UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Criação e Visualização de Domínios Dinâmicos em Ambientes de Gerenciamento de Redes

por

MÁRCIO BARTZ CECCON

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

Profa. Dra. Maria Janilce Bosquiroli Almeida
Orientadora

Porto Alegre, março de 2003

CIP – Catalogação na Publicação

Ceccon, Márcio Bartz
Criação e Visualização de Domínios Dinâmicos em Ambientes de Gerenciamento de Redes / por Márcio Bartz Ceccon. – Porto Alegre: PPGC da UFGRS, 2003.

89p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientadora: Almeida, Maria Janilce Bosquiroli.

1. Redes de computadores. 2. Gerência de redes. 3. Criação e visualização de domínios dinâmicos. I. Almeida, Maria Janilce Bosquiroli. II. Título.

Agradecimentos

Aos meus pais, Luizer e Maria Antonieta Ceccon, pela confiança que sempre depositaram em mim e pelo apoio e carinho que sempre prestaram.

A minha namorada, Patrícia Albuquerque Menti, pelo amor, incentivo e paciência demonstrados.

Ao meu irmão, Gabriel Bartz Ceccon, pelo carinho, ajuda e apoio prestados nos momentos de dificuldade.

A todos os meus colegas do Mestrado em Ciência da Computação da UFRGS, pelo companheirismo, sonhos e alegrias compartilhados. Em especial, a Ricardo Neisse e Leandro Vaguetti.

Aos professores Lisandro Zambenedetti Granville e Maria Janilce Bosquiroli, pelo auxílio, dedicação, amizade e pelos caminhos apontados, sem os quais este trabalho não se completaria.

Sumário

Lista de Abreviaturas

2D	Duas Dimensões
3D	Três Dimensões
ATM	Asynchronous Transfer Mode
ASN.1	Abstract Syntax Notaion One
BB	Bandwith Broker
CIM	Common Information Model
DiffServ	Differentiated Services
DMTF	Distributed Management Task Force
DNS	Domain Name Server
ER	Entidade-relacionamento
GDMO	Guidelines for the Definition of Managed Objects
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
ISSO	International Organization for Standardization
ISP	Internet Service Provider
ITU-T SG4	ITU-T Study Group 4
LDAP	Lightweight Directory Access Protocol
MOF	Managed Object Format
MIB	Management Information Base
MLM	Mid-Level Manager
MRTG	Multi-Router Traffic Grapher
OID	Object Identifier
PBNM	Policy-Based Network Management
PDP	Policy Decision Point
PIB	Policy Information Base
QoS	Quality of Service
RFC	Request for Comments
RSVP	Resource Reservation Protocol
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SPPI	Structure of Policy Provisioning Information
SQL	Structured Query Language
UML	Unified Modeling Language

Lista de Figuras

Resumo

No contexto do gerenciamento de redes de computadores, domínios são recursos utilizados para agrupar objetos gerenciáveis. Os mapas de rede utilizados pelos sistemas de gerenciamento são exemplos bastante comuns do uso de domínios. Domínios são importantes porque as ações de gerenciamento podem ser aplicadas a todos os objetos gerenciáveis membros de um domínio ao mesmo tempo, não sendo necessário, então, repetir a mesma ação em cada objeto gerenciável, um a um. Domínios que necessitam ser rapidamente criados, utilizados e descartados são referenciados nesta dissertação de mestrado como domínios dinâmicos. Atualmente, na maioria dos sistemas de gerenciamento de redes não existem facilidades disponíveis para suportar o conceito desse tipo de domínios. Em relação aos aspectos de visualização, a apresentação visual de domínios deve ser realizada de forma adequada, visto que, atualmente, as GUIs estão presentes na maioria dos sistemas de gerenciamento. Entretanto, os processos de visualização de domínios utilizados pelos sistemas de gerenciamento atuais apresentam limitações em relação à configuração de características visuais dos domínios apresentados. Essas características são estáticas, não permitindo ao usuário, dessa forma, escolher como deseja visualizar um determinado domínio.

Nesta dissertação de mestrado são apresentados a definição e o desenvolvimento de duas novas linguagens que têm por objetivo aperfeiçoar a criação e a visualização de domínios dinâmicos. A primeira linguagem é baseada em um modelo de informação do ambiente gerenciado e é utilizada para criar, de forma automatizada, novos domínios dinâmicos. Tal modelo de informação é formado por classes, atributos e relacionamentos entre as classes. A segunda linguagem, por sua vez, é usada para configurar características visuais dos domínios dinâmicos criados por meio da primeira linguagem.

Esta dissertação apresenta, também, um protótipo desenvolvido para suportar as linguagens de criação e visualização de domínios dinâmicos criadas. A partir desse protótipo, é possível criar domínios dinâmicos relacionados a informações da rede de computadores utilizada, bem como personalizar a visualização dos domínios dinâmicos criados. Além disso, o protótipo possibilita, também, a obtenção de algumas informações dos membros dos domínios dinâmicos criados. O protótipo é baseado na Web e foi desenvolvido utilizando-se as tecnologias PRECCX, PHP, MySQL e SNMP.

Como será visto ao final, as linguagens definidas e o protótipo desenvolvido mostram que o suporte a domínios dinâmicos objetivado pode ser efetivo e melhorar, sensivelmente, os processos de gerenciamento de redes. A integração da implementação desenvolvida junto ao ambiente QAME, por exemplo, permite atualmente que os administradores possam selecionar, através das linguagens, os equipamentos de uma rede com QoS que precisam ser configurados através do processo de aplicação de políticas original do QAME.

Palavras-chave: redes de computadores, gerência de redes, criação e visualização de domínios dinâmicos.

TITLE: "DEFINITION AND VISUALIZATION OF DYNAMIC DOMAINS IN NETWORK MANAGEMENT ENVIRONMENTS"

Abstract

In the computer networks management context, domains are facilities used to group managed objects. The network maps presented in the management systems are the most common examples of the use of domains. Domains are important because the management actions can be applied to all managed objects that are members of a domain at the same time, not being necessary, then, to repeat the same action in each managed object one by one. Domains that need to be quickly created, used and discarded are referenced in this work as dynamic domains. Currently, in the majority of the available network management systems there are no proper facilities to support this type of domains. Concerning the visualization aspects, the domains visual presentation must be carried through of adjusted form, since, currently, the GUIs are present in the majority of the management systems. However, the domains visualization processes used by the current management systems present limitations related to the visual features configuration of the presented domains. These features are static, not allowing the user, this way, to choose how he or she desires to visualize one determined domain.

This work presents the definition and the development of two new languages whose goals is to enhance the creation and the visualization of dynamic domains. The first language is based on a managed environment information model and is used to create, forming an automated fashion, new dynamic domains. Such information model is formed by classes, attributes and relationships between the classes. The second language, on its turn, is used to configure visual features of the dynamic domains created by the first language.

This work also presents a prototype developed to support the defined dynamic domains creation and visualization languages. With this prototype it is possible to create dynamic domains based on the information of the computer network used, as well as customizing the visualization of the dynamic domains created. Moreover, the prototype makes possible the attainment of some members information of the dynamic domains created. The prototype is based on the Web and was developed using technologies as PRECCX, PHP, MySQL and SNMP.

As will be seen in the end, the defined languages and the developed prototype shows that the desired dynamic domains support can be effective and improve, significantly, the network management processes. The integration of the implementation developed together to the QAME environment, for example, allows currently that the administrators can select, through the languages, the QoS network devices that need to be configured through the original policies application process of the QAME.

Keywords: computer networks, network management, definition and visualization of dynamic domains.

1 Introdução

A evolução das telecomunicações e da informática, associada à necessidade de comunicação imposta pelo mundo atual, impulsionou o crescimento do número de redes de computadores existentes. No entanto, para que possam estar sempre operando de forma adequada, as redes necessitam ser gerenciadas. O elevado número de dispositivos presentes nas redes atuais, assim como suas diversidades, têm contribuído para o aumento da complexidade do gerenciamento de redes. Em sistemas tradicionais, os administradores de rede utilizam mapas visuais que permitem acessar as informações dos dispositivos de rede de interesse. Para redes menos complexas, o gerenciamento orientado a dispositivos pode ser satisfatório, mas para redes mais complexas este tipo de gerenciamento não é suficiente. Nesse contexto, o que atualmente é chamado de gerenciamento através de domínios apresenta-se como uma alternativa interessante. No gerenciamento através de domínios, ações de gerenciamento são aplicadas a domínios de dispositivos, ao invés de serem aplicadas a cada dispositivo gerenciável em separado. Outro aspecto relevante em relação ao gerenciamento por domínios diz respeito a sua visualização, a qual pode melhorar os níveis de percepção e entendimento em certas situações.

Domínios são recursos utilizados pelos sistemas de gerenciamento de redes para agrupar objetos gerenciáveis sob um determinado ponto de vista [SLO 89]. Os mapas de rede utilizados por esses sistemas são exemplos bastante comuns do uso de domínios. Esses mapas, geralmente, agrupam os dispositivos que pertencem a um mesmo segmento de rede, mas poderiam agrupar os dispositivos segundo outros aspectos. Por exemplo, domínios podem agrupar dispositivos que possuem funcionalidades ou características semelhantes, como servidores de correio eletrônico ou roteadores. No gerenciamento por domínios, as ações passam a ser aplicadas a todos os dispositivos membros de um domínio ao mesmo tempo, e não mais a cada dispositivo separadamente. As ações de gerenciamento aplicadas aos domínios são automaticamente repassadas aos membros do mesmo, não sendo necessário, então, repetir a mesma ação em cada dispositivo gerenciável, um a um.

A criação de domínios pode ser realizada com o auxílio de algumas facilidades. Por exemplo, nos sistemas de gerenciamento padrão os mapas de rede que contêm os dispositivos, enlaces e sub-redes da rede gerenciada, podem ser gerados através de mecanismos de descoberta de topologia. Por outro lado, o administrador de rede pode também criar os seus próprios domínios de forma manual, por exemplo, arrastando os dispositivos gerenciáveis desejados para novos mapas por meio de facilidades de interface de usuário. Após serem criados, os domínios normalmente são armazenados em base de dados para uso posterior. Existem, porém, algumas situações em que determinados domínios precisam ser rapidamente criados, utilizados e descartados. Para o melhor entendimento dessas situações, será utilizado um cenário no qual um sistema de gerenciamento baseado em políticas (*Policy-Based Network Management* - PBNM) [SLO 94] é adotado no gerenciamento de uma rede com suporte a QoS. Nesse cenário de exemplo, a ocorrência de uma sessão de videoconferência entre os *hosts* A e B, sendo os roteadores intermediários desconhecidos, faz com que o administrador da rede citada tome as seguintes providências:

1. Crie uma política que defina a QoS esperada para a sessão em questão e armazene-a em um repositório de políticas;

2. Descubra os roteadores entre A e B. Os roteadores identificados devem ser adicionados ao domínio da sessão citada. Quando o último roteador entre A e B for descoberto, o caminho inverso, de B para A, deve também ser verificado, já que os caminhos de ida e volta podem ser diferentes. Todos os novos roteadores descobertos devem ser adicionados ao domínio da sessão;
3. Aplique a política de QoS, armazenada anteriormente, no domínio da sessão, ao invés de aplicar a política em cada roteador da sessão entre A e B, um a um;
4. Descarte o domínio da sessão após o término da videoconferência e o fechamento da sessão entre A e B.

O cenário apresentado demonstra duas características interessantes de domínios:

- Algumas vezes, a criação de domínios pode ser lenta, principalmente quando é feita manualmente pelos administradores de rede;
- Alguns domínios têm tempo de vida pequeno, pois são utilizados para um determinado propósito e depois são descartados.

Com relação à primeira característica, o tempo necessário para a criação de um domínio deve ser menor que o tempo de vida desse mesmo domínio. Por exemplo, a criação do domínio da sessão da videoconferência comentada anteriormente deve ser mais rápida que o tempo de vida da videoconferência, senão a política de QoS definida e armazenada será aplicada aos roteadores da sessão após o término da videoconferência. Domínios que precisam ser rapidamente criados, utilizados e descartados são referenciados nesta dissertação de mestrado por meio do termo “domínios dinâmicos”.

Em relação aos aspectos de visualização, a apresentação visual de domínios deve ser realizada de forma adequada, visto que, atualmente, as GUIs estão presentes em qualquer sistema de gerenciamento. A visualização gráfica de domínios pode reduzir o tempo de procura de algumas informações em um sistema de gerenciamento e facilitar a comparação de tais informações e, principalmente, destacar informações não perceptíveis. Normalmente, mapas visuais são usados na apresentação de dispositivos, enlaces e sub-redes. Os componentes desses mapas, na maioria das vezes, são representados por diferentes imagens ou formas geométricas. As cores utilizadas na apresentação visual dos dispositivos podem indicar a ocorrência de algum evento ou até mesmo a ocorrência de um determinado problema. Outras formas visuais de apresentação de domínios também podem ser utilizadas, como por exemplo, tabelas, gráficos ou matrizes.

Os processos de visualização utilizados pelos sistemas de gerenciamento atuais apresentam limitações em relação à configuração de características visuais dos domínios apresentados. Essas características são estáticas, não permitindo, dessa forma, ao usuário escolher como deseja visualizar um determinado domínio. Por exemplo, os dispositivos que apresentam problemas, geralmente são destacados nos mapas de rede com a cor vermelha. Porém, o usuário é incapaz de alterar esta cor ou escolher uma outra cor para destacar os dispositivos que possuem suporte a *DiffServ* (*Differentiated Services*) [BER 2002], por exemplo.

Nesse contexto, dois principais problemas podem ser identificados: aplicações de gerenciamento de redes convencionais possuem pouco ou nenhum suporte ao conceito de

domínios dinâmicos, e o suporte fornecido à visualização de domínios é restrito a poucas opções de configuração. Assim, no presente trabalho são apresentados a definição e o desenvolvimento de duas novas linguagens que têm por objetivo aperfeiçoar a criação e a visualização de domínios dinâmicos. A primeira linguagem é baseada em um modelo de informação dinâmico e é usada para criar, de forma automatizada, novos domínios dinâmicos. A segunda linguagem é utilizada para configurar características visuais dos domínios que forem criados por meio da primeira linguagem.

O trabalho apresenta, também, um protótipo desenvolvido para suportar as linguagens criadas. Pelo protótipo, é possível criar domínios relacionados a informações da rede utilizada, bem como configurar a visualização dos domínios criados. O protótipo é baseado na Web e foi desenvolvido utilizando-se PRECCX [BRE 2002], PHP [PHP 2002], MySQL [YAR 99] e SNMP [CAS 90].

O restante deste trabalho está assim dividido: no capítulo 2, são apresentados os trabalhos relacionados e o estado da arte em relação ao gerenciamento de redes através de domínios, bem como as questões de visualizações em sistemas de gerenciamento de redes. O capítulo 3 apresenta a primeira contribuição desta dissertação, em que são introduzidas as linguagens para criação e visualização de domínios dinâmicos. No capítulo 4 são apresentadas a forma como o protótipo desenvolvido suporta as linguagens criadas e o funcionamento desse protótipo. Por fim, no capítulo 5, são apresentadas as conclusões resultantes desta dissertação e os trabalhos futuros.

2 Trabalhos Relacionados

A noção de domínios tem sido adotada no gerenciamento de redes de computadores de grande porte como uma alternativa ao gerenciamento tradicional. Domínios são estruturas hierárquicas utilizadas para agrupar objetos gerenciáveis sob um determinado aspecto (por exemplo, o agrupamento de roteadores com suporte a QoS). No gerenciamento baseado em domínios, as ações de gerenciamento (por exemplo, o uso de políticas) deixam de ser aplicadas diretamente aos objetos gerenciáveis em separado e passam a ser aplicadas sobre domínios.

Os objetos gerenciáveis utilizados na formação de domínios precisam ser, de alguma forma, definidos. O uso de modelos de informação e de dados no processo de definição e desenvolvimento de objetos gerenciáveis é de grande importância neste contexto, visto que os recursos de rede estão cada vez mais diversificados e complexos. Em gerência de redes, modelos são as abstrações construídas para que objetos gerenciáveis sejam compreendidos antes de serem implementados. O modelo de informação é responsável por descrever os objetos gerenciáveis em um nível conceitual. O modelo de dados, por sua vez, define os objetos em um nível mais baixo de abstração, que inclui detalhes específicos de implementação e de protocolos. O modelo conceitual é mais adequado à interpretação humana, enquanto o modelo de dados é importante porque inclui detalhes necessários à manipulação dos objetos gerenciáveis em uma implementação.

Além de serem importantes porque manipulam e agrupam os objetos gerenciáveis, os domínios também devem ser apresentados pelos sistemas de gerenciamento de redes de forma adequada. A visualização gráfica de domínios aumenta a capacidade de compreensão de certas informações e facilita a execução de tarefas relacionadas ao gerenciamento. Normalmente, as informações de um domínio são representadas visualmente por meio de mapas de rede. Entretanto, outras formas visuais podem ser utilizadas, como por exemplo, gráficos estatísticos.

Neste capítulo, serão vistos o conceito de domínio e o seu uso no gerenciamento de redes, a caracterização dos modelos de informação e de dados e as formas de visualização de informações de gerenciamento utilizadas pelos sistemas atuais. Por fim, o capítulo encerra definindo os problemas investigados nesta dissertação: as aplicações de gerenciamento de redes convencionais possuem pouco ou nenhum suporte ao conceito de domínios dinâmicos, e o suporte fornecido à visualização de domínios é restrito a poucas opções de configuração.

2.1 Domínios

Segundo Bauer [BAU 97], objetos gerenciáveis são abstrações de recursos que podem ser gerenciados em um sistema de gerenciamento de redes de computadores. Interfaces de rede, tabelas de roteamento e processos de conformação de tráfego são exemplos de objetos gerenciáveis. Esses objetos podem ser inspecionados por meio de operações de leitura (por exemplo, a obtenção da tabela de roteamento de um determinado roteador) ou configurados através de operações de escrita (por exemplo, a configuração do processo de conformação de um roteador de borda).

Nos sistemas tradicionais, administradores de rede utilizam mapas gráficos para acessar e manipular os objetos gerenciáveis desejados. Para redes menos complexas, o gerenciamento orientado a objetos gerenciáveis é considerado satisfatório. No entanto, para redes de grande porte, esse tipo de gerenciamento torna-se inadequado devido ao elevado número de objetos gerenciáveis existentes. Nesse contexto, o gerenciamento orientado a domínios apresenta-se como uma solução interessante. No gerenciamento de redes através de domínios, as ações deixam de ser aplicadas diretamente sobre objetos gerenciáveis e passam a ser aplicadas a domínios.

Domínios são conjuntos de objetos gerenciáveis que foram explicitamente agrupados sob algum ponto de vista [SLO 89]. Diversas razões existem para se agruparem objetos gerenciáveis em domínios; algumas são apresentadas a seguir:

- Ocultar a escalabilidade e a complexidade inerente nos recursos gerenciáveis;
- Especificar hierarquias de objetos, nas quais ações de gerenciamento possam ser aplicadas;
- Promover a reunião de objetos gerenciáveis que possuam funções, serviços ou atributos semelhantes;
- Especificar limites de autoridade e responsabilidade no gerenciamento dos objetos e
- Promover o uso de rótulos permitindo a localização e a identificação de grupos de objetos.

Objetos gerenciáveis tornam-se membros de um domínio ao serem agrupados por ele. Por outro lado, um domínio é conhecido como domínio-pai de objetos gerenciáveis ao agrupá-los. No exemplo apresentado na figura 2.1 a, os objetos gerenciáveis números 1 e 2 são membros do domínio D1, enquanto este mesmo domínio é o domínio-pai destes objetos.

Domínios, por sua vez, também são objetos gerenciáveis e podem fazer parte de outros domínios, tornando possível a existência de uma hierarquia de domínios [BOU 94]. Um domínio, ao ser membro de um outro domínio, é chamado de subdomínio do domínio-pai. Por exemplo, o domínio D3 apresentado na figura 2.1 a é um subdomínio dos domínios D1 e D2. O uso de subdomínios possibilita a divisão de grandes conjuntos de objetos gerenciáveis pertencentes a um domínio e a execução de ações de gerenciamento em subconjuntos do grupo original. Objetos gerenciáveis, ao serem membros de subdomínios, são também membros indiretos do domínio ancestral. Por exemplo, na figura 2.1 a, os membros do domínio D3 são membros indiretos do domínio D1, que é o domínio ancestral.

Como apresentado na figura 2.1 a, sob o ponto de vista do usuário, domínios contêm os objetos gerenciáveis que agrupam. No entanto, como apresentado na figura 2.1 b, domínios possuem apenas referências a tais objetos [SLO 93]. Como conseqüência, um objeto gerenciável pode ser membro de diversos domínios ao mesmo tempo, possuindo, assim, diferentes domínios-pai. Por exemplo, o objeto gerenciável número 3 faz parte dos domínios D1 e D2, da mesma forma que o domínio D3 também faz parte desses domínios.

Para cada objeto gerenciável deve existir apenas uma identificação associada. Essa identificação deve ser única em todo o sistema gerenciável de forma a permitir um

gerenciamento consistente. Com a finalidade de tornar o gerenciamento baseado em domínios mais simples, para cada objeto gerenciável deve haver um nome associado. O caminho de um objeto gerenciável é dado pela concatenação do nome de seus ancestrais e o nome do próprio objeto. Na figura 2.1 a, por exemplo, o caminho do objeto gerenciável número 5 é dado pela seqüência D1/D3/5. Como objetos podem pertencer a diferentes domínios simultaneamente, um objeto pode ter mais de um caminho. Por exemplo, o objeto gerenciável número 5 pode ser encontrado através do caminho D1/D3/5 ou pelo caminho D2/D3/5.

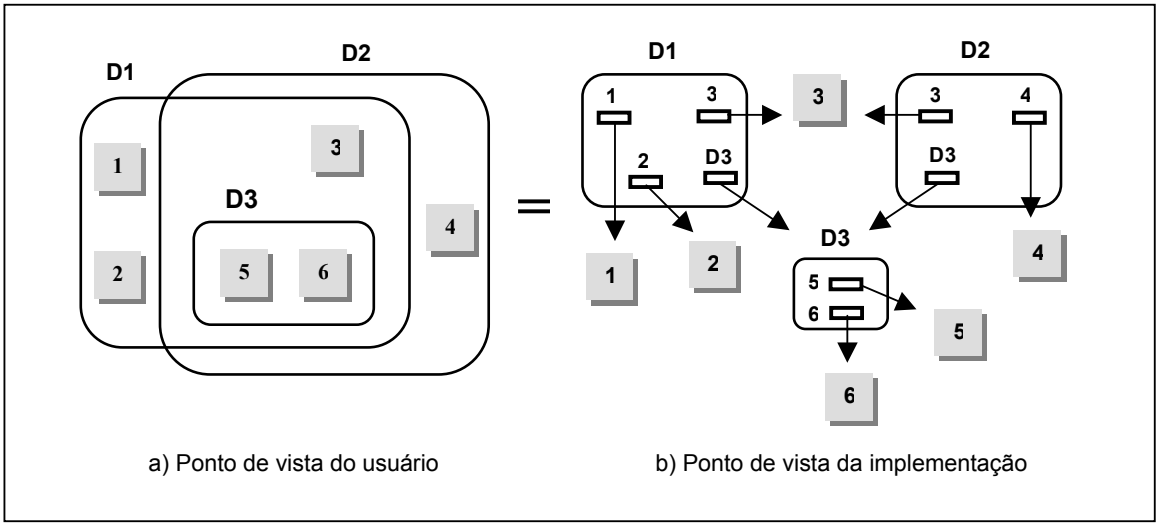

FIGURA 2.1 – Domínios em relação aos pontos de vista do usuário e da implementação

Ações de gerenciamento (por exemplo, a aplicação de políticas) são automaticamente executadas em todos os membros de um domínio quando aplicadas a ele [DAM 2002]. Em outras palavras, pode-se dizer que os objetos gerenciáveis pertencentes a um domínio herdam ações aplicadas aos seus pais diretos e indiretos. Por exemplo, os objetos gerenciáveis números 1, 2, 3, 4, 5 e 6 herdarão qualquer ação aplicada sobre o domínio D1, porém somente os objetos gerenciáveis números 5 e 6 herdarão ações aplicadas sobre o domínio D3.

Domínios podem ser vistos como uma generalização de estruturas de diretório, pois apresentam algumas características semelhantes às mesmas, como por exemplo:

- A utilização de referências a objetos gerenciados como se fossem vínculos para arquivos reais;
- O fornecimento de visibilidade e acesso ao espaço de objetos gerenciáveis a gerentes externos;
- O uso de uma forma flexível e estruturada de rotulação de objetos gerenciáveis.

Diversos projetos relacionados a domínios têm sido desenvolvidos ao longo dos anos. Sloman et al, além de introduzir os conceitos básicos de domínios apresentados anteriormente [SLO 89], demonstrou também como domínios podem ser utilizados no gerenciamento de redes celulares [SLO 93] e aplicados ao gerenciamento baseado em políticas (PBNM) [SLO 94]. No primeiro caso, o emprego de domínios como meio de

especificação e controle do uso de serviços contratados por usuários em uma rede internacional de celulares é abordado. Como estudo de caso foi utilizada a infra-estrutura da rede Pan Européia de Celulares, que é formada por múltiplas operadoras e tem a possibilidade de fornecer diversos serviços. No segundo caso, domínios são utilizados no gerenciamento automatizado de grandes redes e sistemas distribuídos para especificar o escopo de aplicação de políticas, ou seja, para agrupar objetos gerenciáveis em que políticas serão aplicadas.

Kar, Keller e Calo utilizaram a noção de domínios no gerenciamento fim-a-fim de serviços de aplicação fornecidos por ISPs a seus clientes [KAR 2000], como por exemplo, serviços de e-mail e compartilhamento de arquivos, hospedagem de conteúdo para dados com acesso baseado na Web, segurança baseada em *firewall*, entre outros. A rede global da IBM, que possui pontos de presença em mais de 800 cidades e 100 países, foi utilizada como cenário neste estudo.

Miranda, Nogueira e Machado usaram domínios para simplificar a análise de dados originados do gerenciamento de redes de telecomunicações [MIR 2002]. Visto que a quantidade de informações nestes sistemas é muito grande, domínios foram utilizados como espécies de filtros para restringirem os dados a serem analisados. Por exemplo, é possível analisar dados provenientes do gerenciamento por regiões, sub-regiões, entre outros domínios. A empresa brasileira de telecomunicações Telemar foi utilizada como estudo de caso.

2.2 Modelos de Informação e de Dados no Gerenciamento de Redes

Em gerência de redes, o **modelo de informação** tem como principal objetivo a modelagem de objetos gerenciáveis em um nível conceitual, independente de qualquer implementação ou protocolo usado no transporte dos dados [PRA 2002]. O grau de detalhe das abstrações definidas neste modelo depende das necessidades de modelagem da pessoa responsável por projetar os objetos gerenciáveis. O modelo de informação deve ocultar todos os protocolos e detalhes de implementação para tornar o projeto de gerência, como um todo, o mais simples possível. Outra característica importante deste modelo é a possibilidade de definição de relacionamentos entre os objetos gerenciáveis.

Modelos de dados, por outro lado, são definidos em um nível mais baixo de abstração, sendo direcionados aos programadores de objetos gerenciáveis [WES 2001]. Como resultado, estes modelos possuem vários detalhes e informações relacionadas a protocolos específicos. A relação existente entre o modelo de informação e o modelo de dados é apresentada na figura 2.2. Visto que um modelo conceitual pode ser implementado de várias formas, diversos modelos de dados podem ser derivados de um único modelo de informação.

Em algumas situações, ainda que os modelos de informação e de dados tenham diferentes finalidades, pode haver a sobreposição dos mesmos. A ocorrência desse fato faz com que a tarefa de definir exatamente a que modelo uma determinada abstração pertence torne-se uma atividade bastante difícil. As subseções a seguir apresentam, em maiores detalhes, os modelos de informação e de dados.

