Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Ciência da Computação

APLICAÇÃO DO MODELO TMN NA GERÊNCIA DE REDES DE ALTA VELOCIDADE

KETTER OHNES ROGERIO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da UFSC como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

FLORIANÓPOLIS - SC, AGOSTO DE 1999.

APLICAÇÃO DO MODELO TMN NA GERÊNCIA DE REDES DE ALTA VELOCIDADE

KETTER OHNES ROGERIO

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, na área de concentração Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Prof. Carlos Becker Westphall, Dr. – Orientador

Prof. Jorge Muniz Barreto, Dr. – Coordenador

Banca Examinadora:

Prof. Carlos Becker Westphall, Dr. (CPGCC-UFSC)

Profa. Silvia Modesto Nassar, Dra. (CPGCC-UFSC)

Prof. João Bosco da Mota Alves, Dr. (CPGCC-UFSC)

Prof. João Bosco Mangueira Sobral, Dr. (CPGCC-UFSC)

Florianópolis, 9 de agosto de 1999.

ÍNDICE

LISTA DE ABREVIATURAS

ASN.1	*Abstract Syntax Notation 1*
ATM	*Asynchronous Transfer Mode*
CLP	*Cell Loss Priority*
CMIP	*Common Management Information Protocol*
CMIS	*Common Management Information Service*
DCN	*Data Communications Network*
FDDI	*Fiber Distributed Data Interface*
GDMO	*Guidelines for the Definition of Managed Objects*
IAB	*Internet Activities Board*
ILMI	*Integrated Local Management Interface*
ISO	*International Organization for Standardization*
ISODE	*ISO Development Environment*
ITU-T	*International Telecommunications Union – Telecommunication Standardization Sector*
LAN	*Local Area Network*
LLA	*Logical Layered Architecture*
MAN	*Metropolitan Area Network*
MD	*Mediation Device*
MF	*Mediation Function*
MIB	*Management Information Base*
NE	*Network Element*
NEF	*Network Element Function*

NEL	*Network Element Level*
NEML	*Network Element Management Level*
NML	*Network Management Level*
OAM	*Operation, Administration and Maintenance*
OCS	*OpenCon Systems*
OID	*Object Identifier*
OS	*Operation System*
OSF	*Operations Systems Function*
OSI	*Open Systems Interconnection*
OSIMIS	*OSI Management Information Service*
QA	*Q Adapter*
QAF	*Q Adapter Function*
QoS	*Quality of Service*
RDN	*Relative Distinguished Name*
RDSI-FE	Rede Digital de Serviços Integrados – Faixa Estreita
RDSI-FL	Rede Digital de Serviços Integrados – Faixa Larga
RFC	*Request for Comments*
SCR	*Sustainable Cell Rate*
SMK	*Shared Management Knowledge*
SNMPv1	*Simple Network Management Protocol version 1*
SNMPv2	*Simple Network Management Protocol version 2*
TCP/IP	*Transport Control Protocol/Internet Protocol*
TL1	*Transaction Language 1*
TMN	*Telecommunications Management Network*

UNI	*User Network Interface*
VC	*Virtual Channel*
VCC	*Virtual Channel Connection*
VCL	*Virtual Channel Link*
VP	*Virtual Path*
VPC	*Virtual Path Connection*
VPL	*Virtual Path Link*
WAN	*Wide Area Network*
WS	*Work Station*
WSF	*Work Station Function*

RESUMO

A expansão das redes ATM (*Asynchronous Transfer Mode*) e a crescente integração dos serviços oferecidos têm aumentado a complexidade da gerência de redes. A estes fatos, adicionam-se o crescimento do mercado de telecomunicações e o aumento do número de fabricantes de equipamentos e provedores de serviços de telecomunicação. Consequentemente, a tarefa de gerenciamento exige uma arquitetura com um modelo de informação abrangente e flexível, capaz de representar os diversos tipos de recursos e conexões existentes no ambiente, com um nível maior ou menor de abstração. As redes apresentam ainda uma diversidade de arquiteturas, entre as quais se destaca a arquitetura *Internet*. Devido a simplicidade deste modelo, grande parte dos recursos ATM existentes são providos com bases de informação de gerenciamento (MIBs - *Management Information Bases*) compatíveis com o protocolo de gerência deste modelo, o SNMP (*Simple Network Management Protocol*). O modelo TMN (*Telecommunications Management Network*) tem se destacado como arquitetura para a gerência destes ambientes, pois possui características que atendem aos seus requisitos. Mas devido ao alcance do modelo *Internet*, a escolha de um sistema de gerência não pode ignorar a existência das MIBs SNMP.

Este trabalho apresenta a construção de um elemento de rede TMN para a gerência de redes ATM e de um mecanismo de interface responsável pela comunicação entre gerentes TMN e agentes SNMP. Além das diferenças sintáticas existentes, os modelos apresentam funcionalidades distintas, exigindo uma interface mais sofisticada que um conversor de sintaxe. Desta forma, este trabalho propõe a extensão do mecanismo de interface implementado com a adição de métodos para a aplicação da funcionalidade TMN em redes compostas por recursos que dispõem de MIBs SNMP.

Palavras-chave: ATM, TMN, SNMP, Interface, Adaptador Q e Elemento de Rede

ABSTRACT

The expansion of ATM networks and the crescent service integration have magnified the complexity of network management. In addition to these facts, the growing demand for telecommunications services and the increasing number of equipment vendors and service suppliers have also added new requirements to network management. At present, pieces of equipment from diverse vendors are used to build up a telecommunication environment. Moreover, several protocols and architectures are used to connect these devices. Consequently, this diversity demands a management architecture with a flexible information model, able to represent the existing features in the environment. Among the different architectures used nowadays, the Internet model can be highlighted. Many efforts have been made in order to supply SNMP MIBs to ATM devices. In spite of TMN model defines a more encompassing and flexible architecture, the existing investments cannot be discarded.

The present work proposes the development of a TMN managed element to control ATM based networks. An adapter is also presented to enable SNMP devices to be integrated into TMN platform. Not only syntactic differences must be considered since these models have distinct functionality. Therefore, the proposed adapter must be able to allow the use of TMN management resources in networks composed by pieces of equipment supplied with SNMP interfaces.

Keywords: ATM, TMN, SNMP, Interface, Q-Adapter and Network Element

1 INTRODUÇÃO

O crescente uso das redes de computadores, somado à expansão destas pela utilização de equipamentos provenientes de diferentes fabricantes, implica em aumentos de complexidade na tarefa de seu gerenciamento. Estes fatores tornaram insuficientes os sistemas de gerência proprietários dirigidos a equipamentos de apenas um fabricante. Surgiram propostas para padronizar o gerenciamento de redes heterogêneas, entre as quais se destacam os modelos de gerência OSI (*Open Systems Interconnection*) e *Internet*. O segundo, proposto pelo IAB (*Internet Activities Board*), tornou-se um padrão *de facto*. O primeiro, elaborado pela ISO (*International Organization for Standardization*), possui rica funcionalidade e é mais abrangente que o modelo *Internet*. O modelo de gerência OSI é reconhecido como padrão *de juri*.

Adicionalmente, o aumento do poder de processamento dos computadores e a melhoria dos meios de transmissão existentes possibilitaram a disponibilização de um maior número de serviços, incentivando a utilização de aplicações multimídia em redes de computadores. As informações provenientes destas aplicações possuem requisitos distintos, como a não tolerância a erros dos serviços de transmissão de textos ou a alocação de grande largura de banda exigida durante a realização de vídeo-conferências. Para suprir estas necessidades, as provedoras de serviços de telecomunicação passaram a oferecer as RDSIs (Redes Digitais de Serviços Integrados) como opção de serviço capaz de transportar vários tipos de informação, respeitando seus requisitos e características.

Atualmente as RDSIs são classificadas em RDSI-FE (Faixa Estreita) ou RDSI-FL (Faixa Larga) de acordo com a largura de banda suportada pela rede. A principal característica destas redes é a possibilidade de acesso integrado aos vários serviços disponíveis [SOLE 95].

A adoção de redes ATM (*Asynchronous Transfer Mode*) constitui um suporte à implementação e difusão das RDSIs-FL, pois um ambiente ATM possui características, como grande confiabilidade e altas taxas de transmissão, que o fazem uma boa opção para a transferência de dados originados por mídias distintas. Para a integração dos serviços, as redes ATM devem interligar as redes dos usuários das operadoras de serviços de telecomunicação, aumentando a variedade de equipamentos e arquiteturas presentes. A administração de redes caracterizadas por esta diversidade exige a utilização de um sistema de gerência abrangente, que permita a representação e o controle de forma eficiente dos diversos tipos de recursos, arquiteturas e serviços disponíveis.

Para suprir estas novas necessidades, foi desenvolvido pela ITU-T (*International Telecommunications Union - Telecommunication Standardization Sector*) um conjunto de normas técnicas, denominado série M.3000, que descreve uma arquitetura para gerenciar redes de telecomunicação. Esta arquitetura é chamada de modelo TMN (*Telecommunications Management Network*) e define uma rede de gerência para controlar equipamentos e serviços de telecomunicação. O modelo informacional e a arquitetura física contidos no modelo TMN são amplos e genéricos, possibilitando sua aplicação em redes heterogêneas, onde são transmitidos dados com diferentes necessidades de qualidade de serviço. Os

conceitos presentes na série M.3000 podem ser aplicados a diversas arquiteturas de rede e são especializados na recomendação G.atmm [G.atmm] para atender a requisitos de redes ATM.

Devido a simplicidade dos serviços e protocolos do modelo *Internet,* grande parte dos equipamentos ATM em uso possuem interfaces e MIBs (*Management Information Bases*) compatíveis com o protocolo de gerência do modelo *Internet*, o SNMP (*Simple Network Management Protocol*). Para a aplicação do modelo TMN na gerência destes recursos, torna-se necessária a implementação de interfaces compatíveis com TMN ou a utilização de adaptadores que possibilitem a interação entre os modelos, pois o modelo informacional TMN utiliza o serviço CMIS (*Common Management Information Service*) do modelo de gerência OSI.

A necessidade de coexistência destes dois ambientes originou recomendações de organismos internacionais padronizando a integração dos protocolos CMIP e SNMP. Além dos esforços de padronização podem ser mencionadas ferramentas disponíveis para a integração destes dois ambientes como o *Solstice SNMP/TMN Q-Adaptor* e o *OpenCon Systems TMN Gateway*. Este último possibilita ainda a conexão com sistemas TL1 (*Transaction Language* 1), que de acordo com [TL1 99] é o protocolo de gerenciamento mais utilizado em ambientes de telecomunicação.

A implementação apresentada neste trabalho utiliza a ferramenta de gerenciamento OSIMIS (*OSI Management Information Service*), desenvolvida pela *University College London* sobre a plataforma ISODE (*ISO Development Environment*). A primeira fornece um conjunto de classes e funções para a

construção de sistemas de gerência e a última implementa recursos para o desenvolvimento de aplicações OSI sobre a pilha de protocolos TCP/IP (*Transport Control Protocol / Internet Protocol*). Em adição aos recursos de gerenciamento OSI, a plataforma OSIMIS implementa um adaptador Q para a obtenção de dados em MIBs SNMP.

Neste contexto, é descrita a implementação, sobre a plataforma OSIMIS, de objetos gerenciados da recomendação G.atmm para a composição de um elemento de rede. Devido a disponibilidade de MIBs SNMP em equipamentos ATM, a aplicação destes objetos na gerência de recursos ATM exige a implementação de funções que possibilitem a interação entre os modelos. Assim, também é apresentada a construção de um mecanismo de interface que utiliza o *gateway* SNMPv1/CMIP (*Common Management Information Protocol*) da plataforma OSIMIS para a obtenção de dados das MIBs SNMP.

Porém, a simples interação com as MIBs SNMP presentes nos recursos gerenciados impede que a funcionalidade do modelo TMN seja corretamente utilizada na administração de redes pois, além das diferenças sintáticas, estes dois modelos possuem funcionalidades distintas. Este trabalho propõe então a extensão do mecanismo de interface com a construção de funções para a adaptação dos recursos presentes nas MIBs SNMP à funcionalidade do modelo TMN.

Este trabalho está estruturado em oito capítulos, sendo o primeiro esta introdução. O segundo capítulo apresenta conceitos do modelo de gerenciamento TMN e descreve suas arquiteturas e funções. A aplicação do modelo TMN à gerência de redes ATM está descrita no terceiro capítulo, onde também são

apresentados requisitos e recomendações para a administração de redes ATM e suas conexões. A construção dos objetos gerenciados especificados nas recomendações G.atmm e M.3100 está no quarto capítulo. Mecanismos para acesso a bases de informações de gerência SNMP a partir de ambientes TMN são descritos no quinto capítulo. O sexto capítulo apresenta o mecanismo de interface implementado para permitir a aplicação do modelo TMN em redes ATM com MIBs SNMP. A aplicação do mecanismo construído no ambiente de testes disponível é descrita no capítulo sete. Finalizando este trabalho estão a conclusão (capítulo 8), as referências bibliográficas (capítulo 9) e os anexos (capítulo 10).

2 MODELO TMN

Com o objetivo de oferecer um suporte eficiente ao gerenciamento de equipamentos e serviços de telecomunicação foi desenvolvido pela ITU-T um conjunto de normas técnicas, denominado série M.3000, que descreve uma arquitetura para gerenciar redes de telecomunicação. Esta arquitetura é chamada de modelo TMN e define uma rede genérica para controlar equipamentos e serviços de telecomunicação.

Esta rede pode ser descrita como uma rede de gerência que possui diversos pontos de comunicação com o ambiente gerenciado [BRISA]. O sistema de gerência utiliza a rede TMN para a troca de informações, podendo fazer uso da rede sob sua administração. Devido a grande diversidade dos ambientes de telecomunicação, as especificações contidas no modelo TMN permitem a construção de sistemas para a administração de redes interligadas por meios físicos distintos, com diferentes topologias e alcances (LANs – *Local Area Networks*, MANs – *Metropolitan Area Networks* e WANs – *Wide Area Networks*). O ambiente TMN deve possibilitar o gerenciamento da própria rede TMN, pois esta também é uma rede de telecomunicações.

Os conceitos apresentados pelo modelo TMN abrangem o conjunto de funções contido no modelo OSI e estão em conformidade com os princípios do gerenciamento OSI. Portanto, o modelo TMN possui o suporte às cinco classes funcionais de gerenciamento: configuração, falhas, contabilização, segurança e desempenho.

Nas subseções seguintes serão apresentadas as três arquiteturas básicas que compõem o modelo TMN: funcional, informacional e física. Estas arquiteturas devem ser consideradas no planejamento de um novo ambiente TMN e podem ser desenvolvidas separadamente.

2.1. Arquitetura Funcional

Nesta arquitetura estão especificadas funções genéricas, necessárias ao gerenciamento, agrupadas em conjuntos denominados blocos funcionais. A divisão das funções em blocos funcionais permite a distribuição das funções de gerência, sendo realizada de acordo com a área de atuação destas funções. A comunicação entre os blocos é realizada em pontos de referência estabelecidos pela arquitetura funcional. Os blocos funcionais presentes no modelo TMN e suas respectivas áreas de atuação são:

- **Bloco Funcional Sistemas de Suporte a Operações (OSF)**: provê funções para o gerenciamento dos serviços de telecomunicação e da rede de gerência;
- **Bloco Funcional Elemento de Rede (NEF)**: contém funções para representar objetos passíveis de gerenciamento na rede TMN. Apenas as funções que representam o objeto estão incluídas na rede TMN, as funções de comunicação com o recurso gerenciado não estão no ambiente TMN;
- **Bloco Funcional Estação de Trabalho (WSF)**: inclui funções para a apresentação das informações disponíveis na rede TMN de forma compreensível ao operador;
- **Bloco Funcional Adaptador Q (QAF)**: as funções contidas neste bloco realizam a comunicação entre uma rede TMN e recursos que não oferecem suporte às

interfaces TMN. As funções de conversão presentes no Adaptador Q estão fora do escopo do ambiente TMN;

- **Bloco Funcional de Mediação (MF)**: dispõe funções para a comunicação entre os blocos funcionais OSF e NEF ou OSF e QAF, sendo responsável pela conversão das informações enviadas em um formato compreensível pelo bloco receptor.

As funções diretamente envolvidas no gerenciamento que estão presentes na arquitetura funcional e distribuídas pelos blocos são: Função de Sistemas de Suporte a Operações (*OSF - Operations Systems Function*), Função de Mediação (*MF - Mediation Function*), Função de Elemento de Rede (*NEF - Network Element Function*), Função de Estação de Trabalho (*WSF – Work Station Function*) e Função de Adaptador Q (*QAF - Q Adapter Function*).

Os pontos de referência do modelo TMN são agrupados em classes de acordo com os blocos funcionais entre os quais atuam. As seguintes classes estão na arquitetura funcional:

- **q**: ponto de referência entre blocos funcionais OSF, QAF, MF e NEF;
- **f**: utilizado para a conexão com o bloco funcional WSF;
- **x**: ponto de referência entre dois blocos OSF de redes TMN distintas ou entre um bloco OSF e um bloco equivalente em um ambiente não TMN;
- **g**: tem como objetivo definir a interface entre blocos funcionais WSF e operadores humanos;
- **m**: classe de interfaces entre blocos funcionais QAF e entidades gerenciadas não TMN.

As classes de pontos de referência **m** e **g**, apesar de estarem definidas no modelo TMN, não estão sujeitas à padronização e portanto estão fora do escopo de um ambiente TMN.

Além da divisão em blocos funcionais, é possível apresentar uma hierarquia de funções de gerenciamento TMN, onde o nível inferior apresenta uma visão à próxima camada para a realização das tarefas de gerência:

- **Elemento de Rede (NEL – *Network Element Level*)**: o nível menos abstrato desta hierarquia, onde são incluídos os recursos e serviços gerenciados;
- **Gerenciamento do Elemento de Rede (NEML – *Network Element Management Level*)**: neste nível estão as funções para o gerenciamento individual de recursos da rede;
- **Gerenciamento de Rede (NML – *Network Management Level*)**: esta categoria estabelece funções para o gerenciamento de atividades da rede. Para possibilitar o gerenciamento, este nível apresenta uma visão geral da rede pelo agrupamento dos recursos gerenciados;
- **Gerenciamento de Serviço**: funções para o gerenciamento dos serviços da rede de telecomunicações, o que inclui tarefas como cadastros de usuário e cobrança;
- **Gerenciamento de Negócios**: atua como um sistema de suporte a decisões administrativas. Contém funções para o gerenciamento do empreendimento e acompanhamento das metas administrativas, dentre outras.

Adicionalmente, pode-se estabelecer uma associação entre as funções contidas nos blocos funcionas e os níveis hierárquicos. As funções do bloco OSF

estão incluídas nas categorias de Gerenciamento de Negócios, Gerenciamento de Serviços, Gerenciamento de Rede e Gerenciamento do Elemento de Rede. As funções do bloco MF pertencem à categoria de Gerenciamento de Elemento de Rede. E finalmente, as funções do Elemento de Rede estão na camada Elemento de Rede. A Figura 2.1 apresenta uma possível conexão entre as funções desta arquitetura.

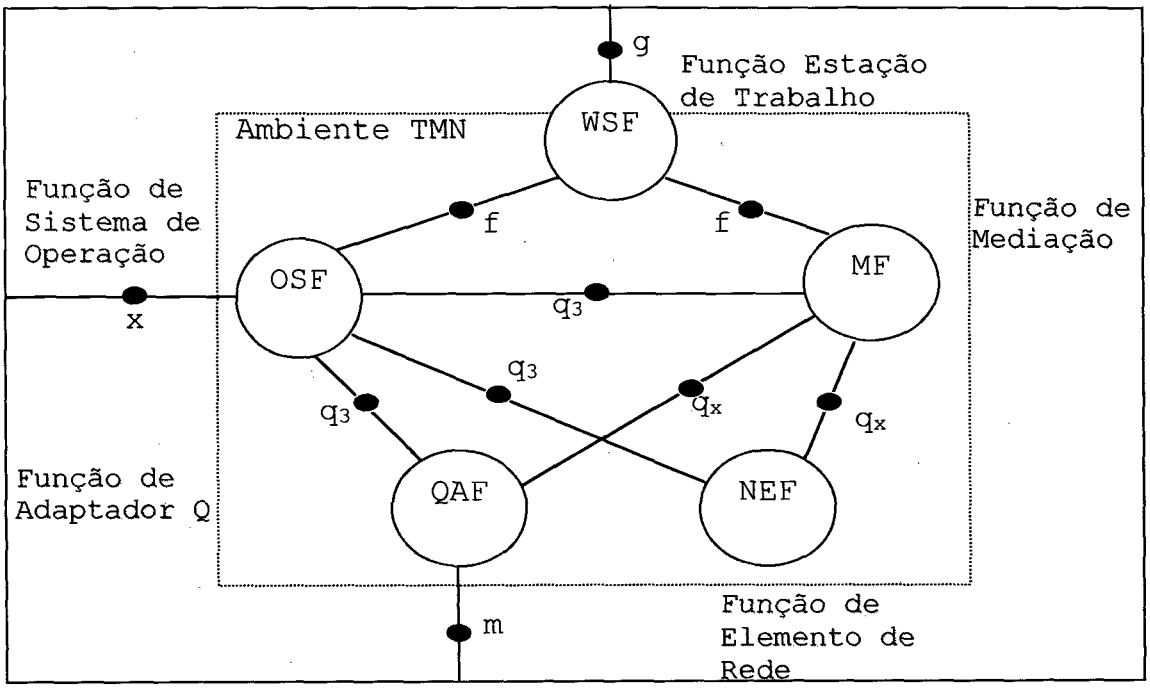

Figura 2.1 – Conexão entre as funções da Arquitetura Funcional TMN

Os pontos de interface q, sub-divididos em q_3 e q_x, atuam na comunicação entre as Funções de Sistema de Operação, Mediação, Adaptador Q e Elemento de Rede (Figura 2.1). A Função Estação de Trabalho é conectada ao ambiente TMN pelo ponto de interface f. A apresentação das informações ao operador é definida pelo ponto de interface g. Os recursos que não possuem interfaces compatíveis com TMN podem ser integrados a este ambiente com a utilização da Função de Adaptador Q e tem a interação com a rede TMN definida pelo ponto de interface m.

