMP recebidas e é do tipo *Counter32*. Com este objeto é projetada uma aplicação ANEMONA que informa ao administrador que está ocorrendo um possível ataque.

Para possibilitar a construção desta aplicação foi necessário monitorar o comportamento do objeto acima em várias situações para que, com base nos dados obtidos, fosse possível arbitrar os valores críticos utilizados no programa. Para este teste foi utilizada uma máquina com o sistema operacional *Linux*, com o agente NET-SNMP, com os protótipos construídos para as MIB's *Event* e *Expression* e com o tradutor ANEMONA; uma máquina com sistema operacional *Linux* e um computador com o sistema operacional *Windows NT*. Os computadores descritos estavam conectados em uma rede *Ethernet* e possuíam interfaces de rede de 10 Mbps.

Como o objeto *icmp.icmpInEchos* é um contador, este deve ser amostrado como delta. As amostras, todas delta, utilizando intervalo de 6 segundos, foram obtidas com o uso da *Event MIB* e da *Expression MIB* e do tradutor ANEMONA.

Durante o funcionamento normal da rede, sem requisições de eco ICMP, o valor de *icmp.icmpInEchos* manteve-se em zero. Com requisições enviadas através do comando *ping* de uma máquina, o valor do delta de *icmp.icmpInEchos* passou a ser 6 e, ao receber requisições de duas máquinas este valor passou para 12.

É possível concluir, com base no parágrafo anterior e nas definições do comando *ping*, que para cada *ping* sendo executado será gerada uma requisição de eco ICMP por segundo.

Quando o comando *ping* for evocado com o parâmetro '-f', o *host* destino será inundado com requisições, tão rapidamente quanto forem respondidas. Fazendo uso de *ping –f* foram colhidos os valores delta constantes na tabela da figura 12, em intervalos de 1 minuto.

Tempo de invasão	delta de *icmp.icmpInEchos* (intervalo de 6 s)
0 min.	3089
1 min.	4034
2 min.	4107
3 min.	4113
4 min.	4126
5 min.	4130
6 min.	4119
7 min.	4126

Figura 12: Ataque ICMP Flooding

Com base nos resultados destes testes, foi arbitrado como valor crítico 1000 para o valor do delta de *icmp.icmpInEchos*, para um intervalo de 6 segundos entre as amostras, que seria equivalente a 166 *ping's* simultâneos. Também foi observado que durante todo o ataque o delta de *icmp.icmpInEchos* foi superior a *1000*.

O programa ANEMONA a seguir implementa esta aplicação:

```
watch: denes.cce.ufpr.br using private
icmp.icmpInEchos.0 is Counter32: delta
begin
        when (icmp.icmpInEchos.0 > 1000)
        do
                notify denes.cce.ufpr.br private icmp.icmpInEchos.0
        end
end
```

Neste programa, o *host* denes.cce.ufpr.br é monitorado, tendo o objeto *icmp.icmpInEchos* de sua MIB amostrado como delta. Quando o valor deste objeto for superior a 1000, será gerada uma notificação para denes.cce.ufpr.br contendo *icmp.icmpInEchos.0*.

As ações equivalentes àquelas geradas pela tradução deste programa estão disponíveis no Anexo 4.1, onde é possível observar que são necessários 14 comandos para programar a aplicação, além das operações necessárias para encontrar entradas livres nas tabelas.

Com o computador em condições normais de operação, isto é, sem nenhum ataque sendo executado, este programa foi traduzido e, em seguida foram gerados ataques através do comando *ping –f* a partir de outra máquina. Após 4,3 segundos, em média, foram recebidas notificações no *host* monitorado.

O tempo observado acima poderia ser de até duas vezes o valor do intervalo de amostragem do delta, dependendo do valor inicial e final da amostra no período.

7.1.2 Detecção de um Ataque TCP Syn Flooding

O ataque conhecido como *TCP Syn Flooding* consiste em provocar uma falha no mecanismo de estabelecimento de conexão do protocolo TCP. Este estabelecimento de conexão é feito através de um *handshake* de três vias [8], o qual é iniciado com um cliente emitindo um segmento TCP para o servidor com o *flag Syn* setado em seu cabeçalho. Normalmente o servidor responde com um segmento com os *flags Syn* e *Ack* setados para o endereço do cliente especificado no cabeçalho IP. Por fim, o cliente envia um *Ack* para o servidor e a conexão está estabelecida.

O ataque *TCP Syn Flooding* [38] é caracterizado por inundar um dado servidor com segmentos TCP com o *flag Syn* setado em seu cabeçalho, porém com o endereço de origem no cabeçalho IP adulterado. Este endereço de origem corresponde ao de um *host* inacessível. Desta maneira, quando o servidor receber este segmento, enviará um segundo segmento com os *flags Syn* a *Ack* setados para o endereço constante no cabeçalho IP do primeiro segmento. O segmento transmitido pelo servidor não será respondido, até atingir o tempo limite, não permitindo ao TCP completar o *handshake* de três vias.

Durante um ataque, o *host* atacante envia diversas requisições *Syn* para uma ou mais portas TCP do servidor atacado. O número destas requisições deverá ser suficiente para inundar as filas de requisições, causando a indisponibilidade dos serviços executados nas portas atacadas.

Para a detecção de um provável ataque deste tipo, o objeto de gerência *tcp.tcpAttemptFails* computa o número de vezes que a máquina de estados TCP passa para o estado *CLOSED* a partir de um estado *SYN-SENT* ou *SYN-RCVD*, mais o número de vezes que passa do estado *SYN-RCVD* para o estado *LISTEN*.

Um servidor em funcionamento, aguardando conexões, permanece no estado *LISTEN* até receber uma requisição *Syn*. Ao receber esta requisição, que no caso de um ataque terá o endereço de origem adulterado para um endereço inacessível, o servidor enviará à origem um segmento com os *flags Syn* e *Ack* setados e passará ao estado *SYN-RCVD*, onde

aguardará a confirmação do segmento transmitido. Como não haverá esta confirmação, o servidor permanecerá no estado *SYN-RCVD* até atingir o tempo limite, quando então, passará para o estado *CLOSED*, provocando um incremento de *tcp.tcpAttemptFails*.

Para este estudo de caso foi utilizado o programa *Neptune* [38], que gera um número determinado de segmentos adulterados, com endereços de origem e destino e porta fornecidos pelo usuário.

O instrumental utilizado foi um computador com sistema operacional *Linux*, agente NET-SNMP, MIB's *Event* e *Expression*, tradutor ANEMONA e os servidores *Apache*, WU-FTP e *telnetd* e outro computador com sistema operacional *Linux*, com o serviço *telnetd* e com *Neptune*. Ambos estavam conectados a uma rede *Ethernet* através de interfaces de 10 Mbps.

Como o objeto *tcp.tcpAttemptFails* é um contador, este foi amostrado como delta, com intervalo de 6 segundos, usando a *Expression* MIB e o tradutor ANEMONA.

Durante o funcionamento normal da rede, foram realizadas conexões para os serviços de FTP, HTTP e *telnet*, onde o *host* monitorado agiu como cliente e também como servidor, e o valor de *tcp.tcpAttemptFails.0* manteve-se em zero, constatando que os erros assinalados pelo objeto não são freqüentes.

Foi, então, feito o primeiro ataque, contra o serviço da porta 23 (*telnet*), usando 5000 segmentos e foram colhidos os valores delta constantes na tabela da figura 13, em intervalos de 6 segundos. Após 30 segundos, o valor delta de *tcp.tcpAttemptFails* se estabilizou em 300. Ao final do ataque, o valor absoluto de *tcp.tcpAttemptFails.0* era 4887.

Tempo de invasão	delta de *tcp.tcpAttemptFails* (intervalo de 6 s)
0 s	0
6 s	0
12 s	170
18 s	300
24 s	301
30 s	300

Figura 13: Ataque TCP Syn Flooding

Com base nos resultados destes testes, foi arbitrado como valor crítico 25 para o valor do delta de *tcp.tcpAttemptFails*, para um intervalo de 6 segundos entre as amostras, que seria equivalente a quatro falhas por segundo.