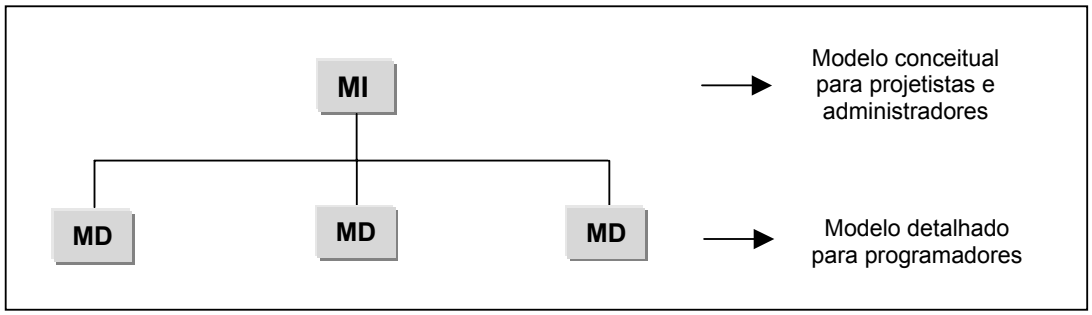

FIGURA 2.2 – Relação entre os modelos de informação e de dados

Modelo de Informação

Modelos de informação são úteis para a descrição do ambiente gerenciado por parte dos projetistas, para os administradores entenderem os objetos gerenciáveis modelados e para orientar os programadores na descrição e codificação de funcionalidades nos modelos de dados. Os termos modelo conceitual e modelo abstrato, encontrados na literatura, são utilizados como sinônimos de modelo de informação. Esses modelos podem ser implementados de diferentes maneiras e mapeados em diversos protocolos. Assim sendo, pode-se dizer que os modelos de informação são protocolos neutros [PRA 2002].

A especificação de relacionamentos entre objetos gerenciáveis é uma importante característica dos modelos de informação. Por meio dessa característica, organizações podem utilizar o conteúdo de um modelo de informação para delimitar as funcionalidades que podem ser incluídas em um modelo de dados.

Linguagens naturais podem ser utilizadas na definição informal de modelos de informação. Cita-se como exemplo a RFC 3290 [BER 2002], na qual o inglês foi utilizado na descrição do modelo conceitual de um roteador *DiffServ* e na especificação dos relacionamentos entre os componentes deste dispositivo. No IETF (*Internet Engineering Task Force*), os modelos de informação devem ser explicitamente descritos, e estes modelos e os modelos de dados devem ser especificados em RFCs distintas. Neste caso, o documento especificando o modelo de informação normalmente é uma RFC informativa, enquanto o documento responsável por definir o modelo de dados segue o padrão determinado na RFC 2026 [BRA 96]. Em geral, a maioria das RFCs que definem módulos de MIBs (*Management Information Base*) SNMP (*Simple Network Management Protocol*) incluem descrições informais para explicar a modelagem destes módulos. No entanto, por serem descrições rudimentares e incompletas, podem ser consideradas como um esboço de um modelo de informação ideal. Cita-se, como exemplo, a MIB do grupo interfaces definida na RFC 2863 [MCC 2000].

Entretanto, modelos de informação podem ser definidos por meio do uso de uma linguagem formal ou uma linguagem semiformal estruturada. Uma das possibilidades de especificação formal de um modelo de informação é através do uso de diagramas de classe UML [OMG 2001]. Apesar desses diagramas serem raramente utilizados pelo IETF na definição de modelos de informação, muitas outras organizações os usam. Citam-se como exemplos: DMTF (*Distributed Management Task Force*), ITU-T SG4 (*ITU-T Study Group 4*), o ATM fórum, entre outros. A principal vantagem da utilização de diagramas de classes

UML é a possibilidade de representar os objetos gerenciáveis e os relacionamentos entre eles de uma forma gráfica padronizada, fazendo com que projetistas e administradores tenham uma compreensão mais simples do modelo de gerenciamento em questão. No entanto, existem outras técnicas de representação gráfica de objetos e relacionamentos, como por exemplo, os diagramas entidade-relacionamento (ER).

Modelo de Dados

Modelos de dados definem objetos gerenciáveis em um nível mais baixo de abstração que os modelos de informação. Esse nível de abstração inclui detalhes específicos de implementação e de protocolos. A maioria dos modelos de gerenciamento padronizados são modelos de dados. Citam-se como exemplos:

- Os módulos das MIBs definidos pelo IETF. A linguagem (sintaxe) utilizada para definir estes modelos de dados é chamada de SMI (*Structure of Management Information*) [MCC 99] e é derivada da ASN.1 (*Abstract Syntax Notaion One*) [IOF 87].
- Os módulos das PIBs (*Policy Information Base*) definidos pelo IETF. Sua sintaxe é definida pela linguagem SPPI (*Structure of Policy Provisioning Information*) [MCC 2001], que é semelhante ao SMI, e é, também, derivada do ASN.1.
- Os módulos das MIBs definidos pela ISO e, atualmente, mantidos e aperfeiçoados pelo ITU-T. A sintaxe destes modelos de dados é especificada no GDMO (*Guidelines for the Definition of Managed Objects*) [ITU 92]. Os módulos das MIBs GDMO fazem uso de princípios da orientação a objetos.
- Os esquemas CIM (*Common Information Model*) [DMT 99] desenvolvidos pelo DMTF. Esses esquemas são publicados em duas formas: gráficas e textuais. A forma gráfica faz uso de diagramas UML e não é padronizada. A forma textual é padronizada e escrita na linguagem MOF (*Managed Object Format*) [DMT 99]. Os esquemas CIM são orientados a objetos.

A figura 2.3 resume os exemplos de modelos de dados apresentados anteriormente. As linguagens utilizadas na definição dos mesmos estão entre parênteses.

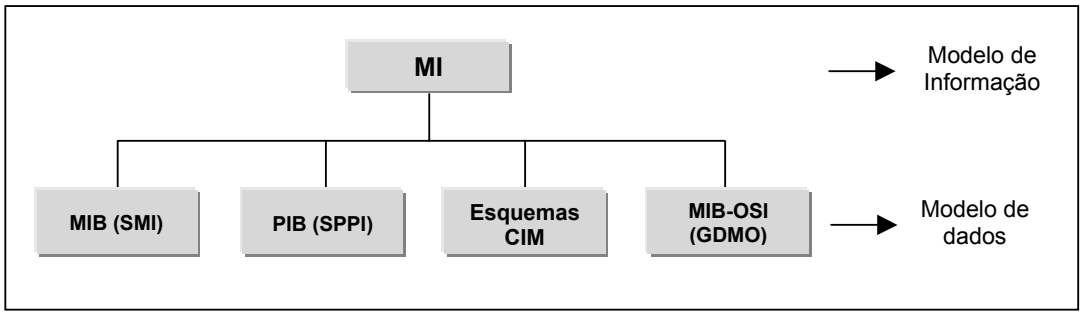

FIGURA 2.3 – Exemplos de modelos de dados padronizados

Com o objetivo de ilustrar os detalhes incluídos em um modelo de dados, os módulos das MIBs do IETF serão considerados como exemplo. Diferentemente do que

ocorre nos modelos de informação, os módulos das MIBs do IETF incluem detalhes como associações de identificadores de objetos (OIDs) e estruturas de indexação. Os relacionamentos definidos nos modelos de informação são implementados nestes módulos como ponteiros para OIDs ou por meio de relacionamentos de indexação especificados nas cláusulas INDEX. Vários outros detalhes específicos de implementação são incluídos nestes módulos, como por exemplo, as cláusulas MAX-ACCESS (permissão de acesso a um determinado objeto gerenciável, por exemplo: somente leitura e, leitura e escrita) e STATUS (indicação se o objeto deve ser obrigatoriamente implementado) apresentadas em negrito na figura 2.4.

```
ifType OBJECT-TYPE

    SYNTAX      IANAifType

    MAX-ACCESS  read-only

    STATUS      current

    DESCRIPTION

       "The type of interface. Additional values for
       ifType are assigned by the Internet Assigned
       Numbers Authority (IANA), through updating
       the syntax of the IANAifType textual
       convention."

    ::= { ifEntry 3 }
```

FIGURA 2.4 – Exemplo de uso das cláusulas MAX-ACCESS e STATUS

2.3 Visualização Gráfica no Gerenciamento de Redes

Os processos utilizados pelos sistemas de gerenciamento para mapear informações de rede em representações gráficas minimizam a complexidade das tarefas relacionadas à administração e operação de redes de computadores. A análise de informações por meio de mapas, gráficos e figuras possibilita aos administradores uma observação diferenciada da rede gerenciada. Como resultado, eles podem identificar problemas ocorridos com maior rapidez e, se possível, solucioná-los através do mesmo sistema de gerenciamento usado na análise da rede.

As informações de redes de computadores estão diretamente relacionadas com seus dispositivos e enlaces. Elas, geralmente, dividem-se em duas classes [EIC 96]:

1. Informações estatísticas associadas aos relacionamentos entre os dispositivos;
2. Informações referentes à estrutura da rede.

Os sistemas de gerenciamento normalmente utilizam gráficos estatísticos para representar a primeira classe de informações (1). Esses gráficos possuem formas variadas e, na maioria das vezes, apresentam informações referentes à monitoração de enlaces e segmentos de rede. Um exemplo de gráfico em forma de colunas é demonstrado na figura

2.5. Ele foi gerado através do *software* LanExplorer [SUN 2002] com a finalidade de identificar os protocolos de aplicação mais utilizados no segmento de rede monitorado. As informações computadas para gerar o gráfico apresentado foram coletadas em um período de aproximadamente 20 minutos. Entretanto, esse período pode ser estendido. O *software* MRTG (*Multi-Router Traffic Grapper*) [OET 2001], por exemplo, possibilita a geração de gráficos em que a escala de tempo pode variar entre dias, semanas e até meses.

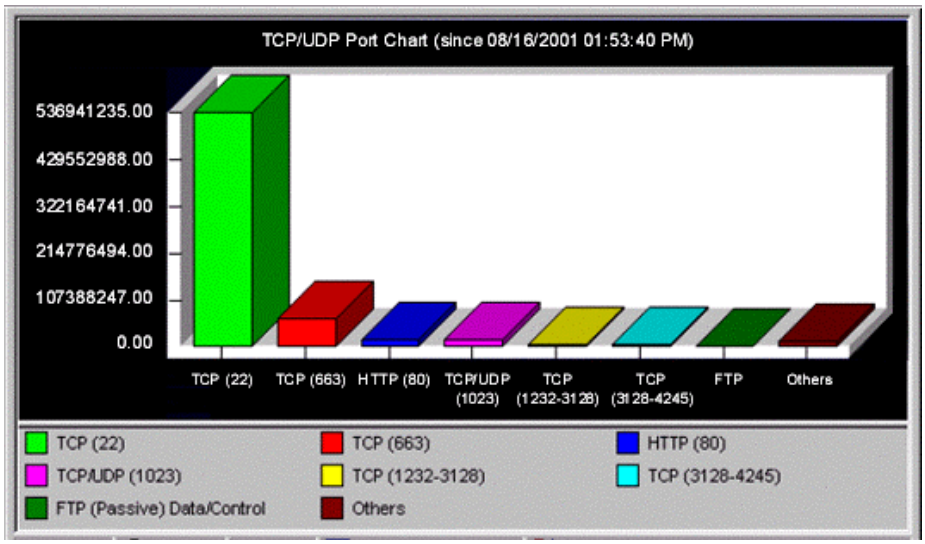

FIGURA 2.5 – Gráfico gerado pelo *software* LanExplorer

Outros tipos de gráficos podem ser utilizados na apresentação de informações associadas aos relacionamentos entre dispositivos. Cita-se, como exemplo, o gráfico gerado pelo *software* Sniffer Investigator [SNI 2002], apresentado na figura 2.6. Este gráfico mostra a matriz de acesso de uma rede de computadores qualquer. As linhas mais densas do gráfico indicam os pares de dispositivos com maior atividade.

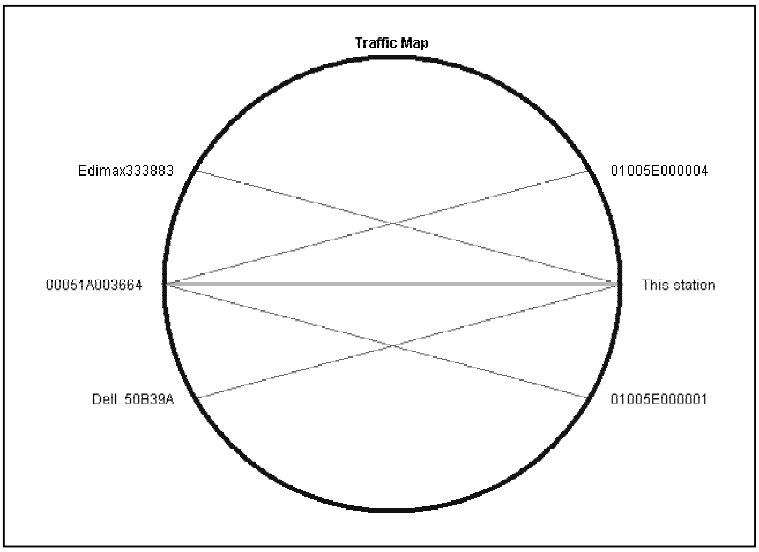

FIGURA 2.6 – Gráfico gerado pelo *software* Sniffer Investigator

A classe de informações referente à estrutura da rede (2) geralmente é apresentada pelos sistemas de gerenciamento por meio de mapas topológicos. Nestes, os dispositivos e sub-redes são representados por formas geométricas ou figuras, enquanto os enlaces são representados por linhas entre os dispositivos. Um mapa de rede que segue essas características é apresentado na figura 2.7. Para distinguir os dispositivos e sub-redes, o *software* Netplus Management and Control [CEC 2001a] [CEC 2001b], usado na geração do mapa da figura 2.7, utiliza diferentes figuras. Elas representam *hosts*, *switches,* roteadores e sub-redes (nuvens).

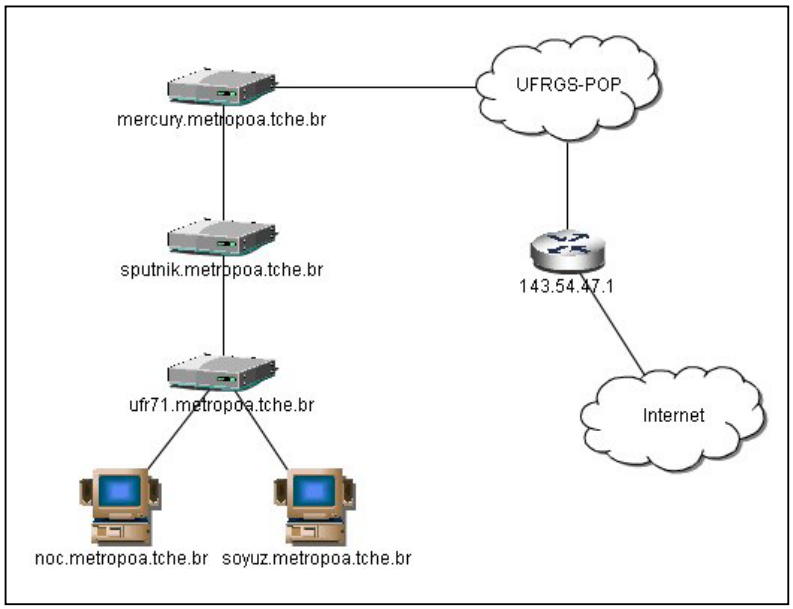

FIGURA 2.7 – Mapa de rede gerado pelo *software* Netplus Management and Control

Os dispositivos, sub-redes e enlaces apresentados nos mapas de rede podem ser destacados de forma especial para indicar a ocorrência de um determinado evento ou problema. No mapa de rede gerado pelo *software* HP OpenView [HEW 2001], apresentado na figura 2.8, por exemplo, a cor cinza-escuro (na verdade, vermelho nos gráficos gerados pelo OpenView) é utilizada para destacar os dispositivos que apresentam problemas de conexão com a rede, enquanto a cor cinza-claro (na verdade, verde) é usada para indicar os dispositivos que estão operando corretamente.

Os sistemas de gerenciamento podem também utilizar mapas de rede para apresentar a primeira classe de informações (1). O mapa gerado pelo *software* Patrol Visualis [BMC 2002], apresentado na figura 2.9, por exemplo, demonstra os dispositivos e enlaces que trafegam as maiores quantidades de dados em uma determinada rede. Esses dispositivos (A) e enlaces (B) são destacados com dimensões maiores que as dos seus semelhantes. Além disso, os enlaces que possuem tráfego excessivo são destacados, também, com a cor cinza-escuro (equivalente à cor vermelha neste *software*).

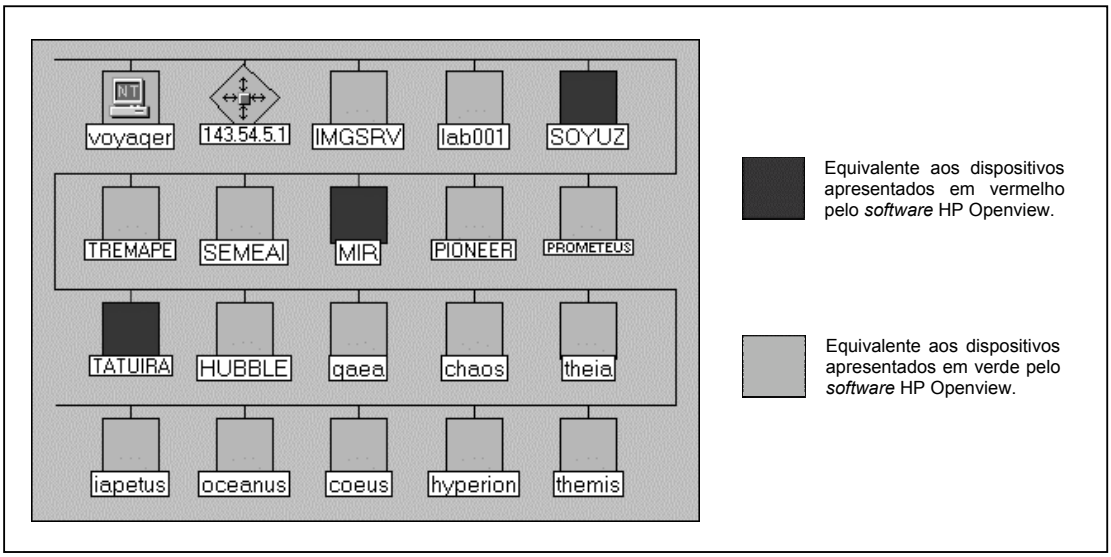

FIGURA 2.8 – Mapa de rede gerado pelo *software* HP OpenView

Os sistemas de gerenciamento citados anteriormente utilizam planos de duas dimensões (2D) para apresentar graficamente os dispositivos, sub-redes e enlaces da rede gerenciada. Entretanto, existem sistemas, como o *software* Unicenter TNG [STR 98], que possibilitam a apresentação de mapas de rede em planos de três dimensões (3D). Esses mapas, em algumas situações, evitam a sobreposição de dispositivos e o cruzamento de linhas que representam os enlaces. No entanto, seu uso não é freqüente por serem de difícil manipulação em relação aos mapas de duas dimensões.

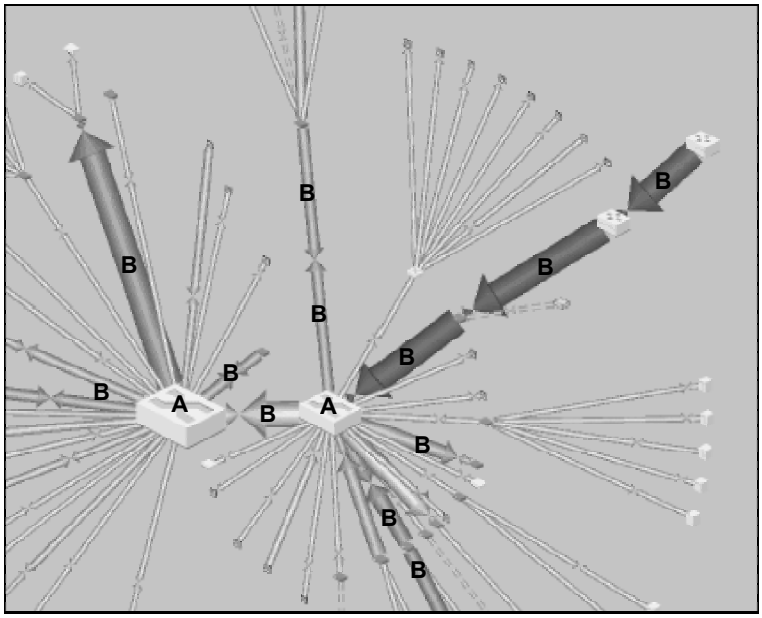

FIGURA 2.9 – Mapa de rede gerado pelo *software* Patrol Visualis

Visto que a necessidade em relação à visualização de informações de rede difere de administrador para administrador, alguns sistemas de gerenciamento permitem a configuração de parâmetros gráficos associados às informações. A possibilidade de interagir com tais sistemas pode trazer algumas vantagens para os administradores de rede, como por exemplo:

- A obtenção de apresentações visuais diferentes das impostas pelos sistemas de gerenciamento (visualizações padrão);
- O aproveitamento mais eficaz do conteúdo da apresentação visual;
- A redução do tempo gasto com a análise e compreensão dos dados visualizados.

Um exemplo de *software* (MapNet [THE 2002]) que possibilita a configuração de parâmetros relacionados às informações apresentadas é demonstrado na figura 2.10. A finalidade desse *software* é representar graficamente, em um mapa geográfico, os nodos e enlaces de alguns *backbones*. O *software* permite que, por meio de alguns botões, sejam escolhidos os *backbones* (comerciais ou não comerciais) que serão apresentados, a visão do mapa (mundo, EUA, Ásia ou Europa), a espessura das linhas que representam os enlaces dos *backbones* escolhidos, o zoom sobre o mapa, o tipo de coloração (por ISPs ou largura de banda) usada nos enlaces, entre outras opções.

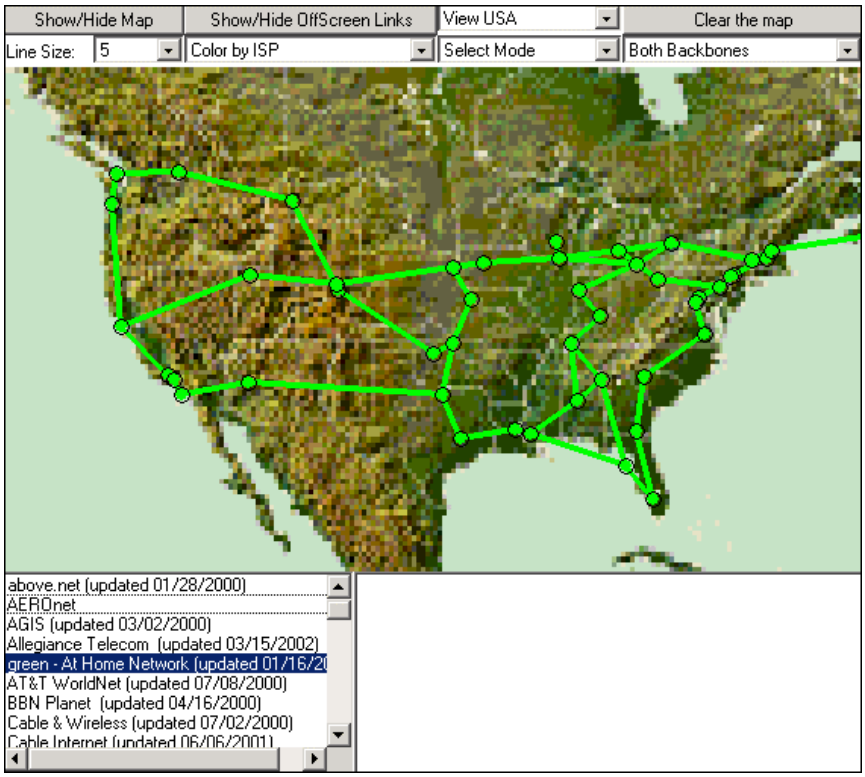

FIGURA 2.10 – Mapa de geográfico de *backbones* gerado pelo *software* MapNet

2.4 Definição do Problema

Como visto na introdução deste trabalho, existem situações em que domínios precisam ser rapidamente criados, utilizados e descartados. Por exemplo, para aplicar uma política de QoS em uma sessão de videoconferência entre dois *hosts*, um administrador deve criar uma política que defina a QoS esperada, descobrir os roteadores dos caminhos de ida e volta entre os dois *hosts* e adicioná-los ao domínio da sessão, aplicar a política de QoS no domínio da sessão e descartar o domínio da sessão após o término da videoconferência.

No entanto, a política de QoS somente terá efeito se o tempo necessário para a criação do domínio da sessão for menor que o tempo de vida da videoconferência, pois senão a política definida e armazenada será aplicada aos roteadores da sessão após o término da videoconferência, o que obviamente não é adequado.

Visto que a criação de domínios pode ser lenta, principalmente se realizada de forma manual, a existência de um mecanismo que possibilite a criação automatizada de domínios torna-se imprescindível. No entanto, as aplicações de gerenciamento de redes convencionais possuem pouco ou nenhum suporte ao conceito de domínios dinâmicos (termo usado neste trabalho para referenciar os domínios que precisam ser rapidamente criados, utilizados e descartados).

Além disso, o suporte fornecido pelos sistemas de gerenciamento à visualização de domínios também deixa a desejar, pois é restrito a poucas opções de configuração. As características visuais configuráveis normalmente são estáticas e limitadas, não permitindo, dessa forma, que o usuário escolha como deseja visualizar um determinado domínio. Por exemplo, no *software* MapNet apresentado na figura 2.10, o administrador de rede tem a opção de escolher entre dois tipos diferentes de coloração a serem usados nos nodos e enlaces dos *backbones* visualizados: por ISPs ou por largura de banda. Porém, ele não tem a possibilidade de escolher as cores utilizadas em cada uma dessas opções, pois as cores são definidas pelo *software*.

Assim, o problema investigado nesta dissertação é a falta de suporte à criação de domínios dinâmicos nas aplicações de gerenciamento atuais e a falta de suporte adequado à visualização de domínios nessas mesmas aplicações.

O capítulo seguinte apresenta a primeira contribuição desta dissertação: a definição de linguagens para a criação e visualização de domínios dinâmicos. A primeira linguagem permite a criação de domínios de forma automatizada, enquanto a segunda linguagem possibilita a configuração de características visuais dos domínios criados com a primeira linguagem.

3 Criação e Visualização de Domínios Dinâmicos

O capítulo anterior identificou a ocorrência de dois problemas: as aplicações tradicionais de gerenciamento de redes de computadores possuem pouco ou nenhum suporte ao conceito de domínios dinâmicos, e o suporte fornecido à visualização de domínios é restrito a poucas opções de configuração. Este capítulo apresenta a primeira contribuição desta dissertação, que é a definição de duas novas linguagens que têm por objetivo aperfeiçoar a criação e a visualização de domínios dinâmicos. A primeira linguagem é usada para criar, de forma automatizada, novos domínios dinâmicos. A segunda linguagem é utilizada para configurar características visuais dos domínios que forem criados através da primeira linguagem.

A linguagem de criação de domínios dinâmicos opera sobre as informações disponíveis no ambiente de gerência da rede onde domínios serão criados e utilizados. Conseqüentemente, o ambiente da rede gerenciada deve ser definido, assim como o seu modelo de informação, que tem por objetivo descrever a organização das informações de gerência disponíveis.

Assim, inicialmente, o capítulo apresenta o ambiente de rede a ser gerenciado utilizado durante o desenvolvimento deste trabalho. A seguir, o modelo de informação desse ambiente é demonstrado. Por fim, as linguagens de criação e visualização de domínios dinâmicos são respectivamente apresentadas.

3.1 Ambiente Gerenciado

A linguagem proposta para automatizar a criação de domínios dinâmicos é baseada no modelo de informação da rede de computadores na qual os domínios dinâmicos serão criados e utilizados. Como conseqüência, a rede de computadores usada deve ser descrita para que as informações de gerenciamento possam ser identificadas e, posteriormente, organizadas em um modelo de informação.

A figura 3.1 apresenta o ambiente de rede utilizado no desenvolvimento deste trabalho. Inicialmente, este ambiente foi considerado como um conjunto de dispositivos heterogêneos que possuem funcionalidades e métodos de acesso distintos. Entre os dispositivos apresentados, podem-se observar *hosts*, *switches* e roteadores (figura 3.1, elementos A). A comunicação entre *hosts* de um mesmo segmento de rede é realizada através de conexões físicas com um *switch*. A comunicação entre *hosts* de diferentes segmentos de rede, por sua vez, é realizada através de conexões físicas entre *switches* e roteadores.

Além desses dispositivos, alguns serviços também foram levados em conta, como por exemplo, os serviços de diretórios (por exemplo, LDAP) e de rede (por exemplo, DNS) apresentados na figura 3.1, elementos B. Entretanto, para que esses serviços possam operar corretamente, a infra-estrutura de rede utilizada deve apresentar um bom funcionamento, pois se a rede parar por algum motivo, os serviços também param.

O modelo de gerência centralizado foi adotado no gerenciamento da rede. Conseqüentemente, apenas uma estação de gerenciamento é utilizada (figura 3.1 – elemento C). No entanto, em um ambiente de gerenciamento distribuído, outros elementos

de gerência poderiam ser usados, como por exemplo, MLMs (*Mid-Level Managers*) [GOL 96], BBs (*Bandwith Brokers*) [NIC 99] e PDPs (*Policy Decision Points*) [QOS 99].