2.2. Arquitetura Informacional

A arquitetura informacional das redes TMN incorpora o modelo de informações do gerenciamento OSI. Consequentemente, os conceitos de orientação a objetos, gerente, agente e objetos gerenciados também estão presentes no modelo TMN [GoNo 95]. Para atender a requisitos da gerência de redes de telecomunicação, o conceito de Arquitetura Lógica em Camadas (LLA – *Logical Layered Architecture*) foi acrescido aos conceitos OSI incorporados.

A LLA implementa a divisão recursiva das funções de gerenciamento, tendo como objetivo o agrupamento das atividades de gerência em domínios funcionais. Os conjuntos formados com esta divisão estão sob a administração de um bloco OSF e recebem a denominação domínios-OSF. Como a arquitetura LLA é recursiva, os domínios formados podem ainda ser subdivididos em domínios mais específicos, gerenciados por um OSF [BRISA][GoNo 95].

Como um sistema de gerenciamento TMN pode ser composto por diversos OSFs, o projeto da arquitetura informacional deve garantir que as funções de gerência tenham acesso compartilhado às informações necessárias ao controle de recursos. O conjunto dos dados compartilhados entre dois sistemas é chamado SMK (*Shared Management Knowledge*) e pode conter informações relacionadas aos protocolos de comunicação, às funções de gerenciamento, às classes de objetos e às instâncias disponíveis dos objetos gerenciados. Assim como no modelo OSI, as entidades que compõem o gerenciamento TMN se comunicam utilizando o protocolo CMIP e o serviço CMIS.

2.3. Arquitetura Física

A arquitetura física é composta por blocos que possibilitam a implementação das características funcionais do modelo TMN. Um bloco representa um componente responsável pela execução de algumas funções e comunica-se através de interfaces com os demais blocos. Os componentes desta arquitetura devem ser flexíveis para possibilitar a utilização do modelo TMN na gerência das diversas topologias de redes existentes, garantir que mensagens críticas sejam corretamente enviadas ao seu destino e não sejam atrasadas por eventuais congestionamentos. O ambiente deve ser planejado de forma que o tráfego das mensagens de gerência não onere a performance da rede.

Os blocos que constituem a arquitetura física são:

- **Bloco de Sistema de Suporte a Operações (OS – *Operations System*)**: atua como gerente e deve permitir que as atividades de gerência sejam distribuídas ou centralizadas. Este bloco corresponde ao bloco funcional OSF;
- **Rede de Comunicações de Dados (DCN – *Data Communications Network*)**: corresponde aos três primeiros níveis da arquitetura OSI [GoNo 95] e deve, preferencialmente, seguir este modelo. A DCN contém as funções para a comunicação entre os demais blocos TMN;
- **Elemento de Rede (NE – *Network Element*)**: suas funções realizam tarefas relativas a telecomunicação e fornecem o suporte necessário a funções de gerência. Um NE pode estar subordinado a diversos OSFs, MDs e QAs;
- **Dispositivo de Mediação (MD – *Mediation Device*)**: realiza a tradução das informações entre OS e NE ou OS e QA. Deve garantir que a informação esteja no formato adequado à entidade receptora e provê facilidades de

gerenciamento local a um grupo de NEs semelhantes ou a um único NE. Este bloco pode ser implementado em um processo independente ou como parte do elemento de rede (NE);

- **Adaptador Q (QA – *Q Adapter*)**: corresponde ao bloco funcional QAF e atua na conversão de informações TMN para que sejam interpretadas por entidades que não oferecem suporte a interfaces TMN;
- **Estação de Trabalho (WS – *Work Station*)**: atua na adaptação das informações TMN a um formato compreensível a operadores humanos. As estações de trabalho correspondem aos blocos funcionais WSF e podem estar conectadas aos blocos físicos MD e OSF. A comunicação entre o bloco WS e os demais é feita com utilização da DCN.

A Figura 2.2 ilustra uma simples representação de conexões entre blocos físicos TMN.

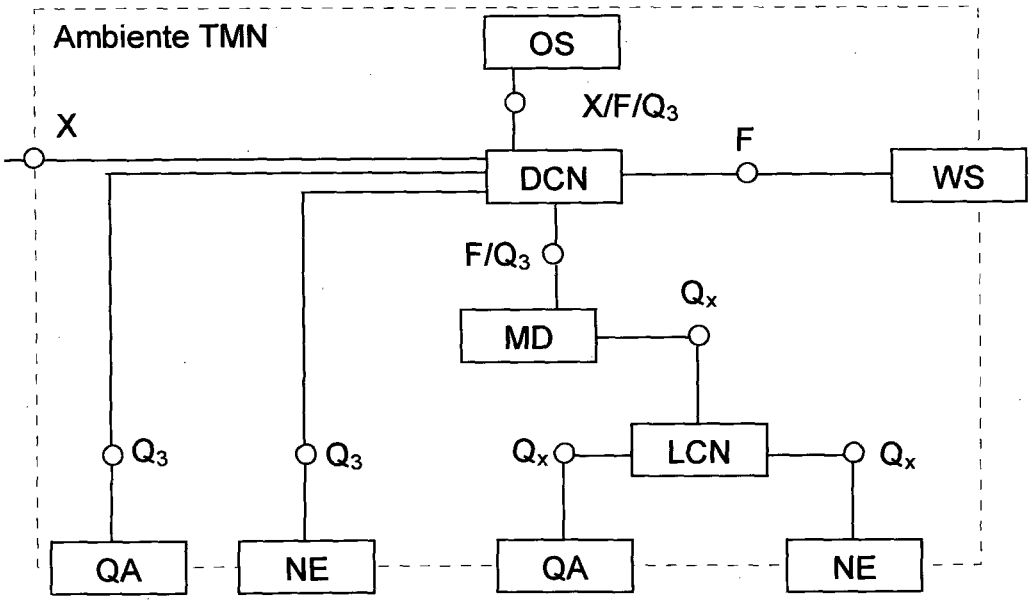

Figura 2.2 Conexão entre blocos físicos TMN.

Assim como na arquitetura funcionai, os blocos físicos se comunicam em pontos de interface (Figura 2.2). As interfaces existentes no modelo físico são Q, X e F. Os pontos de referência q, classificados em Q_3 e Q_x, são interfaces do tipo Q. A interface F, que atua no ponto de referencia f, realiza a comunicação entre OS e WS ou MD e WS. A interface X conecta OSs de redes TMNs distintas, ou uma OS de uma rede TMN a um bloco físico equivalente de um ambiente não TMN. Como ilustrado na Figura 2.2, as informações trocadas pelos blocos trafegam pela DCN.

3 UTILIZAÇÃO DE TMN PARA A GERÊNCIA DE REDES ATM

A arquitetura de redes ATM tem se firmado como suporte para a construção das RDSIs-FL (Redes Digitais de Serviços Integrados – Faixa Larga). As RDSIs-FL são adotadas como solução para o transporte de dados multimídia, resultantes da integração dos diversos tipos de informação. Logo, as redes ATM podem transportar dados de serviços distintos com diferentes necessidades de segurança, tolerância a erros e desempenho.

Por exemplo, a transmissão de textos não pode estar sujeita a erros, mas não impõe grandes restrições quanto ao retardo ou velocidade da conexão. Já as aplicações de vídeo conferência podem estar sujeitas a uma determinada taxa de erros, mas não podem estar sujeitas a retardos e quedas na velocidade de transmissão.

Esta integração de serviços resultará em uma rede única interconectando as diversas redes de usuários existentes atualmente. Assim, além da heterogeneidade de serviços, as redes ATM apresentam uma diversidade arquitetural, pois estarão conectadas a redes de diferentes arquiteturas (FDDI – *Fiber Distributed Data Interface*, *Ethernet* e outras).

Apesar de serem pouco susceptíveis a erros e possuírem uma grande largura de banda, as redes ATM ainda estão sujeitas a erros e ao mau uso dos seus recursos. Desta forma, a atividade de gerenciamento torna-se necessária para evitar congestionamentos que impeçam o atendimento aos requisitos de qualidade de serviço solicitados por aplicações que possuem dados trafegando na rede.

Devido à grande diversidade de serviços e arquiteturas presentes nas redes ATM, a arquitetura de gerenciamento utilizada deve possuir um modelo de informação que represente vários tipos de recursos e fornecer uma visão completa da rede. Para facilitar a administração, deve ser possível dividir a rede em domínios de gerenciamento, possibilitando o agrupamento dos recursos de acordo com critérios estabelecidos pelo gerente.

O modelo TMN adapta-se à heterogeneidade das redes ATM utilizando as características distribuídas dos Elementos de Rede (NE) e Sistemas de Suporte a Operações (OS), juntamente com as interfaces fornecidas pelo protocolo CMIP, para a construção de complexas aplicações de gerenciamento [KorRo97a] [KorRo97b]. A arquitetura TMN utiliza o conceito de Arquitetura Lógica em Camadas (LLA) que permite a divisão recursiva das tarefas de gerenciamento em domínios, o que possibilita ao gerente o agrupamento das funções de gerência.

3.1. Aspectos do gerenciamento de redes ATM

Como apresentado na seção anterior, apesar da alta velocidade e grande confiabilidade das redes ATM, estas necessitam de gerenciamento para possibilitar o uso de toda sua capacidade. Devido a esta necessidade, organismos como a ITU-T e o ATM Fórum têm estabelecido normas para o gerenciamento de redes ATM.

Dentro deste contexto, a recomendação M.3100 contém um conjunto de objetos gerenciados e suas interfaces para a troca de informações TMN. Estas classes de objetos são aplicáveis a diferentes tecnologias e podem também ser utilizadas para o gerenciamento de redes ATM. Os objetos desta recomendação são especializados em G.atmm para aplicação específica nestas redes.

Uma das características que possibilitam às redes ATM a utilização de altas taxas de transferência é o uso de serviços orientados a conexão. A gerência destes serviços é objeto deste trabalho. Nas redes ATM, estes serviços são caracterizados pelas conexões de canais e caminhos virtuais, onde são aplicadas funções de gerenciamento ATM classificadas em: monitoração de desempenho, relatório de falhas, testes e gerenciamento de tráfego. Estes funções habilitam o sistema de gerência a realizar tarefas como consulta e alteração do estado dos elementos da rede e das tabelas de roteamento das conexões e caminhos virtuais (VPs – *Virtual Paths* / VCs – *Virtual Channels*).

Funções para detectar falhas nos elementos de rede, envio de notificações sobre as falhas para o sistema de gerência e coleta de estatísticas referentes ao envio de células incorretas estão inclusas na área de falhas. A área de performance inclui funções que podem ser utilizadas para auxiliar as decisões de gerenciamento e para a cobrança de taxas sobre os serviços prestados aos usuários. Estas funções coletam dados referentes a utilização dos recursos e serviços disponibilizados na rede.

3.2. Gerenciamento de conexões

Objetos para gerenciamento das conexões estabelecidas em um ambiente ATM estão definidos nas recomendações G.atmm e M.3100. Os objetos destes documentos são utilizados para monitorar e controlar diferentes tipos de conexões, classificadas segundo sua função na rede.

Para os padrões de gerenciamento TMN uma rede pode ser decomposta em níveis, onde o nível inferior fornece serviços às entidades do nível superior. As

entidades presentes em cada uma destas camadas podem ainda ser agrupadas para representar a estrutura da camada. A decomposição em níveis e o agrupamento das entidades presentes em cada nível seguem respectivamente os conceitos de camada e partição. Estes conceitos podem ser observados na Figura 3.1.

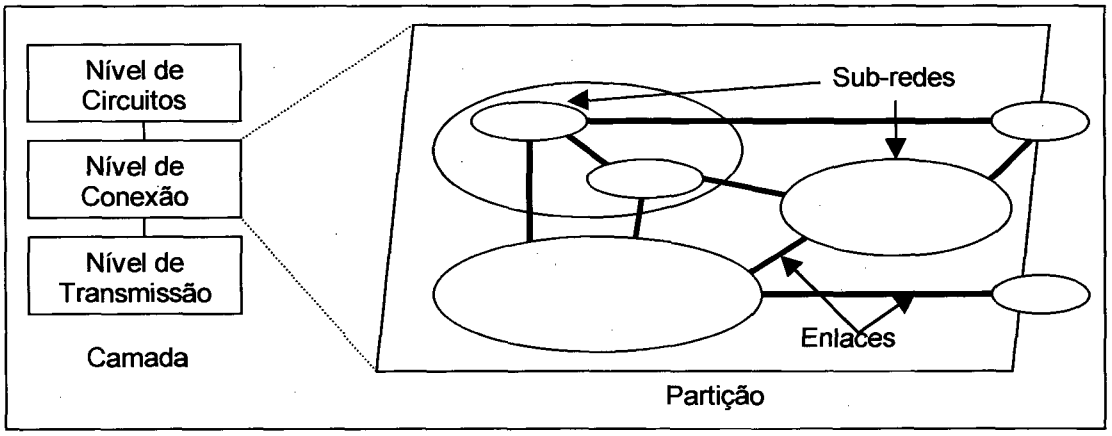

Figura 3.1 - Visão dos conceitos de partição e camada [G.803].

Cada camada de rede agrupa um conjunto de funções similares, podendo ser independentemente definida. A estrutura dos recursos em cada camada é definida pelo particionamento dos recursos em sub-redes e seus enlaces.

3.2.1 Conexões sob o conceito de partição

A conexão entre dois pontos terminais de uma camada de rede é denomina *trail*. Do ponto de vista do conceito de partição, um *trail* é a composição de conexões de sub-rede, conexões de enlace e pontos de terminação. As conexões de sub-rede são realizadas por recursos localizados na mesma sub-rede. Já uma conexão de enlace é responsável por conectar elementos presentes em sub-redes distintas. A composição das conexões de enlace e de sub-rede é chamada de enlace topológico. A Figura 3.2 apresenta conexões observadas segundo o conceito de

partição. Também estão contidas nesta figura os pontos de terminação de conexão, onde localizam-se as funções de terminação.

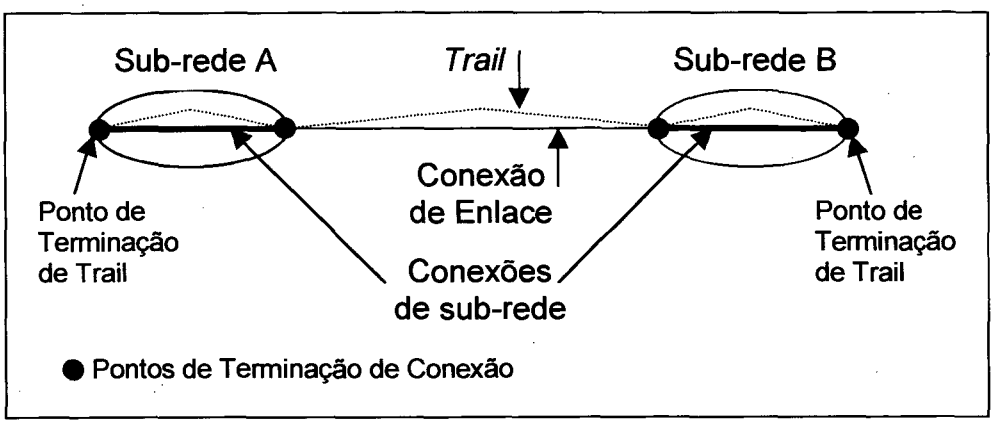

Figura 3.2 – Conexões segundo o conceito de partição.

3.2.2 Conexões sob o conceito de camadas

Para o conceito de camada, as conexões de sub-rede e de enlace da camada cliente são serviços prestado pela camada servidora para ligar pontos terminais das conexões. Cada uma das conexões estabelecidas na camada cliente utiliza um *trail* na camada servidora. Esta definição pode ser observada na Figura 3.3.

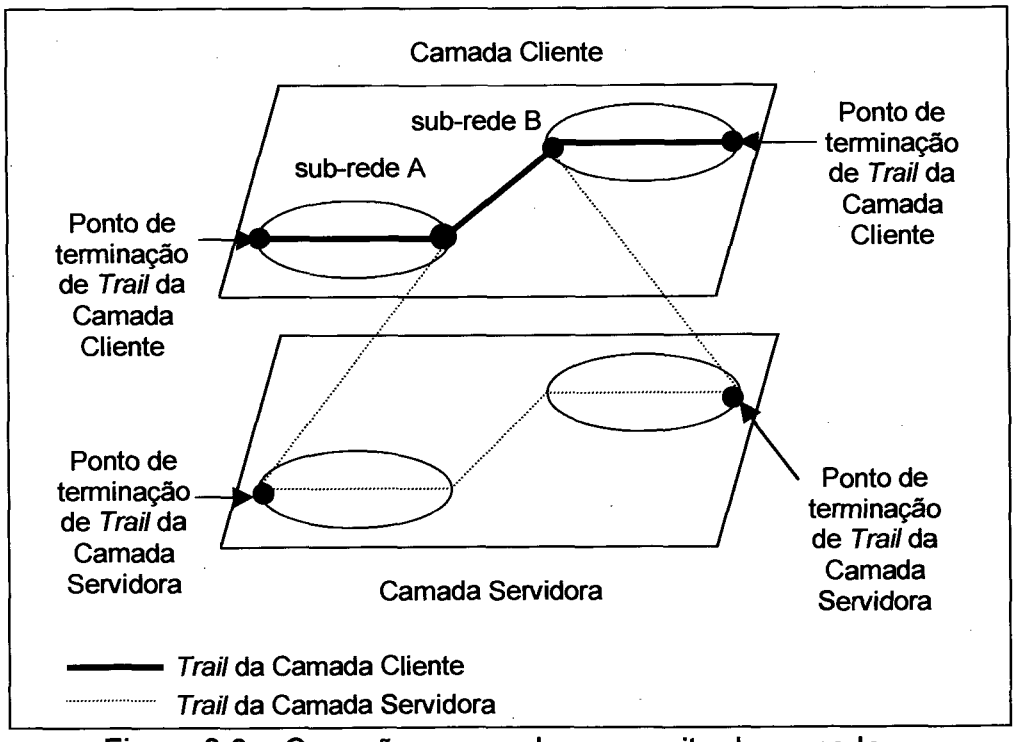

Figura 3.3 – Conexões segundo o conceito de camada.

A Figura 3.3 ilustra um *trail* na camada cliente e sua composição por conexões de sub-rede e uma conexão de enlace conectando as duas sub-redes presentes na camada. As conexões formadoras do *trail* da camada cliente são providas por *trails* da camada servidora. Na figura, apenas o *trail* da camada servidora que implementa a conexão de enlace entre as duas sub-redes da camada cliente é apresentado.

4 ELEMENTO DE REDE TMN

Um elemento de rede TMN consiste na composição de gerentes, agentes e objetos gerenciados. Os gerentes e agentes formam a aplicação de gerenciamento responsável pelo controle e monitoração dos objetos gerenciados e estes representam os recursos e/ou serviços existentes na rede.

As atividades de gerência atribuídas ao elemento de rede podem ser agrupadas em blocos com funções específicas que atuam em diferentes áreas funcionais do gerenciamento ATM. A implementação do elemento de rede apresentada neste trabalho está fundamentada na recomendação G.atmm, cujo modelo de informação descreve classes de objetos para o gerenciamento de VPs (*Virtual Paths*)e VCs (*Virtual Channels*) da camada ATM.

As funções de gerenciamento contidas em G.atmm atuam sobre aspectos de gerenciamento do elemento de rede TMN e são divididas em três blocos: camadas, conexões e desempenho. O gerenciamento de camadas possui funções para configuração dos serviços das camadas de convergência de transmissão e de conexão de canal (VC) e caminho virtual (VP). Tarefas como configuração de interfaces e de equipamentos ATM e realocação da largura de banda de uma conexão são atribuídas a este bloco. Funções para detecção e notificação de falhas das camadas de convergência de transmissão e de conexão de canal e caminho virtual estão presentes neste bloco.

Atividades de estabelecimento e liberação de conexões encontram funções correspondentes no bloco de gerenciamento de conexões. Neste bloco estão as

funções para alocação de identificadores e detecção de queda de conexão. As funções do bloco de gerenciamento de desempenho contêm recursos para monitorar dados das atividade das conexões, tais como pacotes enviados e recebidos.