O programa ANEMONA a seguir implementa esta aplicação:

```
watch: denes.cce.ufpr.br using private
tcp.tcpAttemptFails.0 is Counter32: delta
begin
        when tcp.tcpAttemptFails.0 > 25
        do
                notify denes.cce.ufpr.br private tcp.tcpAttempFails.0
        end
end
```

Neste programa, o *host* denes.cce.ufpr.br é monitorado, tendo o objeto *tcp.tcpAttemptFails* de sua MIB amostrado como delta. Quando o valor do delta deste objeto for superior a 25, será gerada uma notificação para denes.cce.ufpr.br contendo *tcp.tcpAttemptFails.0*. As ações geradas pela tradução deste programa estão disponíveis no Anexo 4.2.

Com o computador em condições normais de operação, isto é, sem nenhum ataque sendo executado, este programa foi traduzido e, em seguida foram gerados ataques idênticos ao anterior a partir da outra máquina. Em um resultado representativo, após 7 segundos foi recebida uma notificação no *host* monitorado.

O ataque foi detectado no tempo indicado acima, mas mesmo que a invasão fosse abortada logo que descoberta, isto não seria suficiente para impedir a interrupção da continuidade do serviço na porta 23. Isto se deve ao fato de que a máquina de estados TCP somente passará do estado SYN-RCVD para o estado CLOSED após o *timeout*, quando o serviço já terá sido interrompido.

7.2 Detecção da Saturação do Enlace

Este experimento consiste na geração de uma notificação quando a utilização de um enlace estiver saturada. Para simular uma saturação, a utilização do enlace foi medida com base no número de datagramas IP, sendo utilizado um programa gerador de fluxo, especialmente projetado para este fim e cujo código fonte encontra-se no Anexo 5.

Este programa transmite pacotes UDP, com um tamanho mínimo, em grande quantidade, com o objetivo de saturar o enlace. O protocolo escolhido foi o UDP porque, ao contrário do TCP, não possui algoritmos de controle de fluxo.

Os objetos de gerência *ip.ipInReceives* e *ip.ipOutRequests* contabilizam o número de datagramas IP recebidos e transmitidos, respectivamente. O número de datagramas do enlace é dado pela soma dos valores destes dois objetos.

Os recursos de *software* e *hardware* utilizados foram um computador com sistema operacional *Linux*, agente NET-SNMP, MIB's *Event* e *Expression*, tradutor ANEMONA e servidor e cliente do gerador de fluxo UDP e outro computador com sistema operacional *Linux*, com servidor e cliente do gerador de fluxo UDP, conectados a uma rede *Ethernet* através de interfaces de 10 Mbps.

Como os objetos *ip.ipInReceives* e *ip.ipOutRequests* são contadores, foram amostrados como deltas, utilizando intervalo de 6 segundos. O programa ANEMONA a seguir amostra os objetos mencionados como deltas, calcula a sua soma e atribui o resultado a uma instância da tabela de resultados da *Expression MIB*, denotada por *utilization* e cujo endereço será reportado ao usuário após a tradução do programa.

```
watch: denes.cce.ufpr.br using private
ip.ipInReceives.0 is Counter32: delta
ip.ipOutRequests.0 is Counter32: delta
begin
        bind utilization to (ip.ipInReceives.0 + ip.ipOutRequests.0);
end
```

Os dados obtidos a partir do programa acima foram coletados em duas situações distintas: sem utilização dos recursos da rede e com utilização pesada dos recursos da rede.

Sem fazer uso de aplicações que requeiram recursos da rede, os valores representativos do número de datagramas na rede, dados pela soma dos deltas de *ipInReceives* e *ipOutRequests*, registrados em intervalos de 1 minuto foram 80, 88 e 84, respectivamente.

Para garantir a utilização massiva dos recursos da rede, foram estabelecidas conexões FTP ao servidor residente no computador monitorado. Embora a elevada taxa de utilização de um enlace durante um FTP esteja diretamente relacionada com o tamanho dos datagramas IP, esta aplicação garante um tráfego contínuo de pacotes no sentido servidor-cliente e é relevante para o experimento.

Primeiramente foi estabelecida uma sessão FTP, em seguida duas, três e quatro sessões. A cada nova sessão, foram feitas três leituras dos valores do número de datagramas na rede, dado pela soma dos deltas de *ipInReceives* e *ipOutRequests*, conforme mostrado na figura 14.

	Tempo Decorrente		
Número de Sessões FTP	1 minuto	2 minutos	3 minutos
1 Sessão	2055	2926	3003
2 Sessões	3052	3244	2762
3 Sessões	3686	3318	3837
4 Sessões	3796	3569	3230

Figura 14: Tráfego em sessões FTP

Em seguida, foi monitorada a quantidade de datagramas na rede durante a execução do gerador de fluxo UDP. Primeiramente, foi monitorado o fluxo num único sentido, ou seja, com o servidor rodando no *host* monitorado e cliente no outro computador da rede. Depois foi monitorado o fluxo em dois sentidos, ou seja, com servidores e clientes sendo executados em cada um dos computadores utilizados. A figura 15 mostra os valores representativos do número de datagramas na rede, utilizando intervalo de 6 segundos, coletados de 6 em 6 segundos, para execução do gerador de fluxo em um sentido e nos dois sentidos.

Tempo de Execução	1 sentido	2 sentidos
0 segundos	90	86
6 segundos	16091	96463
12 segundos	55483	57489
18 segundos	81360	62784
24 segundos	65859	125463
30 segundos	78524	124944

Figura 15: Número de Datagramas IP

Em vista dos dados obtidos acima, foi arbitrado 8000 como valor crítico do delta do número de datagramas na rede. O programa abaixo amostra *ip.ipInReceives.0* e *ip.ipOutRequests.0* como deltas, calcula a sua soma, atribuindo o resultado a uma entrada da tabela de resultados da *Expression MIB*, denotada por *utilization*. Quando o valor vinculado a *utilization* for superior ao valor crítico, no caso 8000, será gerada uma notificação para o *host* especificado com a instância de objeto vinculada a *utilization*.

```
watch: denes.cce.ufpr.br using private
ip.ipInReceives.0 is Counter32: delta
ip.ipOutRequests.0 is Counter32: delta
begin
        bind utilization to (ip.ipInReceives.0 + ip.ipOutRequests.0);
        when utilization > 8000
        do
                notify denes.cce.ufpr.br private utilization
        end
end
```

As ações geradas pela tradução dos programas contidos nesta seção estão disponíveis no Anexo 4.3, onde o leitor pode comparar os comandos necessários para a programação manual da aplicação com os programas em ANEMONA.

Com o computador em condições normais de operação, isto é, sem estar executando o gerador de fluxo UDP, este programa foi traduzido e, em seguida foi evocado o servidor do gerador de fluxo e, em um outro computador conectado na rede, o seu cliente. Em um experimento representativo, após 19 segundos foi recebida uma notificação no *host* monitorado.

7.3 Considerações Finais

Este capítulo documentou os estudos de caso realizados para monitorar um dado sistema, visando detectar ataques do tipo *Denial of Service* e também detectar a saturação da capacidade de um enlace.

Durante estes testes foi confirmada a efetiva contribuição da ferramenta ANEMONA para a programação da *Event MIB* e da *Expression MIB*, poupando o usuário de idealizar e digitar uma série de comandos complexos do agente NET-SNMP e de exercer um controle efetivo sobre as tabelas das MIB's referidas. Ao invés disso, o usuário faz uso de uma interface de alto nível, fornecida pela linguagem ANEMONA.

Quanto ao desempenho da ferramenta, é possível afirmar que as ações geradas por ela são exatamente as mesmas que seriam geradas pela configuração manual das MIB's, porém esperando-se uma menor incidência de erros e maior rapidez, aumentando a produtividade do administrador de redes.

Capítulo 8

Conclusões

Neste capítulo são descritas as principais contribuições apresentadas no escopo desta dissertação. Na primeira seção são comentados os resultados obtidos a partir da utilização das ferramentas desenvolvidas. Em seguida são propostos alguns trabalhos futuros que darão continuidade a esta pesquisa e, por fim, as considerações finais.

8.1 Resultados Obtidos

Foi proposta uma linguagem para a configuração das MIB's *Event* e *Expression*, denominada ANEMONA (*A NEtwork MONitoring Application*) com o objetivo principal de facilitar o trabalho de programação dessas MIB's.