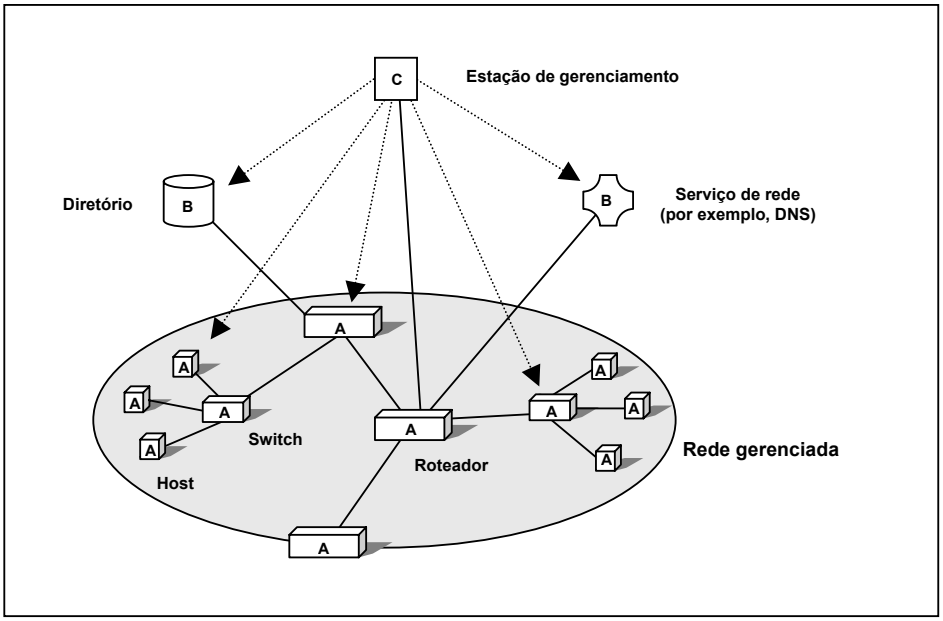

FIGURA 3.1 – Representação do ambiente de rede gerenciado

3.2 Modelo de Informação

Para que a linguagem de criação de domínios dinâmicos possa operar sobre o ambiente de rede utilizado, as informações de gerenciamento disponíveis no mesmo devem ser organizadas em um modelo de informação. A figura 3.2 apresenta, em notação UML, o modelo de informação do ambiente de rede gerenciado descrito na seção anterior. Esse modelo foi definido por meio de classes que possuem atributos e de relacionamentos com cardinalidades entre as classes.

O modelo de informação apresentado é formado pelas seguintes classes:

- *topologies* – Classe responsável por armazenar informações (atributos) das topologias (sub-redes) pertencentes à rede gerenciada.
- *devices* – Classe responsável por armazenar informações dos dispositivos que são membros de alguma topologia do ambiente gerenciado. Uma topologia pode possuir vários dispositivos, e um mesmo dispositivo pode ser membro de mais de uma topologia.
- *interfaces* – Classe responsável por armazenar informações das interfaces de rede dos dispositivos da rede gerenciada. Um único dispositivo pode possuir várias interfaces de rede. Uma interface de rede, porém, pode pertencer a somente um dispositivo.
- *ds_dropper* – Classe responsável por armazenar informações dos descartadores *DiffServ* que estão associados às interfaces de rede dos dispositivos. Uma única

interface de rede pode ter vários descartadores associados, porém um determinado descartador pode estar associado a apenas uma interface de rede.

- *ds_tokenbucket* – Classe responsável por armazenar informações dos *token buckets DiffServ* associados às interfaces de rede dos dispositivos. Uma única interface de rede pode ter vários *token buckets* associados, porém um determinado *token bucket* pode estar associado a somente uma interface.
- *rsvp* – Classe responsável por armazenar informações das sessões RSVP (*Resource Reservation Protocol*) associadas às interfaces de rede dos dispositivos. Por uma interface de rede podem passar várias sessões RSVP e uma sessão RSVP pode passar por várias interfaces de diferentes dispositivos.

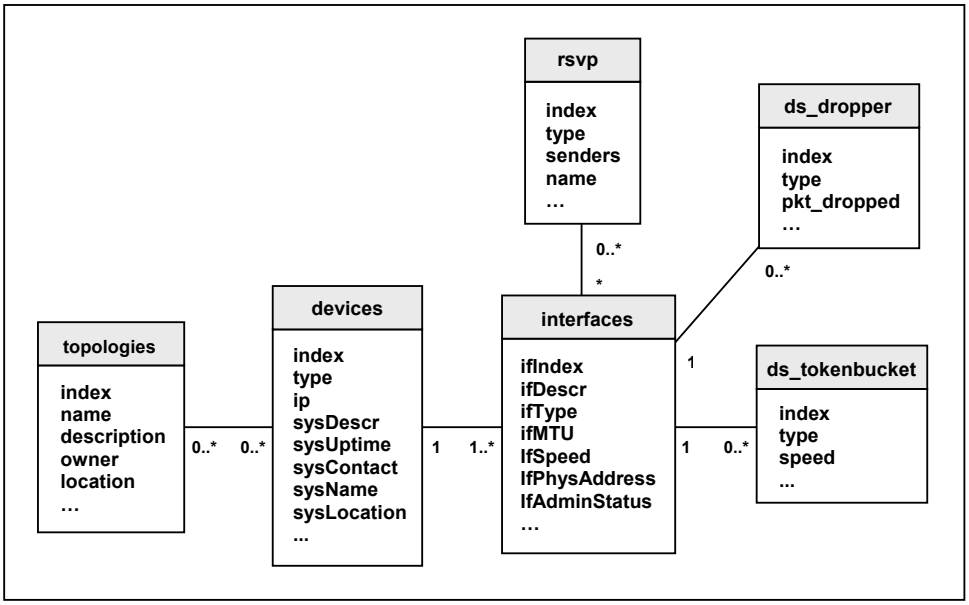

FIGURA 3.2 – Modelo de informação

Este modelo de informação, obviamente, pode ser aplicado a um conjunto limitado de redes de computadores, pois a quantidade de informações de gerenciamento é reduzida. No entanto, para possibilitar que a linguagem de criação de domínios dinâmicos não fique limitada a operar em uma única rede de computadores, assumiu-se que o modelo de informação é dinâmico e pode ser estendido para acomodar novos objetos gerenciáveis. Por exemplo, se uma nova MIB for compilada no ambiente de gerência, uma nova classe pode ser adicionada ao modelo de informação para que os objetos gerenciáveis descritos em tal MIB sejam suportados. Além disso, classes existentes no modelo também podem ser removidas.

3.3 Linguagem de Criação de Domínios Dinâmicos

As seguintes características foram consideradas na definição da linguagem de criação de domínios dinâmicos apresentada nesta seção.

- A linguagem deve operar de acordo com um modelo de informação dinâmico, no qual novas classes possam ser adicionadas e outras classes possam ser removidas;
- A linguagem deve possuir uma sintaxe simples e fácil de usar, pois os administradores de rede não estão habituados a usar linguagens, por exemplo, como o SQL;
- A linguagem deve fornecer um mecanismo capaz de selecionar objetos gerenciáveis e agrupá-los em domínios dinâmicos. A manipulação desses objetos por meio de operações de escrita não deve fazer parte do escopo da linguagem e deve ser realizada por facilidades externas à linguagem.

Assim, a definição de domínios dinâmicos é realizada a partir da elaboração de expressões de seleção que seguem a BNF apresentada na figura 3.3.

```
domain     ::= select expression
expression ::= term {from term}
term       ::= classdata {::classdata}
classdata  ::= class {.attribute[value]}
```

FIGURA 3.3 – BNF da linguagem de criação de domínios dinâmicos

Nesta BNF, os elementos essenciais para a formulação de uma expressão de seleção são os seguintes:

- `class` – elemento utilizado para identificar uma classe de um modelo de informação (por exemplo, o modelo de informação apresentado na seção 3.2);
- `attribute` – elemento utilizado para identificar um atributo de uma classe de um modelo de informação;
- `value` – elemento utilizado para selecionar os objetos (ocorrências) de uma classe que possuem um determinado atributo com um valor específico.

A linguagem de criação de domínios dinâmicos definida por meio desta BNF não é limitada a operar de acordo com um conjunto específico de classes e atributos. Conseqüentemente, a linguagem pode ser utilizada com diferentes modelos de informação. Para melhor compreensão da linguagem, alguns exemplos serão apresentados a seguir, considerando o modelo de informação abordado na seção 3.2.

```
1) select topologies
2) select topologies.owner["secAdm"]
3) select topologies.owner["secAdm"].location["admBuilding"]
```

A primeira expressão de seleção apresentada faz uso apenas do elemento `class` para identificar os objetos gerenciáveis que serão selecionados e agrupados em um domínio dinâmico. Expressões de seleção como esta geram domínios dinâmicos nos quais seus membros são todos os objetos (ocorrências) da classe indicada pelo elemento `class`. Por exemplo, na expressão número 1, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies` do modelo de informação apresentado na figura 3.2) existentes no ambiente gerenciado.

A segunda expressão de seleção apresentada utiliza o elemento `class` em conjunto com os elementos `attribute` e `value`. O elemento `attribute` é associado ao elemento `class` através de um ponto (“.”). O elemento `value`, por sua vez, é associado ao elemento `attribute` através de colchetes (“[]”). Expressões de seleção como esta geram domínios dinâmicos, nos quais seus membros são todos os objetos da classe indicada pelo elemento `class` que possuam o atributo indicado pelo elemento `attribute` com o valor indicado pelo elemento `value`. Por exemplo, na expressão número 2, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`).

Os elementos `attribute` e `value` devem ser usados em conjunto nas expressões de seleção. O uso de somente um deles não é previsto pela BNF desta linguagem. Entretanto, tais elementos podem ser associados diversas vezes a um elemento `class`. Por exemplo, na expressão de seleção número 3, os elementos `attribute` (`owner` e `location`) e `value` (`secAdm` e `admBuilding`) são utilizados duas vezes com um elemento `class` (`topologies`). Nesta expressão de seleção, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`) que estão localizadas (atributo `location` da classe `topologies`) no prédio da administração (`admBuilding`).

As próximas expressões apresentadas demonstram o uso do conector “`::`”. Esse conector permite que atributos (elemento `attribute`) de uma classe (elemento `class`) sejam utilizados para selecionar objetos de uma outra classe. Entretanto, para que isso seja possível, as duas classes conectadas por “`::`” precisam relacionar-se no modelo de informação utilizado.

```
4) select topologies::devices.type[“DSRouter”]
5) select topologies.owner[“secAdm”]::devices.type[“DSRouter”]
6) select topologies::devices::interfaces.ifType[“ATM”]
```

O conector “`::`” pode associar dois elementos `class` que possuam ou não elementos `attribute` e `value`. Por exemplo, na expressão número 4, apenas a segunda ocorrência do elemento `class` (`devices`) possui os elementos `attribute` (`type`) e `value` (`DSRouter`) associados. Expressões de seleção como esta geram domínios dinâmicos, no qual seus membros são todos os objetos da classe indicada pelo primeiro elemento `class` que possuam objetos da classe indicada pelo segundo elemento `class` que, por sua vez, tenham o atributo indicado pelo elemento `attribute` com o valor indicado pelo elemento `value`. Por exemplo, na expressão número 4, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies`) que possuam dispositivos (objetos da

classe `devices`) que sejam do tipo (atributo `type` da classe `devices`) roteador *DiffServ* (`DSRouter`).

Na quinta expressão de seleção apresentada, ambos os elementos `class` (`topologies` e `devices`) utilizados possuem elementos `attribute` e `value` associados. É importante salientar que cada elemento `class` pode ter vários elementos `attribute` e `value` associados. Nesta expressão, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`) que possuam dispositivos (objetos da classe `devices`) do tipo (atributo `type` da classe `devices`) roteador *DiffServ* (`DSRouter`).

A expressão de seleção número 6 faz uso de três elementos `class`, sendo que apenas o último possui elementos `attribute` e `value` associados. Nesta expressão, o domínio dinâmico criado será formado por todas as topologias (objetos da classe `topologies`) que possuam dispositivos (objetos da classe `devices`) com interfaces (objetos da classe `interfaces`) do tipo (atributo `ifType` da classe `interfaces`) `ATM`. Nesse caso, os seguintes relacionamentos devem existir no modelo de informação para que o domínio dinâmico seja criado de acordo com o que é solicitado na expressão de seleção:

- `topologies` → `devices`;
- `devices` → `interfaces`;

A ordem de ocorrência das classes associadas pelo conector "`::`" em uma expressão de seleção deve respeitar a hierarquia das classes correspondentes no modelo de informação utilizado. Por exemplo, de acordo com a hierarquia das classes do modelo de informação utilizado (seção 3.2), apresentada na figura 3.4, as expressões de seleção números 4, 5 e 6 estão elaboradas corretamente, porém as expressões números 7 e 8 apresentam erros em relação à ordem correta de suas classes.

```
7) select devices.type["DSRouter"]::topologies.owner["secAdm"]
8) select topologies::interfaces::devices.type["DSRouter"]
```

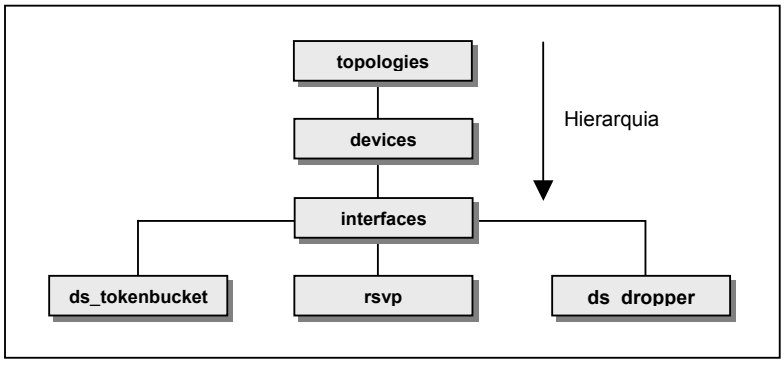

FIGURA 3.4 – Hierarquia das classes do modelo de informação utilizado (seção 3.2)

Em uma expressão de seleção, a primeira ocorrência do elemento `class` identifica a classe responsável por fornecer objetos para composição do domínio dinâmico que será formado. Por exemplo, nas expressões de seleção apresentadas anteriormente (exceto as

expressões números 7 e 8), os domínios dinâmicos formados eram compostos por objetos da classe `topologies`. Expressões que possuem objetos de outras classes como resultado de uma seleção serão apresentadas a seguir. Além disso, essas expressões demonstram o uso de operadores lógicos e relacionais no elemento `value`.

```
9)  select devices.ip["200.132.73.36" or "200.132.73.37"]
10) select devices.sysName["MIR" or "HUBBLE" or "VOYAGER" or "NOC"]
11) select interaces.ifType["ATM" or "RS232" and "ETHERNET"]
12) select interfaces.ifType["ATM"].ifSpeed[155]
13) select devices::interfaces.ifSpeed[>=100 and <=155]
```

O domínio dinâmico criado pela expressão de seleção número 9 será formado pelos dispositivos (objetos da classe `devices`) que possuírem os endereços IPs (atributo `IP` da classe `devices`) `200.132.73.36` ou `200.132.73.37`. Nesta expressão, o operador lógico `or` é utilizado no elemento `value` para possibilitar a seleção de mais de um objeto da classe `devices`. O operador lógico `and` também pode ser utilizado no elemento `value`. Entretanto, ao ser utilizado na expressão número 9, o operador `and` retornaria uma resposta somente se existisse um mesmo dispositivo com os IPs `200.132.73.36` e `200.132.73.37`. Os operadores lógicos `or` e `and` podem ser utilizados várias vezes em um elemento `value`. Por exemplo, na expressão número 10, o operador lógico `or` é utilizado para possibilitar a seleção dos dispositivos (objetos da classe `devices`) que tenham os seguintes nomes (atributo `sysName` da classe `devices`): `MIR`, `HUBBLE`, `VOYAGER` ou `NOC`. É importante mencionar que esta expressão de seleção será satisfeita mesmo que não existam dispositivos com todos esses nomes. No entanto, pela implementação atual da linguagem de criação de domínios dinâmicos, os operadores lógicos `or` e `and` não podem ser usados em um mesmo elemento `value`, como apresentado na expressão número 11.

O domínio dinâmico criado pela expressão de seleção número 12 será formado pelas interfaces (objetos da classe `interfaces`) que forem do tipo (atributo `ifType` da classe `interfaces`) ATM e tiverem velocidade (atributo `ifSpeed` da classe `interfaces`) de `155` Mbps. Os operadores relacionais <, >, <= e >= podem ser utilizados no elemento `value` quando este for associado a um atributo numérico (por exemplo, o atributo `ifSpeed` da classe `interfaces`). A expressão de seleção número 13 demonstra o uso desses operadores. Nesta expressão, o domínio dinâmico criado será formado pelos dispositivos (objetos da classe `devices`) que possuírem interfaces (objetos da classe `interfaces`) com velocidade (atributo `ifSpeed` da classe `interfaces`) entre `100` e `155` Mbps. O operador lógico `or` também pode ser utilizado com os operadores relacionais em elementos `value`. Por exemplo, se este operador fosse utilizado na expressão número 13 ao invés do operador `and`, o domínio dinâmico criado seria formado pelos dispositivos que possuíssem interfaces com velocidade maior ou igual a `100` Mbps e pelos dispositivos que possuíssem interfaces com velocidade menor ou igual a `155` Mbps.

As próximas expressões apresentadas demonstram o uso da cláusula `from`. Essa cláusula também permite que atributos (elemento `attribute`) de uma classe (elemento `class`) sejam utilizados para selecionar objetos de uma outra classe. Entretanto, a cláusula `from` é diferente do conector "`::`", pois a ordem de avaliação da expressão é alterada. Com

isso, pode-se controlar a escolha da classe que define os objetos de retorno de uma expressão de seleção. As expressões 14 e 15 são semelhantes, pois a ordem de avaliação nos dois casos será a seguinte: `topologies` e `devices`. Entretanto, na expressão 14 o resultado será um conjunto de topologias, enquanto na expressão 15 o resultado será um conjunto de dispositivos. Logo, a cláusula `from` foi utilizada na expressão 15 para deslocar a classe `topologies` do início da expressão para o seu final, permitindo que a classe `devices` passasse a ser a primeira classe da expressão, e por conseqüência, a classe que define os objetos de retorno. Ainda que a cláusula `from` tenha deslocado, na expressão 15, a classe `topologies`, a ordem de avaliação se mantém, isto é, a classe `topologies` é avaliada antes da classe `devices`. A parte da expressão de seleção localizada à direita da cláusula `from` deve possuir apenas uma classe e, pelo menos, um atributo dessa classe. Esta classe deve relacionar-se no modelo de informação com a primeira classe da parte da expressão de seleção à esquerda da cláusula `from`. Por exemplo, a classe `topologies` relaciona-se com a classe `devices` no modelo de informação da figura 3.2. A classe à direita da cláusula `from` deve estar localizada em um nível mais alto da hierarquia de classes (figura 3.4) do que a primeira classe da parte da expressão à esquerda desta cláusula. Por exemplo, a classe `topologies` está situada um nível acima do que a classe `devices` na hierarquia de classe apresenta na figura 3.4.

```
14) select topologies.owner["secAdm"]::devices.type["ISRouter"]
15) select devices.type["ISRouter"] from topologies.owner["secAdm"]
```

O domínio dinâmico criado pela expressão de seleção número 14 será formado pelas topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`) que tiverem dispositivos (objetos da classe `devices`) do tipo (atributo `type` da classe `devices`) roteador *IntServ* (`ISRouter`). Já o domínio dinâmico criado pela expressão de seleção número 15 será formado pelos dispositivos (objetos da classe `devices`) do tipo (atributo `type` da classe `devices`) roteador *IntServ* (`ISRouter`) pertencentes às topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`).

Expressões de seleção mais complexas podem ser elaboradas combinando-se os elementos "`.`", "`::`" e `from`, como demonstrado na expressão número 18. Nesta expressão, o domínio dinâmico criado será formado por dispositivos (objetos da classe `devices`) do tipo (atributo `type` da classe `devices`) `switch` que possuírem interfaces (objetos da classe `interfaces`) com velocidade (atributo `ifSpeed` da classe `interfaces`) de `100` Mbps e pertencerem a topologias (objetos da classe `topologies`) gerenciadas (atributo `owner` da classe `topologies`) pelo administrador de segurança (`secAdm`).

```
18) select devices.type["switch"]::interfaces.ifSpeed[100] from
                                        topologies.owner["secAdm"]
```

Na prática, os domínios dinâmicos criados através desta linguagem são formados por conjuntos de atributos que são chave-primária dos objetos selecionados de uma classe. Por exemplo, nas expressões de seleção números 1, 2, 3, 4, 5 e 6, os domínios dinâmicos criados serão formados pelo conjunto de valores do atributo `index` dos objetos selecionados da classe `topologies`. Valores de atributos que são chave-primária dos

objetos selecionados de uma classe são armazenados em uma área de memória temporária para que a linguagem de visualização de domínios dinâmicos possa usá-los. De posse desses valores, esta linguagem poderá apresentar graficamente os domínios dinâmicos criados e ainda possibilitar configurações visuais dos mesmos. Após o seu uso, os valores são descartados, e a área de memória temporária é esvaziada.

3.4 Linguagem de Visualização de Domínios Dinâmicos

O objetivo da linguagem apresentada nesta seção é possibilitar visualizações personalizadas dos domínios criados por meio da linguagem de criação de domínios dinâmicos. O processo de personalização da visualização desses domínios é baseado na configuração de atributos da facilidade visual utilizada. Uma facilidade visual é o recurso utilizado para gerar uma determinada apresentação gráfica. Tabelas, mapas topológicos, gráficos e matrizes são exemplos de tais facilidades. Espera-se que o ambiente gerenciado possua um conjunto de facilidades visuais disponíveis e que o administrador de rede precise apenas especificar qual dessas facilidades será utilizada para apresentar os domínios criados por ele.

A definição de uma nova visualização personalizada deve seguir os seguintes passos:

- Criação dos domínios dinâmicos desejados por meio da linguagem de criação de domínios dinâmicos;
- Escolha da facilidade visual responsável por apresentar os domínios dinâmicos criados;
- Escolha, dentre os domínios dinâmicos criados, daqueles que serão apresentados por meio da facilidade visual escolhida. Normalmente, apenas um domínio é apresentado através de uma facilidade visual. Entretanto, vários domínios podem ser apresentados ao mesmo tempo por meio da mesma facilidade;
- Configuração dos atributos da facilidade visual escolhida conforme os domínios dinâmicos criados.

Os últimos três passos apresentados são suportados pela linguagem de visualização de domínios dinâmicos. Essa linguagem é definida conforme a BNF apresentada na figura 3.5.

```
visualization ::= [selection] show dmview {and dmview}
selection     ::= using visualfacility [customization]
dmview        ::= domain [customization]
customization ::= with attribute=value {, attribute=value}
```

FIGURA 3.5 – BNF da linguagem de visualização de domínios dinâmicos

Nesta BNF, os elementos essenciais para a geração e personalização de visualizações de um ou mais domínios dinâmicos são os seguintes:

- `domain` – representa um domínio criado com a linguagem de criação de domínios dinâmicos;
- `visualfacility` – indica a facilidade visual a ser usada para apresentar os domínios dinâmicos;
- `attribute` – identifica um determinado atributo da facilidade visual escolhida;
- `value` – associa um valor específico ao atributo identificado pelo elemento `attribute`.

Primeiramente, antes de demonstrar o funcionamento da linguagem de visualização, alguns domínios de exemplo serão definidos através da linguagem de criação de domínios dinâmicos. Para que esses domínios possam ser manipulados pela linguagem de visualização, eles devem ser armazenados em variáveis. Os exemplos 1, 2 e 3 apresentam a definição desses domínios e a atribuição dos mesmos a variáveis.

```
1) DiffServRouters = select devices.type["DSRouter"]
2) IntServRouters  = select devices.type["ISRouter"]
3) RSVP33Routers   = select devices.type["Router"]::interfaces::
                                                    rsvp.index[33]
```

A expressão de seleção apresentada no exemplo número 1 criará um domínio dinâmico formado pelos dispositivos (objetos da classe `devices`) do ambiente gerenciado que forem do tipo (atributo `type` da classe `devices`) roteador *DiffServ* (`DSRouter`). Este domínio é atribuído à variável `DiffServRouters` para que o mesmo possa ser manipulado posteriormente pela linguagem de visualização.

A expressão de seleção apresentada no exemplo número 2, por sua vez, criará um domínio dinâmico formado pelos dispositivos (objetos da classe `devices`) do ambiente gerenciado que forem do tipo (atributo `type` da classe `devices`) roteador *IntServ* (`ISRouter`). Este domínio é atribuído à variável `IntServRouters` para que o mesmo também possa ser manipulado posteriormente pela linguagem de visualização.

Por último, a expressão de seleção apresentada no exemplo número 3 criará um domínio dinâmico formado pelos dispositivos (objetos da classe `devices`) do ambiente gerenciado que forem do tipo (atributo `type` da classe `devices`) roteador (`Router`) e possuírem interfaces (objetos da classe `interfaces`) com a sessão RSVP número 33 (atributo `index` da classe `rsvp`) associada. Este domínio é atribuído à variável `RSVP33Routers` para que o mesmo possa ser manipulado posteriormente pela linguagem de visualização.

Os exemplos apresentados nos números 4 e 5 demonstram a utilização das cláusulas `using` e `show` da linguagem de visualização de domínios dinâmicos. A cláusula `using` é utilizada para indicar a facilidade visual a ser utilizada (`table` e `topology`) para apresentar os domínios dinâmicos desejados. A cláusula `show` é usada para escolher os domínios dinâmicos (`DiffServRouters`, `IntServRouters` e `RSVP33Routers`) que serão

apresentados através da facilidade escolhida. Para demonstrar a linguagem de visualização de domínios dinâmicos através de exemplos, foi utilizado um ambiente com duas facilidades visuais: `table` e `topology`.

```
4) using table show DiffServRouters
5) using topology show IntServRouters and RSVP33Routers
```

A expressão apresentada no exemplo número 4 utiliza a cláusula `using` para escolher a facilidade visual `table` e a cláusula `show` para indicar que o domínio `DiffServRouters` deve ser apresentado através dessa facilidade. As visualizações geradas pela facilidade `table` são em forma de tabela, e as suas colunas são formadas pelos valores dos atributos dos objetos membros do(s) domínio(s) criado(s). A tabela demonstrada na figura 3.6 é um exemplo de visualização gerada pela facilidade `table` para apresentar os membros do domínio `DiffServRouters`. As colunas desta tabela são formadas pelos valores dos atributos (`index`, `type`, `ip`, `sysDescr`, `sysObjectID`, `sysUptime`, `sysContact`, `sysName`, `sysLocation`) dos objetos da classe `devices` (do modelo de informação) membros do domínio `DiffServRouters`.

index	type	ip	sysDescr	sysObjectID	sysUptime	sysContact	sysName	sysLocation
5	DSRouter	200.18.246.1	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (522451) 1:27:04.51	Leonardo Carvalho <suporte@cla.com.br>	2216-DSR	Sala de Operações 1
8	DSRouter	200.18.246.7	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (40437300) 4 days, 16:19:33.00	Leonardo Carvalho <suporte@cla.com.br>	2218-DSR	Sala de Operações 3
10	DSRouter	200.18.246.2	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (525610900) 60 days, 20:01:49.00	Leonardo Carvalho <suporte@cla.com.br>	2222-DSR	Sala de Operações 3
12	DSRouter	200.18.246.5	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (337306136) 39 days, 0:57:41.36	Leonardo Carvalho <suporte@cla.com.br>	2215-DSR	Sala de Operações 2

FIGURA 3.6 – Facilidade visual `table`

A expressão apresentada no exemplo número 5 utiliza a cláusula `using` para escolher a facilidade visual `topology` e a cláusula `show` para indicar que o domínio `IntServRouters` deve ser apresentado por meio dessa facilidade. A cláusula `and` da linguagem de visualização é usada para agregar novos domínios a uma visualização. Por exemplo, na expressão apresentada, a cláusula `and` indica que o domínio `RSVP33Routers` também deve ser apresentado através da facilidade `topology`. Os membros dos domínios criados com a linguagem de criação de domínio dinâmicos são apresentados pela facilidade `topology` através de mapas topológicos. O mapa topológico demonstrado na figura 3.7 é um exemplo de visualização gerada pela facilidade `topology` para apresentar os membros dos domínios `IntServRouters` e `RSVP33Routers`.