As próximas subseções (4.1, 4.2 e 4.3) descrevem objetos apresentados por recomendações da ITU-T e do ATM Fórum para o gerenciamento de conexões e sua implementação sobre a plataforma de gerenciamento OSIMIS TMN.

4.1. Enlaces e *Trails*

Como apresentado no terceiro capítulo uma associação entre pontos terminais é formada por conexões entre pontos intermediários, denominadas conexões de sub-rede e conexões de enlace. Em um ambiente ATM estes enlaces são representados pelos canais e caminhos virtuais. Para a gerência destas conexões foram implementadas duas classes de objetos gerenciados do padrão G.atmm: `vcCTPBirectional` e `vpCTPBirectional`. A primeira classe é instanciada para representar enlaces de canal virtual em redes ATM. Um objeto da classe `vpCTPBirectional` é utilizado para gerenciar os enlaces de caminho virtual.

Atributos contidos nestas classes indicam os pontos de terminação de enlaces conectados ou disponíveis para conexão. Estes pontos são utilizados pelo sistema de gerência para a inserção de células de teste OAM (*Operation, Administration and Maintenance)*. Uma das ações incluídas nestes objetos insere uma célula e obtém o tempo de retorno. Esta funcionalidade permite verificar o retardo existente na rede.

As variáveis vcCTPId e vpCTPId, respectivamente das classes `vcCTPBirectional` e `vpCTPBirectional`, são utilizadas como identificadores de RDN (*Relative Distinguished Name*) na hierarquia de *containment*. Estes atributos contém os identificadores de canal e caminho virtual, que podem ser atribuídos pelo sistema de gerência no momento da criação do objeto ou atribuídos automaticamente. Nesta situação o valor do identificador deve ser enviado como retorno da função de criação do objeto.

A classe da qualidade de serviço e descritores de tráfego estão contidos nestas classes de objetos. Estes atributos possibilitam a obtenção de dados como a taxa pico do fluxo de células e o retardo tolerado no enlace.

Para a gerência de *trails* (conexões entre pontos terminais) foram implementadas as classes de objetos `vcTTPBirectional` e `vpTTPBirectional`, descritas em G.atmm. Objetos da primeira classe estão associados a *trails* estabelecidos em canais virtuais. Já os *trails* de caminhos virtuais são gerenciados com o uso da classe *vpTTPBirectional*.

Estão presentes nestas classes funções que permitem detectar o atraso no envio de células pela conexão. Um célula `OAMCellLoopback` é inserida na conexão de *trail* e o tempo de retorno é obtido para verificar se a performance não está degradada.

A implementação destas classes de objetos gerenciados foi feita com a utilização do compilador GDMO (*Guidelines for the Definition of Managed Objects*) fornecido com a plataforma OSIMIS. Este compilador recebe como entrada uma

especificação GDMO e apresenta como saída um arquivo com o código em C++ para a integração da classe de objeto gerenciado com a plataforma OSIMIS.

4.2. Implementação dos objetos gerenciados

As classes `vcCTPBidirectional` e `vpCTPBirectional` da recomendação G.atmm são subclasses de `connectionTerminationPoint Bidirectional` que, por sua vez, descende de `connectionTerminationPoint Source` e `connectionTerminationPointSink`. Estas classes representam os pontos de início e término de um enlace, e por sua vez são especializações da classe `terminationPoint`.

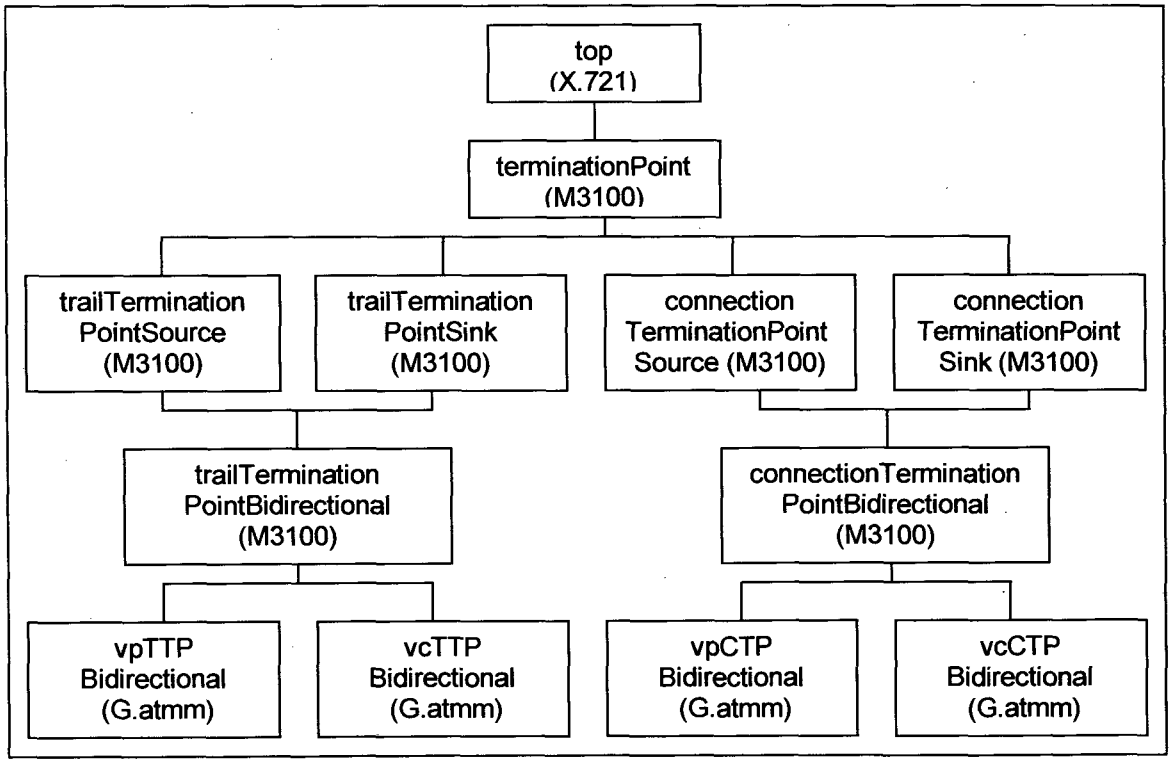

Figura 4.1 – Hierarquia de herança dos objetos implementados.

A Figura 4.1 apresenta a hierarquia de herança das classes `vpTTPBidirectional` e `vcTTPBirectional`. Estas classes herdam características de `trailTerminationPointBidirectional`, subclasse de

`trailTerminationPointSource` e `trailTerminationPointSink` que representam os pontos de início e fim de um *trail* e são subclasses de `terminationPoint`.

4.3. Implementação de atributos

Os objetos gerenciados especificados segundo o padrão OSI possuem variáveis que representam o estado do recurso sendo monitorado. Para tornar padronizada a troca de informações entre os processos, as sintaxes dos atributos são especificadas utilizando a notação ASN.1 (*Abstract Syntax Notation 1*). A plataforma OSIMIS fornece um mecanismo de suporte à sintaxe ASN.1, mas é necessário representar estas estruturas em linguagem de programação C++. Por conseguinte, os atributos não são integrados à plataforma de gerenciamento em ASN.1, mas com a utilização de estruturas de dados C++ que implementam esta sintaxe.

O primeiro passo para a construção das estruturas em C++ que implementam a sintaxe ASN.1 dos atributos é a utilização do compilador *Pepsy*. Este compilador é utilizado para a construção das estruturas de dados em C que representam a sintaxe. O compilador recebe como entrada as especificações ASN.1 e obtém-se como saída os arquivos que contém as estruturas representadas em C. Às funções geradas pelo compilador *Pepsy* foram adicionadas funções para a manipulação de valores das variáveis.

Após a utilização do compilador *Pepsy* é necessário implementar a semântica dos atributos. Nesta tarefa são codificadas classes de objetos de atributos que serão utilizadas na construção dos objetos gerenciados. As classes de atributos fornecem

funcionalidades para apresentação dos atributos e sua manipulação pelo uso de primitivas CMIP, tais como GET e SET.

A Tabela 4.1 apresenta as sintaxes de atributos utilizadas pelos objetos gerenciados implementados e que não estavam presentes na plataforma OSIMIS. A primeira coluna apresenta o nome da sintaxe ASN.1 do atributo e o nome da classe de atributos C++ codificada para implementar o comportamento do atributo. A estrutura em C gerada pelo compilador *Pepsy* está na segunda coluna.

Tabela 4.1 – Sintaxes de atributos implementadas.

Sintaxe e Classe do atributo	**Estrutura em linguagem C**
BurstTolerance	type_AtmMIBMod_BurstTolerance
CDVTolerance	type_AtmMIBMod_CDVTolerance
PeakCellRate	type_AtmMIBMod_PeakCellRate
QosClass	type_AtmMIBMod_QosClass
SustainableCellRate	type_AtmMIBMod_SustainableCellRate
OAMPeakCellRate	type_ASN1TypeModule_OAMPeakCellRate
AlarmStatus	type_ASN1DefinedTypesModule_AlarmStatus
CurrentProblemList	type_ASN1DefinedTypesModule_CurrentProblemList
SystemTimingSource	type_ASN1DefinedTypesModule_SystemTimingSource
SystemTitle	type_ASN1DefinedTypesModule_SystemTitle
ObjectList	type_ASN1DefinedTypesModule_ObjectList
ConnectivityPointer	type_ASN1DefinedTypesModule_ConnectivityPointer
CrossConnectionObjectPointer	type_ASN1DefinedTypesModule_CrossConnectionObjectPointer
DownstreamConnectivityPointer	type_ASN1DefinedTypesModule_DownstreamConnectivityPointer
SupportableClientList	type_ASN1DefinedTypesModule_SupportableClientList
PointerorNull	type_ASN1DefinedTypesModule_PointerorNull

Os atributos `BurstTolerance`, `CDVTolerance`, `PeakCellRate`, `SustainableCellRate` e `OAMPeakCellRate` compõem o descritor de tráfego da conexão sendo monitorada. A classe de qualidade de serviço (QoS – *Quality of Service*) é representada pelo atributo `QoSClass`.

Para atender a requisitos do gerenciamento de falhas são descritas duas sintaxes de atributos para os objetos gerenciados: `AlarmStatus` e `Current ProblemList`. O primeiro atributo contém o grau de severidade da falha indicada pelo alarme. Em `CurrentProblemList` é armazenada a lista de falhas ocorridas, e que levaram ao disparo do alarme.

Os objetos gerenciados implementados neste trabalho têm como objetivo gerenciar conexões estabelecidas em um ambiente ATM, o que torna necessário a referência aos pontos terminais das conexões. Para identificar os objetos que representam os terminadores de conexão e *trail* têm-se as sintaxes de atributos `ConnectivityPointer`, `CrossConnectionObjectPointer` e `DownStream ConnectivityPointer`.

5 INTERFACE COM RECURSOS SNMP

Conforme descrito no capítulo 3, a arquitetura TMN descreve um modelo informacional amplo que o torna apto a administrar a heterogeneidade dos equipamentos de telefonia e fornecer suporte aos diferentes requisitos do gerenciamento de redes de alta velocidade. Devido a simplicidade dos serviços e protocolos do modelo *Internet,* grandes investimentos foram feitos para prover os recursos ATM com interfaces e MIBs compatíveis com o protocolo SNMP. Para a aplicação do modelo TMN na gerência destes recursos, torna-se necessária a implementação de um modelo de informação TMN integrado ao recurso real ou a utilização de adaptadores que possibilitem a interação entre os protocolos SNMP e CMIP.

Neste trabalho, propõe-se a aplicação da segunda abordagem pois possibilita a interação de um sistema de gerência TMN com as interfaces existentes nos recursos. Esta alternativa torna desnecessária a construção de *drivers* específicos para cada modelo de equipamento utilizado, mas exige a disponibilização de mecanismos que permitam a interação entre sistemas TMN e recursos gerenciáveis pelo protocolo SNMP.

Diversas soluções de gerenciamento de redes apresentam ferramentas que possibilitam esta interação. Entre as implementações disponíveis encontram-se o adaptador Q da plataforma OSIMIS, o *OCS TMN System* da OCS (*OpenCon Systems*) e o *Solstice TMN/SNMP Q-Adaptor* da *Sun Microsystems*. Bibliotecas de programação, como a *CMU SNMP Library* da *Carnegie Mellon University* (CMU),

também podem ser utilizadas para implementar a comunicação com bases de informações SNMP.

Porém, a utilização isolada destes recursos resulta apenas na conversão das informações SNMP em um formato compatível com GDMO. Conforme apresentado na Figura 5.1, as variáveis são obtidas das MIBs SNMP, convertidas em um formato GDMO definido previamente e então apresentadas na MIB TMN.

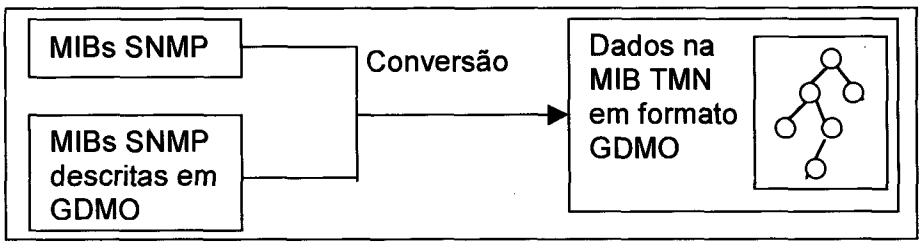

Figura 5.1 - Estrutura Básica para a Conversão de Sintaxe SNMP/TMN.

Devido a diferença semântica entre os modelo TMN e *Internet*, esta abordagem impede que a funcionalidade presente no modelo TMN seja inteiramente aplicada no gerenciamento destes recursos. Para a comunicação entre estes ambientes é requerido um mecanismo com habilidade de se conectar a MIBs SNMP e atuar como uma interface totalmente adaptada às requisições do elemento de rede.

Além da diferença de funcionalidade entre o modelo TMN e o modelo *Internet*, as MIBs deste modelo contém dados desnecessários ao elemento de rede. O mecanismo de interface deve prover ao elemento de rede apenas as informações solicitadas [MiRa97], reduzindo o tráfego de informações.

A Figura 5.2 apresenta uma situação onde o mecanismo de interface atua na adaptação dos atributos requeridos pelo elemento de rede. No exemplo, o elemento

de rede solicita à MIB o atributo $NE\_Atributo_1$. Para responder à solicitação, o mecanismo de interface deve obter da MIB SNMP os valores de $ATM\_Atributo_3$ e $ATM\_Atributo_n$. O resultado solicitado pelo elemento de rede é obtido com a aplicação de uma função de conversão sobre as varáveis consultadas.

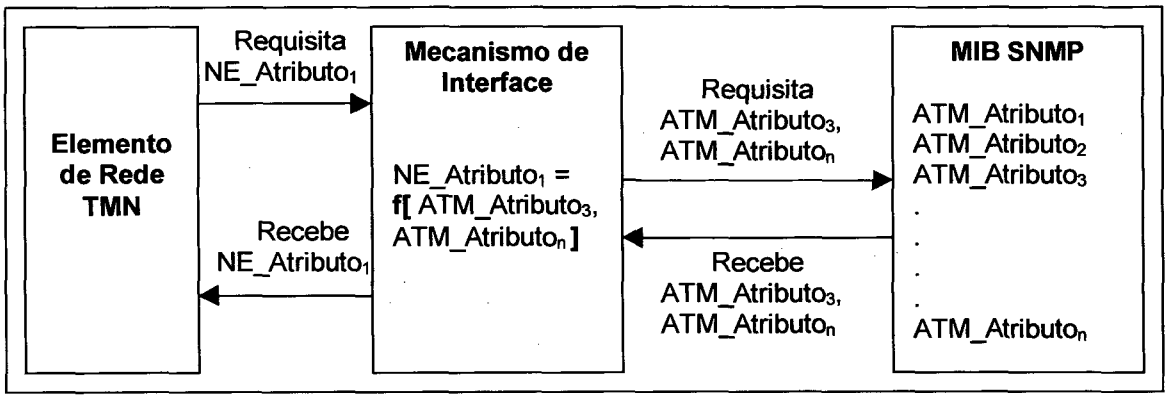

Figura 5.2 - Exemplo de Atuação do Mecanismo de Interface.

O mecanismo de interface deve permitir adaptações para possibilitar a interação com bases de informações de gerenciamento SNMP utilizando os recursos disponíveis na plataforma de gerência escolhida. Nas próximas seções são apresentadas algumas das soluções existentes para a integração de ambientes TMN a interfaces SNMP. As seções 5.1 e 5.2 descrevem o adaptador Q fornecido pela plataforma OSIMIS e algumas modificações realizadas com o objetivo de minimizar o tráfego de dados. Os adaptadores Q da *Sun Microsystems* e da *OpenCon Systems* são apresentados na seção 5.3. A biblioteca CMU SNMP é brevemente descrita na seção 5.4.

5.1. *Gateway* SNMPv1/CMIP da Plataforma OSIMIS

O adaptador Q presente na plataforma OSIMIS possibilita sua integração ao ambiente multi-protocolar das redes de telecomunicação e de computadores. Este conversor está implementado na sétima camada do modelo de referência OSI, a

camada de aplicação. A coexistência dos protocolos SNMP e CMIP prevê duas diferentes tarefas: a conversão de MIBs SNMP em MIBs CMIP e o mapeamento das operações CMIP em primitivas SNMP. Como esta ferramenta não possui acesso direto às MIBs SNMP, deve solicitar informações ao agente remoto responsável pela MIB onde estão as informações necessárias (Figura 5.3).

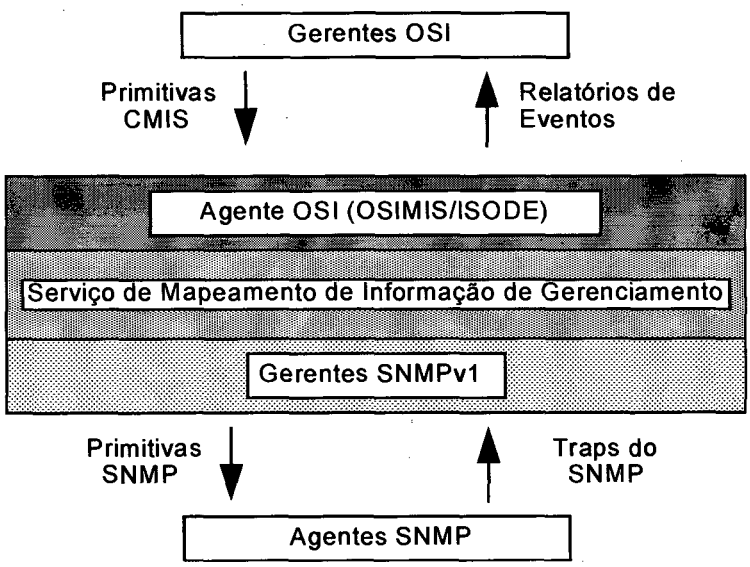

Figura 5.3 - Arquitetura Funcional do Gateway [SoMa93].

A funcionalidade do *gateway* pode ser observada na Figura 5.3. Os comandos de gerência provenientes de agentes OSI são enviados a agentes específicos para o processo de conversão. Após este processo, o *gateway* atua como um gerente SNMP enviando os comandos para um agente SNMP. Os quatro blocos funcionais que compõem o adaptador Q SNMPv1/CMIP são apresentados na Figura 5.4.

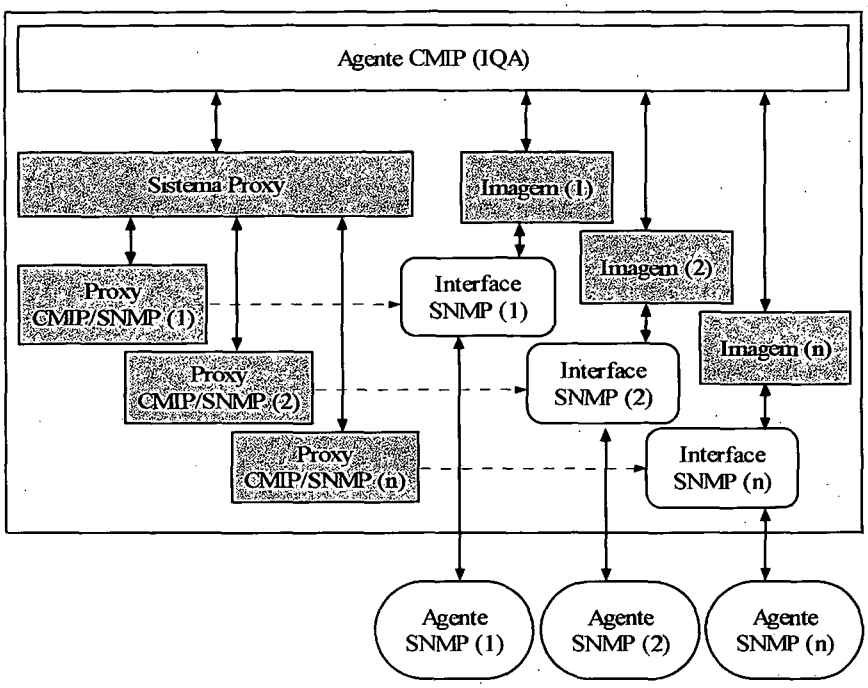

Figura 5.4 – Blocos Funcionais do Adaptador Q Existente na Plataforma OSIMIS [MiRa97].