A linguagem ANEMONA oferece ao usuário uma interface de alto nível para especificar o comportamento destas MIB's; permite a utilização das duas MIB's em conjunto; gera automaticamente ações para configurar a *Expression MIB* e a *Event MIB*, a partir de uma mesma especificação; reduz a incidência de erros devido ao fato de minimizar a interação com o usuário, durante a fase de configuração; exonera o usuário do exercício do controle sobre as diversas tabelas das MIB's e poupa o usuário do trabalho meramente repetitivo de atribuir valores a objetos de gerência.

A ferramenta proposta torna viável o uso, individual ou combinado, das MIB's *Event* e *Expression*, que até então possuíam elevada complexidade de operação.

Para demonstrar a viabilidade desta abordagem foi construída uma ferramenta prática, em linguagem C, cujas partes integrantes foram:

- Subconjunto da *Expression MIB* com base no subconjunto de [6] disponível no Anexo 1 (1093 linhas).
- Subconjunto da *Event MIB* com base no subconjunto de [7] disponível no Anexo 2 (1637 linhas).
- Um tradutor para ANEMONA, composto por um analisador léxico, programado em *Flex* (69 linhas), um analisador sintático, programado em *Bison* (448 linhas) e um arquivo com bibliotecas em C (507 linhas).

Foram ainda, realizados exaustivos testes, para os quais foram elaborados programas em ANEMONA.

Foram também realizados estudos de caso com aplicações que podem detectar ataques iminentes e a saturação da capacidade de um enlace.

Estes testes e estudos de caso mostraram não apenas a viabilidade da ferramenta como as vantagens no seu uso, se comparadas às ações executadas diretamente contra as MIB's *Event* e *Expression*.

8.2 Trabalhos Futuros

Como continuação da pesquisa desenvolvida no decorrer deste trabalho, alguns tópicos relevantes podem vir a ser estudados:

- Agregação das MIB's do DISMAN *Management Target* e *Notification* ao conjunto, possibilitando que a *Event MIB* possa monitorar expressões de objetos em *hosts* remotos e utilizando todos os recursos disponíveis para as notificações do SNMP versão 3.
- Um ambiente gráfico para a ferramenta, onde o usuário possa construir programas manipulando elementos gráficos.
- Portar a presente ferramenta para outras plataformas, como *Windows NT*, por exemplo.

8.3 Considerações Finais

Os estudos realizados nesta dissertação produziram a especificação de uma linguagem para a configuração das MIB's *Event* e *Expression* e a implementação de um modelo para testes.

Durante os testes e estudos de caso realizados, a ferramenta mostrou ser de grande valia para aplicações de gerência de redes.

Anexo 1

Definições do Subconjunto da Expression MIB

Este anexo fornece a especificação em ASN.1 para o subconjunto construído para a *Expression MIB*. A partir desta especificação, foi gerado o código em linguagem C do módulo *Expression MIB* para o agente NET-SNMP, com auxílio da ferramenta MIB2C e cujo comportamento dos objetos, bem como métodos para leitura e escrita de valores foram programados manualmente sobre a estrutura do código gerado automaticamente.

1.1 Especificação em ASN.1

```
UFPR-EXPRESSION-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Unsigned32,
    Counter32, Counter64, IpAddress,
    TimeTicks, mib-2                        FROM SNMPv2-SMI
    SnmpAdminString           FROM SNMP-FRAMEWORK-MIB;

ufprExpressionMIB MODULE-IDENTITY
    LAST-UPDATED "200010260000Z" -- 26 October 2000
    ORGANIZATION "Universidade Federal do Parana"
    CONTACT-INFO "Henrique Denes Fernandes
                  Universidade Federal do Parana
                  Centro Politecnico S/N,
                  Curitiba PR.
                  Phone: +54 41 361 3028
                  Email: denes@cce.ufpr.br"
    DESCRIPTION
     "The MIB module for defining expressions of MIB objects for
      management purposes."

-- Revision History
```

```
        REVISION      "200010260000Z" -- 26 October 2000"
        DESCRIPTION   "This is the initial version of this MIB"

    ::= { mib-2 98 }


ufprExpressionMIBObjects OBJECT IDENTIFIER ::= { ufprExpressionMIB 1 }

expDefine OBJECT IDENTIFIER ::= { ufprExpressionMIBObjects 1 }
expValue  OBJECT IDENTIFIER ::= { ufprExpressionMIBObjects 2 }

-- Definições
--
-- Tabela de Definicao da Expressao
--

expExpressionTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ExpExpressionEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A table of expression definitions."
    ::= { expDefine 1 }

expExpressionEntry OBJECT-TYPE
    SYNTAX      ExpExpressionEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "Information about a single expression. To create an expression
     first write in this entry in this table. For expression evaluate
     to succeed, all related entries in expExpressionTable and
     expObjectTable must be active."

    INDEX       { expExpressionNumber }
    ::= { expExpressionTable 1 }

ExpExpressionEntry ::= SEQUENCE {
    expExpressionNumber          INTEGER,
    expExpression                OCTET STRING,
    expExpressionValueType       INTEGER,
    expExpressionComment         SnmpAdminString
}

expExpressionNumber OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "The index of the expression."
    ::= { expExpressionEntry 1 }

expExpression OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (1..1024))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
     "The expression to be evaluated. Variables are expressed as a
     dollar sign ('$') and an integer that corresponds to an index
     to object (expObjectIndex). An example: ($1 - $5) * 100.
```

```
      Expressions must be not recursive. The only allowed operators are
      (, ), -(unario), +, -, *, /, %, &&, ||, ==, !=, >, >=, <, <=.
      The only constant types defined are int(32), long(64), unsigned
      int, unsigned long. The default is int. Conditional expressions
      result in a unsigned 32 bits, of value 0 for false or non-zero is
      true. Rules for the resulting data type from an operation, based on
      operador:
      For &&, ||, ==, !=, <, <=, >, >= the result is always Unsigned32.
      For unary - the result is always Integer32.
      For +, -, *, /, %, the result is promoted according the
      following properties:
      If left and right are the same type, use that.
      If either side is TimeTicks, use that.
      If either side is Counter32, use that.
      Otherwise use Unsigned32.
      The following rules say what operators apply with what data types.
      Any combination not explicitly defined does not work."
     ::= { expExpressionEntry 2}

expExpressionValueType OBJECT-TYPE
     SYNTAX       INTEGER { counter32(1), unsigned32(2), timeTicks(3),
                   integer32(4), ipAddress(5), octetString(6),
                   objectId(7) }
     MAX-ACCESS   read-write
     STATUS       current
     DESCRIPTION
      "The type of the expression value."
     DEFVAL       { counter32 }
     ::= { expExpressionEntry 3}

expExpressionComment OBJECT-TYPE
     SYNTAX       SnmpAdminString
     MAX-ACCESS   read-write
     STATUS       current
     DESCRIPTION
      "A commnet to explain the use or meaning of the expression."
     DEFVAL       { ''H }
     ::= { expExpressionEntry 4}
--
-- Tabela Objetos
--

expObjectTable OBJECT-TYPE
     SYNTAX       SEQUENCE OF expObjectEntry
     MAX-ACCESS   not-accessible
     STATUS       current
     DESCRIPTION
      "A table with object definitions to be used in expExpression."
     ::= { expDefine 2 }

expObjectEntry OBJECT-TYPE
     SYNTAX       ExpObjectEntry
     MAX-ACCESS   not-accessible
     STATUS       current
     DESCRIPTION
      "Information about an object. Values of objects in this table
      may be changed at any time."
     INDEX        { expObjectIndex }
     ::= { expObjectTable 1 }
```

```
ExpObjectEntry ::= SEQUENCE {
    expObjectIndex                          INTEGER,
    expObjectID                             OBJECT IDENTIFIER,
    expObjectSampleType                     INTEGER
}

expObjectIndex OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A numeric identification for an object. Prefixed with $ this
     value is used to reference the object in the corresponding
     expExpression."
    ::= { expObjectEntry 1 }

expObjectID OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
     "The OBJECT IDENTIFIER (OID) of this object."
    ::= { expObjectEntry 2 }

expObjectSampleType OBJECT-TYPE
    SYNTAX      INTEGER { absoluteValue(1), deltaValue(2),
                          changedValue(3) }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
     "The method of sampling the selected variable. 'absoluteValue'
     is the present value of this object. 'deltaValue' is the present
     value minus the previous value. 'changedValue' is a boolean true
     whether the present value is different from the previous value."
    DEFVAL      { absoluteValue }
    ::= { expObjectEntry 3 }

--
-- Tabela Valor da Expressao
--

expValueTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ExpValueEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A table of values from evaluated expressions."
    ::= { expValue 1 }

expValueEntry OBJECT-TYPE
    SYNTAX      ExpValueEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A single value from an evaluated expression. For a given
     instance, only one 'Val' will be instanciated, that is, the one
     with appropriate type for the value."
    INDEX       { expValueIndex}
    ::= { expValueTable 1 }
```

```
ExpValueEntry ::= SEQUENCE {
    expValueIndex              INTEGER,
    expValueCounter32Val       Counter32,
    expValueUnsigned32Val      Unsigned32,
    expValueTimeTicksVal       TimeTicks,
    expValueInteger32Val       Integer32,
    expValueIpAddressVal       IpAddress,
    expValueOctetStringVal     OCTET STRING,
    expValueOidVal             OBJECT IDENTIFIER
}

expValueIndex    OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "The index of the table."
    ::= {expValueEntry 1}

expValueCounter32Val OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'counter32'."
    ::= { expValueEntry 2 }

expValueUnsigned32Val OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'unsigned32'."
    ::= { expValueEntry 3 }

expValueTimeTicksVal OBJECT-TYPE
    SYNTAX      TimeTicks
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'timeTicks'."
    ::= { expValueEntry 4 }

expValueInteger32Val OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'integer32'."
    ::= { expValueEntry 5 }

expValueIpAddressVal OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'ipAddress'. Available for
     compatibility."
    ::= { expValueEntry 6 }
```

```
expValueOctetStringVal OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0..65536))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'octetString'. Available for
     compatibility."
    ::= { expValueEntry 7 }

expValueOidVal OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
     "The value when the expression is 'objectId'. Available for
     compatibility."
    ::= { expValueEntry 8 }

END
```