As facilidades visuais (por exemplo, `table` e `topology`) conseguem suportar a visualização simultânea de vários domínios desde que os mesmos sejam formados por objetos provenientes da mesma classe. Assim, para que diferentes domínios possam ser apresentados ao mesmo tempo em uma visualização, eles devem possuir elementos da mesma classe do modelo de informação utilizado. Por exemplo, diferentes domínios formados por dispositivos (objetos da classe `devices`) podem ser apresentados por meio da

facilidade `topology`. Entretanto, domínios distintos formados por dispositivos e por interfaces (objetos da classe `interfaces`) não podem ser apresentados por meio da facilidade `table`.

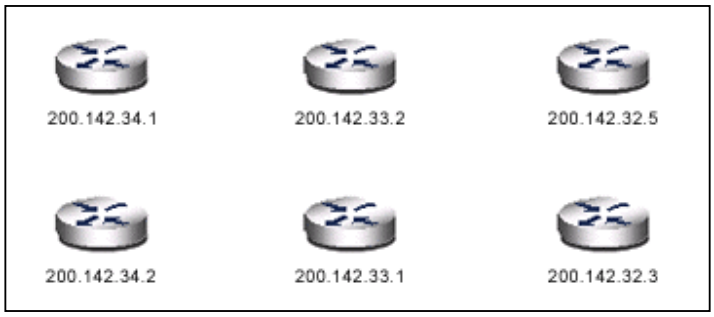

FIGURA 3.7 – Facilidade visual `topology`

As facilidades visuais possuem atributos próprios que podem ser configurados para modificar determinadas características gráficas das visualizações que serão apresentadas. A habilidade para configurar esses atributos é a principal característica da linguagem de visualização de domínios dinâmicos. A cláusula `with` desta linguagem é o elemento responsável por possibilitar a personalização das visualizações geradas por uma facilidade visual. Os atributos configurados pela cláusula `with` podem ser aplicados a todos os domínios de uma visualização ou a cada domínio em separado. Os exemplos apresentados nos números 6, 7 e 8 demonstram o uso da cláusula `with`.

```
6) using table with cellcolor=blue show IntServRouters and
                                                  RSVP33Routers

7) using table show IntServRouters with cellcolor=yellow and
                            RSVP33Routers with cellcolor=green

8) using table with cellcolor=blue show IntServRouters and
                       RSVP33Routers  with cellcolor=red and
                       DiffServRouters with cellcolor=yellow
```

Em uma expressão, a cláusula `with` pode estar situada após a facilidade visual escolhida, após domínios específicos ou em ambos. O primeiro caso é utilizado para configurar atributos visuais de uma facilidade que serão aplicados à visualização de todos os domínios apresentados. Por exemplo, na expressão número 6, a cláusula `with` é utilizada para configurar um atributo da facilidade `table` que será aplicado à visualização dos domínios `IntServRouters` e `RSVP33Routers`. O elemento `attribute`, da BNF da linguagem de visualização indica os atributos a serem configurados, e o elemento `value` atribui valores a esses atributos. No exemplo número 6, os elementos `cellcolor` e `blue` modificarão a cor padrão (cinza) das células da tabela gerada para apresentar os domínios `IntServRouters` e `RSVP33Routers` para a cor azul. Vários atributos podem ser utilizados após uma cláusula `with`. Entretanto, os conjuntos atributo e valor devem ser separados por vírgula (por exemplo, "`cellcolor=blue`, `cellborder=10`"). É importante mencionar que os atributos de uma facilidade visual assumem valores padrão quando não forem configurados.

A cláusula `with`, quando situada após um domínio específico, é utilizada para configurar atributos que sejam válidos somente para tal domínio. Por exemplo, na expressão número 7, para cada domínio (`IntServRouters` e `RSVP33Routers`) há um atributo e um valor associado. O primeiro atributo utilizado (`cellcolor`) configura a facilidade `table` para apresentar as informações dos membros do domínio `IntServRouters` em células de cor amarela (`yellow`). O segundo atributo utilizado (`cellcolor`), por sua vez, configura a facilidade `table` para apresentar as informações dos membros do domínio `RSVP33Routers` em células de cor verde (`green`). A figura 3.8 demonstra como é a visualização gerada pela facilidade `table` quando a expressão número 7 é executada. Na tabela apresentada nesta figura, as células de cor cinza-escuro equivalem a células de cor amarela e as células de cor cinza-claro equivalem a células de cor verde.

index	type	Ip	sysDescr	sysObjectID	sysUptime	sysContact	sysName	sysLocation
2	ISRouter	200.142.34.1	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (442050900) 51 days, 03:55:09.00	Marcelo Oliveira <suporte@ws.com.br>	3242-ISR	Setor A
7	ISRouter	200.142.34.2	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (564981120) 65 days, 09:23:31.20	Marcelo Oliveira <suporte@ws.com.br>	3254-ISR	Setor A
8	ISRouter	200.142.33.2	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (525610900) 60 days, 20:01:49.00	Marcelo Oliveira <suporte@ws.com.br>	2993-ISR	Setor D
13	ISRouter	200.142.33.1	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (337306136) 39 days, 0:57:41.36	Marcelo Oliveira <suporte@ws.com.br>	3284-ISR	Setor D
15	Router	200.142.32.5	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (62742933) 7 days, 06:17:09.33	Marcelo Oliveira <suporte@ws.com.br>	1343-R	Setor C
22	Router	200.142.32.3	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	OID: enterprises.2.6.131	Timeticks: (370963600) 42 days, 22:27:16.00	Marcelo Oliveira <suporte@ws.com.br>	2122-R	Setor C

FIGURA 3.8 – Uso da cláusula `with` em domínios específicos

A cláusula `with` pode ser utilizada após a facilidade visual escolhida e após domínios específicos em uma mesma expressão. Por exemplo, na expressão número 8, a cláusula `with` é utilizada inicialmente após a facilidade `table` e, então, após os domínios `RSVP33Routers` e `DiffServRouters`. A primeira cláusula `with` utilizada configura a facilidade `table` para apresentar as informações dos membros dos domínios `IntServRouters`, `RSVP33Routers` e `DiffServRouters` em células (`cellcolor`) azuis (`blue`). Entretanto, somente os membros do domínio `IntServRouters` serão apresentados em células desta cor, pois os domínios `RSVP33Routers` e `DiffServRouters` possuem atributos específicos associados que sobrepõem a configuração do primeiro `with` da expressão. Assim, os membros do domínio `RSVP33Routers` serão apresentados em células de cor vermelha (`red`) e os membros do domínio `DiffServRouters` serão apresentados em células de cor amarela (`yellow`).

As informações apresentadas em uma visualização também podem ser controladas por meio da configuração de determinados atributos das facilidades visuais. Por exemplo, as colunas das tabelas geradas pela facilidade `table` podem ser determinadas pela configuração do atributo `columns`. A expressão apresentada no exemplo número 9 demonstra a configuração desse atributo. Os nomes das colunas escolhidas para serem

apresentadas devem ser separados por vírgula e estarem entre chaves (“{}”). A visualização correspondente à expressão número 9 é apresentada na figura 3.9.

```
9)  using  table  with  columns={index,  type,  ip,  sysDescr,
                              sysLocation} show IntServRouters

10) using topology show IntServRouters with alpha=30 and
                                                  RSVP33Routers

11) using topology with show IntServRouters with color=blue and
                             RSVP33Routers with color=red
```

index	type	ip	sysDescr	sysLocation
2	ISRouter	200.142.34.1	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	Setor A
7	ISRouter	200.142.34.2	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	Setor A
8	ISRouter	200.142.33.2	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	Setor D
13	ISRouter	200.142.33.1	CISCO ROUTER ABC, MODEL XYZ VER. 11.0	Setor D

FIGURA 3.9 – Uso do atributo `columns` da facilidade `table`

As expressões apresentadas nos exemplos números 10 e 11 demonstram a configuração de alguns atributos da facilidade `topology`. No exemplo número 10, a facilidade `topology` foi configurada para apresentar o domínio `RSVP33Routers` na forma padrão, pois nenhum atributo foi associado à facilidade visual ou ao domínio em questão. Entretanto, essa facilidade foi configurada para apresentar o domínio `IntServRouters` com apenas 30% da sua transparência (`alpha`) normal. Como resultado, os membros deste domínio serão apresentados em tons de cores mais fracos que os membros dos domínios `RSVP33Routers`. A visualização correspondente ao exemplo número 10 é apresentada na figura 3.10.

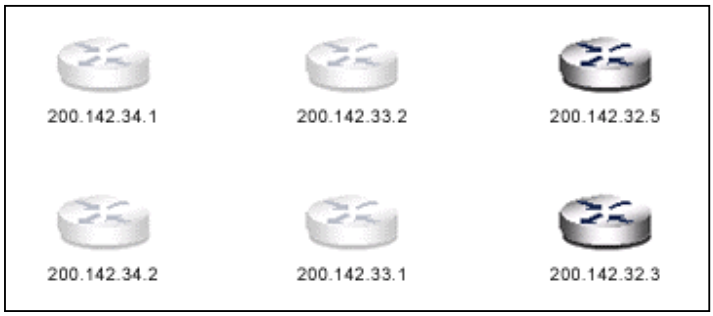

FIGURA 3.10 - Uso do atributo `alpha` da facilidade `topology`

No exemplo número 11, a facilidade `topology` foi configurada para apresentar os membros dos domínios `IntServRouters` e `RSVP33Routers` em diferentes cores. Os membros do domínio `IntServRouters` serão apresentados na cor (`color`) azul (`blue`) e os membros do domínio `RSVP33Routers` serão apresentados na cor (`color`) vermelha (`red`).

As facilidades visuais disponíveis em um ambiente gerenciado possuem vários atributos (por exemplo, o atributo `cellcolor` da facilidade `table`). Cada um desses atributos possui um valor padrão associado que será utilizado nas visualizações se outros valores não forem atribuídos por meio da linguagem de visualização. De maneira similar, cada classe do modelo de informação utilizado possui uma facilidade visual padrão. Como conseqüência, a cláusula `using`, responsável por indicar a facilidade visual utilizada para apresentar determinados domínios, pode ser omitida da expressão de visualização. Por exemplo, assumindo que `topology` é a facilidade visual padrão da classe `devices`, a expressão demonstrada no exemplo número 12 apresentará os membros dos domínios `IntServRouters` e `RSVP33Routers` por meio de um mapa topológico.

```
12) show IntServRouters and RSVP33Routers
```

A configuração de atributos poderá continuar a ser realizada por meio da cláusula `with` nestes casos. No entanto, essa cláusula deverá ser associada a domínios específicos (por exemplo, `IntServRouters` e `RSVP33Routers`). O próximo capítulo apresentará a forma como protótipo desenvolvido foi implementado para suportar as linguagens de criação e visualização de domínios dinâmicos abordadas neste capítulo.

4 Implementação do Protótipo

O capítulo anterior apresentou as linguagens de criação e visualização de domínios dinâmicos definidas. Este capítulo apresenta a segunda contribuição desta dissertação: o desenvolvimento de um sistema capaz de suportar as linguagens demonstradas no capítulo 3.

Inicialmente são apresentadas algumas considerações gerais sobre o protótipo desenvolvido, como por exemplo, a sua interface e as tecnologias utilizadas. A seguir, o suporte à linguagem de criação de domínios dinâmicos é abordado. Por fim, o capítulo encerra demonstrando como a linguagem de visualização de domínios dinâmicos é suportada pelo protótipo.

4.1 Considerações Gerais

O suporte às duas linguagens demonstradas no capítulo 3 foi implementado através de um sistema baseado na Web apresentado na figura 4.1. A interface deste sistema é formada por caixas de texto e botões que possibilitam a elaboração e a execução de expressões das linguagens de criação e visualização de domínios dinâmicos. O número de expressões usadas para criar os domínios desejados é variável e pode ser configurado a partir da própria interface do sistema (figura 4.1 – letras A). O número de expressões utilizadas para personalizar a visualização dos domínios criados, por sua vez, é fixo. A partir de uma única expressão de visualização, todos os domínios criados podem ser configurados com as características visuais desejadas. Por exemplo, na figura 4.1, é possível observar que existem caixas de texto (letras B_1, B_2, C_1 e C_2) para a elaboração de mais de uma expressão de criação de domínios e apenas uma caixa de texto (letra D) para a formulação da expressão de visualização dos domínios desejados. A avaliação e o processamento das expressões inicia após o botão *Query* (letra E) ser pressionado. Maiores detalhes sobre o funcionamento do protótipo serão apresentados na seção 4.4 deste capítulo.

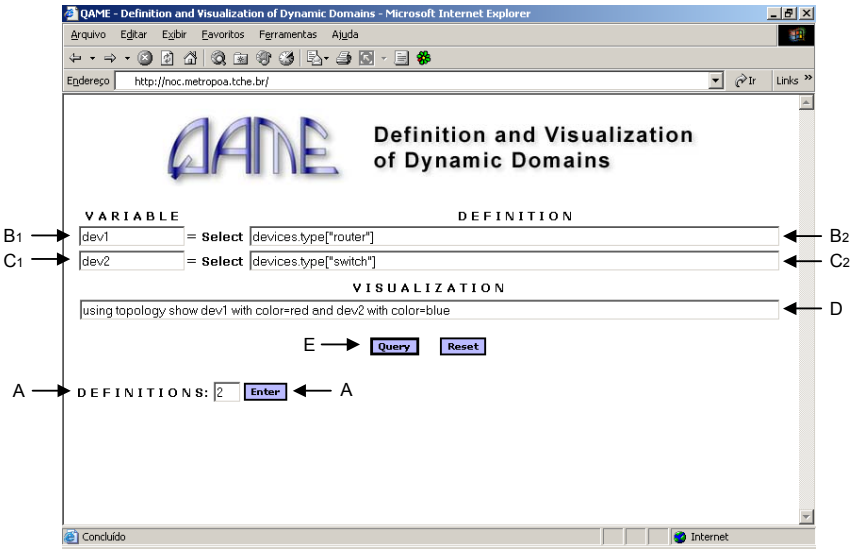

FIGURA 4.1 – Protótipo desenvolvido para suportar as linguagens criadas

O protótipo foi desenvolvido sobre o sistema operacional Linux [WEL 95] e servidor Web Apache [APA 2001]. As interações entre o usuário e o protótipo são realizadas por meio de requisições e respostas do protocolo HTTP (*Hypertext Transfer Protocol*) [FIE 97]. A implementação do protótipo englobou principalmente a utilização das seguintes tecnologias:

- PRECCX [BRE 2002] – Programa utilizado para gerar os analisadores sintáticos das linguagens de criação e visualização de domínios dinâmicos;
- PHP4 [PHP 2001] – Linguagem de *script* utilizada para analisar e processar as expressões das linguagens de criação e visualização de domínios dinâmicos;
- MySQL [YAR 99] – Banco de dados utilizado no armazenamento de informações do ambiente gerenciado, do modelo de informação deste ambiente e de informações relacionadas às linguagens de criação e visualização de domínios dinâmicos;
- NET-SNMP [UNI 2001] - Pacote de *software* que implementa o suporte ao protocolo SNMP integrado à linguagem PHP4.

Além dessas tecnologias, foram utilizados também JavaScript [WIN 2000] e Flash [MAC 2001] de forma complementar.

4.2 Suporte à Criação de Domínios Dinâmicos

As expressões de criação (uma ou mais expressões) e visualização (apenas uma expressão) de domínios dinâmicos são enviadas a um *script* PHP4 principal após o botão *Query* da interface do sistema desenvolvido (figura 4.1 – letra E) ser pressionado. Este *script* é responsável por analisar e executar as expressões recebidas. Todo esse processo ocorre de acordo com os passos apresentados na figura 4.2.

Expressões de criação (por exemplo, as expressões números 1 e 2 da figura 4.2) e visualização (por exemplo, a expressão número 3 da figura 4.2) de domínios dinâmicos são enviadas, através de uma única requisição HTTP, a um *script* PHP4 principal armazenado no servidor. As expressões de criação são analisadas e avaliadas antes da expressão de visualização, pois os membros dos domínios a serem criados precisam ser primeiramente identificados para somente então serem visualizados. As ações correspondentes às expressões de criação são apresentadas na parte superior da figura 4.2, e as ações referentes à expressão de visualização são apresentadas na parte inferior da mesma figura.

Para cada expressão de criação recebida pelo *script* PHP4 principal, os seguintes passos são executados:

1. Análise sintática da expressão;
2. Análise semântica da expressão;
3. Execução da expressão (busca dos valores que satisfazem a expressão).

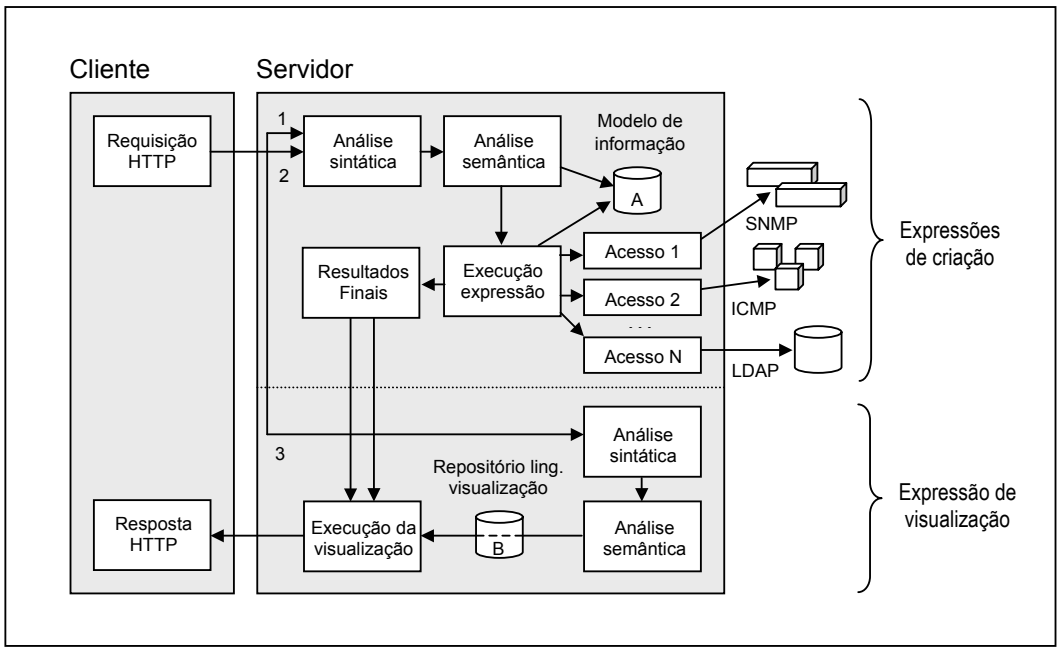

FIGURA 4.2 – Avaliação das expressões de criação e visualização de domínios dinâmicos

Análise Sintática

O primeiro passo executado é a análise sintática. Essa análise é realizada através de um programa escrito na linguagem C gerado automaticamente pelo *software* PRECCX a partir da BNF da linguagem de criação de domínios dinâmicos. O *script* PHP4 principal inicia este programa passando uma expressão de criação. O programa em C, após ser executado, gera um resultado como saída que é capturado pelo *script* PHP4 principal. Esse resultado é analisado e, não ocorrendo erros sintáticos, a expressão passada é avaliada semanticamente. Entretanto, se erros existirem, a execução da expressão é cancelada e outra expressão passa a ser avaliada. Os erros identificados são armazenados para serem informados ao usuário do sistema.

A BNF completa utilizada para gerar o analisador sintático da linguagem de criação de domínios dinâmicos por meio do *software* PRECCX é apresentada no anexo 1.

Análise Semântica

A análise semântica de uma expressão de criação é realizada a partir da verificação da existência, no modelo de informação do ambiente gerenciado, das classes e dos atributos usados na expressão avaliada. Por exemplo, de acordo com o modelo de informação da figura 3.2, a expressão de criação apresentada no exemplo número 1 está correta semanticamente, pois a classe `interfaces` existe neste modelo, e os atributos `ifType` e `ifSpeed` existem nesta classe. Entretanto, a expressão apresentada no exemplo número 2 possui dois erros semânticos, pois a classe `topolog` não existe no modelo citado, e o atributo `tipes` não existe na classe `devices` deste modelo. O atributo `owner` da expressão apresentada no número 2 não chega a ser avaliado, porque a classe `topolog` não existe no modelo de informação utilizado.

```
1) select interfaces.ifType["ATM"].ifSpeed[>100]
2) select topolog.owner["secAdm"]::devices.tipes["DSRouter"]
```

O modelo de informação do ambiente gerenciado deve estar armazenado em memória secundária para que a existência das classes e dos atributos de uma expressão de criação possa ser verificada em tempo de execução. Como resultado, o modelo de informação utilizado no desenvolvimento deste trabalho (figura 3.2) foi mapeado para uma base de dados MySQL. As classes deste modelo foram armazenadas na tabela *classes*, e os atributos destas classes foram armazenados na tabela *attributes* desta base de dados. As tabelas *classes* e *attributes* são apresentadas na figura 4.3.

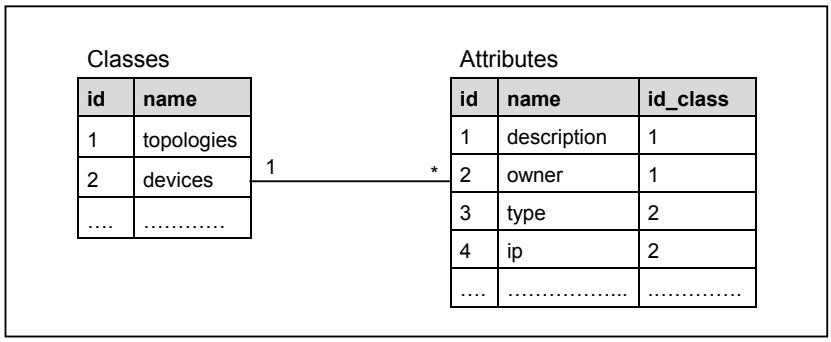

FIGURA 4.3 – Modelo de informação na base MySQL

A existência das classes do modelo de informação utilizado é verificada a partir de consultas ao campo *name* da tabela *classes*. A existência dos atributos de uma classe deste modelo, por sua vez, é verificada por meio de consultas aos campos *name* e *id_class* da tabela *attributes*. O campo *id_class* é responsável por relacionar os atributos (identificados pelo campo *name*) da tabela *attributes* com as classes da tabela *classes*. Na figura 4.3, por exemplo, a partir deste campo é possível verificar que os atributos `description` e `owner` da tabela *attributes* pertencem à classe `topologies` da tabela *classes*, e que os atributos `type` e `ip` da tabela *attributes* pertencem à classe `devices` da tabela *classes*. Nas tabelas *classes* e *attributes* demonstradas na figura 4.3, foram apresentados somente os campos úteis a avaliação semântica de uma expressão de criação.

O *script* PHP4 principal inicia o processo de avaliação semântica de uma expressão de criação recebida através da invocação de funções armazenadas em um outro *script*. Inicialmente, a expressão de criação é segmentada e seus elementos (classes, atributos e valores dos atributos) são armazenados em uma matriz para que sejam manipulados com maior facilidade. Em seguida, a existência das classes e dos atributos armazenados nesta matriz é verificada a partir de consultas às tabelas *classes* e *attributes* da base MySQL. Os valores dos atributos de uma expressão de criação também são armazenados na matriz de elementos para que ela possa ser utilizada no processo de obtenção dos valores que satisfaçam a expressão recebida (próxima subseção).

As expressões de criação são segmentadas, para que suas classes, atributos e valores destes atributos possam ser extraídos e armazenados na matriz de elementos. A segmentação de uma expressão de criação é efetuada a partir da cláusula `from`, do conector

"::", do ponto (".") e dos colchetes ("[]"). A forma como as expressões de criação recebidas pelo *script* PHP4 principal são segmentadas é demonstrada a seguir por meio de alguns exemplos.

```
3) select devices.type["DSRouter"]::interfaces.ifType["ATM"]
                .ifSpeed[100] from topologies.location["UFRGS-II"]
4) select topologies::devices.type["DSRouter"]::interfaces
                                                    .ifType["ATM"]
5) select devices.type["DSRouter"]
6) select devices
```

A expressão de criação apresentada no exemplo número 3 inicialmente é segmentada a partir da cláusula `from`. A seguir, a parte da expressão situada à esquerda da cláusula `from` (`devices.type["DSRouter"]::interfaces.ifType["ATM"].ifSpeed[100]`), resultante da primeira divisão, é segmentada através do conector "::". As três partes da expressão resultantes dessas segmentações (`topologies.location["UFRGS-II"]`, `devices.type["DSRouter"]` e `interfaces.ifType["ATM"].ifSpeed[100]`) são, então, secionadas a partir dos pontos e dos colchetes.

A expressão de criação apresentada no exemplo número 4 primeiramente é segmentada a partir dos conectores "::", pois a cláusula `from` não foi utilizada na formulação dessa expressão. Em seguida, as partes da expressão resultantes desta divisão (`topologies`, `devices.type["DSRouter"]` e `interfaces.ifType["ATM"]`) são, então, segmentadas a partir dos pontos e dos colchetes.

A expressão de criação apresentada no exemplo número 5 é segmentada somente a partir do ponto e dos colchetes, pois a cláusula `from` e o conector "::" não foram utilizados na formulação desta expressão. A expressão de criação apresentada no exemplo número 6, por sua vez, não precisa ser segmentada, pois a cláusula `from`, o conector "::", o ponto e os colchetes não foram utilizados na formulação dessa expressão.

Classes, atributos e valores de atributos são armazenados na matriz de elementos após serem extraídos de uma expressão de criação. A matriz resultante da segmentação da expressão apresentada no exemplo número 3 é demonstrada na figura 4.5. As linhas da matriz de elementos que armazenam classes, recebem o rótulo *class*, e as linhas que armazenam atributos, recebem o rótulo *attrib*. Os atributos são armazenados nas linhas da matriz situadas após as linhas das classes a que pertencem. Os valores dos atributos, por sua vez, são armazenados nas colunas situadas após as colunas dos atributos a que são associados. Os elementos (classes, atributos e valores dos atributos) do segmento de uma expressão de criação localizado à direita da cláusula `from` (por exemplo, na expressão número 3, o segmento `topologies.location ["UFRGS-II"]`) possuem prioridade de armazenamento em relação aos elementos de outros segmentos da expressão. Os elementos dos primeiros segmentos de uma expressão de criação divididos pelo conector "::" (por exemplo, na expressão número 3, o segmento `devices.type["DSRouter"]`) possuem prioridade de armazenamento em relação aos elementos dos últimos segmentos divididos por este conector (por exemplo, na expressão número 3, o segmento `interfaces.ifType["ATM"].ifSpeed[100]`). A ordem como as classes, atributos e valores da expressão número 3 foram armazenados na matriz de elementos pode ser

verificada percorrendo-se em profundidade, da esquerda para direita, a árvore apresentada na figura 4.4. Os números 1, 2 e 3 das figuras 4.4 e 4.5 são correspondentes.

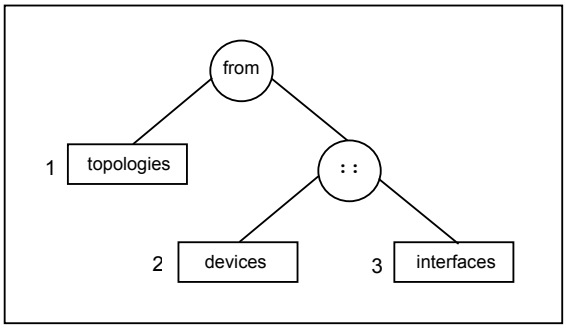

FIGURA 4.4 – Ordem de armazenamento dos elementos da matriz da expressão número 3

A existência das classes e dos atributos armazenados na matriz da expressão avaliada começa a ser verificada assim que a matriz for gerada. As linhas e colunas da matriz são percorridas, e a existência das classes e dos atributos é verificada a partir de consultas SQL às tabelas *classes* e *attributes* da base MySQL (figura 4.2 – letra A). A não-existência de uma classe ou atributo de uma classe da matriz na base MySQL gera um erro semântico. Os erros semânticos ocorridos são armazenados para serem apresentados ao usuário do sistema. A busca pelos valores que satisfaçam a expressão de criação inicia senão ocorrerem erros semânticos. Entretanto, se os erros existirem, o processamento da expressão é cancelada e outra expressão passa a ser avaliada.

Os erros ocorridos nas análises sintática e semântica das expressões de criação não encerram a execução geral do sistema, pois ao final deste processo um histórico de erros é gerado para ser apresentado ao usuário.