O primeiro bloco funcional, Sistema *Proxy* (Figura 5.4), tem como objetivo iniciar a associação com agentes SNMP remotos. Para cada agente SNMP monitorado são instanciados objetos das classes que constituem o bloco funcional *Proxy* CMIP/SNMP. O mapeamento das informações e operações requisitadas pelo sistema de gerência é implementado pelo bloco funcional Interface SNMP. As informações obtidas e convertidas são apresentadas ao agente CMIP pelo bloco Imagem.

5.2. Mecanismo de interface

Apenas parte do conteúdo presente em bases de informações SNMP é efetivamente utilizada pelo elemento de rede implementado, tornando desnecessária a representação integral da MIB oferecida pelo bloco Imagem do adaptador Q (Figura 5.4). Este trabalho apresenta uma solução alternativa, construída sobre o

adaptador Q da plataforma OSIMIS, que possibilita a obtenção dos atributos requeridos, sem o envio do conteúdo completo da MIB SNMP.

A composição do novo mecanismo pode ser observada na Figura 5.5. O bloco Imagem foi suprimido pois a representação integral da MIB SNMP não é requerida. O bloco Sistema *Proxy* foi mantido com a mesma funcionalidade. Os blocos *Proxy* CMIP/SNMP e Interface SNMP foram coligidos no bloco *Proxy* SNMP. Desta forma, as funções de conversão de sintaxe, mapeamento de operações e associação a agentes remotos SNMP foram agrupadas em um único bloco funcional.

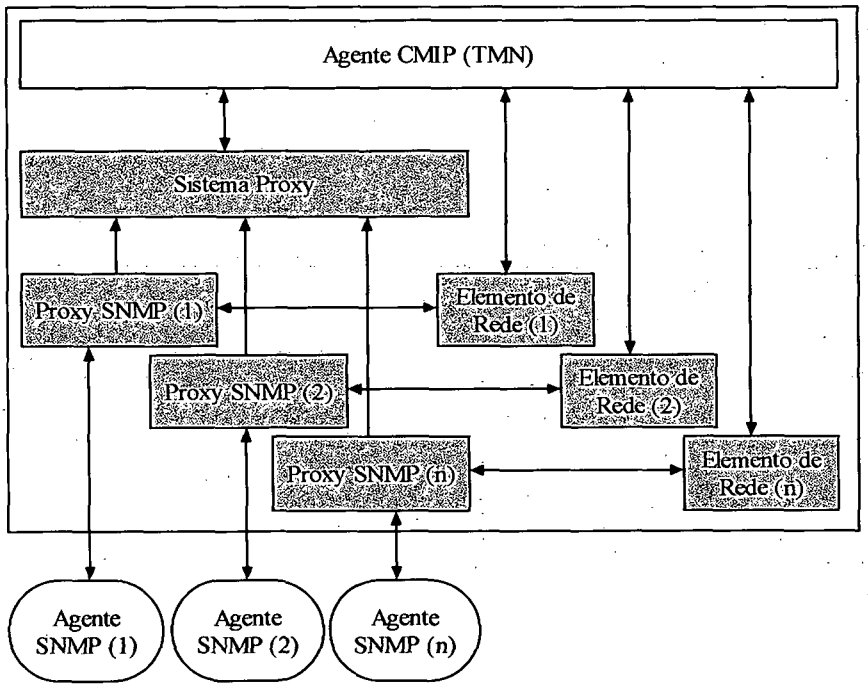

Figura 5.5 - Mecanismo de Interface Proposto para o Elemento de Rede [MiRa 97].

As solicitações de gerenciamento enviadas ao elemento de rede são transmitidas ao bloco *Proxy* SNMP (Figura 5.5). Este bloco realiza a conversão sintática da operação e a envia ao agente remoto. Os valores transferidos na operação são convertidos de tipos SNMP em CMIP, ou de CMIP em SNMP, e enviados ao agente remoto ou ao elemento de rede.

A operação *CMIS M-Get* pode ser mapeada diretamente para um *SNMP Get*. Como o SNMP não oferece um serviço equivalente ao *CMIS M-Cancel-Get*, esta operação será de responsabilidade do agente OSI. Uma operação *CMIS M-Set* será mapeada em um *SNMP Set*. No SNMP, apenas *table entries* podem ser criadas ou apagadas. Para a criação de um *table entry* o agente SNMP precisa enviar uma primitiva *Set* com os valores de cada coluna, assim uma operação *M-Create* informará todos os valores iniciais do objeto e será mapeada para um *SNMP Set*. Para apagar um objeto deve-se atribuir a determinada coluna um valor que indique linha inválida.

O mapeamento das operações *CMIS M-Event-Report* e *SNMP Trap* no *gateway* da plataforma OSIMIS ainda continua em estudos. Esta operação tem seu procedimento dificultado pois é composta por comunicações originadas por um agente e enviadas ao gerente [SoMa93].

A estrutura do mecanismo de interface desenvolvido para interconectar o elemento de rede TMN às bases de informações de gerenciamento SNMP é composta pelas classes C++ ilustradas na Figura 5.6.

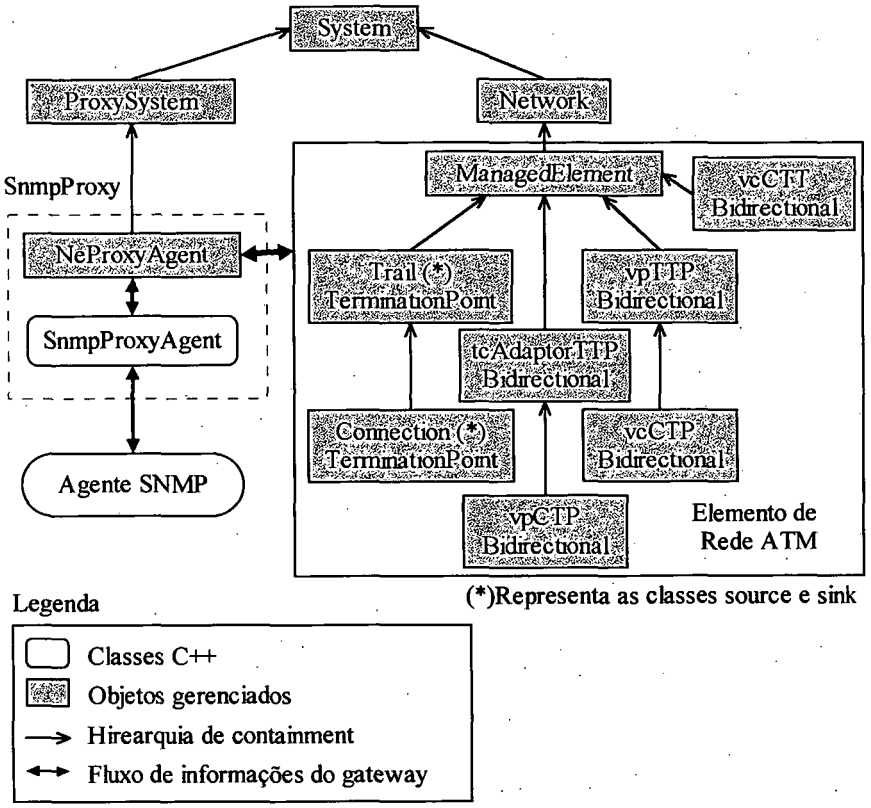

Figura 5.6 - Estrutura do Mecanismo de Interface [MiRa 97].

O bloco funcional Sistema *Proxy* é implementado pela classe `ProxySystem`. Esta classe consulta a lista das bases de informações de gerenciamento que serão utilizadas pelo elemento de rede. A lista é armazenada no arquivo "ne.conf" (Figura 5.7) e fornece informações como o número do endereço IP do recurso que contém a MIB SNMP, o numero da porta UDP, o direito de acesso definido pela `community string` e o nome do equipamento a ser gerenciado. Os identificadores de atributos da MIB SNMP são mantidos em arquivos diferentes cujos nomes estão no arquivo de configuração indicados pela entrada `mibDef`. Na Figura 5.7, o valor contido em `mibDef` aponta para o arquivo "comutadorA.def".

```
[remote mibs]
comutador_a = "IBM 8260"
comutador_b = "centillium 100"

[comutador_a]
hostname = comutador_a.ufsc.br
address = 192.168.0.1
udpport = 161
community = "public"
magic = 1
mibdef = "comutadorA.def"
rdn = "managedElementId=1"
parentdn = "networkId=1"
classname = "managedElementR1"
```

Figura 5.7 - Fragmento do Arquivo "ne.conf".

Para cada sistema remoto definido no arquivo de configuração será instanciado um conjunto de objetos das classes que constituem o bloco *Proxy* SNMP: `NeProxyAgent` e `SnmpProxyAgent`. A classe `SnmpProxyAgent` representa a interface com o agente remoto, implementando funções capazes de controlar o estabelecimento e a liberação de uma associação com o agente remoto. A classe `NeProxyAgent` é responsável pela conversão dos tipos SNMP em atributos de classes de objetos gerenciados e vice-versa.

A validação dos atributos de localização da MIB remota (`address`, `udpport`, `community`) contidos no arquivo de configuração é realizada por uma instância de `neProxyAgent`. Se o teste de validação for bem sucedido, será criado um objeto da classe `SnmpProxyAgent` para o acesso ao sistema remoto. Após o estabelecimento da comunicação, o objeto `NeProxyAgent` realiza uma consulta à MIB remota. Em caso de falha, as instâncias que contém informações referentes ao sistema remoto, área pontilhada da Figura 5.6, serão removidas.

Após a conclusão da fase de associação, o elemento de rede está habilitado a interagir com o agente remoto. As classes de objetos gerenciados que compõem o

elemento de rede enviam solicitações diretamente ao objeto `NeProxyAgent`. Este objeto pesquisa uma lista contendo identificadores de atributos CMIP e correspondentes OIDs (*Object Identifiers*) SNMP para a devida conversão sintática entre estes dois padrões.

5.3. Gateways Proprietários

A necessidade de coexistência e integração de ambientes multi-protocolares exige dos fabricantes de plataformas de gerência a implementação de adaptadores Q que possibilitem esta tarefa. As implementações são também motivadas pelo crescimento verificado do mercado de telecomunicação e pelo aumento do número de fabricantes de equipamentos e operadoras de serviços.

O *TMN Gateway* da *OpenCon Systems* [Open 99] constitui um adaptador Q responsável pela conversão de informações entre os modelo TMN, SNMP e TL1. O *Solstice TMN/SNMP Q-Adaptor* da *Sun* [Sun 99] prevê, assim como a plataforma OSIMIS, a interação entre os modelo TMN e SNMP.

As ferramentas citadas nesta seção oferecem suporte a determinadas MIBs, mas fornecem meios para o aprimoramento e integração do adaptador Q a novas MIBs proprietárias ou padronizadas. Como a plataforma OSIMIS, estes adaptadores exigem que as estruturas SNMP sejam previamente convertidas em estruturas GDMO e então integradas ao sistema de gerência.

O *Solstice TMN/SNMP Q-Adaptor* inclui suporte à MIB II e SNMPv1 [Sun 99]. Também estão incorporadas as funcionalidades requeridas pelo Gerenciamento de Objetos (ISO 10165-1), Gerenciamento de Estados (ISO 10165-2), Relatório de Alarmes (ISO 10165-4), Filtro de Eventos (ISO 10165-5) e Monitoração de Atributos

(ISO 10165-11). Implementa adicionalmente a conversão de *traps* SNMP em notificações CMIP, recurso não encontrado na plataforma OSIMIS.

5.4. Biblioteca CMU SNMP

Construída na *Carnegie Mellon University*, esta biblioteca é composta por um conjunto de funções e procedimentos que implementam o envio e a busca de informações em MIBs SNMP. Este conjunto de funções não objetiva converter as informações obtidas em um formato compatível com TMN, mas pode ser utilizado pelo elemento de rede ou pelo mecanismo de interface para interagir diretamente com as bases de informações de gerenciamento presentes nos recursos reais.

A biblioteca CMU SNMP foi construída em C++ e está disponível para as plataformas Solaris, Irix, Linux, Microsoft Windows NT, Microsoft Windows 95 e SCO Unix [CMU 99]. As funções de acesso à MIB são compatíveis com SNMP versões 1 e 2. Juntamente com esta biblioteca são oferecidas aplicações para a interação com interfaces SNMP:

- `snmpget`: implementa o envio da primitiva SNMP GET à MIB SNMP;
- `snmpgetnext`: envio de uma solicitação SNMP GETNEXT;
- `snmpwalk`: percorre todas as variáveis da MIB retornando seu resultado ao sistema de gerência executando sucessivos SNMP GETNEXT;
- `snmptrap`: cria *traps* SNMP compatíveis com a versão 1 do SNMP;
- `snmptrap2`: emissão de *traps* SNMP compatíveis com a versão 2 do SNMP;
- `snmptrapd`: servidor responsável pelo recebimento das *traps* provenientes dos recursos e agentes SNMP;

- `snmpstatus`: Obtém o estado de um equipamento ou sistema presente no ambiente gerenciado.

6 TMN EM REDES COM MIBS SNMP

O conjunto de atributos dos objetos implementados tem como objetivo monitorar os descritores de tráfego, controlar células OAM e gerenciar a qualidade de serviço e estado administrativo de canais e caminhos virtuais. Como descrito no capítulo 4, foram implementados dois tipos de classes para o gerenciamento destas conexões virtuais. O primeiro é constituído pelas classes `vpCTPBidirectional` e `vcCTPBidirectional` e monitora segmentos de canais e caminhos virtuais. O segundo compõe a interface com *trails* e abrange as classes `vpTTPBidirectional` e `vcTTPBidirectional`.

Apenas os objetos de interface com os segmentos de canais e caminhos virtuais realizam o controle da qualidade de serviço e dos descritores de tráfego. Os objetos `vpTTPBidirectional` e `vcTTPBidirectional` implementam o controle de células OAM, assim como os objetos de interface com segmentos.

Como os objetos apresentados não implementam a comunicação direta com o *hardware* dos recursos reais, as MIBs SNMP presentes nos equipamentos gerenciados constituem a interface entre os comutadores e o sistema de gerência. Desta forma, para realizar as atividades de gerenciamento, os objetos devem obter os dados necessários e implementar o controle dos comutadores ATM com a manipulação destas bases de informações de gerenciamento.

As MIBs SNMP definidas nos documento RFC 1695 (*Request for Comments - Definitions of Managed Objects for ATM Management version 8)* e ILMI (*Integrated Local Management Interface Specification version 4*) foram analisadas para obter os

dados necessários aos objetos implementados neste trabalho. A base de informações proposta pelo documento RFC 1695 contém dados relevantes ao controle de tráfego presente nos objetos gerenciados TMN. As operações definidas em G.atmm envolvendo células OAM podem ser emuladas por atributos presentes na MIB definida em ILMI.

A simples utilização das MIBs SNMP poderia atender a requisitos de gerência TMN inclusos nas classes implementadas, porém imporia um maior grau de dificuldade como conseqüência da falta de mecanismos de abstração. Estes mecanismos podem ser implementados em aplicativos que utilizam dados das bases de informações SNMP para compor visões abstratas dos recursos e conexões estabelecidas. Esta abordagem é utilizada neste trabalho, contudo, a aplicação desenvolvida atende a padrões de organismos internacionais.

6.1. Descrição do ambiente de testes

O Núcleo de Processamento de Dados (NPD) da Universidade Federal de Santa Catarina (UFSC) dispõe de um ambiente de redes ATM em uso e parte desta rede está disponível para a realização de testes. Os comutadores que compõem a rede UFSC possuem as bases de informações necessárias a este trabalho, possibilitando a utilização do ambiente para a verificação da funcionalidade dos objetos construídos. O conjunto de equipamentos disponíveis para testes está conectado ao restante da rede UFSC por uma linha de 155 Mbps (Figura 6.1). Duas linhas de 155 Mbps conectam o comutador ligado à rede UFSC a outros dois comutadores ATM.

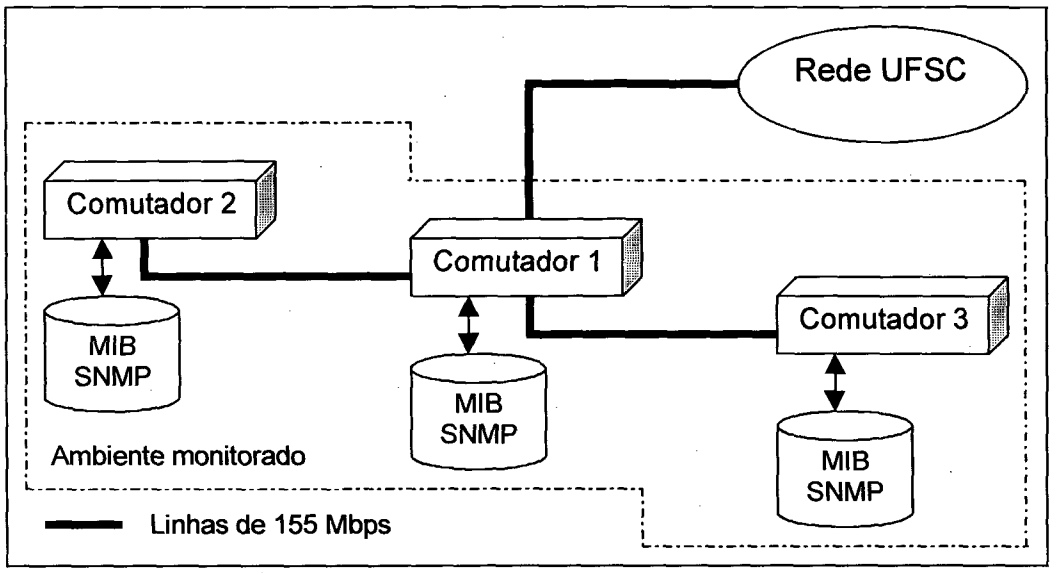

Figura 6.1 - Topologia do ambiente ATM disponível para testes na rede UFSC.

Para a representação dos canais e caminhos virtuais, as MIBs SNMP também utilizam o conceito de enlace topológico do modelo TMN. Segundo [RFC 1695], uma conexão virtual em um ambiente ATM consiste em uma série de enlaces virtuais concatenados. A união dos enlaces ocorre em comutadores ATM (Figura 6.2). Os enlaces virtuais que formam uma VCC (*Virtual Channel Connection*) são denominados VCL (*Virtual Channel Link*) e uma VPC (*Virtual Path Connection*) é o resultado da concatenação de VPLs (*Virtual Path Links*).

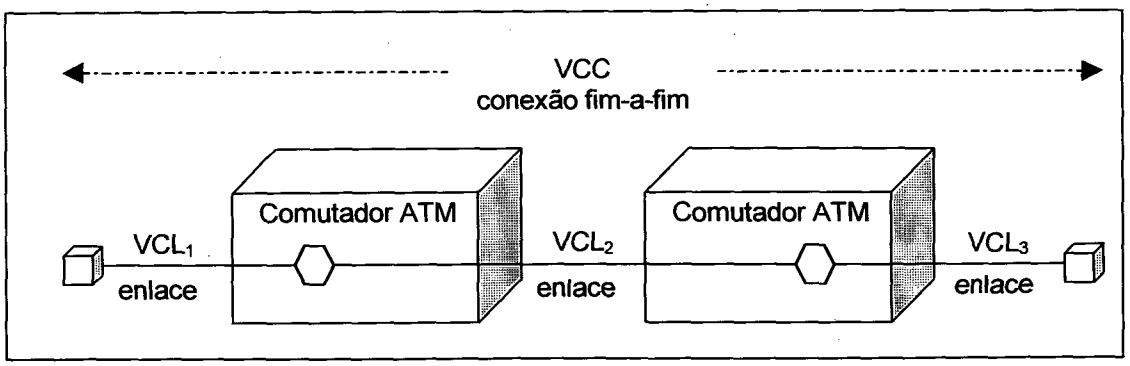

Figura 6.2 – Enlaces e Conexões de Canal Virtual.

Os repositórios de informações SNMP estudados e presentes nos equipamentos ATM disponíveis para teste não suportam o gerenciamento completo do sistema, pois fornecem informações apenas sobre o equipamento onde estão instalados e, em alguns casos, sobre equipamentos imediatamente adjacentes. Adicionalmente, estas bases de informações representam recursos físicos, assim os VCLs e VPLs iniciam e terminam em dispositivos ATM. No gerenciamento TMN, um elemento de rede não representa obrigatoriamente um único equipamento, podendo modelar diversos dispositivos interligados. Sob esta abordagem, as conexões de enlace podem ser delimitadas por equipamentos ou sub-redes, o que permite que apenas as informações relevantes sejam apresentadas ao gerente da rede.

6.2. Conversibilidade entre os modelos de informação Internet e TMN

A primeira etapa para a utilização das MIBs SNMP consiste no mapeamento das informações disponíveis em atributos dos objetos gerenciados TMN e o estabelecimento de uma equivalência entre os dados disponíveis nos dois ambientes. No modelo de gerência SNMP as informações são armazenadas em tabelas, porém na arquitetura TMN os recursos e serviços são representados com a utilização de classes de objetos gerenciados.