Anexo 2

Definições do Subconjunto da Event MIB

Este anexo fornece a especificação em ASN.1 para o subconjunto construído para a *Event MIB*. A partir desta especificação, foi gerado o código em linguagem C do módulo *Event MIB* para o agente NET-SNMP, com auxílio da ferramenta MIB2C e cujo comportamento dos objetos, bem como métodos para leitura e escrita de valores foram programados manualmente sobre a estrutura do código gerado automaticamente.

2.1 Especificação em ASN.1

```
UFPR-EVENT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, mib-2          FROM SNMPv2-SMI
    TruthValue                FROM SNMPv2-TC
    SnmpAdminString           FROM SNMP-FRAMEWORK-MIB;

ufprEventMIB MODULE-IDENTITY
    LAST-UPDATED "200130030000Z"            -- 30 March 2001
    ORGANIZATION "UFPR Universidade Federal do Parana"
    CONTACT-INFO "Henrique Denes Fernandes
                  Universidade Federal do Parana
                  Departamento de Informatica
                  Centro Politecnico s/n
                  Curitiba PR 81.531-970.
                  Phone: +55 41 361 3028
                  Email: denes@cce.ufpr.com.br"
    DESCRIPTION
     "The MIB module for defining event triggers and actions for
     network management purposes."
-- Revision History
```

```
        REVISION     "200130030000Z"            -- 30 March 2001
        DESCRIPTION  " "
    ::= { mib-2 99 }

ufprEventMIBObjects OBJECT IDENTIFIER ::= { ufprEventMIB 1 }

-- Management Triggered Event (MTE) objects

mteTrigger            OBJECT IDENTIFIER ::= { ufprEventMIBObjects 1 }
mteEvent              OBJECT IDENTIFIER ::= { ufprEventMIBObjects 2 }

--
-- Secao Trigger
--

--
-- Trigger Table
--

mteTriggerTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF MteTriggerEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table of management event trigger information."
    ::= { mteTrigger 1}

mteTriggerEntry OBJECT-TYPE
    SYNTAX      MteTriggerEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Information about a single trigger. "
    INDEX       { mteTriggerIndex }
    ::= { mteTriggerTable 1 }

MteTriggerEntry ::= SEQUENCE {
    mteTriggerIndex                     INTEGER,
    mteTriggerComment                   SnmpAdminString,
    mteTriggerTest                      INTEGER,
    mteTriggerSampleType                INTEGER,
    mteTriggerValueID                   OBJECT IDENTIFIER,
    mteTriggerEnabled                   TruthValue,
    mteTriggerExistenceTest           INTEGER,
    mteTriggerEvent                 INTEGER,
    mteTriggerBooleanComparison         INTEGER,
    mteTriggerBooleanValue              Integer32
}

mteTriggerIndex OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The index of Trigger table."
    ::= { mteTriggerEntry 1 }

mteTriggerComment OBJECT-TYPE
    SYNTAX      SnmpAdminString
```

```
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "A description of trigger's use and function."
    DEFVAL { ''H }
    ::= { mteTriggerEntry 2 }

mteTriggerTest OBJECT-TYPE
    SYNTAX      INTEGER { existence(0), boolean(1) }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The type of test to perform.  For 'boolean' test, the object
        mteTriggerValueID must evaluate to an integer.

        For 'existence', the specific test is selected by
        mteTriggerExistenceTest.  When an object appears, vanishes
        or change of changes of value, the trigger fires. If the object
        appearence caused the trigger firing, the object must vanish
        before the trigger can be fired again for it, and vice versa.
        If the trigger fired due to a change in the object's value, it
        will be fired again on every sucessive value change for that
        object.

        For 'boolean', the specific test is selected by
        mteTriggerBooleanTest. If the test result is true the trigger
        fires. The trigger will not fire again until the value has
        false and come back to true."

    DEFVAL { boolean }
    ::= { mteTriggerEntry 3 }

mteTriggerSampleType OBJECT-TYPE
    SYNTAX      INTEGER { absoluteValue(1), deltaValue(2) }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The type of sampling to perform.

        An 'absoluteValue' sample requires only a single sample to be
        meaningful, and is exactly the value of the object at
        mteTriggerValueID at the sample time.

        A 'deltaValue' requires two samples to be meaningful and is
        thus not available for testing until the second and subsequent
        samples after the object at mteTriggerValueID is first found to
        exist. It is the difference between the two samples.

        For SNMP counters to be meaningful they should be sampled as a
         'deltaValue'."
    DEFVAL { absoluteValue }
    ::= { mteTriggerEntry 4 }

mteTriggerValueID OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The OID of the object to sample."
    ::= { mteTriggerEntry 5}
```

```
mteTriggerEnabled OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "A control to allow a trigger to be configured and not used.
        When the value is 'false', the trigger is not sampled."
    DEFVAL { false }
    ::= { mteTriggerEntry 6 }

mteTriggerExistenceTest OBJECT-TYPE
    SYNTAX      INTEGER { present(0), absent(1), changed(2) }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The type of existence test to perform. The trigger fires
        when an object at mteTriggerValueID is seen to go from present
        to absent, from absent to present, or to have your value
        modified, depending on which the tests are selected:

        present(0) - When this test is selected, the trigger fires when
        the mteTriggerValueID object goes from absent to present.
        absent(1)  - The trigger fires when the mteTriggerValueID object
        goes from present to absent.
        changed(2) - The trigger fires when the mteTriggerValueID object
        value changes.

        Once the trigger has fired for either presence or absence it
        will not fire again for that state until the object has been
        to the other state."
    DEFVAL { present }
    ::= { mteTriggerEntry 7 }

mteTriggerEvent OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The index of the event to be called when the trigger fires."
    DEFVAL { ''H }
    ::= { mteTriggerEntry 8 }

mteTriggerBooleanComparison OBJECT-TYPE
    SYNTAX      INTEGER { unequal(1), equal(2),
                 less(3), lessOrEqual(4),
                 greater(5), greaterOrEqual(6) }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The type of boolean comparison to perform.

        The value at mteTriggerValueID is compared to
        mteTriggerBooleanValue, so for example if
        mteTriggerBooleanComparison is 'less' the result would be
        true.
        If the value at mteTriggerValueID is less than the value of
        mteTriggerBooleanValue."
    DEFVAL { unequal }
    ::= { mteTriggerEntry 9 }
```

```
mteTriggerBooleanValue OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The value for use for the test specified by
         mteTriggerBooleanTest."
    DEFVAL { 0 }
    ::= { mteTriggerEntry 10 }

--
-- Secao Event
--

--
-- Event Table
--

mteEventTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF MteEventEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table of management event action information."
    ::= { mteEvent 1 }

mteEventEntry OBJECT-TYPE
    SYNTAX      MteEventEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Information about a single event."
    INDEX       { mteEventIndex }
    ::= { mteEventTable 1 }

MteEventEntry ::= SEQUENCE {
    mteEventIndex                       INTEGER,
    mteEventComment                     SnmpAdminString,
    mteEventActions                     INTEGER,
    mteEventSetObject                   OBJECT IDENTIFIER,
    mteEventSetValue                    Integer32,
    mteEventEnabled                 TruthValue,
    mteEventNotification            OBJECT IDENTIFIER
    }

mteEventIndex OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The index of the table."
    ::= { mteEventEntry 1 }

mteEventComment OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "A decsription of the event's function and use."
```

```
    DEFVAL { ''H }
    ::= { mteEventEntry 2 }

mteEventActions OBJECT-TYPE
    SYNTAX      INTEGER { notification(0), set(1) }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The actions to perform when this event occurs.
        For 'notification', Traps are sent.

        For 'set', an SNMP Set operation is performed."
    DEFVAL { notification }
    ::= { mteEventEntry 3 }

mteEventSetObject OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The OID from the MIB object to set if mteEventActions
      has 'set' set."
    ::= { mteEventEntry 4 }

mteEventSetValue OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The value to which to set the object at mteEventSetObject
        if mteEventActions has 'set' set."
    ::= { mteEventEntry 5 }

mteEventNotification OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    MAX-ACCESS    read-write
    STATUS  current
    DESCRIPTION
            "The object identifier from the NOTIFICATION TYPE for the
             notification to use if mteEventActions has
             'notification' set."
    ::= { mteEventEntry 6 }

mteEventEnabled OBJECT-TYPE
    SYNTAX  TruthValue
    MAX-ACCESS    read-write
    STATUS  current
    DESCRIPTION
            "A control to allow an event to be configured and not
            used. When your value is 'false', the event is not
            performed."
    DEFVAL  {false}
    ::= { mteEventEntry 7 }

END
```