	Rótulo	Classe / atrib.	Valor atrib.
1	class	topologies	
	attrib	location	“UFRGS-II”
2	class	devices	
	attrib	type	“DSRouter”
3	class	interfaces	
	attrib	IfType	“ATM”
	attrib	ifSpeed	100

FIGURA 4.5 – Matriz resultante da segmentação da expressão apresentada no exemplo 3

O modelo de informação utilizado foi mapeado em um banco de dados devido à facilidade de realizar consultas e armazenar novas informações. É importante salientar que, se o modelo de informação for expandido ou reduzido, basta acessar as tabelas *classes* e *attributes* e acrescentar ou retirar as informações necessárias. A análise semântica de uma expressão de criação irá adaptar-se automaticamente às atualizações na base de dados, pois a existência das classes e dos atributos armazenados na matriz é verificada de acordo com as tabelas *classes* e *attributes* desta base.

Controle de Execução das Expressões de Criação de Domínios Dinâmicos

O processo de execução de uma expressão de criação é baseado em consultas aos objetos gerenciáveis do ambiente de rede que estão associados aos atributos do modelo de informação utilizados na expressão. Por exemplo, ao processar o atributo `ifSpeed` da classe `interfaces` de uma expressão de criação qualquer, consultas ao objeto *ifSpeed* da MIB *Interfaces* de determinados dispositivos do ambiente de rede são realizadas. O acesso aos objetos gerenciáveis associados aos atributos é realizado por funções especiais que abstraem os detalhes do protocolo de comunicação desses objetos. As classes, atributos e valores dos atributos de uma expressão de criação são processados a partir da matriz de elementos gerada na análise semântica desta expressão.

```
7) select topologies::devices.type[“ISRouter”]
```

As expressões de criação são executadas a partir de segmentos formados por uma classe, zero ou mais atributos e os valores desses atributos. Esses segmentos podem ser observados secionando a expressão de criação por meio da cláusula `from` e/ou do conector “`::`”. Por exemplo, a expressão de criação apresentada no número 7 é formada por dois segmentos: `topologies` e `devices.type[“ISRouter”]`. Esses segmentos também podem ser identificados por meio da matriz de elementos desta expressão que é apresentada na figura 4.6.

Os segmentos de uma expressão de criação são processados em forma seqüencial a partir do primeiro segmento da matriz de elementos dessa expressão. O resultado da execução de cada segmento é formado pelos valores do atributo chave-primária dos objetos selecionados de uma classe. Por exemplo, o resultado obtido no segmento `topologies` da expressão apresentada no número 7 é formado pelos valores do atributo `index` (atributo chave-primária da classe `topologies` no modelo de informação) de todas as ocorrências dos objetos da classe `topologies`. O resultado obtido no segmento `devices.type[“ISRouter”]` da mesma expressão, por sua vez, é formado pelos valores do atributo `index` (atributo chave-primária da classe `devices`) das ocorrências dos objetos da classe `devices` que possuem o atributo `type` igual a `ISRouter`.

Os valores resultantes da execução de cada segmento de uma expressão de criação são armazenados em vetores. Esses vetores recebem como rótulo o nome das classes armazenadas nos segmentos. Por exemplo, os valores da execução do segmento `topologies` da expressão apresentada no número 7 são armazenados no vetor *topologies*, e os valores da execução do segmento `devices.type[“ISRouter”]` da mesma expressão são armazenados no vetor *devices*. Os valores armazenados em um vetor correspondente a um determinado segmento podem ser utilizados no processamento de outros segmentos da mesma expressão de criação.

A execução dos segmentos de uma expressão de criação recebida pelo *script* PHP4 principal é coordenada por funções específicas invocadas a partir de um outro *script*. Essas funções são responsáveis por processar os segmentos das expressões de criação, manipular os vetores de valores desses segmentos, controlar a utilização desses vetores e obter o valor final das expressões de criação executadas. Além disso, essas funções também são responsáveis por manipular vetores intermediários que, em algumas situações, são utilizados para atualizar os vetores dos segmentos das expressões de criação. A seguir, o

processo de execução da expressão apresentada no exemplo número 7 é comentado e ilustrado na figura 4.6 para demonstrar como os segmentos de uma expressão de criação são processados.

Inicialmente, a matriz de elementos da expressão número 7 (figura 4.6) é acessada, e o nome das classes armazenadas em cada segmento é verificado. Após, dois vetores são criados sem valores e recebem como rótulo o nome destas classes: *topologies* e *devices*. Na base de dados do modelo de informação existem informações complementares responsáveis por indicar se os valores dos vetores criados podem ser utilizados no processamento dos segmentos de uma expressão avaliada. Essas informações são acessadas para que a possibilidade do uso dos valores dos vetores *topologies* e *devices* no processamento do primeiro segmento da matriz de elementos da expressão de criação seja avaliada (figura 4.2 - A). É constatado que os valores do vetor *topologies* devem ser utilizados no processamento do segmento `topologies`. Entretanto, como esse vetor ainda está vazio, seus valores não são acessados, e o segmento `topologies` da matriz de elementos é então executado. Os valores resultantes da execução deste segmento são armazenados no vetor *topologies*. (figura 4.6 – seta número 1). A seguir, a base de dados é acessada novamente para que a possibilidade do uso dos valores dos vetores *topologies* e *devices* no processamento do segundo segmento da matriz de elementos seja avaliada. É constatado, então, que os valores do vetor *topologies* e *devices* devem ser utilizados no processamento deste segmento. Entretanto, apenas os valores do vetor *topologies* são utilizados na execução do segmento `devices.type["ISRouter"]` (seta número 2), pois o vetor *devices* ainda não possui valores. A utilização dos valores do vetor *topologies* no processamento do segmento `devices.type["ISRouter"]` é necessária para que a busca aos dispositivos do tipo roteador *IntServ* possa ser realizada sobre as topologias obtidas na execução do segmento `topologies`. Os valores resultantes da execução do segmento `devices.type["ISRouter"]` são armazenados no vetor *devices* e no vetor intermediário *topologies_aux* (seta número 3). O vetor *devices* é responsável por armazenar os valores dos dispositivos do tipo roteador *IntServ* obtidos nas topologias resultantes da execução do segmento `topologies`. O vetor intermediário *topologies_aux*, por sua vez, é responsável por armazenar os valores das topologias dos dispositivos obtidos na execução do segmento `devices.type["ISRouter"]`(setas número 4). Por fim, os valores do vetor intermediário *topologies_aux* são utilizados para atualizar o vetor *topologies* (setas número 5).

O resultado final de uma expressão de criação executada é obtido por meio do acesso aos valores de um ou mais vetores dos segmentos dessa expressão. Esses vetores são identificados a partir da verificação da chave-primária da primeira classe da expressão de criação em questão. Caso esta chave-primária seja formada por atributos da própria classe, o resultado final da expressão é dado pelo vetor de valores correspondente ao segmento desta classe. Caso esta chave-primária seja formada por atributos da própria classe e de uma outra classe, o resultado final da expressão é obtido através do acesso aos vetores correspondentes aos segmentos dessas classes. Por exemplo, o resultado final da expressão apresentada no exemplo número 7 (`topologies::devices.type["ISRouter"])` é obtido por meio do acesso ao vetor *topologies* (figura 4.6 – letra A), pois a chave-primária da classe *topologies* (primeira classe da expressão) é formada pelo atributo `index` da própria classe. É importante mencionar que a primeira classe de uma expressão de criação é identificada a partir da expressão original e não da sua matriz de elementos, pois, em alguns

casos, a primeira classe da matriz de elementos não corresponde à primeira classe da expressão original. Exemplos disso são as expressões que contêm a cláusula `from`.

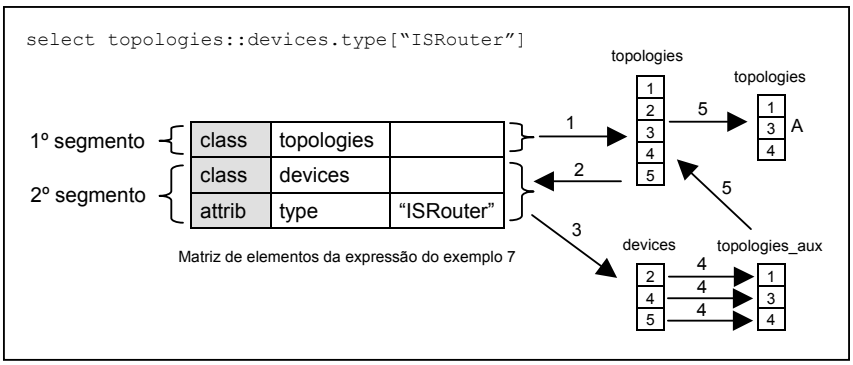

FIGURA 4.6 – Execução da expressão apresentada no exemplo número 7

Os valores resultantes da execução dos segmentos de uma expressão de criação são obtidos através do processamento seqüencial dos atributos desses segmentos. Por exemplo, os valores do vetor *topologies* resultantes da execução do primeiro segmento da expressão apresentada no exemplo número 7 (seta número 1 – figura 4.6) foram obtidos através do processamento do atributo chave-primária (atributo `index`) da classe `topologies`. O atributo chave-primária de uma classe é executado sempre que essa classe não possuir atributos. Já os valores dos vetores *devices* e *topologies_aux* resultantes da execução do segundo segmento da mesma expressão (seta número 3 – figura 4.6) foram obtidos por meio do processamento do atributo `type` da classe `devices`. A seguir, o processo de execução da expressão apresentada no exemplo número 8 é comentado e ilustrado na figura 4.7 para demonstrar em detalhes como os atributos dos segmentos de uma expressão de criação são processados.

```
8) select interfaces.ifType["ATM"].ifSpeed[>100] from
                        devices.ip["200.132.73.33 or 200.132.73.34"]
```

Inicialmente, a matriz de elementos da expressão de criação apresentada no exemplo número 8 (figura 4.7) é acessada, e o nome das classes armazenadas em cada segmento é verificado. Após, dois vetores são criados sem valores e recebem como rótulo o nome dessas classes: *devices* e *interfaces*. A base de dados é, então, acessada para que a possibilidade do uso dos valores desses vetores no processamento do atributo `ip` do primeiro segmento da matriz de elementos seja avaliada. É constatado que os valores do vetor *devices* devem ser utilizados no processamento desse atributo. Entretanto, como esse vetor ainda está vazio, seus valores não são acessados, e o atributo `ip` do primeiro segmento da matriz é então executado. Os valores resultantes da execução deste atributo são armazenados no vetor *devices* (figura 4.7 – seta número 1). A seguir, a base de dados é acessada novamente para que a possibilidade do uso dos valores dos vetores *devices* e *interfaces* no processamento do atributo `ifType` do segundo segmento da matriz seja avaliada. É constatado, então, que os valores dos vetores *devices* e *interfaces* devem ser utilizados no processamento deste atributo. Entretanto, apenas os valores do vetor *devices* são utilizados na execução do atributo `ifType` (seta número 2), pois o vetor *interfaces* ainda não possui valores. O uso dos valores do vetor *devices* no processamento do atributo

`ifType` é necessário para que a busca às interfaces do tipo ATM possa ser realizada sobre os dispositivos obtidos na execução do primeiro segmento da matriz. Os valores resultantes da execução do atributo `ifType` são armazenados no vetor *interfaces* e no vetor intermediário *devices_aux* (seta número 3). O vetor *interfaces* é responsável por armazenar os valores das interfaces do tipo ATM obtidas nos dispositivos resultantes da execução do primeiro segmento da matriz de elementos. O vetor intermediário *devices_aux*, por sua vez, é responsável por armazenar os valores dos dispositivos em que cada interface do tipo ATM foi encontrada (setas número 4). Os valores do vetor intermediário *devices_aux* são, então, utilizados para atualizar o vetor *devices* (setas número 5). Em seguida, a base de dados é acessada outra vez para que a possibilidade do uso dos valores dos vetores *devices* e *interfaces* no processamento do atributo `ifSpeed` do segundo segmento da matriz seja avaliada. É constatado que os valores dos vetores *devices* e *interfaces* devem ser utilizados no processamento deste atributo. Esses valores são, então, acessados, e o atributo `ifSpeed` é executado (setas número 6). A utilização dos valores destes vetores no processamento do atributo `ifSpeed` é necessária para que a busca às interfaces com velocidade superior a 100 Mbits possa ser realizada sobre as interfaces obtidas na execução do atributo `ifType`. As posições dos vetores *devices* e *interfaces* são correspondentes, assim é possível identificar o dispositivo em que uma determinada interface está localizada. Os valores resultantes da execução do atributo `ifSpeed` são usados para atualizar o vetor *interfaces* e o vetor intermediário *devices_aux* (seta número 7). O vetor *interfaces* é responsável por armazenar os valores das interfaces ATM com velocidades acima de 100 Mbits obtidas a partir dos dispositivos resultantes da execução do primeiro segmento da matriz de elementos. O vetor intermediário *devices_aux*, por sua vez, é responsável por armazenar os valores dos dispositivos dessas interfaces (setas número 8). Por fim, os valores do vetor intermediário *devices_aux* são utilizados para atualizar o vetor *devices* (setas número 9).

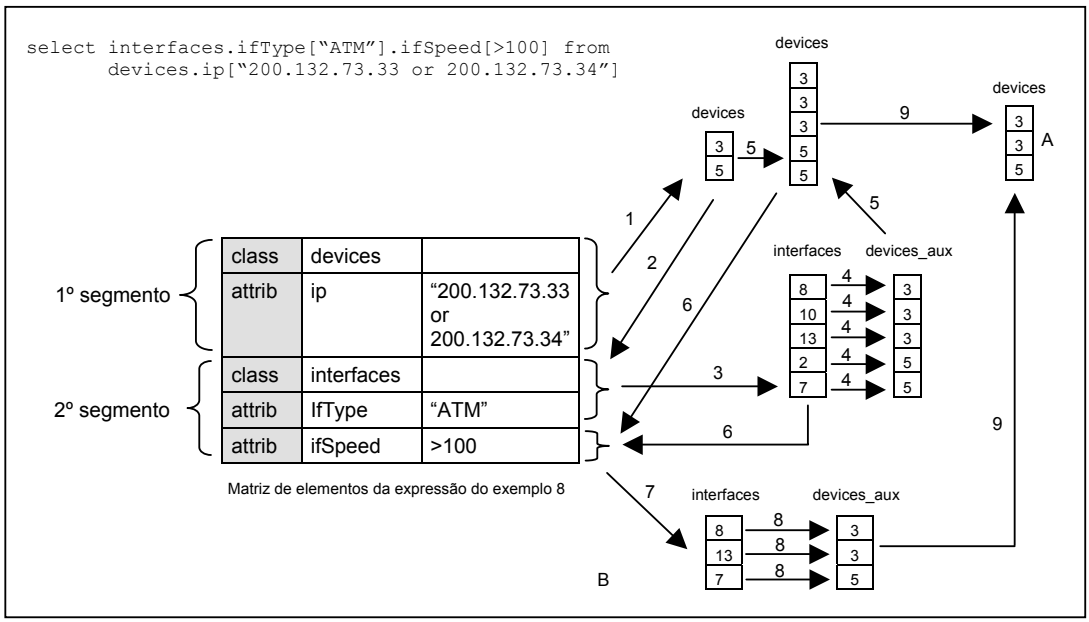

FIGURA 4.7 – Execução da expressão apresentada no exemplo número 8

O resultado final da expressão `interfaces.ifType["ATM"].ifSpeed[>100] from devices.ip["200.132.73.33 or 200.132.73.34"]` é obtido por meio do acesso aos valores dos vetores *devices* e *interfaces* (figura 4.6 – letras A e B), pois a chave-primária da classe `interfaces` (primeira classe da expressão) é formada pelo atributo `index` desta classe e pelo atributo `index` da classe `devices`. A classe `interfaces` possui essa chave-primária porque em algumas situações (por exemplo, a visualização de um domínio formado por interfaces) é necessário obter algumas informações adicionais das interfaces selecionadas e isso somente é possível acessando-se primeiramente os dispositivos dessas interfaces.

Busca de valores

Os atributos do modelo de informação utilizados em uma expressão de criação são executados a partir de consultas aos objetos gerenciáveis do ambiente de rede. Essas consultas são realizadas por funções específicas associadas a cada um dos atributos do modelo de informação. Cada função associada a um determinado atributo acessa os objetos gerenciáveis necessários por meio do protocolo apropriado e obtém os valores solicitados nas expressões de criação. Por exemplo, a função associada ao atributo `type` da classe `devices` acessa, via SQL, o repositório de dispositivos do ambiente gerenciado para verificar a existência de dispositivos com as características solicitadas (por exemplo, dispositivos do tipo roteador *DiffServ*) e obter a identificação desses dispositivos (valor do atributo chave-primária). A função associada ao atributo `ifType` da classe `interfaces`, por sua vez, acessa, via SNMP, o objeto *ifType* da MIB *Interfaces* de determinados dispositivos do ambiente gerenciado para verificar a existência de interfaces de rede com as características solicitadas (por exemplo, interfaces de rede do tipo ATM) e obter a identificação dessas interfaces. Essas funções foram implementadas de acordo com os objetos gerenciáveis do ambiente de rede relacionados a cada um dos atributos do modelo de informação. Visto que esses objetos gerenciáveis estão localizados em uma base de dados e em MIBs de dispositivos, tais funções foram desenvolvidas com base no SQL e no SNMP. Entretanto, dependendo dos objetos gerenciáveis necessários, outros protolocos de acesso poderiam ser utilizados, como por exemplo, o LDAP [WAH 97] e o ICMP [POS 81]. As funções de cada atributo são invocadas pelos procedimentos que coordenam o processamento das expressões de criação.

Essas funções recebem dois tipos de parâmetros: valores associados aos atributos (elemento `value` da BNF da linguagem de criação de domínios dinâmicos) e vetores com valores das classes do modelo de informação (por exemplo, os vetores dos exemplos das figuras 4.6 e 4.7). O primeiro tipo de parâmetro é utilizado para indicar às funções que ocorrências da classe desejada devem ser selecionadas. O segundo tipo de parâmetro, por sua vez, é utilizado para otimizar as buscas realizadas pelas funções dos atributos às ocorrências de uma classe. Por exemplo, a função do atributo `type` da classe `devices` ao receber o valor `DSRouter` e um vetor vazio da classe `topologies` irá selecionar os dispositivos do tipo roteador *DiffServ* de todas as topologias do ambiente gerenciado. Entretanto, essa função, ao receber o mesmo valor e um vetor com valores da classe `topologies`, irá selecionar os dispositivos do tipo roteador *DiffServ* apenas das topologias referentes aos valores contidos no vetor recebido. As funções dos atributos podem receber vetores de diferentes classes. No entanto, todas as funções dos atributos de uma mesma

classe recebem tipos de vetores iguais. Por exemplo, as funções dos atributos da classe `devices` recebem vetores com valores das classes `topologies` e `devices`, e as funções dos atributos da classe `interfaces` recebem vetores com valores das classes `devices` e `interfaces`.

As funções dos atributos retornam como resultado vetores com valores das classes do modelo de informação. Esses valores são decorrentes das buscas aos valores solicitados em uma expressão de criação. As funções dos atributos podem retornar um ou mais vetores distintos. No entanto, todas as funções dos atributos de uma mesma classe retornam tipos de vetores iguais. Por exemplo, as funções dos atributos da classe `topologies` retornam um vetor com valores da própria classe, e as funções dos atributos da classe `interfaces` retornam dois vetores distintos: um vetor com valores da classe `devices` e outro vetor com valores da classe `interfaces`. Os vetores distintos retornados por uma determinada função (dois ou mais) são associados. Por exemplo, no caso dos vetores retornados pelas funções dos atributos da classe `interfaces`, o valor da primeira posição do vetor da classe `devices` corresponde ao valor da primeira posição do vetor da classe `interfaces`, e o valor da segunda posição do vetor da classe `devices` corresponde ao valor da segunda posição do vetor da classe `interfaces` e assim por diante. Neste caso, essas associações são necessárias para que se possa saber a que dispositivo uma determinada interface de rede pertence. Tais associações são realizadas por meio de relacionamentos existentes no modelo de dados do ambiente gerenciado. Os vetores de valores resultantes da execução das funções dos atributos das expressões de criação são utilizados para atualizar os vetores envolvidos no processamento dessas expressões (tal processamento foi demonstrado nos exemplos das figuras 4.6 e 4.7).

As funções dos atributos foram implementadas em arquivos PHP4 que receberam o nome das classes do modelo de informação. Por exemplo, todas as funções dos atributos da classe `devices` foram implementadas no arquivo *devices.php*, e todas as funções dos atributos da classe `interfaces` foram implementadas no arquivo *interfaces.php*. Os nomes dados às funções dos atributos armazenadas nos arquivos PHP4 seguiram o seguinte padrão: *get_<nome da classe do atributo>_<nome do atributo>*. Por exemplo, a função do atributo `ip` da classe `devices` recebeu o nome *get_devices_ip*, e a função do atributo `ifType` da classe `interfaces` recebeu o nome *get_interfaces_ifType*. Esse padrão foi utilizado para que as chamadas às funções dos atributos possam ser montadas dinamicamente pelos procedimentos responsáveis por coordenar o processamento das expressões de criação.

Esses procedimentos acessam a matriz de elementos da expressão de criação avaliada para montar as chamadas às funções dos atributos utilizados e, então, invocá-las. Além disso, tais procedimentos acessam também informações complementares armazenadas na base MySQL que indicam os tipos de vetores aceitos como parâmetros pelas funções dos atributos da expressão avaliada. Essas informações são armazenadas na base de dados por classes, pois todas as funções dos atributos de uma mesma classe recebem os mesmos tipos de parâmetros. A seguir, uma expressão de criação é apresentada (exemplo número 9) para demonstrar a forma como as chamadas às funções dos seus atributos são montadas dinamicamente.

```
9) select devices.type["ISRouter"]::interfaces.ifType["ATM"]
                                                    .ifSpeed[>100]
```

Inicialmente, para montar as chamadas às funções dos atributos da expressão de criação número 9, a matriz de elementos desta expressão, apresentada na figura 4.8, precisa ser acessada. Após, os passos apresentados abaixo são seguidos:

1. As colunas da primeira linha da matriz são lidas e, então, o arquivo *devices.php*, que contém as funções dos atributos da classe `devices`, é invocado (figura 4.8 – número 1);
2. As informações complementares da base MySQL são acessadas para se obter os nomes dos vetores aceitos pelas funções dos atributos da classe `devices`. É constatado que os vetores das classes `topologies` e `devices` são aceitos pelas funções dos atributos da classe `devices`;
3. As colunas da segunda linha da matriz são lidas, e a chamada à função do atributo `type` da classe `devices` é montada (figura 4.8 – número 2). Em seguida, essa função é invocada. É importante mencionar que os vetores com valores das classes são passados para as funções dos atributos mesmo que estejam vazios;
4. As colunas da terceira linha da matriz são lidas e, então, o arquivo *interfaces.php*, que contém as funções dos atributos da classe `interfaces`, é invocado (figura 4.8 – número 3);
5. As informações complementares da base MySQL são acessadas novamente para obterem-se os nomes dos vetores aceitos pelas funções dos atributos da classe `interfaces`. É constatado que os vetores das classes `devices` e `interfaces` são aceitos pelas funções dos atributos da classe `interfaces`;
6. As colunas da quarta linha da matriz são lidas, e a chamada à função do atributo `ifType` da classe `interfaces` é montada (figura 4.8 – número 4). Em seguida, essa função é invocada;
7. As colunas da quinta linha da matriz são lidas, e a chamada à função do atributo `ifSpeed` da classe `interfaces` é montada. Em seguida, essa função é invocada (figura 4.8 – número 5).

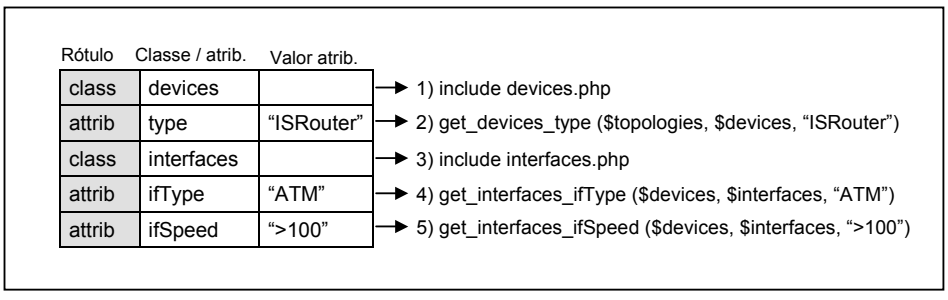

FIGURA 4.8 – Montagem dinâmica das chamadas às funções dos atributos da expressão 9

Os valores retornados pelas funções dos atributos são acessados pelos procedimentos responsáveis por coordenar o processamento das expressões de criação para que os vetores necessários possam ser atualizados, e o resultado final da expressão avaliada

possa ser obtido (como demonstrado nos exemplos das figuras 4.6 e 4.7). O resultado final de uma expressão de criação avaliada é armazenado por tais procedimentos em uma variável. O nome dessa variável é dado pelo próprio usuário que, por meio da interface do sistema, associa uma variável para cada expressão de criação elaborada. Isso é necessário para que a linguagem de visualização de domínios dinâmicos possa acessar os valores resultantes da execução das expressões de criação.

A forma como as chamadas às funções dos atributos utilizados nas expressões de criação são montadas permite que novos atributos sejam adicionados ao modelo de informação do ambiente gerenciado sem que seja necessário realizar grandes alterações no sistema desenvolvido. Por exemplo, para adicionar um novo atributo à classe `interfaces`, os seguintes passos devem ser seguidos:

- Atualização das tabelas da base MySQL que descrevem o modelo de informação do ambiente gerenciado;
- Implementação de uma função para realizar o acesso aos objetos gerenciáveis associados ao novo atributo e a busca aos valores que serão solicitados. Esta função deve seguir o padrão apresentado anteriormente;
- Colocação da função desenvolvida no arquivo *interfaces.php*. Essa função deve receber os mesmos tipos de parâmetros que todas as funções do arquivo *interfaces.php* e retornar os mesmos tipos de resultados que estas funções.

Novas classes também podem ser adicionadas ao modelo de informação do ambiente gerenciado. Para isso, os passos apresentados abaixo devem ser seguidos:

- Atualização das tabelas da base MySQL que descrevem o modelo de informação do ambiente gerenciado;
- Criação de um arquivo PHP4 com o nome da classe desejada;
- Definição e atualização nas informações complementares da base MySQL dos tipos de vetores de valores aceitos e retornados pelas funções dos atributos da nova classe;
- Definição e atualização nas informações complementares da base MySQL da facilidade visual padrão da nova classe;
- Desenvolvimento de atributos (atualização do modelo de informação na base MySQL e implementação das funções necessárias) para a nova classe.

O mecanismo responsável por realizar o processamento das expressões de criação não precisa ser atualizado em nenhum dos casos apresentados acima. Como demonstrado, para expandir o sistema, é preciso somente realizar algumas atualizações na base MySQL, criar alguns arquivos para armazenar as funções dos atributos caso novas classes sejam criadas e desenvolver funções para os novos atributos que forem criados. A próxima seção apresenta como a linguagem de visualização de domínios dinâmicos é suportada pelo sistema desenvolvido.

4.3 Suporte à Visualização de Domínios Dinâmicos

Como demonstrado na figura 4.2, apenas uma expressão de visualização de domínios dinâmicos (por exemplo, a expressão número 3 desta figura) é enviada ao *script* PHP4 principal a cada requisição HTTP transmitida ao servidor. As expressões de visualização são analisadas e executadas após as expressões de criação de domínios dinâmicos. Entretanto, se ocorrerem erros sintáticos ou semânticos nas expressões de criação avaliadas, ou se os valores solicitados nessas expressões não estiverem disponíveis, as expressões de visualização não chegam a ser avaliadas, pois a execução geral do sistema é interrompida. As ações correspondentes às expressões de visualização são apresentadas na parte inferior da figura 4.2.

Para cada expressão de visualização recebida pelo *script* PHP4 principal, os seguintes passos são executados:

1. Análise sintática da expressão;
2. Análise semântica da expressão;
3. Execução da expressão (geração da visualização desejada).

Análise Sintática

O primeiro passo executado é a análise sintática. Essa análise é realizada através de um programa escrito na linguagem C gerado automaticamente pelo *software* PRECCX a partir da BNF da linguagem de visualização de domínios dinâmicos. O *script* PHP4 principal inicia este programa passando uma expressão de visualização. O programa em C, após ser executado, gera um resultado como saída que é capturado pelo *script* PHP4 principal. Esse resultado é analisado e, não ocorrendo erros sintáticos, a expressão passada é avaliada semanticamente. Entretanto, se erros existirem, a execução geral do sistema é encerrada, e os erros identificados são apresentados ao usuário.

A BNF completa utilizada para gerar o analisador sintático da linguagem de visualização de domínios dinâmicos através do *software* PRECCX é apresentada no anexo 1.