Os objetos TMN responsáveis pelo controle de enlaces que compõem uma conexão fim-a-fim, as instâncias das classes `vpCTPBidirectional` e `vcCTPBidirectional`, devem monitorar os VCLs e VPLs definidos em [RFC 1695]. As conexões de canais e caminhos virtuais são monitoradas por objetos de *trail* dos padrões G.atmm e M.3100. A MIB proposta em RFC 1695 contém atributos

que identificam os enlaces que compõem uma conexão fim-a-fim. Estes atributos permitem ao mecanismo de interface informar ao elemento de rede os *trails* estabelecidos e associá-los a objetos gerenciados.

Esta alternativa representa a rede gerenciada no seu menor nível de abstração, onde as conexões são delimitadas por equipamentos físicos. A composição de visões mais genéricas da rede é realizada pela união de comutadores que, segundo a representação, atuam como um único comutador distribuído. Estes comutadores são então monitorados por um único objeto TMN, com mecanismos para manipular as bases de informações SNMP dos equipamentos.

A base de informações RFC 1695 descreve os enlaces de canais e caminhos virtuais nas tabelas `atmVplTable` e `atmVclTable` respectivamente. A estrutura da tabela de VPLs é apresentada na Figura 6.3 e a tabela de VCLs é descrita na Figura 6.4.

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

Figura 6.3 - Estrutura da Tabela de Enlaces de Caminho Virtual.

O atributo `atmVplVpi` da tabela `atmVplTable` (Figura 6.3) identifica o caminho virtual monitorado. A coluna `atmVplAdminStatus`, que pode conter os

valores up(1) ou down(1), é relevante apenas para enlaces que terminam uma conexão fim-a-fim. Os valores atribuídos a esta coluna indicam, respectivamente, que o fluxo de dados está habilitado ou desabilitado. A indicação do funcionamento do enlace é armazenada em atmVplOperStatus. Os valores que podem ser atribuídos a esta variável são: up(1), o enlace está operacional; down(2), fora de operação; e unknown(3), a MIB não pode determinar o estado do enlace. A data armazenada em atmVplLastChange indica o momento da alteração mais recente do estado operacional do VPL.

A descrição dos parâmetros de tráfego da conexão estão em atmVplReceiveTrafficDescrIndex e atmVplTransmitTrafficDescr Index. As informações contidas nestes atributos possuem a mesma sintaxe e indicam as linhas da tabela de parâmetros de tráfego que descrevem a conexão. O primeiro atributo descreve o fluxo de informações em direção ao comutador monitorado e o segundo contém informações sobre o tráfego originado pelo equipamento. Os VPLs que compõem um determinado VPC são identificados pelo atributo atmVplCrossConnectIdentifier. Se dois VPLs pertencem ao mesmo caminho virtual, devem possuir o mesmo valor neste atributo.

A tabela de VCLs contém todos os atributos da tabela de VPLs acrescidos do identificador do enlace de caminho virtual e de outros atributos não utilizados por este trabalho. Os atributos que compõem a tabela de enlaces podem ser observados na Figura 6.4.

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

Figura 6.4 - Estrutura da Tabela de Enlaces de Canal Virtual.

As tabelas de enlaces virtuais utilizam a tabela `atmTrafficDescrParam Table` para armazenar os parâmetros de tráfego e a qualidade de serviço (QoS) associada. Os atributos `atmVclReceiveTrafficDescrIndex`, `atmVplReceive TrafficDescrIndex` (assim como os de transmissão) indicam a linha desta tabela que descreve os parâmetros de tráfego da conexão. A estrutura da tabela é apresentada na Figura 6.5.

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

Figura 6.5 - Estrutura da Tabela de Parâmetros Descritores de Tráfego.

Os atributos `atmTrafficDescrParamX` possuem interpretações distintas de acordo com o tipo de conexão que descrevem. A semântica destes atributos é

definida por `atmTrafficDescrType`. O documento RFC 1695 define os seguintes valores para `atmTrafficDescrType`:

- `atmNoTrafficDescriptor`: nenhum dos parâmetros é utilizado, este valor descreve conexões que exigem a largura de banda disponível;
- `atmNoClpNoScr`: os parâmetros de tráfego descrevem conexões que não exigem largura de banda mínima (SCR - *Sustainable Cell Rate*) e onde não há prioridade para descarte de células (CLP - *Cell Loss Priority*). Apenas o primeiro parâmetro é utilizado e indica a taxa máxima de transferência da conexão em células por segundo;
- `atmNoClpScr`: conexões deste tipo exigem uma largura mínima de banda, mas não utilizam prioridade para descarte de células. O primeiro parâmetro contém a taxa máxima de transferência em células por segundo. O parâmetro dois define a taxa mínima de transferência exigida pela conexão. O último parâmetro descreve o número máximo de células que constituem uma rajada;
- `atmClpNoTaggingNoScr`: descreve enlaces caracterizados pela implementação da prioridade para descarte de células e por não exigirem uma taxa mínima de transferência. Estas conexões não alteram o indicador de prioridade de descarte de células. Apenas os dois primeiros parâmetros possuem relevância e contém, respectivamente, a taxa máxima do conjunto total de células e o pico da taxa de transferência para células com prioridade de envio (CLP=0). Ambos os valores são expressos em células por segundo;

- `atmClpTaggingNoScr`: equivalente à descrição anterior, mas as células com prioridade de envio que excedam a taxa máxima de transferência definida têm a prioridade de envio reduzida. A utilização dos parâmetros é idêntica à anterior;

- `atmClpNoTaggingScr`: o usuário da conexão exige uma taxa mínima de transferência. Também é utilizado o controle de prioridade de descarte, mas sem modificações no indicador de prioridades das células. O primeiro parâmetro contém o pico da taxa de transferência para o conjunto total das células. No segundo parâmetro está a largura de banda exigida para as células com prioridade de envio (CLP=0). O número máximo de células em uma rajada é obtido no atributo `atmTrafficDescrParam3`.

- `atmClpTaggingScr`: conexões semelhantes àquelas identificadas por `atmCLPNoTaggingScr`. No entanto, as células com prioridade de envio que ultrapassem a taxa de transferência definida por `atmTrafficDescrParam2` são marcadas como prioritárias para descarte. A utilização dos parâmetros é idêntica à anterior.

As informações obtidas nas tabelas do modelo de informações SNMP apresentadas devem ser mapeadas para as classes de objetos gerenciados definidas pela arquitetura informacional do modelo TMN. No gerenciamento TMN, as conexões e os enlaces de canais e caminhos virtuais são representados por objetos gerenciados organizados em uma árvore de *containment* onde as conexões estão localizadas sob o elemento de rede monitorado. Como mencionado, os atributos de controle de tráfego do documento G.atmm pertencem apenas aos objetos que

representam enlaces virtuais e constituem o pacote opcional `trafficDescriptorPkg`, cuja sintaxe é apresentada na Figura 6.6.

```
trafficDescriptorPkg PACKAGE
ATTRIBUTES
        ingressPeakCellRate          GET-REPLACE    ADD-REMOVE,
        egressPeakCellRate           GET-REPLACE    ADD-REMOVE,
        ingressCDVTolerance          GET-REPLACE    ADD-REMOVE,
        egressCDVTolerance           GET-REPLACE    ADD-REMOVE,
        ingressSustainableCellRate   GET-REPLACE    ADD-REMOVE,
        egressSustainableCellRate    GET-REPLACE    ADD-REMOVE,
        ingressBurstTolerance        GET-REPLACE    ADD-REMOVE,
        egressBurstTolerance         GET-REPLACE    ADD-REMOVE;
REGISTERED AS {};
```

Figura 6.6 - Especificação GDMO do Pacote `trafficDescriptorPkg`.

De forma análoga às tabelas das bases de informações SNMP, os objetos TMN também representam conexões bidirecionais. Os atributos são replicados para definir o tráfego de entrada e saída em relação ao nó monitorado. Ao nome da característica são adicionadas, como prefixos, as partículas `ingress` e `egress` para a formação do nome do atributo. A sintaxe destes é constituída por dois inteiros, o primeiro é utilizado para o conjunto total de células (CLP=0+1) e o segundo caracteriza o tráfego das células com prioridade de envio (CLP=0). A formação do nome das variáveis que compõem a seqüência é dada pela adição, como sufixos, das partículas CLP0plus1, para o primeiro e CLP0, para o segundo. A especificação ASN.1 dos atributos `ingressPeakCellRate` e `egressPeakCell Rate` é apresentado na Figura 6.7.

```
PeakCellRate ::= Sequence      {
        peakCellRateCLP0Plus1  [1] INTEGER OPTIONAL,
        peakCellRateCLP0       [2] INTEGER OPTIONAL
                               }
```

Figura 6.7 - Sintaxe de Atributos `PeakCellRate`.

Estas classes foram definidas pelo ATM Fórum para atender a especificação da UNI (*User Network Interface*) e são correspondentes àquelas do modelo TMN (`class1`, `class2`, `class3` e `class4` respectivamente).

6.3. Implementação do mecanismo de interface

A construção do mecanismo de interface proposto pode ser dividida em três diferentes aspectos: a pesquisa em MIBs SNMP, a representação local das informações obtidas e a composição da visão requerida pelo sistema de gerência TMN. As classes que implementam este mecanismo foram desenvolvidas com a utilização da linguagem C++ e posteriormente integradas à plataforma OSIMIS.

Como diversas soluções de gerenciamento de redes apresentam ferramentas que possibilitam a troca de informações com MIBs SNMP, a implementação dos métodos de interação com estas MIBs torna-se dependente da plataforma de gerência adotada. Consequentemente, a definição destes métodos deve ser realizada durante a integração do mecanismo ao ambiente gerenciado. Neste trabalho, a comunicação é realizada pela utilização do *gateway* SNMPv1/CMIP da plataforma OSIMIS e da troca de arquivos com aplicativos de consulta a MIBs SNMP.

Para tornar o mecanismo de interface independente dos métodos de consulta a MIBs SNMP, as informações relevantes ao gerenciamento TMN são armazenadas localmente em estruturas de dados semelhantes àquelas descritas em RFC 1695. Este isolamento tem como objetivo evitar que adaptações nos procedimentos de consulta sejam a causa de comportamentos não previstos e requeiram reimplementações em outros módulos do mecanismo.

A representação local das conexões estabelecidas pelos comutadores é implementada por duas listas de instâncias da classe `ATMLink`: uma lista para as conexões de caminho virtual e outra para os canais virtuais. A interface desta classe é apresentada pela Figura 6.9.

```
class AtmLink {
public:
  char MyIpAddress[16];
  char MyNeighborIpAddress[16];
  int Vpi;
  int Vci;
  int AdminStatus;
  int OperStatus;
  int CrossConnectIdentifier;
  TrafficDescriptor receiveParameters;
  TrafficDescriptor sendParameters;
  inline AtmLink(void); }
```

Figura 6.9 - Interface da Classe `AtmLink`.

Os atributos `MyIpAddress` e `MyNeighborIpAddress` da classe `AtmLink` (Figura 6.9) contêm os endereços IP dos comutadores associados pela conexão. Estes valores são obtidos na tabela `atmInterfaceConfTable` da MIB proposta em RFC 1695. Os atributos `Vpi` e `Vci` são os identificadores de caminho e canal virtual da conexão, entretanto o valor contido em `Vci` é ignorado nas instâncias de `AtmLink` que representam um caminho virtual. Os atributos `OperStatus` e `AdminStatus` indicam o estado da conexão, porém o atributo `AdminStatus` está presente apenas em enlaces que delimitam um *trail*. `CrossConnectIdentifier` permite identificar os enlaces que compõem um *trail*. Estes valores são obtidos nas tabelas `AtmVplTable` e `AtmVclTable`.

Para descrever determinados tipos de conexões apenas um sub-conjunto dos atributos do pacote `trafficDescriptorPkg` é necessário. Devido a inexistência de atributos que caracterizem a tolerância à variação de atraso no tráfego de dados nas base de informações SNMP pesquisadas, pode-se estabelecer que nenhuma das conexões passíveis de gerenciamento neste trabalho exige os atributos `ingressCDVTolerance` e `egressCDVTolerance`.

```
qosClassPkg PACKAGE
    ATTRIBUTES
        ingressQOSClass
            GET,
        egressQOSClass
            GET;
REGISTERED AS {atmPackage 4};
```

Figura 6.8 - Especificação GDMO do Pacote `qosClassPkg`.

A classe de qualidade de serviço exigida pelo usuário para a conexão é descrita, no ambiente TMN, por um atributo do pacote `qosClassPkg`, apresentado na Figura 6.8. Os atributos `ingressQOSClass` e `egressQOSClass` podem conter os seguintes valores: `Class1`, `Class2`, `Class3` e `Class4`. Nas bases de informações SNMP, a classe de qualidade de serviço é representada pela coluna `atmTrafficQoSClass` na tabela `atmTrafficDescrParamTable`. Os valores permitidos nesta coluna e sua interpretação estão na Tabela 6.1.

Tabela 6.1 - Classes de Qualidade de Serviço

Valor	Descrição
1	Serviço de Classe A – Vídeo de taxa constante e emulação de circuito
2	Serviço de Classe B – Vídeo ou áudio com taxa variável
3	Serviço de Classe C – Serviços transmissão de dados orientados à conexão
4	Serviço de Classe D – Serviços transmissão de dados não orientados à conexão

Os atributos `ReceiveParameters` e `SendParameters` são instâncias da classe `TrafficDescriptor` e contêm os descritores de tráfego da conexão. A classe `TrafficDescriptor` é apresentada na Figura 6.10 e possui a mesma estrutura da tabela `atmTrafficDescrParamTable` (Figura 6.5).

```
class TrafficDescriptor {
public:
  int atmTrafficDescrType;
  int atmTrafficDescrParam1;
  int atmTrafficDescrParam2;
  int atmTrafficDescrParam3;
  int atmTrafficDescrParam4;
  int atmTrafficDescrParam5;
  int atmQoSClass;
  inline TrafficDescriptor(void);
}
```

Figura 6.10 - Interface da Classe `TrafficDescriptor`.

O atributo `atmTrafficDescrType` da classe apresentada na Figura 6.10 define a interpretação dos outros atributos descritores de tráfego (`atmTrafficDescrParam1` a `atmTrafficDescrParam5`). A qualidade de serviço da conexão é mantida em `atmQoSClass`. As informações disponíveis nestas classes são convertidas e disponibilizadas ao sistema de gerência TMN em listas de instâncias da classe `TMNLink`, cuja interface é descrita pela Figura 6.11.

```
class TMNLink {
  public:
   int Vpi;
   int Vci;
   int OperStatus;
   int Delimitador;
   TMNTrafficAttribute PeakCellRate;
   TMNTrafficAttribute QoSClassTMN;
   TMNTrafficAttribute SustainedCellRate;
   TMNTrafficAttribute MaxBurstSize;
  TMNLink(void);
 }
```

Figura 6.11 - Interface da Classe `TMNLink`.

Os atributos `Vpi`, `Vci` e `OperStatus` da classe `TMNLink` (Figura 6.11) são obtidos diretamente dos atributos `Vpi`, `Vci` e `OperStatus` da classe `AtmLink` (Figura 6.9). O atributo Delimitador determina se o enlace representado delimita um *trail*. O valor deste atributo é determinado pela disponibilidade do atributo `AdminStatus` na classe `AtmLink`.

Os atributos `PeakCellRate`, `QoSClassTMN`, `SustainedCellRate` e `MaxBurstSize` contém os descritores de tráfego com uma sintaxe semelhante àquela apresentada pelo modelo TMN. Estas estruturas são instâncias da classe `TMNTrafficAttribute` e descrevem atributos para conexões bidirecionais, com diferenciação de prioridade de envio entre células (`CLP0` e `CLP0Plus1`). A Figura 6.12 descreve a classe `TMNTrafficAtribute`.

```
class TMNTrafficAttribute
{
  public:
  int egressCLP0;
  int egressCLP0Plus1;
  int ingressCLP0;
  int ingressCLP0Plus1;
  inline TMNTrafficAttribute(void);
}
```

Figura 6.12 - Classe `TMNTrafficAttribute`.

Os valores atribuídos aos descritores de tráfego TMN (Figura 6.12) são obtidos pela interpretação do conteúdo dos parâmetros de tráfego SNMP de acordo com o tipo de descritor de tráfego (Figura 6.10). A Figura 6.13 contém um trecho do arquivo `switch.cc` onde é implementada parte da função de conversão.

```
switch(atmlink->sendParameters.atmTrafficDescrType)
 {
   ...
   case atmClpNoTaggingNoScr:
    tmnlink->PeakCellRate.egressCLP0Plus1 =
        atmlink->sendParameters.atmTrafficDescrParam1;
    tmnlink->PeakCellRate.egressCLP0 =
        atmlink->sendParameters.atmTrafficDescrParam2;
    break;
   ... };
```

Figura 6.13 - Interpretação de Parâmetros de Tráfego SNMP.

O tipo de descritor de parâmetro de tráfego SNMP `atmClpNoTaggingNoScr` (Figura 6.13) determina a interpretação do primeiro parâmetro do descritor como a taxa máxima de células que podem ser enviadas nesta conexão (CLP=0+1). O segundo parâmetro determina a largura de banda máxima, em células por segundo, que pode ser ocupada por células com prioridade de envio (CLP=0). Estes valores

são então atribuídos a uma instância da classe `TMNLink` que será posteriormente enviada ao elemento de rede.

O comportamento do mecanismo de interface implementado está na classe `Switch`, cuja interface é apresentada na Figura 6.14. Os métodos presentes nesta classe são responsáveis pelo acesso à MIB SNMP e o seu armazenamento nas estruturas de dados locais. Posteriormente, estes dados são convertidos ao formato TMN solicitado pelo sistema de gerência e enviados ao elemento de rede.

Os endereços dos equipamentos que participam do nó gerenciado são enviados à instância da classe `Switch` durante a sua criação. O construtor da classe, método `Switch::Switch(char *IpAddresses)`, recebe como parâmetro uma string contendo a lista de endereços IP dos equipamentos. Para alterar a composição do nó é necessário reiniciar o mecanismo de interface.

```
class Switch
{
public:
  char MyIpAddresses[MAXSWITCHESINNODE][16];
  Switch(char *IpAddresses);
  ~Switch(void);
  int iniciaLinks(void);
  int setTrafficParameter(char * ip, int index, int param, int value);
  int setInterfaceParameter(char * ip, int index, char * ipDest);
  TMNLinkList * getVcConnections(void);
  TMNLinkList * getVpConnections(void);
};
```

Figura 6.14 - Especificação da Classe `Switch`.

Os dados são obtidos das MIBs SNMP pelo método `Switch::iniciaLinks(void)` (Figura 6.14) e enviados ao gerente TMN por chamadas aos métodos `Switch::getVcConnections(void)` e `Switch::get`

VpConnections(void). O destrutor da classe, método Switch:: ~Switch(void), é responsável pela liberação da memória reservada pelas estruturas de dados utilizadas.

Uma instância da classe Switch é responsável pela interface com um nó gerenciado. Na integração com o ambiente de testes da rede UFSC, uma instância desta classe foi adicionada á lista de atributos da classe NetworkElement, responsável pela monitoração de um nó no ambiente TMN. Durante a criação de uma instância NetworkElement, o mecanismo de interface correspondente deve ser iniciado. O construtor da classe Switch pesquisa as MIBs SNMP e envia a lista de conexões estabelecidas ao elemento de rede. O trecho do arquivo NetworkElementR1.inc.cc, onde a instância de Switch é inicializada, é apresentado na Figura 6.15.

```
while (link!=NULL)
 {
  miblink = vpCTPBidirectional->create(NULL,self);
  if (miblink != NULL)
  {
   miblink->internalVpId = link->Vpi;
   //********** Ingress peak cell rate
   AtmMIBMod_PeakCellRate.peakCellRateCLPplus1 =
      link->PeakCellRate.ingressCLP0Plus1;
   AtmMIBMod_PeakCellRate.peakCellRateCLP0    = ;
      link->PeakCellRate.ingressCLP0;
   miblink->ingressPeakCellRate->set(& AtmMIBMod_PeakCellRate);
  ...
 }}
```

Figura 6.15 – Inclusão do Objeto Gerenciado na MIB TMN.

Para cada membro presente nas listas retornadas pelas funções getVpConnections e getVcConnections (Figura 6.14) o elemento de rede deve inserir na MIB TMN objetos gerenciados das classes vpCTPBidirectional e

vcCTPBidirectional. Os descritores de tráfego contidos nas instâncias de TMNLink são então atribuídos aos objetos gerenciados e apresentados na MIB ao gerente de rede.

7 APLICAÇÃO DO MECANISMO DE INTERFACE NO AMIBENTE DE TESTES

Como descrito no quinto capítulo, os métodos de obtenção de dados em MIBs SNMP do mecanismo de interface devem ser especializados para utilizar os recursos disponíveis na plataforma de gerência escolhida. Deste modo, o primeiro estágio da implantação do mecanismo consiste na adaptação dos procedimentos de consulta a estas MIBs. A seção 7.1 descreve a implementação realizada para possibilitar a utilização do mecanismo de interface no ambiente de testes.