Anexo 3

Tradutor ANEMONA

Este anexo fornece as definições dos *tokens*, usando o formalismo de expressões regulares estendidas, e a gramática da linguagem ANEMONA, além dos códigos fonte do tradutor implementado. Como o código fonte do tradutor foi gerado com auxílio das ferramentas *Flex* e *Bison,* os arquivos apresentados neste anexo são o fonte do analisador léxico, em *Flex*; o fonte do analisador sintático, em *Bison* e os protótipos das funções do arquivo de rotinas auxiliares, em linguagem C.

3.1 Definições Regulares dos Tokens ANEMONA

A tabela abaixo mostra os padrões válidos para cada *token*, especificados usando a notação de expressões regulares estendidas.

Token	Padrão Reconhecido
DELIM	espaço em branco | tabulação
NEWLINE	Símbolo de nova linha
WS	DELIM+
LETRA	A-Z | a-z | _
DIGITO	0-9
ID	LETRA (LETRA | DIGITO | '-')*
IP1	0-9 | 0(0-9) | 00 (0-9)
IP2	(0-9) (0-9) | 0 (0-9) (0-9)
IP3	1 (0-9) (0-9)
IP4	2 (0-4) (0-9) |25 (0-5)
IP	IP1 | IP2 | IP3 | IP4
HOST	ID (.ID)* | IP.IP.IP.IP
NUMBER	DIGITO+
OID1	ID | NUMBER
OID	.? OID1.OID1 (.OID1)*

3.2 Gramática da Linguagem ANEMONA

A seguir será mostrada a gramática da linguagem ANEMONA, tendo como símbolo de partida 'PROGRAM', como símbolos não terminais aqueles grafados em maiúsculas e como terminais os demais símbolos. Já os símbolos entre '<' e '>', são lexemas reconhecidos por um dos padrões da seção 1 deste anexo.

PROGRAM → HEADER DECLS BODY

HEADER → watch : HOST1 using <id>

HOST1 → <host>
| <id>

DECLS → DECL_LINE DECLS
| Λ

DECL_LINE → <oid> is TYPE : SAMPLE

TYPE → Integer
| OctetString
| OID
| IPAddress
| Counter32
| Unsigned
| TimeTicks

SAMPLE → absolute
| delta
| modified

BODY → begin CODE end

CODE → CODE1 CODE
| Λ

CODE1 → EXPR ;
| COND
| MACRO
| WHEN

```
EXPR      →   EXPR or EXPR
          |   EXPR and EXPR
          |   not EXPR
          |   EXPR1

EXPR1     →   EXPR2 == EXPR2
          |   EXPR2 != EXPR2
          |   EXPR2 < EXPR2
          |   EXPR2 <= EXPR2
          |   EXPR2 > EXPR2
          |   EXPR2 >= EXPR2
          |   EXPR2

EXPR2     →   counter32 ( EXPR2 )
          |   EXPR2 + EXPR2
          |   EXPR2 - EXPR2
          |   EXPR2 * EXPR2
          |   EXPR2 / EXPR2
          |   - EXPR2
          |   EXPR3

EXPR3     →   <number>
          |   <id>
          |   REF1
          |   ( EXPR )

REF1      →   <oid>

REF2      →   <oid> at HOST1
          |   <oid> at HOST1 using <id>

COND      →   if EXPR then NUMBER else NUMBER rec by REF2

MACRO     →   bind <id> to EXPR ;

WHEN      →   when EXPR do ACTIONS end
          |   when exists ( REF2 ) do ACTIONS end
          |   when not exists ( REF2 ) do ACTIONS end

ACTIONS   →   CMD ACTIONS
          |   CMD

CMD       →   set <oid> at HOST1 using <id> to <number>
          |   notify HOST1 <id> <oid>
          |   notify HOST1 <id> <id>
```

3.3 Tradutor para a Linguagem ANEMONA

3.3.1 Fonte do Analisador Léxico

```
%option noyywrap
delim [ \t]
newline [\n]
ws    {delim}+
letra [A-Za-z\_]
digito      [0-9]
id    {letra}({letra}|{digito}|\-)*
ip1   [0-9]|0[0-9]|00[0-9]
ip2   [0-9][0-9]|0[0-9][0-9]
ip3   1[0-9][0-9]
ip4   2[0-4][0-9]|25[0-5]
ip    {ip1}|{ip2}|{ip3}|{ip4}
host  {id}(\.{id})*|{ip}\.{ip}\.{ip}\.{ip}
number      {digito}+
oid1  {id}|{number}
oid   \.?{oid1}\.{oid1}(\.{oid1})*
%%
{ws}        {}
{newline}   {linenum++;}
"-"         {return (MINUS);}
watch       {return (WATCH);}
using       {return (USING);}
is          {return (IS);}
begin       {return (BEG);}
end         {return (END);}
or          {return (OR);}
and         {return (AND);}
not         {return (NOT);}
counter32   {return (FUNCC32);}
at          {return (AT);}
if          {return (IF);}
then        {return (THEN);}
else        {return (ELSE);}
rec         {return (REC);}
by          {return (BY);}
bind        {return (BIND);}
to          {return (TO);}
when        {return (WHEN);}
do          {return (DO);}
exists            {return (FUNCEX);}
set         {return (SET);}
notify            {return (NOTIFY);}
"=="        {return (EQUAL);}
"!="        {return (DIF);}
"<="        {return (MNE);}
">="        {return (MAE);}
Integer           {var_type = 0;return(TYPE);}
OctetString {var_type = 0;return(TYPE);}
OID         {var_type = 0;return(TYPE);}
```

```
IPAddress    {var_type = 0;return(TYPE);}
Counter32    {var_type = 2;return(TYPE);}
Unsigned     {var_type = 1;return(TYPE);}
TimeTicks    {var_type = 3;return(TYPE);}
absolute     {sprintf(var_sample, "1"); return(SAMPLE);}
delta        {sprintf(var_sample, "2"); return(SAMPLE);}
modified     {sprintf(var_sample, "3"); return(SAMPLE);}
{id}         {var_id = strtok(yytext," \t\n");
              strcpy(yylval.str, var_id); return(ID);}
{host}            {var_host = strtok(yytext," \t\n"); return(HOST);}
{number}     {strcpy(yylval.str, strtok(yytext,
                  " \t\n"));return(NUMBER);}
{oid}        {strcpy(var_oid, strtok(yytext," \t\n"));
              strcpy(yylval.str, var_oid); return(OBJID);}
.            {return(yytext[0]);}
%%
char *report_error()
{
      return yytext;
}
```