Análise Semântica

A análise semântica de uma expressão de visualização é realizada em duas partes distintas:

- Verificação da igualdade das classes dos domínios dinâmicos distintos armazenados nas variáveis utilizadas na expressão avaliada. Essas variáveis possuem valores resultantes da execução de expressões de criação;
- Verificação da existência, no repositório de informações da linguagem de visualização (figura 4.2 – letra B), da facilidade visual e dos atributos dessa facilidade utilizados na expressão avaliada.

A primeira parte da análise semântica é executada somente se a visualização de dois ou mais domínios dinâmicos distintos gerados através da linguagem de criação for solicitada. Esta parte da análise semântica é realizada para garantir que os domínios

dinâmicos distintos de uma visualização possuam ocorrências de uma mesma classe. Por exemplo, um erro semântico será identificado na avaliação da expressão apresentada no exemplo número 1 se os domínios dinâmicos armazenados nas variáveis `IntServRouters` e `RSVP33Routers` possuírem ocorrências de classes diferentes. A segunda parte da análise semântica é executada para garantir que a facilidade visual e os atributos dessa facilidade utilizados em uma expressão de visualização sejam válidos. Por exemplo, de acordo com o repositório de informações da linguagem de visualização, a expressão de visualização apresentada no exemplo número 1 está correta semanticamente, pois a facilidade `table` existe neste repositório, e o atributo `cellcolor` faz parte desta facilidade. Entretanto, a expressão apresentada no exemplo número 2 possui um erro semântico, pois a facilidade `map` não existe em tal repositório. O atributo visual `alpha` não chega a ser avaliado porque a facilidade `map` não existe no repositório de informações da linguagem de visualização.

```
1) using table with cellcolor=yellow show IntServRouters and
                                                    RSVP33Routers

2) using map show IntServRouters with alpha=30 and RSVP33Routers
```

Cada variável utilizada nas expressões de visualização armazena os valores de um domínio dinâmico criado em um ou mais vetores. Esses vetores possuem rótulos que identificam as classes dos valores armazenados. Dessa forma, a verificação da igualdade das classes dos domínios dinâmicos distintos armazenados nas variáveis é realizada analisando-se os rótulos dos vetores dessas variáveis.

O repositório de informações da linguagem de visualização é formado por duas tabelas que estão armazenadas na base de dados MySQL. As facilidades visuais existentes no ambiente gerenciado foram armazenadas na tabela *facilities*, e os atributos dessas facilidades foram armazenados na tabela *facilities_attribs*. Essas tabelas são apresentadas na figura 4.9.

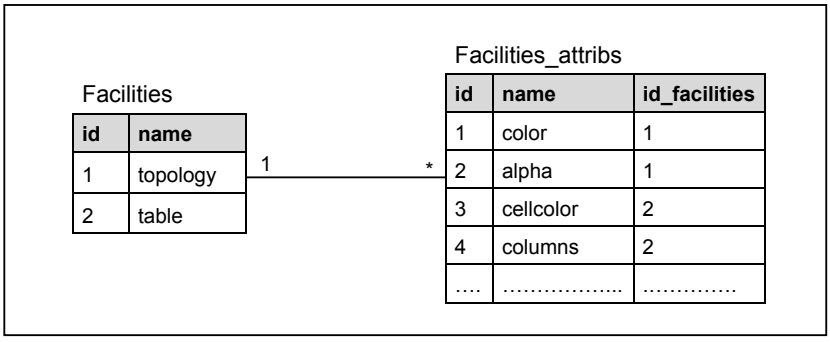

FIGURA 4.9 – Repositório de informações da linguagem de visualização na base MySQL

A existência das facilidades visuais do ambiente gerenciado é verificada a partir de consultas ao campo *name* da tabela *facilities*. A existência dos atributos de uma facilidade visual, por sua vez, é verificada por meio de consultas aos campos *name* e *id_facilities* da tabela *facilities_attribs*. O campo *id_facilities* é responsável por relacionar os atributos (identificados pelo campo *name*) da tabela *facilities_attribs* com as facilidades visuais da tabela *facilities*. Na figura 4.9, por exemplo, a partir desse campo é possível verificar que os atributos `color` e `alpha` da tabela *facilities_atribs* pertencem à facilidade visual `topology`

da tabela *facilities*, e que os atributos `cellcolor` e `columns` da tabela *facilities_attribs* pertencem à facilidade visual `table` da tabela *facilities*. Nas tabelas *facilities* e *facilities_attribs*, demonstradas na figura 4.9, foram apresentados somente os campos úteis à avaliação semântica de uma expressão de visualização.

O *script* PHP4 principal inicia o processo de avaliação semântica de uma expressão de visualização recebida através da invocação de funções armazenadas em um outro *script*. Inicialmente, a expressão de visualização avaliada é segmentada, e seus elementos (facilidade visual, atributos da facilidade e variáveis) são armazenados em uma matriz para que sejam manipulados com maior facilidade. Após, as variáveis armazenadas na matriz são acessadas para que a verificação da igualdade das classes dos domínios dessas variáveis seja realizada. Por último, caso os domínios dinâmicos armazenados em tais variáveis sejam procedentes de uma mesma classe, a existência da facilidade visual e dos seus atributos, também armazenados na matriz, é verificada a partir de consultas às tabelas *facilities* e *facilities_atribs* da base MySQL. Os valores dos atributos da facilidade visual utilizada também são armazenados na matriz de elementos para que ela possa ser utilizada no processo de geração de uma visualização solicitada (próxima subseção).

As expressões de visualização são segmentadas para que a facilidade visual utilizada, os atributos dessa facilidade, os valores desses atributos e as variáveis que armazenam os domínios criados possam ser extraídos e armazenados na matriz de elementos dessas expressões. A segmentação de uma expressão de visualização é efetuada a partir das cláusulas `show`, `with` e `and`; da vírgula (“`,`”) e do igual (“`=`”). A forma como as expressões de visualização recebidas pelo *script* PHP4 principal são segmentadas é demonstrada a seguir através de alguns exemplos.

```
3) using table with cellcolor=blue show IntServRouters and
                              RSVP33Routers with cellcolor=red and
                              DiffServRouters with cellcolor=yellow

4) using topology show IntServRouters with alpha=30, color=blue and
                                                     RSVP33Routers

5) using topology show IntServRouters and RSVP33Routers

6) show IntServRouters and RSVP33Routers
```

A expressão de visualização apresentada no exemplo número 3 inicialmente é segmentada a partir da cláusula `show`. Após, o segmento dessa expressão situado à esquerda da cláusula `show` (`using table with cellcolor=blue`), resultante da primeira divisão, é secionado através da cláusula `with`. O segmento da expressão situado à direita da cláusula `show` (`IntServRouters and RSVP33Routers with cellcolor=red and DiffServ-Routers with cellcolor=yellow`), por sua vez, é secionado a partir da cláusula `and`. As partes `RSVP33Routers with cellcolor=red` e `DiffServRouters with cellcolor=yellow`, resultantes dessa segmentação, são, então, secionadas por meio da cláusula `with`. Por último, os atributos da facilidade visual e os valores desses atributos (`cellcolor=blue`, `cellcolor=red` e `cellcolor=yellow`) são secionados através do igual (“`=`”).

A expressão de visualização apresentada no exemplo número 4 primeiramente é segmentada a partir da cláusula `show`. Após, o segmento da expressão situado à direita da cláusula `show` (`IntServRouters with alpha=30, color=blue and RSVP33Routers`) é secionado a partir da cláusula `and`. A parte desse segmento situada à esquerda da cláusula `and` (`IntServRouters with alpha=30, color=blue`) é, então, secionada por meio da cláusula `with`. A seguir, a parte deste último segmento situada à direita da cláusula `with` (`alpha=30, color=blue`) é secionada a partir da vírgula (“,”). Por último, os atributos da facilidade visual e os valores desses atributos (`alpha=30` e `color=blue`) são segmentados através do igual (“=”).

A expressão de visualização apresentada no exemplo número 5 é segmentada a partir da cláusula `show`. Após, o segmento da expressão situado à direita da cláusula `show` (`IntServRouters and RSVP33Routers`) é secionado a partir da cláusula `and`. A expressão de criação apresentada no exemplo número 6, por sua vez, é segmentada somente a partir da cláusula `and`.

A facilidade visual, os atributos dessa facilidade, os valores desses atributos e as variáveis que possuem os domínios dinâmicos criados são armazenados em uma matriz de elementos após serem extraídos de uma expressão de visualização. A matriz resultante da segmentação da expressão de visualização apresentada no exemplo número 3 é demonstrada na figura 4.10. A linha da matriz de elementos de uma expressão de visualização responsável por armazenar a facilidade visual utilizada recebe o rótulo *facility*, as linhas que armazenam os atributos de uma facilidade visual recebem o rótulo *attrib* e as linhas que armazenam os nomes das variáveis dos domínios criados recebem o rótulo *var*. Os atributos de uma facilidade visual referentes às configurações globais de uma visualização são armazenados nas linhas situadas após a linha dessa facilidade visual. Já os atributos visuais associados às configurações específicas dos domínios distintos de uma visualização são armazenados nas linhas situadas após as linhas das variáveis que possuem os valores desses domínios. Os valores dos atributos visuais são armazenados nas colunas situadas após as colunas dos atributos a que são associados. A facilidade visual de uma expressão de visualização avaliada e os atributos globais dessa facilidade possuem prioridade de armazenamento em relação aos outros elementos dessa expressão. Entretanto, as primeiras variáveis que possuem os domínios criados e os seus atributos visuais específicos possuem prioridade de armazenamento em relação às últimas variáveis dessa expressão.

Rótulo	Facility / var / atrib.	Valor atrib
facility	table	
attrib	cellcolor	blue
var	IntServRouters	
var	RSVP33Routers	
attrib	cellcolor	red
var	DiffServRouters	
attrib	cellcolor	yellow

FIGURA 4.10 – Matriz resultante da segmentação da expressão apresentada no exemplo 3

As classes do modelo de informação possuem facilidades visuais padrão. Estas informações estão armazenadas na base de dados MySQL. Tais informações são acessadas quando a facilidade visual desejada não está especificada em uma expressão de visualização avaliada (por exemplo, a expressão apresentada no exemplo número 6). Assim, a classe dos domínios dinâmicos que serão apresentados visualmente é obtida antes da geração da matriz da expressão avaliada para que se possa verificar na base de dados a sua facilidade padrão. Após, a matriz de elementos da expressão avaliada é preenchida.

As duas partes da análise semântica comentadas anteriormente são executadas assim que a matriz de elementos da expressão avaliada for gerada. As linhas e colunas da matriz são percorridas, e as verificações necessárias são realizadas. Variáveis que armazenam domínios dinâmicos procedentes de classes distintas, e facilidades visuais e atributos não existentes no repositório de informações da linguagem de visualização geram erros semânticos. É importante mencionar que a segunda parte da análise semântica é executada somente se a primeira parte dessa análise for válida. A execução da visualização desejada inicia senão ocorrerem erros semânticos. Entretanto, se os erros existirem, a execução geral do sistema é encerrada, e os erros identificados são apresentados ao usuário.

Execução da Visualização

As facilidades visuais existentes no ambiente gerenciado (`topology` e `table`) foram desenvolvidas em dois arquivos PHP4 específicos que receberam os nomes das próprias facilidades. Em cada um desses arquivos, foi implementada apenas uma função que também recebeu o nome da facilidade associada. Por exemplo, no arquivo *topology.php* foi desenvolvida uma função chamada *topology*, e no arquivo *table.php* foi desenvolvida uma função chamada *table*. Essas funções são responsáveis por gerar as visualizações solicitadas referentes a cada facilidade visual. Tais funções recebem dois tipos de parâmetros: atributos visuais globais e atributos visuais específicos. O primeiro tipo de parâmetro indica os atributos visuais da facilidade escolhida que serão aplicados a todos os domínios da visualização apresentada. O segundo tipo de parâmetro, por sua vez, indica os atributos visuais da facilidade escolhida que serão aplicados somente a domínios específicos da visualização apresentada.

```
7)  using topology with color=blue, alpha=10 show RSVP33Routers
                    with color=red, alpha=10 and DiffServRouters
                    and IntServRouters with color=yellow, alpha 70
```

Os atributos globais utilizados em uma expressão de visualização avaliada (atributos situados após a facilidade visual na matriz de elementos) são extraídos da matriz de elementos dessa expressão e armazenados em uma outra matriz. Por exemplo, a figura 4.12 à esquerda apresenta a matriz responsável por armazenar os atributos globais extraídos da matriz de elementos da expressão de visualização número 7 (figura 4.11). Os atributos específicos de uma expressão de visualização avaliada (atributos situados após as variáveis na matriz de elementos) também são extraídos da matriz de elementos dessa expressão e armazenados em outra matriz. Por exemplo, a figura 4.12 à direita apresenta a matriz responsável por armazenar os atributos específicos da matriz de elementos da expressão de visualização apresentada no exemplo número 7 (figura 4.11). Essas matrizes são passadas

para a função da facilidade visual utilizada na expressão avaliada para que a visualização solicitada possa ser gerada.

Rótulo	Facility / var / atrib.	Valor atrib
facility	topology	
attrib	color	blue
attrib	alpha	10
var	RSVP33Routers	
attrib	color	red
attrib	alpha	50
var	DiffServRouters	
var	IntServRouters	
attrib	color	yellow
attrib	alpha	70

FIGURA 4.11 – Matriz de elementos da expressão de visualização do exemplo 7

A facilidade visual utilizada em uma expressão de visualização avaliada também é extraída da matriz de elementos dessa expressão e armazenada em uma outra variável. Essa variável é usada para invocar o arquivo que possui a função da facilidade visual usada e montar dinamicamente a chamada a esta função. Por exemplo, ao acessar a variável que possui o nome da facilidade visual utilizada na expressão de visualização apresentada no exemplo número 7, os procedimentos responsáveis por processar as expressões de visualização irão invocar o arquivo *topology.php* (por exemplo, *include topology.php*) e montar dinamicamente a chamada à função *topology ($global_attribs , $specific_attribs)*. Os parâmetros *$global_attribs* e *$specific_attribs* correspondem respectivamente aos atributos visuais globais e aos atributos visuais específicos utilizados na expressão de visualização do exemplo número 7.

Rótulo	Var / atrib.	Valor atrib
attrib	color	blue
attrib	alpha	10

Atributos visuais globais

Rótulo	Var / atrib.	Valor atrib
var	RSVP33Routers	
attrib	color	red
attrib	alpha	50
var	IntServRouters	
attrib	color	yellow
attrib	alpha	70

Atributos visuais específicos

FIGURA 4.12 – Matrizes com atributos visuais globais e atributos visuais específicos

A facilidade visual *topology* gera visualizações na forma de mapas topológicos somente para domínios que possuem ocorrências das classes `topologies` e `devices`. Ao ser invocada, a função dessa facilidade inicialmente acessa a matriz de atributos visuais específicos da expressão avaliada (por exemplo, a figura 4.12 à direita) e obtém o nome das variáveis utilizadas nessa expressão. Essa função acessa, então, tais variáveis e obtém os

valores dos domínios dinâmicos criados. É importante relembrar que cada um desses valores corresponde ao valor do atributo chave-primária de um objeto selecionado de uma classe do modelo de informação. Em seguida, tal função gera um arquivo-texto com os valores dos objetos dos domínios criados e associa os parâmetros recebidos (atributos visuais globais e atributos visuais específicos) a esses valores. Além disso, outras informações são associadas a cada valor de um objeto selecionado, como a posição do objeto no mapa que será gerado (por exemplo, coordenadas x e y), o tipo do objeto (por exemplo, roteador, *host*, topologia, etc), o nome do objeto (por exemplo, *voyager*, *hubble*, *mir*, etc) e um *link* para um *script* PHP4 que apresenta algumas informações do objeto. A maioria dessas informações estão armazenadas na base de dados MySQL e são obtidas através do valor de cada objeto. Por último, a função da facilidade *topology* invoca um arquivo desenvolvido no software FLASH [MAC 2001]. Esse arquivo lê as informações dos objetos dos domínios armazenadas no arquivo-texto criado anteriormente e gera a visualização solicitada.

A facilidade visual *table* gera visualizações na forma de tabelas HTML padrão para ocorrências de todas as classes do modelo de informação. Ao ser invocada, a função dessa facilidade inicialmente acessa a matriz de atributos visuais específicos da expressão avaliada (por exemplo, a matriz da figura 4.12 à direita) e obtém o nome das variáveis utilizadas nessa expressão. Essa função acessa, então, tais variáveis e obtém os valores dos domínios dinâmicos criados e a classe desses domínios. Após, os atributos visuais específicos e os atributos visuais globais armazenados nas matrizes recebidas como parâmetros são acessados e utilizados na geração da tabela que apresentará os domínios dinâmicos armazenados nas variáveis utilizadas nessa expressão. Alguns atributos da facilidade *table* (por exemplo, tamanho da borda e colunas da tabela) são aplicados à estrutura geral da tabela apenas uma vez, pois não podem ser aplicados a cada uma das linhas dessa tabela. Entretanto, outros atributos dessa facilidade (por exemplo, cor das células da tabela) são aplicados diversas vezes a uma mesma tabela, pois podem ser aplicados a cada uma das linhas dessa tabela.

Os elementos dos domínios dinâmicos que serão apresentados em uma tabela solicitada são organizados por linhas. Assim, todas as informações apresentadas em uma linha correspondem a somente um elemento de um domínio visualizado. As colunas de uma tabela solicitada são formadas por atributos da classe do modelo de informação da qual fazem parte os elementos dos domínios visualizados. Dessa forma, as linhas de uma tabela gerada apresentam colunas que possuem valores dos atributos de uma classe referentes aos elementos presentes nessas linhas. As colunas de uma tabela solicitada são indicadas por meio do atributo `columns`. No entanto, caso esse atributo visual não seja utilizado, a tabela solicitada será gerada com todas as colunas (atributos) possíveis para os elementos dos domínios que serão apresentados. Na implementação atual, o atributo `columns` é aplicado a uma tabela solicitada somente quando utilizado como atributo visual global. Exemplos do funcionamento da facilidade visual *table* foram apresentados na seção 3.4 do capítulo 3.

Os valores de cada atributo apresentados nas colunas de uma tabela referentes aos elementos dos domínios visualizados são obtidos a partir da execução de funções que estão armazenadas em arquivos PHP4 distintos do arquivo da facilidade *table*. Cada atributo do modelo de informação possui uma função específica associada responsável por acessar os objetos gerenciáveis necessários por meio do protocolo adequado e obter o valor do

atributo em questão para os elementos dos domínios que serão visualizados. Essas funções são invocadas pela função da facilidade *table* no momento da geração de uma nova tabela solicitada.

As funções dos atributos utilizadas na geração das tabelas solicitadas foram armazenadas em arquivos PHP4. Esses arquivos receberam o nome das classes dos atributos que armazenam. Por exemplo, todas as funções dos atributos da classe `topologies` foram armazenadas no arquivo *topologies.php* e todas as funções dos atributos da classe `devices` foram armazenadas no arquivo *devices.php*. Os nomes dados às funções dos atributos armazenados nos arquivos PHP4 seguiram o seguinte padrão: *show_<nome da classe do atributo>_<nome do atributo>*. Por exemplo, a função associada ao atributo `ip` da classe `devices` recebeu o nome *show_devices_ip*, e o atributo `ifType` da classe `interfaces` recebeu o nome *show_interfaces_ifType*. Essas funções recebem como parâmetro o valor de um objeto de um domínio que será apresentado e retornam como resultado o valor do atributo associado para esse objeto.

As chamadas a estas funções são montadas dinamicamente pela função da facilidade *table*. Primeiramente, essa função acessa os valores dos domínios dinâmicos que serão apresentados. Após, tal função verifica os atributos do modelo de informação correspondentes às colunas da tabela solicitada e a classe desses atributos. De posse de todas as informações necessárias, a função da facilidade *table* monta as chamadas às funções necessárias. Essas funções diferem-se das funções dos atributos que são executadas no processamento de uma expressão de criação. As funções dos atributos executadas na geração de uma tabela obtêm os valores dos atributos de uma classe para uma ocorrência dessa classe, enquanto as funções dos atributos executadas no processamento de uma expressão de criação obtêm todas as ocorrências de uma classe que tenham um determinado atributo com um valor específico.

À medida que novos atributos forem acrescentados ao modelo de informação do ambiente gerenciado, funções correspondentes a eles devem ser desenvolvidas para que esses atributos sejam suportados pela facilidade *table*. As funções dos novos atributos devem ser armazenadas nos arquivos correspondentes às classes desses atributos. Caso novas classes forem acrescentadas ao modelo de informação, arquivos PHP4 com o nome dessas classes devem ser criados e as funções dos atributos dessas classes devem ser armazenadas em tais arquivos. O nome das novas funções desenvolvidas, o parâmetro recebido e o resultado retornado por essas funções devem seguir o padrão apresentado anteriormente.

A forma como as facilidades visuais foram desenvolvidas e organizadas permite que novas facilidades sejam adicionadas ao sistema sem que seja necessário realizar grandes alterações. Para isso, os passos apresentados abaixo devem ser seguidos:

- Atualização na base MySQL das informações das facilidades existentes no ambiente gerenciado e dos atributos dessas facilidades;
- Criação de um arquivo PHP4 com o nome da nova facilidade;
- Desenvolvimento nesse arquivo de uma função responsável por gerar as visualizações da nova facilidade. Essa função deve possuir o nome da nova

facilidade e receber os mesmos tipos de parâmetros que as facilidades já existentes.

A próxima seção apresentará maiores detalhes sobre a interface e o funcionamento do protótipo desenvolvido para suportar as linguagens de criação e visualização de domínios dinâmicos.

4.4 Funcionamento do Protótipo

O protótipo desenvolvido foi implementado como um subsistema do ambiente de gerenciamento QAME [GRA 2001]. Ao acessá-lo, uma tela semelhante à demonstrada na figura 4.1 é apresentada. As caixas de texto indicadas pelas letras B_1, B_2, C_1 e C_2 dessa figura são utilizadas para a elaboração de duas expressões de criação de domínios dinâmicos, e a caixa de texto indicada pela letra D da mesma figura é utilizada para a elaboração de uma única expressão de visualização de domínios dinâmicos. As caixas de texto das letras B_1 e C_1 informam ao sistema o nome das variáveis responsáveis por armazenar os elementos dos domínios dinâmicos gerados pelas expressões de criação formuladas nas caixas de texto das letras B_2 e C_2.

Inicialmente, o protótipo disponibiliza ao usuário duas caixas de texto para a elaboração de apenas uma expressão de criação de domínios dinâmicos. Entretanto, o número dessas caixas de texto pode ser expandido por meio da caixa de texto *Selections* e do botão *Enter* (figura 4.1 – letras A) para que mais de uma expressão de criação possa ser executada pelo sistema. Por exemplo, na tela apresentada na figura 4.13 existem caixas de texto para a elaboração de três expressões de criação de domínios dinâmicos (figura 4.13 – letras A_1, A_2, B_1, B_2, C_1 e C_2).

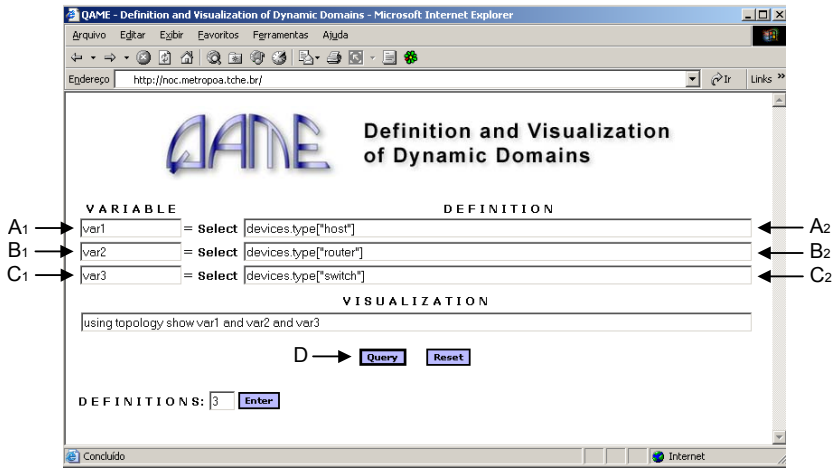

FIGURA 4.13 – Expansão do número de caixas de texto referentes à linguagem de criação

As caixas de texto indicadas pelas letras A_1, B_1 e C_1 na figura 4.13 são utilizadas para informar ao sistema o nome das variáveis responsáveis por armazenar os elementos dos domínios dinâmicos gerados pelas expressões de criação elaboradas nas caixas de texto indicadas pelas letras A_2, B_2 e C_2 na figura 4.13. As expressões de criação e visualização

elaboradas nas caixas de texto disponibilizadas pelo protótipo desenvolvido são processadas após o botão *Query* (figura 4.13 – letra D) ser pressionado.

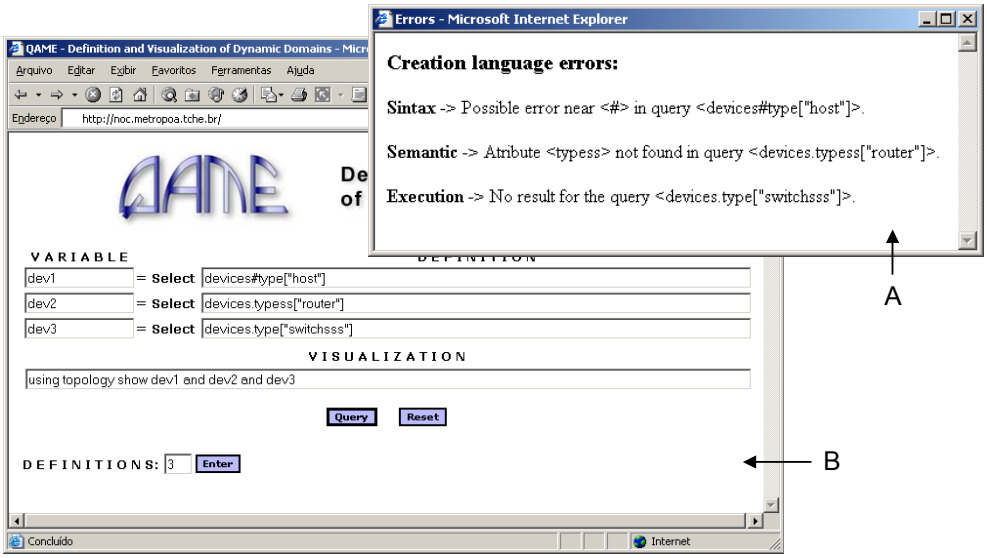

FIGURA 4.14 – Mensagens de erro.

Mensagens de erro semelhantes às demonstradas na figura 4.14 (letra A) são apresentadas ao usuário caso as expressões processadas possuam erros sintáticos ou erros semânticos, ou os valores solicitados nessas expressões não estejam disponíveis. As mensagens de erro apresentadas nessa figura são referentes ao processamento das expressões de criação demonstradas na letra B da figura 4.14. Neste exemplo, cada expressão de criação elaborada possui um erro distinto (erro sintático, erro semântico e resultado não disponível). Caso as expressões de criação e visualização de domínios dinâmicos processadas não possuam erros, a visualização solicitada é gerada. A figura 4.15 demonstra um exemplo de visualização em forma de mapa topológico. A figura 4.16, por sua vez, demonstra um exemplo de visualização em forma de tabela.