A segunda fase da implantação abrange a definição do nível de abstração exigido para a gerência da rede. Neste trabalho, duas diferentes composições de nós gerenciados foram utilizadas para a realização de testes. Na primeira representação, todos os comutadores são individualmente gerenciados e a segunda configuração define o agrupamento dos três equipamentos em um único comutador distribuído. Estas duas representações são descritas nas seções 7.2 e 7.3, respectivamente.

7.1. Comunicação com as MIBs SNMP

A plataforma utilizada nesta implementação, OSIMIS TMN, oferece o *Gateway* SNMPv1/CMIP que, em conjunto com o mecanismo apresentado na seção 5.2, pode ser utilizado na obtenção de dados em bases de informações de gerenciamento SNMP. Porém, a MIB proposta pelo documento RFC 1695 foi definida para a utilização com SNMPv2, o que impossibilita a sua leitura pelo *Gateway* SNMPv1/CMIP.

A utilização do adaptador Q da plataforma OSIMIS para a obtenção de dados da MIB proposta em RFC1695 exigiria a sua atualização para torná-lo compatível com a nova versão do protocolo SNMP, tarefa que não está no escopo deste trabalho. Contudo, este adaptador pode ser utilizado para a obtenção de informações presentes em MIBs compatíveis com SNMPv1.

A pesquisa das informações da MIB RFC 1695 é realizada por um aplicativo implementado em Java que, por meio de arquivos, disponibiliza estas informações ao mecanismo de interface proposto. Em períodos de tempos configuráveis, o mecanismo de interface busca novos arquivos contendo informações atualizadas sobre o estado dos comutadores. A Figura 7.1 contém um trecho do arquivo gerado pela aplicação Java responsável pela interação com a MIB SNMP.

```
.1.3.6.1.2.1.37.1.7.1.3.1.0.34: up (1)
.1.3.6.1.2.1.37.1.7.1.4.1.0.34: up (1)
.1.3.6.1.2.1.37.1.7.1.5.601.0.34: 8564600
.1.3.6.1.2.1.37.1.7.1.6.1.0.34: 4
.1.3.6.1.2.1.37.1.7.1.6.601.0.34: 3
```

Figura 7.1 - Trecho do arquivo gerado pelo aplicativo Java.

As informações presentes no arquivo enviado pelo aplicativo Java são obtidas nas tabelas `atmVplTable`, `atmVclTable`, `atmTrafficDescrParamTable` e `atmInterfaceConfTable` e seguem o formato: "identificador da variável (OID): valor" (Figura 7.1). Em termos gerais, este arquivo contém as informações disponíveis na MIB sobre as conexões estabelecidas no ambiente monitorado.

Cada nova conexão encontrada no arquivo de troca implica na criação de uma nova estrutura de representação de enlace no mecanismo de interface. Os dados de tráfego são associados à estrutura de representação tornando-os

disponíveis para a conversão em estruturas compatíveis com o modelo TMN, definidas pela classe `TMNLink` apresentada na Figura 6.10.

7.2. Gerenciamento individual de comutadores

Esta configuração provê meios para o gerenciamento individual dos equipamentos e serviços da rede, característica que a inclui no nível de gerenciamento de elemento de rede (NEML) proposto pela arquitetura funcional do modelo TMN. O detalhamento requerido pelo NEML é alcançado com a criação de objetos da classe `networkElementR1` associados a cada um dos comutadores presentes no ambiente gerenciado.

A cada um dos objetos `networkElementR1` está associada uma instância do mecanismo de interface. Esta instância é responsável pela obtenção da lista de conexões estabelecidas pelo comutador monitorado. Como todos os equipamentos da rede estão individualmente associados a objetos gerenciados, todos os canais e caminhos virtuais do ambiente de testes são associados a objetos na MIB TMN.

A Figura 7.2 descreve uma configuração de canais e caminhos virtuais estabelecidos no ambiente de testes e os objetos gerenciados associados aos recursos e conexões monitorados. Sob a visão proposta nesta seção, cada um dos comutadores é representado na MIB TMN por uma instância de `NetworkElementR1`, indicada na Figura 7.2 pelo nome *network element*.

Neste cenário, há dois caminhos virtuais delimitados pelos comutadores 1 e 2. Sob a perspectiva do segundo comutador, duas instâncias da classe `vpCTPBidirectional` são criadas para monitorar estas conexões. Na hierarquia

de *containment*, estes objetos estão localizados sob o elemento de rede associado ao comutador 2. O mesmo processo é realizado para as conexões delimitadas pelos outros dois equipamentos do ambiente de testes.

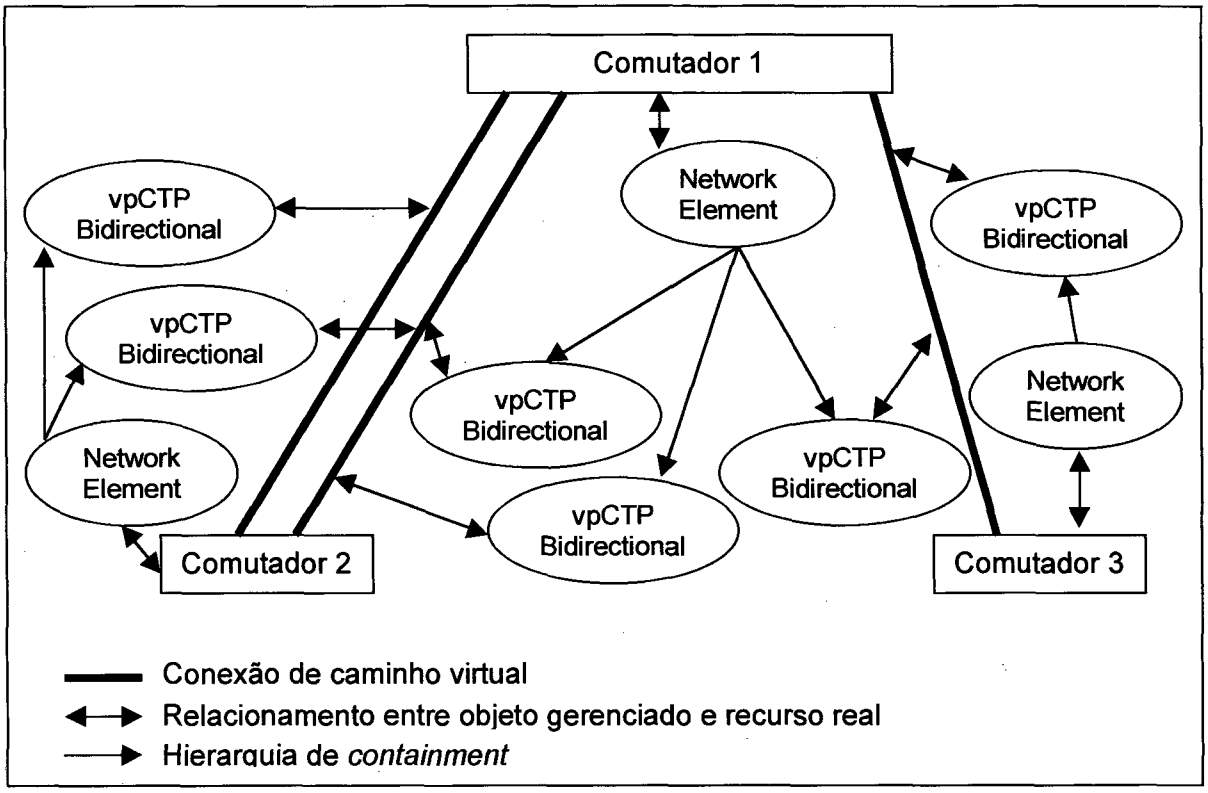

Figura 7.2 - Representação do ambiente de testes.

De acordo com este processo e como indicado na Figura 7.2, os enlaces estabelecidos entre comutadores do ambiente gerenciado são monitorados por duas instâncias de objetos. Esta duplicidade tem como objetivo possibilitar a gerência de conexões sob a perspectiva de ambos os comutadores.

7.3. Constituição de comutadores distribuídos

Esta configuração do mecanismo de interface possibilita o gerenciamento de rede pelo agrupamento de objetos gerenciados e portanto está no nível de gerenciamento de rede (NML) da arquitetura funcional do modelo TMN. Para possibilitar esta visão da rede, o mecanismo de interface atua na interação entre

uma instância da classe `NetworkElementR1` e um conjunto de comutadores para que estes sejam representados na MIB TMN como um único equipamento distribuído.

Sob esta visão, conforme a Figura 7.3, apenas um objeto `NetworkElementR1` foi instanciado para monitorar os três comutadores do ambiente de testes. As conexões estabelecidas entre os comutadores não são visíveis ao elemento de rede e, consequentemente, não estão presentes na MIB TMN. Somente os enlaces estabelecidos entre comutadores do nó gerenciado e equipamentos da Rede UFSC são enviados ao elemento de rede pelo mecanismo de interface.

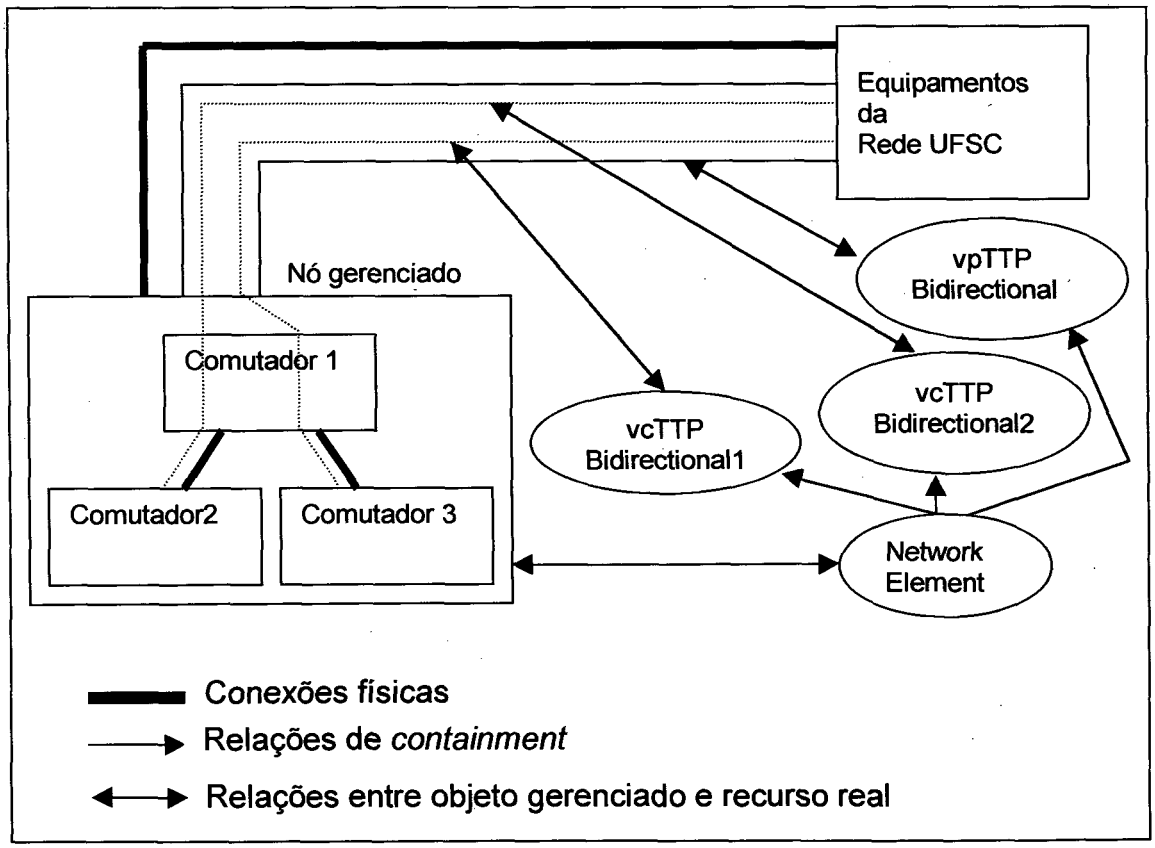

Figura 7.3 - Representação do ambiente de testes como um único comutador distribuído.

As operações solicitadas pelos objetos `vcTTPBidirectional` e `vpTTPBidirectional` que monitoram as conexões do nó gerenciado (Figura 7.3) devem refletir, quando necessário, nas conexões internas ao nó e não visíveis à MIB TMN. Para efetuar estas operações corretamente, o mecanismo de interface identifica os enlaces concatenados à conexão monitorada pela utilização do atributo `atmCrossConnectIdentifier` e efetua a operação solicitada.

Assim, uma alteração da qualidade de serviço solicitada pelo objeto `vcTTPBidirectional1` não deve ficar restrita ao canal virtual apresentado ao elemento de rede. Conforme a descrição da Figura 7.3, este canal virtual é parte de um *trail* que, no ambiente conhecido pelo mecanismo de interface, se estende do comutador 2 à Rede UFSC. Portanto a nova qualidade de serviço deve ser também atribuída ao canal virtual delimitado pelos comutadores 1 e 2.

8 CONCLUSÕES E TRABALHOS FUTUROS

A crescente demanda por serviços mais complexos, com diferentes requisitos de qualidade de transmissão, tem causado o aumento da utilização de redes de grande velocidade e confiabilidade. Dentre estas, podem-se destacar as redes ATM, objeto de estudo deste trabalho. Os serviços providos pelas redes ATM são orientados à conexão e possibilitam o controle de alocação de recursos consumidos por usuários pela monitoração das conexões estabelecidas.

Como apresentado neste trabalho, o modelo de gerência de redes de telecomunicação TMN definido pela ITU-T apresenta arquiteturas funcionais e físicas abrangentes e flexíveis que facilitam sua aplicação na gerência de redes ATM. As recomendações M.3100 e G.atmm definem objetos gerenciados que foram implementados para atuar nesta tarefa. Grande parte dos equipamentos utilizados em redes ATM possuem MIBs compatíveis com o modelo *Internet*. Portanto, os investimentos já feitos sobre o SNMP não podem ser desconsiderados, sob o risco de tornar o sistema de gerência incompatível com os ambientes existentes.

A plataforma OSIMIS TMN, utilizada para a implementação deste trabalho, apresenta interfaces que facilitam o desenvolvimento de aplicações de gerenciamento. Além de estar em conformidade com os conceitos de gerência OSI, esta plataforma permite a interoperabilidade com outros sistemas. Com esta facilidade de interconexão, a OSIMIS segue a tendência atual de um ambiente multi-protocolar para o gerenciamento de redes heterogêneas.

Para atuar na gerência de redes ATM segundo os princípios de gerenciamento TMN foram implementados objetos gerenciados especificados nas recomendações M.3100 e G.atmm, cujo desenvolvimento foi apresentado neste trabalho. Os objetos construídos possuem atributos e ações que permitem o controle e monitoração de atividades nas conexões estabelecidas em redes ATM. O ambiente ATM analisado neste trabalho, assim como grande parte da base instalada, não possui interfaces TMN. Desta forma, as atividades não estiveram restritas à construção destes objetos e incluíram o desenvolvimento de uma interface de conversão de informações entre os modelo TMN e *Internet*.

8.1. Resultados Obtidos

Este trabalho apresentou o desenvolvimento de mecanismos para a aplicação da funcionalidade do modelo TMN a redes de alta velocidade com MIBs especificadas pelo modelo de gerência *Internet*. Esta implementação não restringe a coexistência destes dois modelo à simples apresentação de dados definidos pelo modelo de gerência *Internet* em plataformas TMN, mas possibilita ao gerente de rede aplicar os conceitos definidos pelo modelo TMN na administração de recursos com interfaces SNMP.

Como grande parte dos recursos ATM são fornecidos com modelos de informação compatíveis com SNMP, a integração destes equipamentos a um sistema de gerência TMN exige o desenvolvimento de *drivers* apropriados para a comunicação com o *hardware*. O mecanismo proposto neste trabalho torna esta tarefa desnecessária, pois a gerencia TMN pode ser aplicada sobre o modelo de informações *Internet* sem perda de funcionalidade.

A aplicação da funcionalidade TMN em redes que agregam recursos com interfaces SNMP e a redução dos requisitos necessários à integração de um novo equipamento ao sistema gerência são as duas principais contribuições deste trabalho.

8.2. Trabalhos Futuros

Como mencionado no capítulo 7, a presente implementação do mecanismo de interface pode ser classificada como pertencente aos níveis de gerenciamento de elemento de rede (NEML) e gerenciamento de rede (NML) propostos pela arquitetura funcional TMN. Porém, a composição de nós disponibilizada pelo mecanismo está restrita a união de comutadores que disponham das MIBs propostas por RFC 1695 e ILMI. Sugere-se que futuros trabalhos tenham como objetivo a integração do mecanismo a outros modelos de gerência, como TL1.

Pode-se também propor a integração do mecanismo de interface a outros sistemas para incluir o sistema de gerência nos dois níveis mais elevados da hierarquia funcional TMN. Os objetos do mecanismo de interface podem ser integrados aos objetos especificados pela Função de Medida de Uso do modelo OSI, cuja implementação é descrita em [RoKo 96] e desta forma atender a requisitos do gerenciamento de serviços.

Os dados resultantes da integração proposta poderão então ser utilizados por sistemas de ERP (*Enterprise Resource Planning*), como SAP e BaanERP. A interação entre dados fornecidos pelo sistema de gerência e ferramentas ERP podem compor uma importante ferramenta de apoio à decisões.

9 REFERÊNCIAS BIBLIOGRÁFICAS

ILMI ATM FORUM. **Integrated Management Interface Specification Version 4**, af-ilmi-0065.000. Mountain View, EUA, 1996.

RFC1695 BELL COMMUNICATIONS RESEARCH. **Definitions of Managed Objects for ATM Management using SMIv2**, RFC 1695. Red Bank, EUA, 1994.

BRISA BRISA. **Gerenciamento de Redes: Uma Abordagem de Sistemas Abertos**. São Paulo: Editora Makron Books, 1993. 364p.

CMU 99 CARNEGIE MELLON UNIVERSITY. **CMU SNMP Library**. Disponível na internet. http://www.net.cmu.edu/projects/snmp/. 25 de julho de 1999.

GoNo 95 GOULART, Carlos de C., NOGUEIRA, José Marcos Silva. Utilização do Modelo TMN no Gerenciamento de Redes ATM. In: Simpósio Brasileiro de Rede de Computadores, 13, Belo Horizonte – MG, 1995. **Anais do 13º Simpósio Brasileiro de Redes de Computadores**. Belo Horizonte: Universidade Federal de Minas Gerais, 1995. 655p. p.389-408

G.803 ITU-T. **Architectures of Transport Networks on the Synchronous Digital Hierarchy**, G.803. Helsinki, Finlândia, 1993.

G.atmm ITU-T. **Draft Recommendation**: Asynchronous Transfer Mode (ATM) Management of the Network Element View, G.atmm. Genebra, Suíça, 1995.

M.3100 ITU-T. **Generic Network Model Information**, M3100. Genebra, Suíça, 1995.

KorRo97a KORMANN, Luiz F., ROGERIO, Ketter O., WESTAPHALL, Carlos B. Construction of a TMN Network Element for the Configuration Management of ATM Networks. In: Simpósio Brasileiro de Redes de Computadores, 15, São Carlos – SP, 1997. **Anais do 15° Simpósio Brasileiro de Redes de Computadores**. São Carlos: Universidade Federal de São Carlos, 1997. 554p. p. 447-456.

KorRo97b KORMANN, Luiz F., ROGERIO, Ketter O., WESTPHALL, Carlos B. TMN Network Element for the Configuration Management of a ATM Networks: Implementation Aspects. In: Seminário Integrado de Software e Hardware – SEMISH, 24, Brasília – DF, 1997. **Anais do XXIV SEMISH**. Brasília: Universidade de Brasília, 1997. 556p. p. 203-214.

Korm97 KORMANN, Luiz Fernando. **Novas Funções de Gerência para Redes de Telecomunicações e de Alta Velocidade Usando a Plataforma OSIMIS**. Florianópolis, 1997. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina.

McPa MCCARTHY, K., PAVLOU, G., BHATTI, S., SOUZA, José N. **Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways**. University College Lodon.

MiRa97 MITTMANN, R., RASCHOWETZKI, Ariadne M. et al. **Implementação de um Elemento de Rede TMN para Gerência de Recursos ATM**. Resumo Submetido ao Seminário Franco-Brasileiro, Fortaleza - CE, Novembro 1997.

MuMa95 MUSSI, Clarissa C.; Mazorca, Vivianne N.F. **Desenvolvimento de Novos Processos para Gerência de Redes**. Relatório de Pesquisa – Projeto PLAGERE, Universidade Federal de Santa Catarina, dezembro 1995.

Open99 OPENCON SYSTEMS. **OCS TMN Gateway**: CMISE, SNMP, TL1 translation. Disponível na internet. http://www.opencon.inter.net/tmnwhite.htm. 30 de junho de 1999.

RoKo96 ROGERIO, Ketter Ohnes; KORMANN, Luiz Fernando; CORRÊA, Álvaro Sampaio Neto et. al. Implementação de Novos Serviços de Contabilização para a Plataforma OSIMIS. In: Seminário Integrado de Software e Hardware – SEMISH, 23, Recife – PE, 1996. **Anais do XXIII SEMISH**. Recife: Universidade Federal de Pernambuco, 1996. 564p. p. 493-504.