3.3.2 Fonte do Analisador Sintático

```
%{
#include <stdio.h>
#include <string.h>
#include <ucd-snmp/ucd-snmp-config.h>
#include <ucd-snmp/ucd-snmp-includes.h>
#include <ucd-snmp/system.h>
#define STR1_LEN 40
#define STR2_LEN 65
#define STR3_LEN 300
#define TABLE_SIZE 10
typedef struct {int index;
            oid obj[MAX_OID_LEN];
            int type;
            size_t obj_len;}table1;
typedef struct {char id[STR1_LEN];
            int type;
            char oid[STR1_LEN];
            int entry;}table2;
int linenum = 1;
char monitor[STR1_LEN];
char community[STR1_LEN];
char var_host1[STR1_LEN];
char *var_host;
char *var_id;
char var_oid[STR2_LEN];
int var_type;
char var_sample[2];
char var_expr[STR3_LEN];
table1 obj_table[TABLE_SIZE];
int p_obj_table = 1;
int last_exp = 0;
table2 id_table[TABLE_SIZE];
```

```
int p_id_table = 1;
void rep_macros(void);
int ret_type = 0;
int event_array[TABLE_SIZE];
void start_event_array(void);
char *report_error(void);
char oid_trap[STR1_LEN];
%}
%union {
      char str[80];
      int integer;}

%token MINUS
%token WATCH
%token USING
%token IS
%token BEG
%token END
%token OR
%token AND
%token NOT
%token FUNCC32
%token AT
%token IF
%token THEN
%token ELSE
%token REC
%token BY
%token BIND
%token TO
%token WHEN
%token DO
%token FUNCEX
%token SET
%token NOTIFY
%token EQUAL
%token DIF
%token MNE
%token MAE
%token TYPE
%token SAMPLE
%token <str> ID
%token HOST
%token <str> NUMBER
%token <str> OBJID
%type <str> expr
%type <str> expr1
%type <str> expr2
%type <str> expr3
%type <str> ref1
%type <str> ref2
%left '+' MINUS
%left '*' '/'
%left NEG
%left EQUAL DIF MNE MAE '<' '>'
%left OR
%left AND
%left NOT
```

```
%%
program     :     header decls body       {rep_macros();}
      |     error       {parseerror(report_error());}
      ;
header      :     WATCH ':' host1 USING ID {
                        strcpy(monitor, var_host1);
                        sprintf(community, "%s", var_id);}
      ;
host1 :     HOST  {sprintf(var_host1, "%s", var_host);}
      |     ID    {sprintf(var_host1, "%s", var_id);}
      ;
decls :     decl_line decls
      |
      ;
decl_line : OBJID IS TYPE ':' SAMPLE {install_decl_line();}
      ;
body  :     BEG code END
      ;
code  :     code1 code
      |
      ;
code1 :     expr ';' {sprintf(var_expr, "%s", $1);
                  install_expression(); ret_type = 0;}
      |     cond
      |     macro
      |     when
      ;
expr  :     expr OR expr      {strcpy($$, $1); strcat($$, "||");
                        strcat($$, $3); ret_type = 1;}
      |     expr AND expr     {strcpy($$, $1); strcat($$, "&&");
                        strcat($$, $3); ret_type = 1;}
      |     NOT expr    {strcpy($$, "~"); strcat($$, $2);
                        ret_type = 1;}
      |     expr1 {strcpy($$, $1);}
      ;
expr1 :     expr2 EQUAL expr2 {strcpy($$, $1); strcat($$, "==");
                         strcat($$, $3); ret_type = 1;}
      |     expr2 DIF expr2 {strcpy($$, $1); strcat($$, "!=");
                        strcat($$, $3); ret_type = 1;}
      |     expr2 '<' expr2 {strcpy($$, $1); strcat($$, "<");
                        strcat($$, $3); ret_type = 1;}
      |     expr2 MNE expr2 {strcpy($$, $1); strcat($$, "<=");
                        strcat($$, $3); ret_type = 1;}
      |     expr2 '>' expr2 {strcpy($$, $1); strcat($$, ">");
                        strcat($$, $3); ret_type = 1;}
      |     expr2 MAE expr2 {strcpy($$, $1); strcat($$, ">=");
                        strcat($$, $3); ret_type = 1;}
      |     expr2 {strcpy($$, $1);}
      ;
expr2 :     FUNCC32 '(' expr2 ')'   {strcpy($$, $3); ret_type = 2;}
      |     expr2 '+' expr2 {strcpy($$, $1); strcat($$, "+");
                        strcat($$, $3);}
      |     expr2 MINUS expr2 {strcpy($$, $1); strcat($$, "-");
                        strcat($$, $3);}
      |     expr2 '*' expr2 {strcpy($$, $1); strcat($$, "*");
                        strcat($$, $3);}
      |     expr2 '/' expr2   {strcpy($$, $1); strcat($$, "/");
                        strcat($$, $3);}
      |     MINUS expr2 %prec NEG {strcpy($$, "-"); strcat($$, $2);}
      |     expr3 {strcpy($$, $1);}
```

```
      ;
expr3 :     NUMBER      {strcpy($$, $1);}
      |     ID    {int i; i = find_id($1); if(i) sprintf($$, "v%d",
                   i); else{ printf("Macro indefinida: %s\n", $1);
                   exit(1);}}
      |     ref1  {strcpy($$, $1);}
      |     '(' expr ')' {strcpy($$, "("); strcat($$, $2);
                  strcat($$, ")");}
      ;
ref1  :     OBJID {int a; a = find_oid(var_oid); if(a)
                   sprintf($$, "v%d", a); else{ printf(
                   "Objeto nao declarado: %s\n", var_oid);
                   exit(1);}}
      ;
ref2  :     OBJID AT host1 {strcpy($$, $1);}
      |     OBJID AT host1 USING ID {strcpy($$, $1);}
      ;
cond  :     IF expr THEN NUMBER ELSE NUMBER REC BY ref2
            { sprintf(var_expr, "%s", $2); install_expression();
              install_cond($9, $4, $6); ret_type = 0;
              start_event_array();}
      ;
macro :     BIND ID TO expr ';' {sprintf(var_expr, "%s", $4);
                        install_expression();
                        install_macro($2); ret_type = 0;}
      ;
when  :     WHEN expr DO actions END {sprintf(var_expr, "%s",
$2);
                  install_expression(); install_when(); ret_type=0;
                  start_event_array();}
      |     WHEN FUNCEX '(' ref2 ')' DO actions END
                  {install_w_funcex($4); start_event_array();}
      |     WHEN NOT FUNCEX '(' ref2 ')' DO actions END
                  {install_w_nfuncex($5); start_event_array();}
      ;
actions :   cmd actions
      |     cmd
      ;
cmd   :     SET OBJID AT host1 USING ID TO NUMBER
                  {install_set($2,$8);}
      |     NOTIFY host1 ID OBJID {install_notify($4);}
      |     NOTIFY host1 ID ID {if(find_macro($4))install_notify(oid_trap);
                        else {printf("Macro Indefinida: %s\n", $4);
                        exit(1);}}
      ;
%%
#include "lex.yy.c"
#include "aux.c"
main(){
      start_event_array();
      yyparse();
}
yyerror(s)
char *s;
{
      printf("%s\n", s);
}
parseerror(s) /* reporta erros de sintaxe */
char *s;
```

```
install_decl_line() /* instala declaracao de objeto */

install_expression() /* instala expressao */

install_macro(a) /* instala definicao de macro */
char *a;

install_cond(oset, value1, value2) /* instala operador condicional */
char *oset;
char *value1;
char *value2;

install_notify(a) /* instala acao de notificacao */
char *a;

install_set(a, b) /* instala acao 'set' */
char *a;
char *b;

install_when() /* instala trigger */

install_w_funcex(a) /* instala funcao 'exists' */
char *a;

install_w_nfuncex(a) /* instala neg funcao 'exists' */
char *a;
```