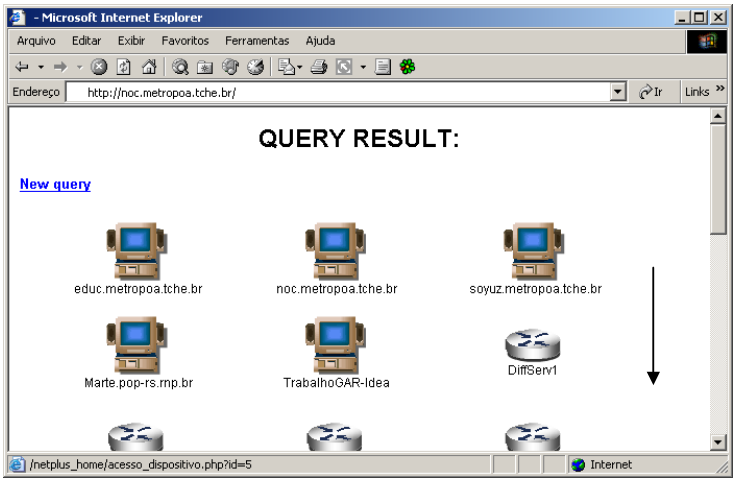

FIGURA 4.15 – Visualização em forma de mapa topológico

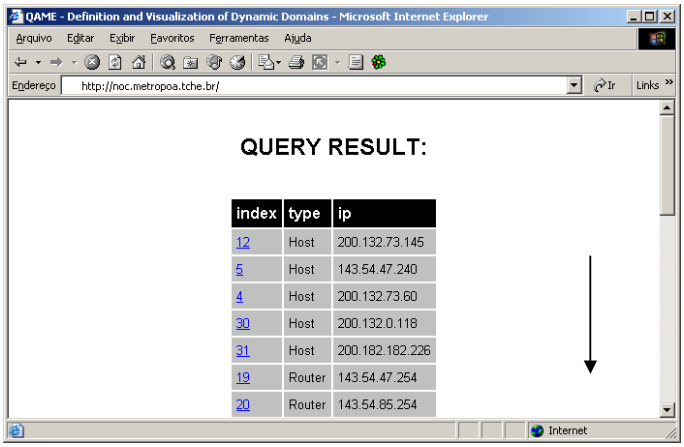

FIGURA 4.16 – Visualização em forma de tabela

Informações adicionais às apresentadas nas visualizações geradas podem ser obtidas quando os membros dos domínios apresentados forem provenientes da classe `devices`. Para isso, basta clicar no elemento desejado de um mapa topológico gerado ou no campo `index` de tal elemento de uma tabela gerada. Ao realizar uma dessas ações, uma tela semelhante à demonstrada na figura 4.17 será apresentada. Nessa tela há algumas informações do elemento escolhido, *links* para algumas MIBs desse elemento e *links* para gráficos gerados pelo *software* MRTG referentes a tal elemento.

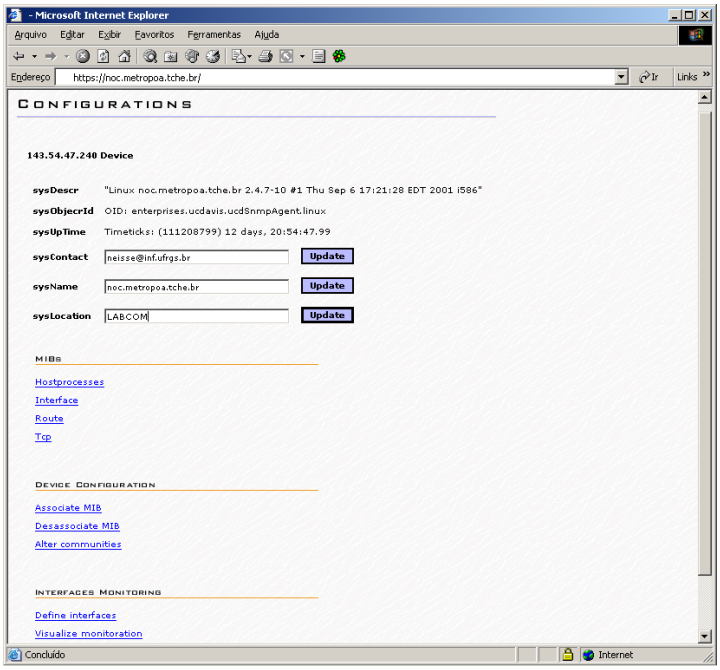

FIGURA 4.17 – Informações adicionais de um dispositivo membro de um domínio visualizado

A seguir são apresentadas as conclusões resultantes desta dissertação e os trabalhos futuros.

5 Conclusões e Trabalhos Futuros

O uso de domínios nos sistemas de gerenciamento de redes é importante porque as ações de gerenciamento (por exemplo, aplicação de políticas) podem ser aplicadas a todos os objetos gerenciáveis de um domínio ao mesmo tempo, não sendo necessário, então, repetir a mesma ação em cada objeto gerenciável, um a um. Existem algumas situações em que determinados domínios precisam ser rapidamente criados, utilizados e descartados. Domínios com essas características foram referenciados nesta dissertação de mestrado por meio do termo "domínios dinâmicos". Atualmente, os sistemas de gerenciamento de redes convencionais possuem pouco ou nenhum suporte ao conceito de domínios dinâmicos. Em relação à visualização de domínios, o suporte fornecido por esses sistemas é restrito a poucas opções de personalização gráfica. Este foi então o ponto principal desta dissertação: o desenvolvimento de um suporte inicial a domínios dinâmicos e o aperfeiçoamento da visualização de domínios.

A tentativa de gerar uma solução automatizada para possibilitar a criação de domínios de forma simples e rápida, e a necessidade de permitir visualizações mais dinâmicas dos domínios criados resultou na primeira contribuição desta dissertação: a definição de linguagens para a criação e a visualização de domínios dinâmicos. A primeira linguagem é baseada em um modelo de informação dinâmico e é usada para criar novos domínios dinâmicos. A segunda linguagem, por sua vez, é utilizada para configurar características visuais dos domínios que forem criados através da primeira linguagem. O desenvolvimento de um protótipo para suportar as linguagens criadas foi a segunda contribuição desta dissertação. Este protótipo é baseado na Web e foi desenvolvido como um subsistema do ambiente QAME através das tecnologias PRECCX, PHP, MySQL e SNMP. O ambiente QAME tem por objetivo o gerenciamento de redes com suporte a QoS. Uma operação normal neste ambiente é a seleção de dispositivos, nos quais, políticas de rede devem ser aplicadas. As linguagens de criação e visualização, no contexto do QAME, facilitarão o processo de aplicação de políticas.

O suporte desenvolvido para a linguagem de criação de domínios dinâmicos proporciona aos administradores de rede simplicidade e rapidez na tarefa de selecionar objetos gerenciáveis para formar um novo domínio, pois não é mais necessário vasculhar manualmente a rede de computadores utilizada a procura dos objetos gerenciáveis necessários a formação do domínio desejado. O tempo gasto na criação de domínios por meio de uma solução automatizada é bastante inferior ao tempo gasto com a criação manual desses domínios. É importante lembrar que o tempo necessário para a criação de um domínio deve ser menor que o tempo de vida desse mesmo domínio, caso contrário, ações de gerenciamento podem ser aplicadas a um domínio obsoleto.

O suporte desenvolvido para a linguagem de visualização de domínios dinâmicos permite aos administradores de rede visualizar e personalizar graficamente os domínios criados com o suporte à linguagem de criação. Além de poderem escolher o tipo de visualização (tabela ou topologia), os administradores podem, também, escolher as cores em que os membros dos domínios criados serão apresentados, entre outros parâmetros visuais configuráveis. A característica dinâmica do suporte à linguagem de visualização faz com que os diferentes domínios dinâmicos apresentados possam ser destacados de forma distinta, proporcionando assim, uma análise diferenciada das informações, pois aumenta a

percepção do administrador. Normalmente, nos sistemas de gerenciamento, os administradores não têm a liberdade de escolher como desejam visualizar um determinado domínio, pois os parâmetros visuais na maioria das vezes são estáticos. Por exemplo, na maioria dos sistemas, os dispositivos de rede que apresentam algum problema são apresentados em vermelho, porém, o administrador de rede não tem a opção de escolher a cor amarela para destacar esses dispositivos.

A possibilidade de expansão dos suportes às linguagens de criação e visualização de domínios dinâmicos é uma das principais características do protótipo desenvolvido. Esta característica é importante, pois permite que o administrador de rede adicione novas funcionalidades ao sistema, conforme a sua necessidade. Sem precisar alterar o núcleo do sistema e realizando apenas algumas alterações na base de dados usada pelo protótipo, é possível adicionar novas classes e atributos ao modelo de informação utilizado pela linguagem de criação de domínios dinâmicos. De forma semelhante, pode-se também incluir novas facilidades visuais ao ambiente utilizado. Essas novas facilidades podem ser utilizadas por meio da linguagem de visualização de domínios dinâmicos.

O objetivo principal desta dissertação foi alcançado, visto que, um suporte inicial a domínios dinâmicos foi desenvolvido. Com relação ao processo de desenvolvimento do protótipo, outras contribuições podem ser enumeradas. Primeiramente, como a criação de domínios dinâmicos é baseada em uma visão comum dos objetos gerenciáveis, foi implementado um único e simples modelo de informação ao invés de diferentes modelos de dados. A obtenção das informações organizadas no modelo de informação é fornecida pela utilização de funções de acesso aos objetos gerenciáveis associados a esse modelo. Além de possibilitar a operação adequada da linguagem de criação de domínios dinâmicos, o modelo de informação desenvolvido faz com que o administrador de rede não precise se preocupar com os protocolos específicos necessários para acessar os objetos gerenciáveis desejados.

Como trabalhos futuros, a linguagem de criação de domínios dinâmicos pode ser expandida para suportar a utilização de vários operadores lógicos `and` e `or` em um único elemento `value` da BNF desta linguagem. Além disso, facilidades de armazenamento em relação aos domínios dinâmicos criados devem ser implementadas no protótipo desenvolvido. Estas facilidades devem gravar não somente os domínios criados a partir da linguagem de criação, mas também as expressões das linguagens de criação e visualização de domínios dinâmicos, e a visualização final fornecida. Por fim, acredita-se que embora as linguagens baseadas em texto sejam fáceis de usar, a utilização de assistentes gráficos para auxiliar o usuário na elaboração das expressões aumentariam a habilidade de criação e visualização dos domínios dinâmicos.

Anexo 1 BNFs das Linguagens de Criação e Visualização de Domínios Dinâmicos

Neste anexo são apresentadas as BNFs completas das linguagens de criação e visualização de domínios dinâmicos. Essas BNFs foram utilizadas para gerar os analisadores sintáticos das linguagens de criação e visualização através do software PRECCX.

BNF da Linguagem de Criação de Domínios Dinâmicos

```
@ from = <'f'><'r'><'o'><'m'>

@ query = class[tmp1][tmp2]

@ class = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> | <'h'>
                 | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> | <'n'> | <'o'>
                 | <'p'> | <'q'> | <'r'> | <'s'> | <'t'> | <'u'> | <'v'>
                 | <'x'> | <'z'> | <'w'> | <'y'> | <'A'> | <'B'> | <'C'>
                 | <'D'> | <'E'> | <'F'> | <'G'> | <'H'> | <'I'> | <'J'>
                 | <'K'> | <'L'> | <'M'> | <'N'> | <'O'> | <'P'> | <'Q'>
                 | <'R'> | <'S'> | <'T'> | <'U'> | <'V'> | <'X'> | <'Z'>
                 | <'W'> | <'K'> | <'Y'> | <'-'> | <'_'>}*

@ tmp1 = {atrib1 | <':'><':'>class}*

@ atrib1 = {<'.'>atribute}<'['><'"'>{value}<'"'>{<','><'"'>{value}<'"'>}*
                 <']'> |{<'.'>atribute}<'['>{value}{<','>{value} }*<']'>

@ atribute = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> |
                      <'h'> | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> |
                      <'n'> | <'o'> | <'p'> | <'q'> | <'r'> | <'s'> |
                      <'t'> | <'u'> | <'v'> | <'x'> | <'z'> | <'w'> |
                      <'y'> | <'A'> | <'B'> | <'C'> | <'D'> | <'E'> |
                      <'F'> | <'G'> | <'H'> | <'I'> | <'J'> | <'K'> |
                      <'L'> | <'M'> | <'N'> | <'O'> | <'P'> | <'Q'> |
                      <'R'> | <'S'> | <'T'> | <'U'> | <'V'> | <'X'> |
                      <'Z'> | <'W'> | <'K'> | <'Y'> | <'-'> | <'_'>}*

@ value  = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> | <'h'>
                  | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> | <'n'> | <'o'>
                  | <'p'> | <'q'> | <'r'> | <'s'> | <'t'> | <'u'> | <'v'>
                  | <'x'> | <'z'> | <'w'> | <'y'> | <'1'> | <'2'> | <'3'>
                  | <'4'> | <'5'> | <'6'> | <'7'> | <'8'> | <'9'> | <'0'>
                  | <'<'> | <'>'> | <'!'> | <'.'> | <'='>}*

@ tmp2 = {<' '>from<' '>class{atrib2}}+

@ atrib2 =  {{<'.'>atribute}<'['><'"'>{value}<'"'>{<','><'"'>{value}
                 <'"'>}*<']'>}+ | {{<'.'>atribute}<'['>{value}{<','>
                 {value}}*<']'>}+
```

BNF da Linguagem de Visualização de Domínios Dinâmicos

```
@ show = <'s'><'h'><'o'><'w'>

@ using = <'u'><'s'><'i'><'n'><'g'>

@ and = <'a'><'n'><'d'>

@ with = <'w'><'i'><'t'><'h'>


@ visualization = {[selection<' '>] show <' '> dview {<' '> and <' '>
                                                        dview}*}

@ selection  = {using <' '> facility [<' '>customization]}

@ dview = {domain [<' '>customization]}

@ customization = {with<' '>atribute<'='>value}{more_atr_value}*

@ more_atr_value = {<','><' '>atribute<'='>value}

@ value = {inter_value} | {final_value}

@ inter_value = {<'{'>final_value{<','><' '>final_value}+<'}'>}

@ facility = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> |
                      <'h'> | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> |
                      <'n'> | <'o'> | <'p'> | <'q'> | <'r'> | <'s'> |
                      <'t'> | <'u'> | <'v'> | <'x'> | <'z'> | <'w'> |
                      <'y'> | <'_'> | <'1'> | <'2'> | <'3'> | <'4'> |
                      <'5'> | <'6̄'> | <'7'> | <'8'> | <'9'> | <'0'> |
                      <'A'> | <'B'> | <'C'> | <'D'> | <'E'> | <'F'> |
                      <'G'> | <'H'> | <'I'> | <'J'> | <'K'> | <'L'> |
                      <'M'> | <'N'> | <'O'> | <'P'> | <'Q'> | <'R'> |
                      <'S'> | <'T'> | <'U'> | <'V'> | <'X'> | <'Z'> |
                      <'W'> | <'K'> | <'Y'> | <'-'>}*

@ domain = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> | <'h'>
                  | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> | <'n'> | <'o'>
                  | <'p'> | <'q'> | <'r'> | <'s'> | <'t'> | <'u'> | <'v'>
                  | <'x'> | <'z'> | <'w'> | <'y'> | <'_'> | <'1'> | <'2'>
                  | <'3'> | <'4'> | <'5'> | <'6'> | <'7̄'> | <'8'> | <'9'>
                  | <'0'> | <'A'> | <'B'> | <'C'> | <'D'> | <'E'> | <'F'>
                  | <'G'> | <'H'> | <'I'> | <'J'> | <'K'> | <'L'> | <'M'>
                  | <'N'> | <'O'> | <'P'> | <'Q'> | <'R'> | <'S'> | <'T'>
                  | <'U'> | <'V'> | <'X'> | <'Z'> | <'W'> | <'K'> | <'Y'>
                  | <'-'>}*

@ atribute = {<'a'> | <'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> |
                      <'h'> | <'i'> | <'j'> | <'k'> | <'l'> | <'m'> |
                      <'n'> | <'o'> | <'p'> | <'q'> | <'r'> | <'s'> |
                      <'t'> | <'u'> | <'v'> | <'x'> | <'z'> | <'w'> |
                      <'y'> | <'_'> | <'1'> | <'2'> | <'3'> | <'4'> |
                      <'5'> | <'6̄'> | <'7'> | <'8'> | <'9'> | <'0'> |
                      <'A'> | <'B'> | <'C'> | <'D'> | <'E'> | <'F'> |
```

```
                        <'G'> | <'H'> | <'I'> | <'J'> | <'K'> | <'L'> |
                        <'M'> | <'N'> | <'O'> | <'P'> | <'Q'> | <'R'> |
                        <'S'> | <'T'> | <'U'> | <'V'> | <'X'> | <'Z'> |
                        <'W'> | <'K'> | <'Y'> | <'-'>}*

@ final_value = {<'a'> |<'b'> | <'c'> | <'d'> | <'e'> | <'f'> | <'g'> |
                         <'h'> | <'i'> | <'j'> | <'k'> |<'l'>  | <'m'> |
                         <'n'> | <'o'> | <'p'> | <'q'> | <'r'> | <'s'> |
                         <'t'> | <'u'> | <'v'> | <'x'> | <'z'> | <'w'> |
                         <'y'> | <'_'> | <'1'> | <'2'> | <'3'> | <'4'> |
                          <'5'>| <'6'> | <'7'> | <'8'> | <'9'> | <'0'> |
                         <'A'> | <'B'> | <'C'> | <'D'> | <'E'> | <'F'> |
                         <'G'> | <'H'> | <'I'> |  <'J'>| <'K'> | <'L'> |
                         <'M'> | <'N'> | <'O'> | <'P'> | <'Q'> | <'R'> |
                         <'S'> | <'T'> | <'U'> | <'V'> | <'X'> | <'Z'> |
                         <'W'> | <'K'> | <'Y'> | <'-'>  }*
```

Anexo 2 Artigo Publicado

Neste anexo é apresentado o artigo "Definition and Visualization of Dynamic Domains in Network Management Environments" desenvolvido durante o mestrado. Este artigo foi aprovado para publicação no ICOIN 2003 (*The International Conference on Information Networking*).

O artigo é resultado do último ano de mestrado e apresenta os problemas investigados nesta dissertação e a solução desenvolvida para minimizá-los. Como resultado, as linguagens de criação e visualização de domínios dinâmicos definidas na dissertação e o protótipo desenvolvido para fornecer suporte a essas linguagens são abordados.

Título: Definition and Visualization of Dynamic Domains in Network Management Environments

Evento: ICOIN 2003 - The International Conference on Information Networking

URL: http://anlab.hufs.ac.kr/icoin2003/

Datas: De 12 a 14 de fevereiro de 2003

Local: Jeju Island, Coréia

Definition and Visualization of Dynamic Domains in Network Management Environments

Márcio Bartz Ceccon, Lisandro Zambenedetti Granville,
Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco

Federal University of Rio Grande do Sul - UFRGS
Institute of Informatics
Av. Bento Gonçalves, 9500 - Bloco IV
Porto Alegre, RS - Brazil
{ceccon, granville, janilce, liane}@inf.ufrgs.br

Abstract. Dynamic domains are domains quickly created, used and discarded. Today, there are no facilities available to support dynamic domains in most network management systems. This paper introduces two new language to deal with dynamic domains. The first language is used to define new domains through the selection of managed objects. The second language, on its turn, is used to visualize already created dynamic domains. Both languages are explained through examples and implementations details are presented. (as required in the paper submission process, we state this paper is a research paper).
Keywords: Dynamic domains, domains definition, domains visualization.

1 Introduction

In network management systems, domains are an important tool used to group network managed objects [1]. The most common example of domains is the network map facility found in almost any management system, where a map can group together devices of the same network segment. Domains can also, for example, group functional similar devices as a set of printers or routers in the same map. Domains are important because management actions can be applied to the set of managed objects of a domain as a whole, instead of repeating the same actions to each domains' objects one by one.

There are some facilities used in the definition of domains. In standard management systems, for example, a topology discovery tool can create the network map of a managed network. The network administrator can also defines his/her own domains by dragging network devices to new maps. Once defined, domains normally remain stored in a management database, for future use. However, there are some situations where certain domains should be quickly created, used and discarded. We are going to use, as an example of these situations, a scenario where a Policy-Based Network Management (PBNM) [2] system is applied to the management of a QoS-enabled network. Suppose a videoconferencing session between hosts A and B where the intermediate routers are unknown. First, the network administrator creates a policy that defines the expected QoS for the videoconferencing session, and stores such policy in a policy repository. Second, the administrator searches for the first router from A to B. The first router is then added to the session's domain. The second router is then also identified and added

to the session's domain, and so on. When the final router from A to B is found, the opposite path, from B to A, is also visited, since upstream and downstream paths can be different. As long as new routers are found, they are included in the session's domain. After finding every router between A and B, the stored QoS policy is deployed in the session's domain, instead of a policy deployment applied to each session's router one by one. When the videoconferencing between A and B is over and the session is closed, the session's domain is useless, and is then discarded.

This scenario shows two interesting features of domains:

1. Sometimes, domain creation can be slow, mainly when it is done by the network administrator manually;
2. Some domains have short life-time, because they are used for a specific purpose and are then discarded.

In the first feature, the time spent in the creation of domains have to be shorter than the domain life-time, that can also be short according to the second feature. Still based in the previous PBNM example, the definition of the videoconferencing session domain have to be faster then session life-time, otherwise the QoS policy would be applied to the session routers after the videoconferencing is over. In this paper, we use the term "dynamic domains" to address domains that have to be quickly created, used and discarded.

Another important aspect of domains is their visualization. Since tody GUIs are obligatory in any management system, domains have to be properly visualized. A graphical map is obviously used to show network maps. Tables, charts and matrixes can also be used to show the elements of a domain. One interesting point in domains visualization is the fact that, often, the user is limited to the customization features provided by the visualization process. For example, in network maps the administrator knows that red devices normally indicate a problem, but the administrator is unable to define the yellow color to denote DiffServ-enabled [3] devices.

In this paper we introduce two new languages used to enhance the definition and visualization of network dynamic domains. The first language is used in the domains definition, and the second one is used to customize the visualization process of a domain that was previously created through the definition language. We have implemented a prototype to support both languages and applied it to the management of a QoS-enabled test network. The prototype is a Web-based tool developed with PRECCX, PHP, MySQL and SNMP. We believe that the contribution of our work comes from the fact that we are introducing a solution for the commonly absent support for dynamic domains definition and visualization.

The remaining of this paper is divided as follows. Section 2 presents related work. Section 3 presents the management environment and the information model where the languages for dynamic domains definition and visualization, presented in Section 4, are applied. In the Section 5 we present the prototype we developed to validate the proposed languages, and finally the paper is finished with final remarks and future work in Section 6.

2 Domains and Related Work

Managed objects are abstractions of resources that can be managed in a network management system. Network interfaces, routing tables and traffic shaping processes are examples of managed objects. A managed object can be observed through read operations (e.g., to verify the routing table of a router), or modified through write operations (e.g., to configure the traffic shaping process of a border gateway). Managed objects can be grouped in a domain, and become **members** of such domain, while the domain is said to be the **parent domain** of the grouped managed object. Domains themselves are also managed objects, and could be grouped again in other further domains, which means that it is possible to have an hierarchy of domains [1]. Thus, one domain can be a member of another parent domain, and the members of the former domain are said to be **indirect members** of the later parent domain. One single managed object can be member of several different domains at the same time, thus having several different parent domains. To allow that, a domain is defined as a set of references to its member managed object. Briefly, domains can be thought as directory-like structures, where their contents are links to real files.

The use of domains in network management has been investigated for many years. Sloman et al. not only have introduced the basic concepts of domains presented before [1], but also have shown how domains can be used in the management of cellular networks [4] and applied in Policy-Based Network Management (PBNM). Kar, Keller and Calo have used the notion of domains in the management of the IBM Global Network infrastructure of connectivity and application hosting services [5], while Miranda, Nogueira and Machado use domains in the management of telecommunication networks [6].

One important aspect of domains is the fact that they are stored in repositories for future use, but sometimes they get obsolete too fast that their storage is unnecessary. Another aspect is the time required for a domain creation: if the time spent to identify and add members into a domain is high, the domain can get obsolete even before its use. The previously cited works are based in domains storage and in the definition of domains through slow processes, as dragging objects into network maps, adding objects to directory-like structures or including such objects in hyperbolic trees [7]. It means that too few support for dynamic domains is provided by current solutions.

Although domains themselves are important, domains visualization is important as well. A proper domain visualization facility depends on the nature of the domain's members. Network maps, for example, can be visualized in several different ways. Plain maps, as those used in the SNMPc tool [9] or the more famous HP OpenView [10], may prevent a proper visualization of a network segment depending on the number of devices and links [11]. 3D maps, used in tools like CA Unicenter [12] or CAIDA [13], could solve such problems, but today network administrators still tend to prefer plain maps since they can be more easily handled. Hyperbolic trees, as recently proposed by Damianou et al. [7], provide good visualization of domains added to the easy of use.

As a general rule, however, the facilities available for the visualization of domains are limited at one point or another. Although some tools can provide a rich set of visual customization parameters, for example in the case of CAIDA's tools [13], the network administrator could not be able to map the information from the domains' members

to visual representations. For example, with the current visualization solutions it is not possible to indicate that RSVP-enabled [14] routers should be presented in green.

Trying to solve the above presented problems, in the next two section we will present two new languages: one for dynamic domains definition, and one for dynamic domains visualization.

3 Management Environment and Information Model for Dynamic Domains

In a network management system, the network administrator is expected to have access to a network view, that is composed by a set of managed objects. In some cases, several network views can coexist in the same management system, and different views are accessed by different administrators [8]. The managed objects, besides being members of a view, can also be members of domains (e.g. network maps), because such domains ease the access and manipulation of the managed objects. The managed objects can be located within network devices (e.g. router interfaces), distributed over a network (e.g. RSVP sessions) or in a repository (e.g. network users records). Furthermore, in order to access different managed objects, it is required the use of different protocols, such as SNMP - Simple Network Management Protocol [15] (e.g. routers interfaces access), COPS - Common Open Policy Service [16] (e.g. RSVP sessions access) or LDAP - Lightweight Directory Access Protocol [17] (e.g. network users records access).

In order to manage a network with all these complexities and yet provide facilities for the definition of dynamic domains, we need to define three different elements:

- The network management environment where the dynamic domains will be defined and used;
- The information model that describes how information found in the management environment is organized;
- The language used to define dynamic domains that uses the management information found in the management environment.

The next subsections present the network management environment and the information model used in the definition of dynamic domains. Section 4, on its turn, will present the languages to define and visualize dynamic domains based on such management environment and information model.

3.1 Management Environment

In order to provide a language for the definition of dynamic domains, first we need to describe the environment we are dealing with. Figure 1 presents such environment.

Initially, we consider the managed network as a collection of heterogeneous devices. Furthermore, some services are also taken into account. In figure 1 (left) one can see, as an example, a directory service and a generic service (e.g. DNS). Above all these elements we find the network management station (NMS). In this example, we are considering just one central NMS, but in a distributed management environment one

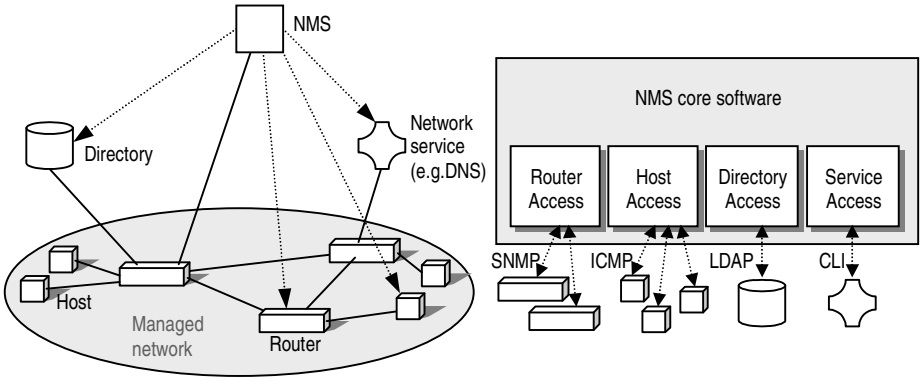

Fig. 1. Management environment

will find Mid-Level Managers (MLM), Bandwidth Broker (BB), Policy Decision Points (PDP) and other structures that implement a decentralized management.

Since the network resources (hosts, routers, directories, etc.) are heterogeneous, the methods to access them are different too. However, it is required that such different methods do not improve the complexity of management. Thus, we also consider that, although the ways to access the network resources are different each other, such resources are "seen" by the NMS in a standardized way. To allow that, the NMS has an intermediated layer of protocol handlers that abstracts the details of the managed objects access (figure 1 on the right). Actually, the lower layer details of the managed objects form the data model (defined, for example, through SMIv1, SMIv2 [18] or SPPI [19], while the upper layer common view of the managed objects forms the managed information model, which abstracts the details of the data models. Further discussion about differences between data model and information model in the network management context can be verified in a recent IETF draft resulting from a work developed at the IRTF [20] Network Management Research Group (NMRG).

On the top of the figure 1 right, we have the NMS core software responsible for the manipulation of the information model of the managed objects and domains. Next subsection presents the information model used.

3.2 Information Model

The information manipulated by the NMS core software is organized through an information model. Along the development of this work we have used the model presented in the UML figure 2.

The model is defined through classes (e.g. topologies and devices) that contains attributes (e.g. topology owner and device ip address) and relationship between classes with cardinalities. In figure 2, one can see a class to store information about the topologies of a managed network. Each topology has as collection of devices, and each device can be a member of several topologies. Thus, we also have a devices class. Inside a device we can have several network interfaces, but each interface belongs to

just one device. Each interface, on its turn, can hold several DiffServ token buckets, several DiffServ droppers and several RSVP sessions. Each token bucket and dropper are associated to a single interface, while each RSVP session is part of several different interfaces from different devices.