SoLe95 SOARES, Luiz Fernando; LEMOS, Guido; COLCHER, Sérgio; **Redes de Computadores – Das LANs, MANs e WANs às Redes ATM**. Rio de Janeiro: Editora Campus LTDA, 1995. 705pp.

SoMa93 SOUZA, J. N.; MACCARTHY, K.; AGOULMINE, N.; PAVLOU, G.; CMIP/SNMPv1 Translation Through Application Level Gateways Using the OSIMIS/ISODE Platform. In: RACE Conference on Intelligence Services and Networks, Paris, França, 1993. **Proceedings of the RACE Conference on Intelligence Services and Networks**. Paris, França: 1993. v. 4: p. 1-16.

Sun 99 SUN MICROSYSTEMS. **Solstice(TM) TMN/SNMP Q-Adaptor**. Disponível na internet. http://www.sun.com/software/white-papers/wp-tmnproducts. 25 de julho de 1999.

TL1 99 **TL1**. Disponível na internet. http://www.tl1.com. 25 de julho de 1999.

10 ANEXOS

10.1. Especificação em GDMO dos Objetos Gerenciados da Recomendação M.3100

```
-- ############ Managed Object templates ###################
net MANAGED OBJECT CLASS
     DERIVED FROM "Recommendation X.721:1992":top;
     CHARACTERIZED BY
    CONDITIONAL PACKAGES
         userLabelPackage,
-- ---------------------------------------
netPackage PACKAGE
     BEHAVIOUR networkDefinition;
     ATTRIBUTES
          networkId       GET;;;
  CONDITIONAL PACKAGES
  userLabelPackage
     PRESENT IF "an instance support it";

REGISTERED AS {m3100ObjectClass 1};

-------------------------------------------------------------------------
managedElement  MANAGED OBJECT CLASS
     DERIVED FROM "Recommendation X.721:1992":top;
     CHARACTERIZED BY
 Conditional Packages
          createDeleteNotificationsPackage,
          attributeValueChangeNotificationPackage,
          stateChangeNotificationPackage,
          audibleVisualLocalAlarmPackage,
          resetAudibleAlarmPackage,
          userLabelPackage,
          vendorNamePackage,
          versionPackage,
          locationNamePackage,
          currentProblemListPackage,
          externalTimePackage,
          systemTimingSourcePackage,
-- ---------------------------------------------

managedElementPackage PACKAGE
     BEHAVIOUR managedElementBehaviour;
     ATTRIBUTES
          managedElementId                    GET,
          systemTitle                                   GET-REPLACE,
          alarmStatus                    GET,
          administrativeState                           GET-REPLACE,
          operationalState                    GET,
          usageState                                    GET;;;
     NOTIFICATIONS
          "Recommendation X.721:1992":environmentAlarm,
          "Recommendation X.721:1992":equipamentAlarm,
          "Recommendation X.721:1992":communicationsAlarm,
          "Recommendation X.721:1992":processingErrorAlarm;;;
```

```
CONDITIONAL PACKAGES
createDeleteNotificationsPackage
      PRESENT IF "the objectCreation and objectDeletion notifications defined in Recommendation X.721 are
supported by an instance of this class.",
attributeValueChangeNotificationsPackage
     PRESENT IF "the attributeValueChange notifications defined in Recommendation X.721 are supported by
an instance of this class.",
stateChangeNotificationPackage
     PRESENT IF "the stateChange notifications defined in Recommendation X.721 are supported by an
instance of this class.",
audibleVisualLocalAlarmPackage
     PRESENT IF "an instance supports it.",
resetAudibleAlarmPackage
     PRESENT IF "an instance supports it.",
userLabelPackage
     PRESENT IF "an instance supports it.",
vendorNamePackage
     PRESENT IF "an instance supports it.",
versionPackage
     PRESENT IF "an instance supports it.",
locationNamePackage
     PRESENT IF "an instance supports it.",
currentProblemListPackage
     PRESENT IF "an instance supports it.",
externalTimePackage
     PRESENT IF "an instance supports it.",
systemTimingSourcePackage
     PRESENT IF "an instance supports it.";

REGISTERED AS {m3100ObjectClass 3};

-------------------------------------------------------------------------------
managedElementR1        MANAGED OBJECT CLASS
        DERIVED FROM        managedElement;
        CHARACTERIZED BY
 Conditional Packages
        alarmSeverityAssignmentPointerPackage,
-- ------------------------------------------------
managedElementR1Package        PACKAGE;;
          NOTIFICATIONS
                "Recommendation X.721:1992":environmentalAlarm
                        "Recommendation Q.821:1992":logRecordIdParameter
                        "Recommendation Q.821:1992":correlatedRecordNameParameter
                        "Recommendation Q.821:1992":suspectObjectListParameter,
                "Recommendation X.721:1992":equipmentAlarm
                        "Recommendation Q.821:1992":logRecordIdParameter
                        "Recommendation Q.821:1992":correlatedRecordNameParameter
                        "Recommendation Q.821:1992":suspectObjectListParameter,
                "Recommendation X.721:1992":communicationsAlarm
                        "Recommendation Q.821:1992":logRecordIdParameter
                        "Recommendation Q.821:1992":correlatedRecordNameParameter
                        "Recommendation Q.821:1992":suspectObjectListParameter,
                "Recommendation X.721:1992":processingErrorAlarm
                        "Recommendation Q.821:1992":logRecordIdParameter
                        "Recommendation Q.821:1992":correlatedRecordNameParameter
                        "Recommendation Q.821:1992":suspectObjectListParameter;;;
 CONDITIONAL PACKAGES
        alarmSeverityAssignmentPointerPackage
```

```
 PRESENT IF "The managed object supports configuration of alarm severities";

REGISTERED AS {m3100ObjectClass 27};
-------------------------------------------------------------------------------
terminationPoint      MANAGED OBJECT CLASS
     DERIVED FROM   "Recommendation X.721:1992":top;
     CHARACTERIZED BY
 Conditional Packages
          createDeleteNotificationsPackage,
     attributeValueChangeNotificationPackage,
          stateChangeNotificationPackage,
          operationalStatePackage,
          crossConnectionPointerPackage,
          characteristicInformationPackage,
          networkLevelPackage,
          tmnCommunicationsAlarmInformationPackage,
     alarmSeverityAssignmentPointerPackage,
-- ------------------------------------------------------
terminationPointPackage PACKAGE
     BEHAVIOUR terminationPointBehaviour;
     ATTRIBUTES
          supportedByObjectList   GET;;;
 CONDITIONAL PACKAGES
 createDeleteNotificationsPackage
     PRESENT IF "the objectCreation and objectDeletion notifications defined in Recommendation X.721 are
supported by an instance of this class.",
 attributeValueChangeNotificationsPackage
     PRESENT IF "the attributeValueChange notifications defined in Recommendation X.721 are supported by
an instance of this class.",
 stateChangeNotificationPackage
     PRESENT IF "the stateChange notifications defined in Recommendation X.721 are supported by an
instance of this class.",
 operationalStatePackage
     PRESENT IF "the resource represented by this managed object is capable of assessing the ability to
generate and/or receive a valid signal.",
 crossConnectionPointerPackage
     PRESENT IF "the termination point can be flexibly assigned, (i.e. cross connected).",
 characteristicInformationPackage
     PRESENT IF "an instance supports it.",
 networkLevelPackage
     PRESENT IF "an instance supports it.",
 tmnCommunicationsAlarmInformationPackage
     PRESENT IF "the communicationsAlarm notification (as defined in Recommendation X.721) is supported
by this managed object.",
 alarmSeverityAssignmentPointerPackage
     PRESENT IF "the tmnCommunicationsAlarmInformationPackage package is present AND the managed
object supports configuration of alarm severities.";

REGISTERED AS {m3100ObjectClass 8};

-------------------------------------------------------------------------------
connectionTerminationPointSink  MANAGED OBJECT CLASS
     DERIVED FROM   terminationPoint;
     CHARACTERIZED BY
 Conditional Packages
          ctpInstancePackage,
          channelNumberPackage,
-- ------------------------------------------
```

```
        connectionTerminationPointSinkPackage    PACKAGE

    BEHAVIOUR connectionTerminationPointSinkBehaviour;
    ATTRIBUTES
        downstreamConnectivityPointer  PERMITTED VALUES

  ASN1DefinedTypesModule.CTPDownstreamPointer    GET SET-BY-CREATE;;;


CONDITIONAL PACKAGES
ctpInstancePackage
    PRESENT IF "the name binding used to create an instance of this object class requires this attribute.",
channelNumberPackage
    PRESENT IF "an instance supports it.";

REGISTERED AS {m3100ObjectClass 6};

-------------------------------------------------------------------------------
connectionTerminationPointSource      MANAGED OBJECT CLASS
    DERIVED FROM    terminationPoint;
    CHARACTERIZED BY
Conditional Packages
        ctpInstancePackage,
        channelNumberPackage,
-- ----------------------------------------
connectionTerminationPointSourcePackage PACKAGE
    BEHAVIOUR connectionTerminationPointSourceBehaviour;
    ATTRIBUTES
        upstreamConnectivityPointer    PERMITTED VALUES
        CTPDownstreamPointer
ASN1DefinedTypesModule.CTPUpstreamPointer      GET SET-BY-CREATE;;;
CONDITIONAL PACKAGES
ctpInstancePackage
    PRESENT IF "the name binding used to create an instance of this object class requires this attribute.",
channelNumberPackage
    PRESENT IF "an instance supports it.";

REGISTERED AS {m3100ObjectClass 7};


-------------------------------------------------------------------------------
connectionTerminationPointBidirectional MANAGED OBJECT CLASS
    DERIVED FROM    connectionTerminationPointSource;
            connectionTerminationPointSink;

REGISTERED AS {m3100ObjectClass 5};

-------------------------------------------------------------------------------------------
trailTerminationPointSink      MANAGED OBJECT CLASS
    DERIVED FROM    terminationPoint;
    CHARACTERIZED BY
Conditional Packages
        administrativeStatePackage,
        supportableClientListPackage,
        ttpInstancePackage,
-- ----------------------------------
        operationalStatePackage,
```

```
    trailTerminationPointSinkPackage PACKAGE
    BEHAVIOUR trailTerminationPointSinkBehaviour;

    ATTRIBUTES
        upstreamConnectivityPointer    GET SET-BY-CREATE;;;
CONDITIONAL PACKAGES
"Recommendation X.721: 1992":administrativeStatePackage
    PRESENT IF "the resource represented by the managed object is capable of being administratively placed
in and out of service.",
supportableClientListPackage
    PRESENT IF "the object class can support more than one type of client.",
ttpInstancePackage
    PRESENT IF "the name binding used to create an instance of this class requires this attribute.";

REGISTERED AS {m3100ObjectClass 10};

-------------------------------------------------------------------------------
trailTerminationPointSource    MANAGED OBJECT CLASS
    DERIVED FROM    terminationPoint;
    CHARACTERIZED BY
Conditional Packages
        administrativeStatePackage,
        supportableClientListPackage,
        ttpInstancePackage,
-- ------------------------------------------------
        operationalStatePackage,
    trailTerminationPointSourcePackage PACKAGE
    BEHAVIOUR trailTerminationPointSourceBehaviour;

    ATTRIBUTES
        downstreamConnectivityPointer    GET SET-BY-CREATE;;;
CONDITIONAL PACKAGES
"Recommendation X.721: 1992":administrativeStatePackage
    PRESENT IF "the resource represented by the managed object is capable of being administratively placed
in and out of service.",
supportableClientListPackage
    PRESENT IF "the object class can support more than one type of client.";
ttpInstancePackage
    PRESENT IF "the name binding used to create an instance of this class requires this attribute.";

REGISTERED AS {m3100ObjectClass 11};

-------------------------------------------------------------------------------
trailTerminationPointBidirectional    MANAGED OBJECT CLASS
    DERIVED FROM    trailTerminationPointSource;
            trailTerminationPointSink;
    CHARACTERIZED BY
trailTerminationPointBidirectionalPackage PACKAGE
    BEHAVIOUR trailTerminationPointBidirectionalBehaviour;;;

REGISTERED AS {m3100ObjectClass 9};

-- ############ Packages templates ##########################

administrativeStatePackage PACKAGE
        ATTRIBUTES
            administrativeState        GET-REPLACE;
REGISTERED AS {m3100Package 1};
```

```
alarmSeverityAssignmentPointerPackage PACKAGE
        ATTRIBUTES
                alarmSeverityAssignmentProfilePointer GET-REPLACE;
REGISTERED AS {m3100Package 3};

attributeValueChangeNotificationPackage   PACKAGE
        NOTIFICATIONS
                attributeValueChange;
REGISTERED AS {m3100Package 4};

audibleVisualLocalAlarmPackage PACKAGE
        ACTIONS
                allowAudibleVisualLocalAlarm,
                inhibitAudibleVisualLocalAlarm;
REGISTERED AS {m3100Package 5};


channelNumberPackage PACKAGE
        ATTRIBUTES
                channelNumber  GET;
REGISTERED AS {m3100Package 6};


characteristicInformationPackage PACKAGE
        ATTRIBUTES
                characteristicInformation  GET;
REGISTERED AS {m3100Package 7};


crossConnectionPointerPackage PACKAGE
        ATTRIBUTES
                crossConnectionObjectPointer      GET;
REGISTERED AS {m3100Package 11};

ctpInstancePackage PACKAGE
        ATTRIBUTES
                cTPId   GET SET-BY-CREATE;
REGISTERED AS {m3100Package 12};

createDeleteNotificationsPackage PACKAGE
        NOTIFICATIONS
                objectCreation,
                objectDeletion;
REGISTERED AS {m3100Package 10};

currentProblemListPackage PACKAGE
        ATTRIBUTES
                currentProblemList        GET;
REGISTERED AS {m3100Package 13};

externalTimePackage PACKAGE
        ATTRIBUTES
                externalTime     GET-REPLACE;
REGISTERED AS {m3100Package 16};

locationNamePackage PACKAGE
        ATTRIBUTES
```

```
                locationName    GET-REPLACE;
REGISTERED AS {m3100Package 17};

networkLevelPackage PACKAGE
        ATTRIBUTES
                networkLevelPointer     GET-REPLACE;
REGISTERED AS {m3100Package 18};

operationalStatePackage PACKAGE
        ATTRIBUTES
                operationalState  GET;
REGISTERED AS {m3100Package 19};

resetAudibleAlarmPackage PACKAGE
        ACTIONS
                "Recommendation Q.821: 1992":resetAudibleAlarm;
REGISTERED AS {m3100Package 23};

supportableClientListPackage PACKAGE
        ATTRIBUTES
                supportableClientList
                        GET;
REGISTERED AS {m3100Package 27};

stateChangeNotificationPackage    PACKAGE
        NOTIFICATIONS
                stateChange;
REGISTERED AS {m3100Package 28};

systemTimingSourcePackage PACKAGE
        ATTRIBUTES
                systemTimingSource
                        GET-REPLACE;
REGISTERED AS {m3100Package 29};

tmnCommunicationsAlarmInformationPackage PACKAGE
        ATTRIBUTES
                alarmStatus
                        GET,
                currentProblemList
                        GET;
        NOTIFICATIONS
                "Recommentdation X.721: 1992":communicationAlarm
                        "Recommentdation Q.821: 1992":logRecordIdParameter
                        "Recommentdation Q.821: 1992":correlatedRecordNameParameter
                        "Recommentdation Q.821: 1992":suspectObjectListParameter;
REGISTERED AS {m3100Package 30};

ttpInstancePackage PACKAGE
        ATTRIBUTES
                tTPId
                        GET SET-BY-CREATE;
REGISTERED AS {m3100Package 31};

userLabelPackage PACKAGE
        ATTRIBUTES
                userLabel
                        GET-REPLACE;
```

```
REGISTERED AS {m3100Package 32};

vendorNamePackage PACKAGE
        ATTRIBUTES
                vendorName
                        GET-REPLACE;
REGISTERED AS {m3100Package 33};

versionPackage PACKAGE
        ATTRIBUTES
                version
                        GET-REPLACE;
REGISTERED AS {m3100Package 34};


-- ############ Attributes templates ##########################

alarmSeverityAssignmentProfilePointer ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.PointerOrNull;
    MATCHES FOR EQUALITY;
        BEHAVIOUR alarmSeverityAssignmentPointerBehaviour;
REGISTERED AS {m3100Attribute 5};

alarmStatus ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.AlarmStatus;
        MATCHES FOR EQUALITY;
        BEHAVIOUR alarmStatusBehaviour;
REGISTERED AS {m3100Attribute 6};

channelNumber ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.ChannelNumber;
        MATCHES FOR EQUALITY, ORDERING;
REGISTERED AS {m3100Attribute 7};

characteristicInformation ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.CharacteristicInformation;
        MATCHES FOR EQUALITY;
        BEHAVIOUR characteristicInformationBehaviour;
REGISTERED AS {m3100Attribute 8};

cTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCL-.SimpleNameType;
        MATCHES FOR EQUALITY;
REGISTERED AS {m3100Attribute 13};

crossConnectionObjectPointer ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.CrossConnectionObjectPointer;
        MATCHES FOR EQUALITY;
        BEHAVIOUR crossConnectionObjectPointerBehaviour;
REGISTERED AS {m3100Attribute 16};

currentProblemList ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.CurrentProblemList;
        BEHAVIOUR currentProblemListBehaviour;
REGISTERED AS {m3100Attribute 17};

downstreamConnectivityPointer ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.DownstreamConnectivityPointer;
```

```
        MATCHES FOR EQUALITY, SET-COMPARISON, SET-INTERSECTION;
        BEHAVIOUR downstreamConnectivityPointerBehaviour;
REGISTERED AS {m3100Attribute 19};

externalTime ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.ExternalTime;
    MATCHES FOR EQUALITY;
        BEHAVIOUR externalTimeBehaviour;
REGISTERED AS {m3100Attribute 21};

locationName ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.LocationName;
        MATCHES FOR EQUALITY, SUBSTRINGS;
        BEHAVIOUR locationNameBehaviour;
REGISTERED AS {m3100Attribute 27};

managedElementId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
        MATCHES FOR EQUALITY;
        BEHAVIOUR managedElementIdBehaviour;
REGISTERED AS {m3100Attribute 28};

networkId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
        MATCHES FOR EQUALITY;
        BEHAVIOUR networkIdBehaviour;
REGISTERED AS {m3100Attribute 29};

networkLevelPointer ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.ObjectInstance;
        MATCHES FOR EQUALITY;
REGISTERED AS {m3100Attribute 31};

supportableClientList ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.SupportableClientList;
        MATCHES FOR EQUALITY, SET-COMPARISON, SET-INTERSECTION;
        BEHAVIOUR supportableClientListBehaviour;
REGISTERED AS {m3100Attribute 39};

supportedByObjectList ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.ObjectList;
        MATCHES FOR EQUALITY, SET-COMPARISON, SET-INTERSECTION;
        BEHAVIOUR supportedByObjectListBehaviour;
REGISTERED AS {m3100Attribute 40};

systemTimingSource ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.SystemTimingSource;
        MATCHES FOR EQUALITY;
        BEHAVIOUR systemTimingSourceBehaviour;
REGISTERED AS {m3100Attribute 41};

tTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
        MATCHES FOR EQUALITY;
        BEHAVIOUR tTPIdBehaviour;
REGISTERED AS {m3100Attribute 48};

upstreamConnectivityPointer ATTRIBUTE
```

```
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.ConnectivityPointer;
    MATCHES FOR EQUALITY;
        BEHAVIOUR upstreamConnectivityPointerBehaviour;
REGISTERED AS {m3100Attribute 49};

userLabel ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.UserLabel;
    MATCHES FOR EQUALITY, SUBSTRINGS;
        BEHAVIOUR userLabelBehaviour;
REGISTERED AS {m3100Attribute 50};

vendorName ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.VendorName;
    MATCHES FOR EQUALITY, SUBSTRINGS;
        BEHAVIOUR vendorNameBehaviour;
REGISTERED AS {m3100Attribute 51};

version ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1DefinedTypesModule.Version;
        MATCHES FOR EQUALITY, SUBSTRINGS;
        BEHAVIOUR versionBehaviour;
REGISTERED AS {m3100Attribute 52};


-- ############ Notifications and actions templates ##########################

allowAudibleVisualLocalAlarm ACTION
        BEHAVIOUR allowAudibleVisualLocalAlarmBehaviour;
REGISTERED AS {m3100Action 3};

inhibitAudibleVisualLocalAlarm ACTION
        BEHAVIOUR inhibitAudibleVisualLocalAlarmBehaviour;
REGISTERED AS {m3100Action 6};

-- Imported Notifications

objectCreation NOTIFICATION
  BEHAVIOUR             objectCreationBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator     sourceIndicator,
    attributeList       attributeList;
  REGISTERED AS         { smi2Notification 6 };


objectDeletion NOTIFICATION
  BEHAVIOUR             objectDeletionBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator     sourceIndicator,
    attributeList       attributeList;
  REGISTERED AS         { smi2Notification 7 };


attributeValueChange NOTIFICATION
  BEHAVIOUR             attributeValueChangeBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.AttrValChangeInfo
  AND ATTRIBUTE IDS
```

```
    sourceIndicator      sourceIndicator,
    attrValChangeDefinition attrValChangeDefinition;
  REGISTERED AS          { smi2Notification 1 };


stateChange NOTIFICATION
  BEHAVIOUR              stateChangeBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.AttrValChangeInfo
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    stateChangeDefmition stateChangeDefinition;
  REGISTERED AS          { smi2Notification 14 };


-- ############## Name Binding Templates ##############

net-net NAME BINDING
        SUBORDINATE OBJECT CLASS        net AND SUBCLASSES;
        NAMED BY
        SUPERIOR OBJECT CLASS           net AND SUBCLASSES;
        WITH ATTRIBUTE                  networkId;
        BEHAVIOUR                  networkCreateBehaviour;
REGISTERED AS {m3100NameBinding 17};

managedElement-net NAME BINDING
        SUBORDINATE OBJECT CLASS        managedElement;
        NAMED BY
        SUPERIOR OBJECT CLASS           net AND SUBCLASSES;
        WITH ATTRIBUTE                  managedElementId;
        BEHAVIOUR                  managedElementCreateBehaviour;
REGISTERED AS {m3100NameBinding 15};

trailTerminationPointSource-managedElement NAME BINDING
        SUBORDINATE OBJECT CLASS        trailTerminationPointSource AND SUBCLASSES;
        NAMED BY
        SUPERIOR OBJECT CLASS           managedElement AND SUBCLASSES;
        WITH ATTRIBUTE                  tTPId;
        BEHAVIOUR                  trailTerminationPointNameBindingBehaviour;
        CREATE                          WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                          ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 23};

trailTerminationPointSink-managedElement NAME BINDING
        SUBORDINATE OBJECT CLASS        trailTerminationPointSink AND SUBCLASSES;
        NAMED BY
        SUPERIOR OBJECT CLASS           managedElement AND SUBCLASSES;
        WITH ATTRIBUTE                  tTPId;
        BEHAVIOUR                  trailTerminationPointNameBindingBehaviour;
        CREATE                          WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                          ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 24};

connectionTerminationPointSource-trailTerminationPointSource NAME BINDING
        SUBORDINATE OBJECT CLASS        connectionTerminationPointSource;
        NAMED BY
        SUPERIOR OBJECT CLASS           trailTerminationPointSource AND SUBCLASSES;
        WITH ATTRIBUTE                  cTPId;
        BEHAVIOUR                  cTPSource-TTPBehaviour;
```

```
        CREATE                                  WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                                  ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 3};

connectionTerminationPointSource-trailTerminationPointBidirectional NAME BINDING
        SUBORDINATE OBJECT CLASS        connectionTerminationPointSource;
        NAMED BY
        SUPERIOR OBJECT CLASS           trailTerminationPointBidirectional AND SUBCLASSES;
        WITH ATTRIBUTE                  cTPId;
        BEHAVIOUR               cTPSource-TTPBehaviour;
        CREATE                          WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                          ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 4};

connectionTerminationPointSink-trailTerminationPointSink NAME BINDING
        SUBORDINATE OBJECT CLASS        connectionTerminationPointSink;
        NAMED BY
        SUPERIOR OBJECT CLASS           trailTerminationPointSink AND SUBCLASSES;
        WITH ATTRIBUTE                  cTPId;
        BEHAVIOUR               cTPSource-TTPBehaviour;
        CREATE                          WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                          ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 5};

connectionTerminationPointSink-trailTerminationPointBidirectional NAME BINDING
        SUBORDINATE OBJECT CLASS        connectionTerminationPointSink;
        NAMED BY
        SUPERIOR OBJECT CLASS           trailTerminationPointBidirectional AND SUBCLASSES;
        WITH ATTRIBUTE                  cTPId;
        BEHAVIOUR               cTPSource-TTPBehaviour;
        CREATE                          WITH-AUTOMATIC-INSTANCE-NAMING;
        DELETE                          ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS {m3100NameBinding 6};
```