3.3.3 Protótipos das Funções Auxiliares

```
#include <ucd-snmp/ucd-snmp-config.h>
#include <ucd-snmp/ucd-snmp-includes.h>
#include <ucd-snmp/system.h>
#define TABLE_SIZE 10

int find_empty_entry_1(void) /* retorna indice de entrada livre na tabela
                                de objetos */

int find_empty_entry_2() /* retorna indice de entrada livre na tabela
                            de expressoes */

void settable(char *a, char b, char *c) /* atribui um valor a uma entrada
                                           especifica de uma tabela */

int find_oid(char *a) /* pesquisa a tabela de objetos do tradutor pelo oid
                         e retorna um indice */

int find_id(char *a) /* pesquisa a tabela de macros do tradutor pelo
                        identificador da macro e retorna um indice */

void rep_macros(void) /* reporta identificadores de objetos associados a
                         macros */

int inf_type(int a) /* faz a inferencia de tipos entre dois operandos
                       durante a avaliação de expressoes */
```

```
find_empty_entry_3() /* retorna indice de entrada livre na tabela
                        de eventos */

int find_empty_entry_4() /* retorna indice de entrada livre na tabela
                            de triggers */

void start_event_array() /* inicializa lista de eventos */

int find_macro(char *a) /* pesquisa a tabela de macros e retorna endereco
                           de objeto associado a identificador para ser
                           enviado com notificacao */
```

Anexo 4

Ações Geradas Durante Estudos de Caso com ANEMONA

A fim de permitir uma comparação das ações geradas automaticamente pela ferramenta ANEMONA com operações equivalentes geradas manualmente, este anexo ilustra estas ações. Os comandos exibidos foram definidos supondo que as MIB's *Event* e *Expression* estavam com todas as entradas disponíveis em cada uma de suas tabelas.

4.1 Estudo de Caso 7.1.1

A seguir são mostradas as ações equivalentes ao programa que detecta uma invasão do tipo *ICMP Flooding*, ilustrado na seção 7.1.1.

```
snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.1 o icmp.icmpInEchos.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.1 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.1.1.2.1 s "$1 > 1000"
snmpset denes.cce.ufpr.br private 98.1.1.1.1.3.1 i unsigned32
snmpset denes.cce.ufpr.br private 99.1.1.1.1.3.1 i boolean
snmpset denes.cce.ufpr.br private 99.1.1.1.1.4.1 i absoluteValue
snmpset denes.cce.ufpr.br private 99.1.1.1.1.5.1 o 98.1.2.1.1.3.1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.8.1 i 1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.9.1 i unequal
snmpset denes.cce.ufpr.br private 99.1.1.1.1.10.1 i 0
snmpset denes.cce.ufpr.br private 99.1.2.1.1.3.1 i notification
snmpset denes.cce.ufpr.br private 99.1.2.1.1.6.1 o icmp.icmpInEchos.0
snmpset denes.cce.ufpr.br private 99.1.2.1.1.7.1 i true
snmpset denes.cce.ufpr.br private 99.1.1.1.1.6.1 i true
```

4.2 Estudo de Caso 7.1.2

Aqui são mostradas as ações equivalentes ao programa que detecta uma invasão do tipo *TCP Syn Flooding*, ilustrado na seção 7.1.2.

```
snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.1 o tcp.tcpAttemptFails.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.1 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.1.1.2.1 s "$1 > 25"
snmpset denes.cce.ufpr.br private 98.1.1.1.1.3.1 i unsigned32
snmpset denes.cce.ufpr.br private 99.1.1.1.1.3.1 i boolean
snmpset denes.cce.ufpr.br private 99.1.1.1.1.4.1 i absoluteValue
snmpset denes.cce.ufpr.br private 99.1.1.1.1.5.1 o 98.1.2.1.1.3.1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.8.1 i 1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.9.1 i unequal
snmpset denes.cce.ufpr.br private 99.1.1.1.1.10.1 i 0
snmpset denes.cce.ufpr.br private 99.1.2.1.1.3.1 i notification
snmpset denes.cce.ufpr.br private 99.1.2.1.1.6.1 o tcp.tcpAttemptFails.0
snmpset denes.cce.ufpr.br private 99.1.2.1.1.7.1 i true
snmpset denes.cce.ufpr.br private 99.1.1.1.1.6.1 i true
```

4.3 Estudo de Caso 7.2

A seção 7.2 trata do estudo de caso envolvendo a detecção da saturação do enlace e ilustra dois programas. O primeiro deles permite o cálculo do total de datagramas no dispositivo durante um intervalo definido e foi utilizado para colher resultados experimentais. As ações equivalentes a este programa são mostradas abaixo.

```
snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.1 o ip.ipInReceives.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.1 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.2 o ip.ipOutRequests.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.2 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.1.1.2.1 s "$1 + $2"
snmpset denes.cce.ufpr.br private 98.1.1.1.1.3.1 i counter32
```

Após a execução deste programa, o endereço da entrada correspondente na tabela de resultados da *Expression MIB* associado à macro *utilization* será *98.1.2.1.1.2.1.*

O segundo programa permite a detecção da saturação do enlace e as ações equivalentes são mostradas a seguir.

snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.1 o ip.ipInReceives.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.1 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.2.1.2.2 o ip.ipOutRequests.0
snmpset denes.cce.ufpr.br private 98.1.1.2.1.3.2 i deltaValue
snmpset denes.cce.ufpr.br private 98.1.1.1.1.2.1 s "$1 + $2"
snmpset denes.cce.ufpr.br private 98.1.1.1.1.3.1 i counter32
snmpset denes.cce.ufpr.br private 99.1.1.1.1.3.1 i boolean
snmpset denes.cce.ufpr.br private 99.1.1.1.1.4.1 i absoluteValue
snmpset denes.cce.ufpr.br private 99.1.1.1.1.5.1 o 98.1.2.1.1.2.1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.8.1 i 1
snmpset denes.cce.ufpr.br private 99.1.1.1.1.9.1 i unequal
snmpset denes.cce.ufpr.br private 99.1.1.1.1.10.1 i 0
snmpset denes.cce.ufpr.br private 99.1.2.1.1.3.1 i notification
snmpset denes.cce.ufpr.br private 99.1.2.1.1.6.1 o 98.1.2.1.1.2.1
snmpset denes.cce.ufpr.br private 99.1.2.1.1.7.1 i true
snmpset denes.cce.ufpr.br private 99.1.1.1.1.6.1 i true

Anexo 5

Gerador de Fluxo UDP

Este anexo apresenta o código fonte do gerador de fluxo baseado no protocolo UDP, construído para os estudos de caso realizados. Este programa se trata de um sistema cliente-servidor, que transfere uma grande quantidade de pequenos datagramas UDP. O programa é composto por dois arquivos fonte, ambos em linguagem C: o cliente e o servidor.