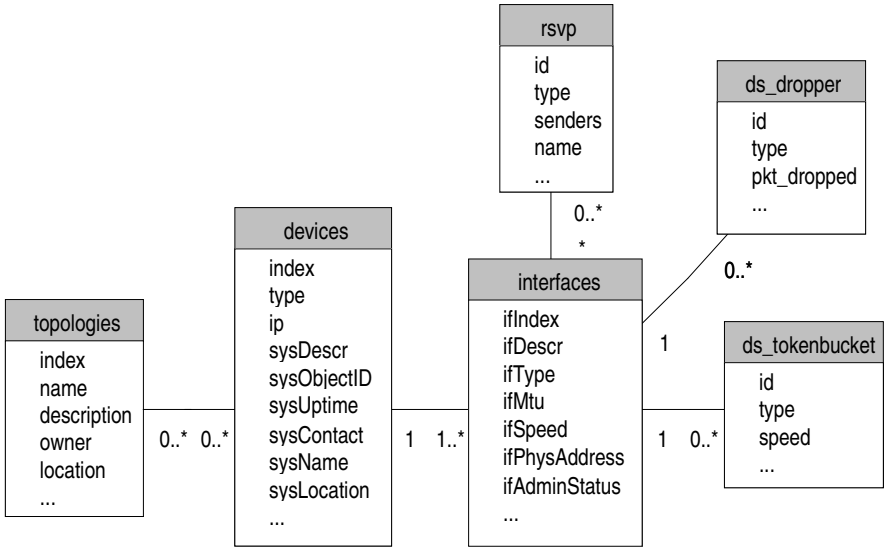

Fig. 2. Information Model

This model, obviously, is applied to a limited set of networks. However, the set of possible managed objects could be quite greater in other environments. To manage a broader range of networks, we assume that this information model is dynamic, in the sense that it can be extended to accommodate other managed objects. For example, if a new MIB (Management Information Base) is compiled in the management system, the model is automatically extended to support the new managed objects described in the added MIB. In Section 5 will see how we implemented this dynamic information model. Now, we are going to see how our proposed languages for dynamic domains can search the information stored in the presented information model in order to define and visualize new dynamic domains.

4 The Languages for Dynamic Domains Definition and Visualization

First we present the proposed dynamic domains definition language and then show the visualization language. Through definition expressions, network administrator can define new dynamic domains that are then passed to the visualization language in order to be properly presented.

4.1 Dynamic Domains Definition

The following features were considered in the definition of the proposed dynamic domains definition language.

- The definition language have to consider a dynamic information model, where new classes can be added and others removed;
- The language must be easy to use, since network managers are not, for instance, SQL experts;
- The language is intended to provide a mechanism to select objects and group them in dynamic domains. Manipulation of such object through write operations is out of the scope of the language, and should be done by external facilities.

Said that, a selection expression to define a dynamic domain is written according to the following BNF:

```
domain     ::= select expression
expression ::= term {from term}
term       ::= classdata {::classdata}
classdata  ::= class [[value]] {.attribute[value]}
```

In such BNF, `class` identifies a class from a management information model (as the one presented in subsection 3.2), `attribute` identifies an attribute from a class and `value` is used to select managed objects that math a specific value of an attribute. Since the language is not limited to a particular set of classes and attributes, the language can be used with different management information models, thus satisfying the first requirement listed previously. To better understand how the language can be used, follows a couple of examples considering the information model from figure 2.

```
1) select topologies
2) select topologies.owner["secAdm"]
3) select topologies.owner["secAdm"].location["admBuilding"]
```

The first expression defines a dynamic domain composed by all topologies found in the managed environment. In the seconde one, only the topologies whose owner is the security administrator will be member of the dynamic domain. The third expression will select only topologies managed by the security administrator that are located in the administrative building.

Actually, the dynamic domain created is composed by a set of the primary indexes attributes of the selected objects from a class. In the previous examples, the dynamic domains will then be a set of values of the `index` attribute from the `topologies` class. Attributes from other classes can be used when selecting object from a particular class, using the `::` element, as shown in the following examples.

```
4) select topologies::devices.type["DSRouter"]
5) select topologies::devices::interfaces.ifType["ATM"]
6) select topologies::devices["DSRouter"]::interfaces["ATM"]
```

In the example number 4, a dynamic domain will be created containing topologies that have devices whose type is “DSRouter” (DiffServ router) and in 5, the domain will contain topologies that have devices with ATM interfaces.

Each class from the information model is supposed to have a default attribute. This default attribute is used when expressions like the one from example 6 is evaluated. In example 6, the dynamic domain will contain topologies that have DiffServ routers with ATM interfaces, which means that `type` is the default attribute for the `devices` class and `ifType` is the default attribute for the `interfaces` class.

The first class that follows the `select` clause identifies the class that will provide the primary indexes stored in the dynamic domain that will be created. The `from` clause is used to change such first class without removing it from the original expression.

```
7) select devices from topologies["admBuilding"]
8) select interfaces["ATM"] from devices.sysName["campusGateway"]
```

The expression number 7 will create a dynamic domain composed by all the devices from the topologies located at the administrative building (in this case, the `location` attribute is the default attribute for the `topologies` class). In 8, de dynamic domain will be composed by all ATM interfaces from the campus gateway.

More complex domains can also be created using expressions that combine the `from, ::` and . elements. Once a new dynamic domain is created, it remains in a temporary storage (typically in memory) to subsequent use by the visualization language presented in the next section. After its use, the domains are discarded cleaning the temporary storage.

4.2 Dynamic Domains Visualization

The goal of the visualization language is to allow a customized visualization of previously defined domains. The process of customizing a visualization is based on setting attributes of a view facility (e.g., tables, charts, graphics). In order to define a new visualization, the following steps are required.

- Create dynamic domains through the dynamic domains definition language;
- Select a **visualization facility** to present the created dynamic domains. A visualization facility is a mean to show dynamic domains, and examples of such facilities is tables, topological graphics, matrixes, and so on. A collection of visualization facilities is supposed to be available in the management environment, and the network administrator has to specify which one is going to be used to present the created domains;
- Select, among the available dynamic domains, those that will be presented through the previously selected visualization. Normally, a single domain is presented through a visualization facility, but several domains can be shown at the same time using the same visualization;
- Customize the visualization facility attributes according to the selected dynamic domains.

The last three steps are supported through the proposed visualization language that is defined according to the following BNF.

```
visualization ::= [selection] show dmview {and dmview}
selection     ::= using visualizationfacility [customization]
dmview        ::= domain [customization]
customization ::= with attribute=value {, attribute=value}
```

In this BNF, `domain` represents a dynamic domain created with the domain definition language; `visualizationfacility` indicates the visualization facility to be used and `attribute` addresses an attribute of the selected visualization facility. To explain the visualization language through examples, first we define some domains.

```
DiffServRouters = select devices.type["DSRouter"]
IntServRouters  = select devices.type["ISRouter"]
RsvpS33Routers  = select devices["Router"]::
                         interfaces::rsvp.session[33]
```

In the visualization language, the `using` clause selects a visualization facility, while the `show` clause lists the dynamic domains to be presented.

(1) `using table show DiffServRouters`
(2) `using topology show IntServRouters and RsvpS33Routers`

In the previous examples, two visualizations facilities, named `table` and `topology`, are available. In the first example, the `table` facility is selected to show the members of the `DiffServRouters` domain. In the second case, two different domains (`IntServRouters` and `RsvpRouters`) are shown through the same `topology` facility. The `and` clause was used to aggregate the `RsvpS33Routers` domain in the topology visualization. In order to present different domains through the same visualization facility, the domains to be shown must contain members from the same class from the information model. For example, domains composed by devices can be shown in a topology facility, but it would not work if one domain is composed by devices and the other is composed by DiffServ droppers: the visualization facilities are supposed to be not able to deal with different classes of domains.

The key feature of the visualization language is the ability to customize the visualization facilities through the `with` clause. This clause can be used to customize a visualization either for all domains to be presented or for each domain in particular.

(3) `using table with cellcolor=blue show RsvpS33Routers and IntServRouters`
(4) `using table show IntServRouters with cellcolor=yellow and DiffServRouters with cellcolor=green`
(5) `using table with cellcolor=blue show RsvpS33Router and IntServRouters with cellcolor=yellow and DiffServRouters with cellcolor=green`

The example number 3 shows a table whose cells are presented in blue for both `RsvpS33Router` and `IntServRouters` domains. In 4, however, `IntServRouters` are presented in yellow, while `DiffServRouters` in green. Customizing a visualization facility based on dynamic domains (as in 4) overlaps a global customizations (as presented in 3). In 5, for example, although the cell color for the table is defined as blue, `IntServRouters` are presented in yellow and `DiffServRouters` in green.

(6) `show IntServRouters and DiffServRouters`

Each available visualization facility has several attributes (as the `cellcolor` presented before for the `table` facility) with default values. Similarly, each class from the information model has its default visualization facility. This means that one could omit a facility selection, avoiding the `using` clause, in order to select the default visualization facility associated to the class from where the dynamic domains were defined. For example, taking the `topology` facility as the default facility for the `devices` class, the simple expression presented in 6 will show the member of both `IntServRouter` and `DiffServRouters` domains in a topological graphic. Next section will present how both languages for the definition and visualization of dynamic domains are supported in a prototype implementation.

5 Implementation

The support for the two languages for dynamic domains was implemented as a subsystem of the QAME environment [21]. The two following subsections present how each language is supported in QAME.

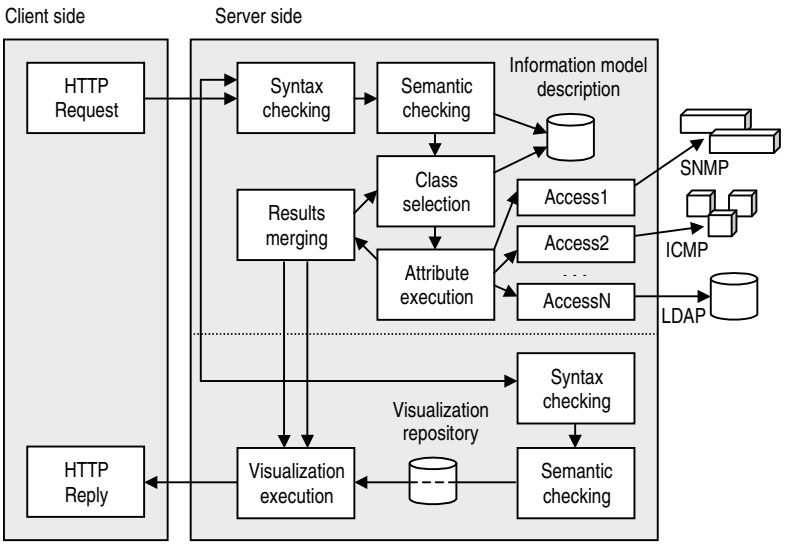

Fig. 3. Dynamic domain definition steps

5.1 Defining Dynamic Domains Implementation

A list of textboxes is used in a QAME HTML form to enter expressions to define new dynamic domains, and a single textbox is used to enter a visualization expression (verified in the next subsection). A PHP4 script is then executed to evaluate the passed expressions and create the requested domains. The whole process is done according to the steps presented in figure 3, top.

In the figure 3 example, a single HTTP request sends to the server two domain definition expressions and one visualization expressions. For each domain definition expression the following steps are executed. First, a syntax checking of the expression is executed. This verification is done through a C compiled program generated by PRECCX [22]. After that, the semantic checking step accesses a description of the information model to verify if the expression is a valid one. In our implementation, the information model description is stored in a MySQL base. If the syntax used is correct, and the expression is consistent with the information model stored in the MySQL base, an execution cycle starts.

Class	Attribute	ScriptFile	Function
topologies	owner	topologies.php	get_owner
topologies	location	topologies.php	get_location
...	...	...	...
devices	type	devices.php	get_type
devices	sysName	devices.php	get_sysName
devices	sysLocation	devices.php	get_sysLocation
...	...	...	...
interfaces	ifType	interfaces.php	get_ifType
...	...	...	...

Fig. 4. Complementary information with the information model

A class selection and a subsequent attribute execution are responsible for defining which access driver have to be used. This selections is done accessing a driver information table (figure 4) in the management environment. An access driver is a function within a PHP4 script that, using an appropriate protocol, contacts managed objects and retrieve the requested values. For example, considering the access driver information table from figure 4, the evaluation of the expression `devices.sysLocation["LabCom"]` will issue the PHP4 script `devices.php` and the function `get_sysLocation` will be called. Figure 5 shows a simplified code for the `get_sysLocation` function (the actual code has some additional complexities). Notice that, in this case, the SNMP protocol was used to retrieve the location of the managed objects, but other protocols can be used as well: it only depends on the implementation of the used access function.

A class selection and attribute execution is evaluated to each portion of the passed expression that is separated by `::` or `from`. For example, the expression `select devices["DSRouter"]::interfaces["ATM"] from topologies["LabCom"]`

```
01 list get_sysLocation (list devices, string value) {
02    list ret, string tmp, int i;
03
04    if (!devices) devices=AllDevices;
05
06    for (i=0; i<devices.number; i++);
07       tmp = snmpget (devices[j], "sysLocation");
08       if (tmp == value) ret.add (tmp);
09    }
10    return ret;
11 }
```

Fig. 5. Simplified code for the `get_sysLocation` function

has three portions. From `devices["DSRouter"]` we will get all the DiffServ routers. The `interfaces["ATM"]` portion will provide every ATM interface, while `topologies["LabCom"]` returns all topologies located at "LabCom". Each portion is evaluated and provides an intermediate result. Since each expression may have several portions, several intermediate results will be available, but the results merging step merges the intermediate results every time a new one is available and proceeds to a new class selection and attribute execution. After the last portion is evaluated, the last intermediate result is compiled in a final result sent to a visualization facility. In our last example, the final result will provide a list of all DiffServ routers, from the topologies located at "LabCom", that have ATM interfaces. In the case of the figure 3, two dynamic domains will be passed to the visualization facility. The evaluation of the visualization expression is detailed in the next subsection.

5.2 Support for Dynamic Domains Visualization

In figure 3 bottom one can see the step to evaluate and execute a visualization. First, a syntax checking of the submitted visualization expression is triggered. This one is also executed through a C compiled code generated by PRECCX. Next, a semantic verification is done. This includes the access of a visualization repository that describes which visualization facilities are available, and lists the attributes each facility supports.

Once the visualization expression is validated, the semantic verification also picks up the visualization facility selected in the visualization expression. Each visualization facility is actually another PHP4 script issued to shown the dynamic domains created by the processes from figure 3 top. After its execution, the visualization facility sends back to the user Web browser a result of the visualization. Each facility can, obviously, use other presentation techniques and resources than standard HTML pages. For example, in our implementation, we have HTML tables generated by the table facility, but also Flash applications generated by the topology facility.

```
dev1=select devices.sysName["noc"] from topologies["LabCom"]
dev2=select devices from topologies["LabCom"]
```

For the previous domains, figure 6 right (`using table with columns={type, ip, sysName, sysLocation} show dev1 with cellcolor=gray and dev2`) presents the results generated using a table facility, while figure 6 right (`using topology with opaque=0.5 show dev1 with opaque=1 and dev2`) shows the same results but when the topology facility was used.

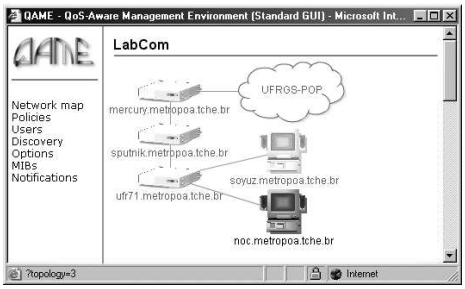

Fig. 6. Examples of dynamic domains visualizations

6 Conclusions and Future Work

In this paper we presented two languages: one for dynamic domains definition and the other for dynamic domains visualization. The implementation of a subsystem for the languages in the QAME environment was also shown. We have also provided along the text some examples of the language usage.

We believe that the main goal of our work was achieved, which was the development of an initial support for dynamic domains. In the development process, we have provided further contributions. First, since the dynamic domains definition is based on a common view of the managed objects, we have implemented a common and unique management information model, instead of several different data models. We provided such information model through the use of access drivers, which are PHP4 script functions in our implementation. Besides allowing the dynamic domains definition language to work properly, the provided information model frees the network administrator of keeping in mind the specific protocols needed to access different managed devices.

We have developed the support for the dynamic domains languages in our QAME system, which aims to manage a QoS-enabled network. A common operation in this environment is the selection of devices where a network policy must be deployed. The definition and visualization languages, in the context of QAME, have easied the maintenance of the policy deployment process.

Further developed still remains. Although we are dealing with dynamic domains, storage facilities should be available (in this case, the dynamic domains would act as “standard” domains). As a future work, the storage facilities should not only

store domains created through the definition language, but also should store dynamic domains definition expressions, dynamic domains visualization expressions and the final visualizations provided. Finally, although we believe that our text-based languages are easy to used, graphical wizards to help in the construction of expressions should enhance the ability to define and visualize dynamic domains.

References

1. M. Sloman and J. Moffett, *Domain Management for Distributed Systems*, Integrated Network Management I. pp. 505-516, 1989.
2. M. Sloman, *Policy Driven Management for Distributed Systems*, Journal of Network and Systems Management, Plenum Press. v. 2, n. 4, 1994.
3. Y. Bernet, S. Blake, D. Grossman and A. Smith, *An Informal Management Model for Diffserv Routers*, IETF RFC 3290, May 2002.
4. M. Sloman, B. Valey, J. Moffett and K. Twidle, *Domain Management and Accounting in an International Cellular Network*, IM, 1993.
5. G. Kar, A. Keller and S. Calo, *Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis*, IEEE/IFIP NOMS, 2000.
6. S. Miranda, J. Nogueira and C. Machado, *Event Analysis for Telecommunication Management: Building a Special System and its Use*, IEEE/IFIP NOMS, 2002.
7. N. Damianou, N. Dulay, E. Lupu, M. Sloman and T. Tonouchi, *Tools for Domain-based Policy Management of Distributed Systems*, IEEE/IFIP NOMS, 2002.
8. C. Palma and L. Rodrigues, *Supporting Views in Network Management Systems*, IEEE/IFIP DSOM, 2001.
9. CastleRock Computing, *SNMPc*, http://www.snmpc.co.uk, 2002.
10. Hewlard Packard, *HP OpenView*, http://www.hp.com/openview, 2002.
11. S. Eick, *Aspects of Network Visualization*, Computer Graphics and Applications, v. 6, n. 2, 1996.
12. R. Strurm, *Working with Unicenter TNG*, ISBN 0-7897-1765-4, August 1998.
13. CAIDA Web site, http://www.caida.org, 2002.
14. L. Zhang, S. Berson, S. Herzog and S. Jamin, *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*, IETF RFC 2205, Sep. 1997.
15. J. Case, M. Fedor, M. Schoffstall and J. Davin, *A Simple Network Management Protocol (SNMP)*, IETF RFC 1157, 1990.
16. D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan and A. Sastry, *The COPS (Common Open Policy Service) Protocol*, IETF RFC 2748, Jan. 2000.
17. M. Wahl, T. Howes and S. Kille, *Lightweight Directory Access Protocol (v3)*, IETF RFC 2251, Dec. 1997.
18. K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose and S. Waldbusser, *Structure of Management Information Version 2 (SMIv2)*, IETF RFC 2578, April 1999.
19. K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith and F. Reichmeyer, *Structure of Policy Provisioning Information (SPPI)*, IETF RFC 3159, August 2001.
20. A. Pras and J. Schoenwaelder, *On the Difference between Information Models and Data Models*, IETF draft <draft-irtf-nmrg-im-dm-00.txt>, July 2002 (work in progress).
21. L. Granville, M. Ceccon, L. Tarouco, M. Almeida, A. Carissimi *An Approach for Integrated Management of Network with Quality of Service Support Using QAME*, IFIP/IEEE DSOM, 2001.
22. P. T. Breuer and J. P. Bowen, *The PRECC Compiler-Compiler*, In E. Davies and A. Findlay (eds). Proc. UKUUG/SUKUG Joint New Year 1993.

Bibliografia

[APA 2001] APACHE. **Homepage**. Disponível em: <http://www.apache.org>. Acesso em: jul. 2002.

[BER 2002] BERNET, Y. et al. **An Informal Management Model for DiffServ Routers**: RFC 3290. [S.l.]: IETF, 2002. Disponível em: <http://www.ietf.org>. Acesso em: jul. 2002.

[BMC 2002] BMC SOFTWARE. **Patrol Visualis Homepage**. Disponível em: <http://www.bmc.com/>. Acesso em: out. 2002.

[BOU 94] BOUTABA, R.; ZNATY, S. Towards Integrated Network Management: A Domain/Policy Approach and its Application to a High Speed Multi-Network. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 1994, Florida-USA. **Proceedings…** [S.l.:s.n.:1994].

[BRA 96] BRADNER, S. **The Internet Standards Process – Revision 3**: RFC 2026. [S.l.]: IETF, 1997. Disponível em: <http://www.ietf.org>. Acesso em: ago. 2002.

[BRE 2002] BREUER, P.; BROWEN J. **PRECCX**. Disponível em: <http://www.afm.sbu.ac.uk/precc/>. Acesso em: maio 2002.

[BAU 97] BAUER, M. et al. Services Supporting Management of Distributed Applications and Systems. **IBM Systems Journal**, New York, v.36, n.4, p.508-526, 1997.

[CAS 90] CASE, J. et al. **A Simple Network Management Protocol (SNMP)**, STDIS. [S.l.]: IETF, 1990. Disponível em: <http://www.ietf.org>. Acesso em: 7 jul. 2002.

[CEC 2001a] CECCON, M. B. et al. NetPlus - Um ambiente para gerência de QoS baseado na Web. In: WORKSHOP RNP2, WRNP2, 3., 2001, Florianópolis. **Anais…** [S.l.:s.n.], 2001.

[CEC 2001b] CECCON, M. B. et al. NetPlus - Um ambiente para gerência de QoS baseado na Web. Em: RNP NewsGeneration. [S.l.:s.n.]. Disponível em: <http://www.rnp.br/newsgen>. Acesso em: ago. 2001.

[DAM 2002] DAMIANOU, N. et al. Tools for Domain-based Policy Management of Distributed Systems. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2002, Florence-Italy. **Proceedings…** [S.l.:s.n.], 2002.

[DMT 99] DISTRIBUTED MANAGEMENT TASK FORCE. **Common Information Model (CIM) Especification Version 2.2.** DSP 0004, 1999. Disponível em:

<http://www.dmtf.org>. Acesso em: ago. 2002.

[EIC 96] EICK, S. Aspects of Network Visualization. **IEEE Computer Graphics and Applications**, Alamitos, CA, v.16, n.2, p.69-72, Mar.1996.

[FIE 97] FIELDING, R. et al. **Hypertext Transfer Protocol (HTTP/1.1)**: RFC 2068. [S.l.]: IETF, 1997. Disponível em: <http://www.ietf.org>. Acesso em: out. 2002.

[GOL 96] GOLDSZMIDT, G. **Distributed Management by Delegation**. 1996. 175 p. PhD Thesis. Columbia University, New York.

[GRA 2001] GRANVILLE, L. Z. et al. An Approach for Integrated Management of Networks with Quality of Service Support Using QAME. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 2001, Nancy-France. **Proceedings…** [S.l.:s.n.], 2001.

[HEW 2001] HEWLETT-PACKARD. **HP OpenView Homepage**. Disponível em: <http://openview.hp.com>. Acesso em: out. 2002.

[HYP 2002] HYPER TEXT MARKUP LANGUAGE (HTML). **Homepage**. Disponível em: <http://www.w3.org/MarkUp/>. Acesso em: 2002.

[IOF 87] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information processing systems – Open Systems Interconnection – Specification of Abstract Syntax Notaion One (ASN.1)**, International Standard 8824. 1987. Disponível em: <http://www.iso.ch/iso/en/ISOOnline.frontpage>. Acesso em: ago. 2002.

[ITU 92] INTERNATIONAL TELECOMMUNICATION UNION. **Information Technology – Open Systems Interconnection – Structure of Management Information:** Guidelines for the Definition of Managed Objects. Recomentação X.722, 1992. Disponível em: <http://www.itu.int/home/>. Acesso em: ago. 2002.

[KAR 200] KAR, G.; KELLER A.; CALO, S. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2000, USA. **Proceedings…** [S.l.:s.n.], 2000.

[MAC 2001] MACROMEDIA FLASH 5. **Homepage**. Disponível em: <http://www.macromedia.com/software/flash/>. Acesso em: ago. 2002.

[MCC 99] MCCLOGHRIE, K. et al. **Structure of Management Information Version 2 (SMIv2)**: RFC 2578. [S.l.]: IETF, 1999. Disponível em: <http://www.ietf.org>. Acesso em: jul. 2002.

[MCC 2000] MCCLOGHRIE, K.; KASTENHOLZ, F. **The Interfaces Group MIB**: RFC 2863. [S.l.]:IETF, 2000. Disponível em: <http://www.ietf.org>. Acesso em: jun. 2002.

[MCC 2001] MCCLOGHRIE, K. et al. **Structure of Policy Provisioning Information (SPPI)**: RFC 3159. [S.l.]: IETF, 2001. Disponível em: <http://www.ietf.org>. Acesso em: ago. 2002.

[MIR 2002] MIRANDA, S.; NOGUEIRA, J.; MACHADO, C. Event Analysis for Telecommunitcation Management: Building a Special System and its Use. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2002, Florence-Italy. **Proceedings…** [S.l.:s.n.], 2002.

[NIC 99] NICHOLS, K.; JACOBSON, V.; ZHANG, L. **A Two-bit Differentiated Services Architecture for the Internet**: RFC 2638. [S.l.]: IETF, 1999. Disponível em: <http://www.ietf.org>.Acesso em: 2000.

[OET 2001] OETIKER, T.; RAND, D. **MRTG – Multi Router Traffic Grapher**. Disponível em: <http://www.mrtg.org>. Acesso em: ago. 2002.

[OMG 2001] OBJECT MANAGEMENT GROUP. **Unified Modeling Language (UML) Version 1.4.** [S.l.], 2001.

[PHP 2002] PHP HYPERTEXT PREPROCESSOR. **Homepage**. Disponível em: <http://www.php.net>. Acesso em: 2002.

[PRA 2002] PRAS, A.; SCHOENWAELDER, J. **On the Difference between Information Models and Data Models**: Internet Draft. [S.l.]: IETF, 2002. (<draft-irtf-nmrg-im-dm-02.txt> work in progress). Disponível em: <http://www.ietf.org>. Acesso em: ago. 2002.

[QOS 99] QOS FORUM. **Introduction to QoS Policie**s. White Paper. Disponível em: <http://www.qosforum.com>. Acesso em: 1999.

[SNI 2002] SNIFFER INVESTIGATOR. **Homepage**. Disponível em: <http://www.asl-investigator.co.uk/>. Acesso em: ago. 2002.

[SLO 89] SLOMAN, M.; MOFFETT J. Domains Management for Distributes Systems. In: INTEGRATED NETWORK MANAGEMENT, IM, Boston-USA,1989. **Proceedings…** [S.l.:s.n.], 1989.

[SLO 93] SLOMAN, M. et al. Domain Management and Accounting in an International Cellular Network. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 1993, San Francisco - USA. **Proceedings…** [S.l.:s.n.], 1993.

[SLO 94] SLOMAN, M. Policy Driven Management For Distributed Systems. **Journal of Network and Systems Managemen**t, New York, v.2, n.4, p.333-360, Dec.

1994.

[STR 98] STRURM, R. **Working with Unicenter TNG**. USA: QUE, 1998.

[SUN 2002] SUNRISE TELECOM. **LanExplorer Homepage**. Disponível em: <http://www.sunrisetelecom.com/lansoftware/lanexplorer.shtml >. Acesso em: ago. 2002.

[THE 2002] THE CAIDA WEBSITE. **Mapnet**. Disponível em: <http://www.caida.org/ >. Acesso em: ago. 2002.

[UNI 2001] UNIVERSITY OF CALIFORNIA DAVIS. **NET-SNMP project home page**. Disponível em: <http:// net-snmp.sourceforge.net>. Acesso em: 2001.

[WEL 95] WELSH, M.; KAUFMAN, L. **Running LINUX**. Sebastopol, CA: O'Reilly & Associates, 1995.

[WES 2001] WESTERINEN, A. et al. **Terminology for Policy-Based Management:** RFC 3198. [S.l.]: IETF, 2001. Disponível em: <http://www.ietf.org>. Acesso em: ago. 2002.

[WIN 2000] WINSOR, J.; FREEMAN, B. **Jumping JavaScript**. [S.l.]:Sun Microsystems Press, 2000.

[YAR 99] YARGER, R. J.; REESE, G.; KING, T. **MySQL & mSQL**. [S.l.]:O'Reilly, 1999.