5.1 Arquivo Fonte do Servidor

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <stdio.h>
#define TAMFILA      5
#define MAXHOSTNAME 30

main ( int argc, char *argv[] )
  {
      int s, t;
      int i;
        char buf [BUFSIZ + 1];
      struct sockaddr_in sa, isa;  /* sa: servidor, isa: cliente */
      struct hostent *hp;
      char localhost [MAXHOSTNAME];

        if (argc != 2) {
           puts("Uso correto: servidor <porta>");
           exit(1);
        }

      gethostname (localhost, MAXHOSTNAME);

      if ((hp = gethostbyname( localhost)) == NULL){
```

```
            puts ("Nao consegui meu proprio IP");
            exit (1);
        }

        sa.sin_port = htons(atoi(argv[1]));

        bcopy ((char *) hp->h_addr, (char *) &sa.sin_addr, hp->h_length);

        sa.sin_family = hp->h_addrtype;


        if ((s = socket(hp->h_addrtype,SOCK_DGRAM,0)) < 0){
             puts ( "Nao consegui abrir o socket" );
             exit ( 1 );
        }

        if (bind(s,&sa,sizeof(sa)) < 0){
             puts ( "Nao consegui fazer o bind" );
             exit ( 1 );
        }

         while (1) {
         i = sizeof(isa);
         puts("Vou bloquear esperando mensagem.");
         recvfrom(s, buf, BUFSIZ, 0, &isa, &i);
         printf("Sou o servidor, recebi a mensagem----> %s\n", buf);
         sendto(s, buf, BUFSIZ, 0, &isa, i);
        }
}
```

5.2 Arquivo Fonte do Cliente

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

main(int argc, char *argv[])

 {  int sockdescr;
    int numbytesrecv;
    struct sockaddr_in sa;
    struct hostent *hp;
    char buf[BUFSIZ+1];
    char *host;
    char *dados;

    int lix,n;

    if(argc != 4) {
      puts("Uso correto: <cliente> <nome-servidor> <porta> <dados>");
      exit(1);
    }

    host = argv[1];
```

```
    dados = argv[3];

    if((hp = gethostbyname(host)) == NULL){
      puts("Nao consegui obter endereco IP do servidor.");
      exit(1);
    }

    bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
    sa.sin_family = hp->h_addrtype;

    sa.sin_port = htons(atoi(argv[2]));

    if((sockdescr=socket(hp->h_addrtype, SOCK_DGRAM, 0)) < 0) {
      puts("Nao consegui abrir o socket.");
      exit(1);
    }

while(1){
    if(sendto(sockdescr, dados, strlen(dados), 0, &sa, sizeof sa) !=
strlen(dados)){
      puts("Nao consegui mandar os dados");
      exit(1);
    }
}

    lix = sizeof sa;

    n = recvfrom(sockdescr, buf, BUFSIZ, 0, &sa, &lix);

    if (n < 0) puts("Nao funcionou bem o receive....");
    printf("Sou o cliente, recebi: %s\n", buf);

    close(sockdescr);
    exit(0);
}
```

Referências Bibliográficas

[1] HARRINGTON, D.; PRESUHN, P. and WIJNEN, B. An Architecture for Describing SNMP Management Frameworks. *Request for Comments 2571*, May 1999.

[2] CASE, J.; HARRINGTON, D.; PRESUHN, P. and WIJNEN, B. Message Processing and Dispatching for the SNMP. *Request for Comments 2572*, May 1999.

[3] LEVI, D.; MEYER, P. and STEWART, B. SNMPv3 Applications. *Request for Comments 2573*, May 1999.

[4] CASE, J.; MUNDY, R.; PARTAIN, D. and STEWART, B. Introduction to Version 3 of the Internet-Standard Network Management Framework. *Request for Comments 2570*, April 1999.

[5] CASE, J.; MCCLOGHRIE, K.; ROSE, M. and WALDBUSSER, S. Management Information Base for Version 2 of the SNMP. *Request for Comments 1907*, January 1996.

[6] KAVASSERI, R.; STEWART, B. Distributed Management Expression MIB. *Request for Comments 2982*, October, 2000.

[7] KAVASSERI, R.; STEWART, B. Event MIB. *Request for Comments 2981*, October 2000.

[8] COMER, D. *Interligação em Rede com TCP/IP*, v. 1, *Princípios, Protocolos e Arquitetura.* Editora Campus, 3ª Edição, 1998.

[9] ROSE, M. *The Simple Book, An Introduction to Internet Management.* Prentice Hall, second edition, 1994.

[10] STALLINGS, W. *SNMP, SNMPv2, SNMPv3, RMON and RMON2: Practical Network Management.* Addison-Wesley, third edition, 1998.

[11] LEINWAND, A. and CONTOY, K. *Network Management: a Practical Perspective.* Addison-Wesley, second edition, 1996.

[12] PARTRIDGE, C. and TREWITT, G. High-level Entity Management System (HEMS). *Request for Comments 1021*, 1987.

[13] FURLAN, D. *Especificação Formal do SNMPv3 Usando Semântica de Ações.* Dissertação (Mestrado em Informática) - Departamento de Informática, Universidade Federal do Paraná, 2000.

[14] CASE, J.; FEDOR, M.; SCHOFFSTALL, M. and DAVIN, J. A Simple Network Management Protocol (SNMP). *Request for Comments 1157*, May 1990.

[15] CASE, J.; MCCLOGHRIE, K.; ROSE, M. and WALDBUSSER, S. Protocol Operations for Version 2 of the SNMP. *Request for Comments 1905*, January 1996.

[16] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Information Processing Systems – Open Systems Interconnection – Specification of Absctract Syntax Notation One (ASN.1). *International Standard 8824*, December 1987.

[17] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Information Processing – Open Systems Interconnection – Absctract Syntax Notation One (ASN.1) – Addendum 1: Extensions to ASN.1. *International Standard 8824/AD 1*, 1987.

[18] MCCLOGHRIE, K.; PERKINS, D. and SCHOENWAELDER, J. Conformance Statements for SMIv2. *Request for Comments 2580*, April 1999.

[19] MCCLOGHRIE, K.; PERKINS, D. and SCHOENWAELDER, J. Structure of Management Information Version 2 (SMIv2). *Request for Comments 2578*, April 1999.

[20] MCCLOGHRIE, K.; PERKINS, D. and SCHOENWAELDER, J. Textual Conventions for SMIv2. *Request for Comments 2579*, April 1999.

[21] DUARTE JUNIOR., E. and MUSICANTE, M. "Formal Specification of SNMP MIBs Using Action Semantics: The Routing Proxy Case Study". *In Proc. of the Sixth IFIP/IEEE Int'l Symp. On Integrated Network Management*, pp. 417-430, IEEE Publishing, Boston, USA, May 1999.

[22] MOSSES, P. *Action Semantics*. Number 26 in Cambridge Tracts in Theorical Computer Science. Cambridge University Press, 1992.

[23] WATT, D. *Programming Language Syntax and Semantics*. Prentice Hall International Series in Computer Science. Prentice Hall, 1991.

[24] TANENBAUM, A. *Redes de Computadores*. Editora Campus, terceira edição.

[25] LEVI, D.; SCHOENWAELDER, J. Definitions of Managed Objects for Scheduling Management Operations. *Request for Comments 2591*, May 1999.

[26] LEVI, D.; SCHOENWAELDER, J. Definitions of Managed Objects for the Delegation of Management Script. *Request for Comments 2592*, May 1999.

[27] WHITE, K. Definitions of Managed Objects for Remote Ping, Traceroute and Lookup Operations. *Request for Comments 2925*, September 2000.

[28] KAVASSERI, R. Notification Log MIB. *Request for Comments 3014*, November 2000.

[29] CHISHOLM, S. and ROMASCANU, D. *Alarm MIB.* Internet Draft. April 2000.

[30] LEVI, D.; MEYER, P. and STEWART, B. SNMP Applications. *Request for Comment 2573*, April 1999.

[31] KAVASSERI, R.; STEWART, B. *Distributed Management Expression MIB.* Internet Draft, June, 2000.

[32] KAVASSERI, R.; STEWART, B. *Distributed Management Event MIB.* Internet Draft, June, 2000.

[33] The NET-SNMP Project Home Page. Disponível em: <http://net-snmp.sourceforge.net>

[34] AHO, A.; SETHI, R. e ULLMAN, J. *Compiladores, princípios, técnicas e ferramentas.* Editora LTC, 1995.

[35] GREENLAW, R. and HOOVER, J. *Fundamentals of the Theory of Computation, Principles and Practice.* San Francisco: Morgan Kaufmann Publishers, 1998.

[36] GNU. *Bison Reference Manual.* 1999.

[37] JOHNSON, S. C. "YACC – Yet Another Compiler Compiler". *Computing Science Technical Report 32.* AT&T Bell Laboratories. Murray Hill, 1975.

[38] "Project Neptune". *Phrack Magazine. Volume Seven. Issue Forty-Eight.* July, 1996.

[39] KERNIGHAN, B. W. and RITCHIE, D. M. *The C Programming Language.* Englewood Cliffs: Prentice Hall, 1978.