10.2. Implementação de objetos da recomendação M.3100

10.2.1 Arquivo ManagedElement.inc.h

```
protected:
    int   buildReport (int, int, void*&, Bool&);
public:
        int createRR (AVA*&, void* = NULLVD, int = -1);
```

10.2.2 Arquivo ManagedElement.inc.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>

int managedElement::createRR (AVA*& err, void*, int) {
 static Bool classInitialised;
 {
  char* val = (char*) userLabel() -> get();
  if (!strlen(val)){
   userLabel() -> set("Network");
  }
  if (!classInitialised){
    _classInfo->setDefault(I_userLabel, new String("Network"));
  }
 }
 {
  char* val = (char*) vendorName() -> get();
  if (!strlen(val)){
   vendorName() -> set("Default Vendor");
  }
  if (!classInitialised){
   _classInfo->setDefault(I_vendorName, new String("Default Vendor"));
  }
 }
 {
  char* val = (char*) version() -> get();
  if (!strlen(val)){
   version() -> set("Default Version");
  }
  if (!classInitialised){
   _classInfo->setDefault(I_version, new String("Default Version"));
  }
 }
 {
  char* val = (char*) locationName() -> get();
  if (!strlen(val)){
   locationName() -> set("Default Anyware");
  }
  if (!classInitialised){
   _classInfo->setDefault(I_locationName, new String("Default Anyware"));
  }
 }

 {
```

```
  char* val = (char*) currentProblemList() -> get();
  if (!strlen(val)){
   currentProblemList()->set("{localValue: cleared}");
  }

  if (!classInitialised){
   _classInfo->setDefault(I_currentProblemList,
    new String("{localValue: cleared}"));
  }

  if (!classInitialised){
   struct type_ASN1DefinedTypesModule_CurrentProblemList *list;
   list = CurrentProblemList_parse("{alarmStatus  :activeReportable_IndeterminateprobableCause :localValue
:789}");
    if (list) {
      currentProblemList()->set(list);
     }
}
 if (!classInitialised){
  classInitialised = True;
 }
 return OK;
}

int managedElement::buildReport (int eventId, int classLevel,
                                 void*& info, Bool& freeFlag)
{
  if (classLevel != _level)
     return Top::buildReport(eventId, classLevel, info, freeFlag);
  freeFlag = False;  // not to free the event info for efficiency -
                     // it is free'd in the destructor
  switch (eventId) {
   case I_objectCreation:
     printf("A managedElementObject Instance was created ...\n");
     return Top::buildReport(MO_objectCreation, classLevel, info, freeFlag);
  case I_objectDeletion:
     printf("A managedElementObject Instance was deleted ...\n");
     return Top::buildReport(MO_objectDeletion, classLevel, info, freeFlag);
  case I_attributeValueChange:
     printf("A managedElementObject Instance had its value changed  ...\n");
     return Top::buildReport(MO_attributeValueChange, classLevel, info, freeFlag);
  case I_stateChange:
     printf("An managedElementObject Instance had its state changed ...\n");
     return Top::buildReport(MO_stateChange, classLevel, info, freeFlag);
  default:
     return NOTOK;
   };
   return OK;
}
```

10.2.3 Arquivo ManagedElementR1.inc.h

```
protected:
 static int _managedElementR1Id;
 inline static int GetNewManagedElementR1Id(){ return(++_managedElementR1Id );};
 int createVirtualChannels(TMNLinkList * list);
 int createVirtualPaths(TMNLinkList * list);
public:
```

```
 int createRR (AVA*&, void* = NULLVD, int = -1);
 int deleteRR (AVA*&, void* = NULLVD, Bool = False, int = -1);

 int wakeUp (char*);
 inline int _cancelWakeUps(char *wake) { return cancelWakeUps("manElemR1WakeUps");}
```

10.2.4 Arquivo managedElementR1.inc.cc

```
#include <stdio.h>
#include <vcCTPBidirectional.h>
#include <vpCTPBidirectional.h>
#include "switch.h"

managedElementR1::_managedElementR1Id = 0;

// The following function is called to create the RDN to a new
// object instace.

// ------------------------------------------------------------------
//                              RDNs
// ------------------------------------------------------------------
RDN managedElementR1::makeRdn (MO*)
{
 char buf[64];
 srintf(buf, "managedElementId=%d", GetNewManagedElementR1Id());
 return str2rdn(buf);
}; // managedElementR1::makeRdn

// ------------------------------------------------------------------
//                    POLLING METHODS
// ------------------------------------------------------------------

int managedElementR1::createRR(AVA*& err, void*, int)
{
 Switch *node=NULL;
 TMNLinkList * linklist;

 #if DEBUG
 printf ("managedElementR1::createRR called\n");
 #endif

 // Verifing inital configuration
 node = new Switch(myIps);
 link
 linklist = node->getVpConnections();
 createVirtualPaths(linklist);
 delete linklist;
 linklist = node->getVcConnectcions();
 createVirtualChannels(linklist);

 // Scheduler start-up
 scheduleWakeUps(5, "manElemR1WakeUps", FALSE);     // for notifications
 return OK;
}; // managedElementR1::createRR

int managedElementR1::deleteRR(AVA*& err, void* info, Bool checkOnly, int)
{
 if (!checkOnly)
```

```
  cancelWakeUps ("manElemR1WakeUps");
 return Top::deleteRR(err, info, checkOnly); // correct OO-style
}; // managedElementR1::deleteRR

int managedElementR1::wakeUp(char*)
{
 #if DEBUG
 printf("The method wakeUp from managedElementR1=%d was called\n", managedElementId);
 #endif
 return OK;
}; //managedElementR1::wakeUp

// ------------------------------------------------------------------
//                  LOOKING FOR LINKS IN SNMP MIBS
// ------------------------------------------------------------------
int managedElementR1::createVirtualPaths(TMNLinkList * list)
{
 type_AtmMIBMod_PeakCellRate AtmMIBMod_PeakCellRate;
 type_AtmMIBMod_QosClass AtmMIBMod_QosClass;
 type_AtmMIBMod_SustainableCellRate AtmMIBMod_SustainableCellRate;
 type_AtmMIBMod_BurstTolerance AtmMIBMod_BurstTolerance;
 type_AtmMIBMod_CDVTolerance AtmMIBMod_CDVTolerance;

 TMNLink * link=NULL;
 list->startGetNext();
 link = list->getNext();
 vpCTPBidirectional * miblink= NULL;

 while (link!=NULL)
 {
  miblink = vpCTPBidirectional->create(NULL,self);
  if (miblink != NULL)
  {
   miblink->internalVpId = link->Vpi;
   //********** Ingress peak cell rate
   AtmMIBMod_PeakCellRate.peakCellRateCLPplus1 =
     link->PeakCellRate.ingressCLP0Plus1;
   AtmMIBMod_PeakCellRate.peakCellRateCLP0    = ;
     link->PeakCellRate.ingressCLP0;
   miblink->ingressPeakCellRate->set(& AtmMIBMod_PeakCellRate);

   //********** Egress peak cell rate
   AtmMIBMod_PeakCellRate.peakCellRateCLPplus1 =
     link->PeakCellRate.egressCLP0Plus1;
   AtmMIBMod_PeakCellRate.peakCellRateCLP0    = ;
     link->PeakCellRate.egressCLP0;
   miblink->egressPeakCellRate->set(& AtmMIBMod_PeakCellRate);

   //********** Ingress Sustanable cell rate
   AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0plus1 =
     link->SustainedCellRate.ingressCLP0Plus1;
   AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0    = ;
     link->SustainedCellRate.ingressCLP0;
   miblink->ingressSustainableCellRate->set(& AtmMIBMod_SustainableCellRate);

   // ********** Egress Sustanable cell rate
   AtmMIBMod_SustainableCellRate.SustainableCellRateCLPplus1 =
     link->SustainedCellRate.egressCLP0Plus1;
```

```
    AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0    = ;
      link->SustainedCellRate.egressCLP0;
    miblink->egressSustainableCellRate->set(& AtmMIBMod_SustainableCellRate);

    //********** Ingress Burst Tolerance
    AtmMIBMod_BurstTolerance.BurstToleranceCLP0plus1 =
      link->MaxBurstSize.ingressCLP0Plus1;
    AtmMIBMod_BurstTolerance.BurstToleranceRateCLP0    = ;
      link->MaxBurstSize.ingressCLP0;
    miblink->ingressBurstTolerance->set(& AtmMIBMod_BurstTolerance);

    // ********** Egress Burst Tolerance
    AtmMIBMod_BurstTolerance.SustainableCellRateCLPplus1 =
      link->MaxBurstSize.egressCLP0Plus1;
    AtmMIBMod_BurstTolerance.SustainableCellRateCLP0    = ;
      link->MaxBurstSize.egressCLP0;
    miblink->egressBurstTolerance->set(& AtmMIBMod_BurstTolerance);

    //********** Ingress QoS Class
    AtmMIBMod_QosClass.parm =
      link->QoSClassTMN.ingressCLP0Plus1 - 1;
    miblink->ingressQOSClass->set(& AtmMIBMod_QosClass);

    // ********** Egress QoS Class
    AtmMIBMod_QosClass.parm =
      link->QoSClassTMN.egresssCLP0Plus1 - 1;
    miblink->egressQOSClass->set(& AtmMIBMod_QosClass);

    link = list->getNext();
   };
  };
};

int managedElementR1::createVirtualChannels(TMNLinkList * list)
{
 type_AtmMIBMod_PeakCellRate AtmMIBMod_PeakCellRate;
 type_AtmMIBMod_QosClass AtmMIBMod_QosClass;
 type_AtmMIBMod_SustainableCellRate AtmMIBMod_SustainableCellRate;
 type_AtmMIBMod_BurstTolerance AtmMIBMod_BurstTolerance;
 type_AtmMIBMod_CDVTolerance AtmMIBMod_CDVTolerance;

 TMNLink * link=NULL;
 list->startGetNext();
 link = list->getNext();
 vpCTPBidirectional * miblink= NULL;

  while (link!=NULL)
  {
   miblink = vcCTPBidirectional->create(NULL,self);
   if (miblink != NULL)
   {
    miblink->internalVpId = link->Vpi;
    miblink->internalVcId = link->Vci;
    //********** Ingress peak cell rate
    AtmMIBMod_PeakCellRate.peakCellRateCLPplus1 =
      link->PeakCellRate.ingressCLP0Plus1;
    AtmMIBMod_PeakCellRate.peakCellRateCLP0    = ;
      link->PeakCellRate.ingressCLP0;
```

```
    miblink->ingressPeakCellRate->set(& AtmMIBMod_PeakCellRate);

    //********** Egress peak cell rate
    AtmMIBMod_PeakCellRate.peakCellRateCLPplus1 =
      link->PeakCellRate.egressCLP0Plus1;
    AtmMIBMod_PeakCellRate.peakCellRateCLP0    = ;
      link->PeakCellRate.egressCLP0;
    miblink->egressPeakCellRate->set(& AtmMIBMod_PeakCellRate);

    //********** Ingress Sustanable cell rate
    AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0plus1 =
      link->SustainedCellRate.ingressCLP0Plus1;
    AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0    = ;
      link->SustainedCellRate.ingressCLP0;
    miblink->ingressSustainableCellRate->set(& AtmMIBMod_SustainableCellRate);

    // ********** Egress Sustanable cell rate
    AtmMIBMod_SustainableCellRate.SustainableCellRateCLPplus1 =
      link->SustainedCellRate.egressCLP0Plus1;
    AtmMIBMod_SustainableCellRate.SustainableCellRateCLP0    = ;
      link->SustainedCellRate.egressCLP0;
    miblink->egressSustainableCellRate->set(& AtmMIBMod_SustainableCellRate);

    //********** Ingress Burst Tolerance
    AtmMIBMod_BurstTolerance.BurstToleranceCLP0plus1 =
      link->MaxBurstSize.ingressCLP0Plus1;
    AtmMIBMod_BurstTolerance.BurstToleranceRateCLP0    = ;
      link->MaxBurstSize.ingressCLP0;
    miblink->ingressBurstTolerance->set(& AtmMIBMod_BurstTolerance);

    // ********** Egress Burst Tolerance
    AtmMIBMod_BurstTolerance.SustainableCellRateCLPplus1 =
      link->MaxBurstSize.egressCLP0Plus1;
    AtmMIBMod_BurstTolerance.SustainableCellRateCLP0    = ;
      link->MaxBurstSize.egressCLP0;
    miblink->egressBurstTolerance->set(& AtmMIBMod_BurstTolerance);

    //********** Ingress QoS Class
    AtmMIBMod_QosClass.parm =
      link->QoSClassTMN.ingressCLP0Plus1 - 1;
    miblink->ingressQOSClass->set(& AtmMIBMod_QosClass);

    // ********** Egress QoS Class
    AtmMIBMod_QosClass.parm =
      link->QoSClassTMN.egresssCLP0Plus1 - 1;
    miblink->egressQOSClass->set(& AtmMIBMod_QosClass);
    link = list->getNext();
  };
 };
};
```

10.3. Especificação em GDMO dos Objetos da Recomendação G.atmm

```
-- ############ Managed Object templates ###################
tcAdaptorTTPBidirectional MANAGED OBJECT CLASS
        DERIVED FROM "ITU-T M.3100:1992":trailTerminationPointBidirectional;
        CHARACTERIZED BY

 Conditional Packages
                 alarmSeverityAssignmentPointerPackage,
                cellScramblingEnabledPkg,
                 tmnCommunicationAlarmInformationPackage,
                 createDeleteNotificationPackage,
                 stateChangeNotificationPackage,

tcAdaptorTTPBidirectionalPkg PACKAGE
     BEHAVIOUR tcAdaptorTTPBidirectionalBeh;
     ATTRIBUTES
          tcTTPId       GET;;;
 CONDITIONAL PACKAGES
 "ITU-T M.3100:1992":alarmSeverityAssignmentPointerPackage
          PRESENT IF "the managed objct supports configuration of alarm severity",
 cellScramblingEnabledPkg
          PRESENT IF "cell scrambling may be activated and deactivated for the supporting ATM interface.";
REGISTERED AS {atmManagedObjectClass 1};

------------------------------------------------------------------------

vcCTPBidirectional MANAGED OBJECT CLASS
        DERIVED FROM "ITU-T M.3100:1992":
                connectionTerminationPointBidirectional;
        CHARACTERIZED BY

 Conditional Packages
trafficDescriptorPkg,
oamTrafficDescriptorPkg,
 "ITU-T M.3100:1992":AdministrativeStatePkg,
qosClassPkg,
oamCellLoopbackPkg,
        "ITU-T M.3100:1992":attributeValueChangeNotificationPackage,
        "ITU-T M.3100:1992":createDeleteNotificationsPackage,
        "ITU-T M.3100:1992":crossConnectionPointerPackage,
        vcCTPBidirectionalPkg PACKAGE
                BEHAVIOUR vcCTPBidirectionalBeh;
                ATTRIBUTES
                        vcCTPId                 GET,
                        segmentEndPointDEFAULT VALUE AEmMod.booleanFalseDefault
                                        GET-REPLACE;;;

 CONDITIONAL PACKAGES
 trafficDescriptorPkg
 PRESENT IF "This package must be present when the upstreamConnectivityPointer and
downstreamConnectivityPointer attributes point to an instance of the vcTTPBidirectional object class.",
 oamTrafficDescriptorPkg
 PRESENT IF "This package must be present when the   upstreamConnectivityPointer attributes point to an
instance of the vcTTPBidirectional object class.",
 "ITU-T M.3100:1992":AdministrativeStatePkg
 PRESENT IF "supported by the Network Element",
```

```
qosClassPkg
 PRESENT IF "QOS Class Information is supplied by the managing system",
oamCellLoopbackPkg
 PRESENT IF "the link termination point supports OAM cell Loopbacks";
REGISTERED AS {atmManagedObjectClass 2};

--------------------------------------------------------------------------
vcTTPBidirectional MANAGED OBJECT CLASS
        DERIVED FROM "ITU-T M.3100:1992":trailTerminationPointBidirectional;
        CHARACTERIZED BY
 Conditional Packages
   oamCellLoopbackPkg,
  "Rec. X.721|ISO/IEC 10165-2":administrativeStatePackage,
  "ITU-T M.3100:1992":attributeValueChangeNotificationPackage,
  "ITU-T M.3100:1992":createDeleteNotificationPackage,

vcTTPBidirectionalPkg PACKAGE
     BEHAVIOUR vcTTPBidirectionalBeh;
     ATTRIBUTES
          vcTTPId       GET;;;
 CONDITIONAL PACKAGES
    oamCellLoopbackPkg
     PRESENT IF "the VCC termination point supports OAM cell Loopbacks";
REGISTERED AS {atmManagedObjectClass 3};

--------------------------------------------------------------------------
vpCTPBidirectional MANAGED OBJECT CLASS
        DERIVED FROM "ITU-T M.3100:1992":
                connectionTerminationPointBidirectional;
        CHARACTERIZED BY

 Conditional Packages
trafficDescriptorPkg,
oamTrafficDescriptorPkg,
 "ITU-T M.3100:1992":AdministrativeStatePkg,
        qosClassPkg,
        oamCellLoopbackPkg,
        "ITU-T M.3100:1992":attributeValueChangeNotificationPackage,
        "ITU-T M.3100:1992":createDeleteNotificationsPackage,
        "ITU-T M.3100:1992":crossConnectionPointerPackage,
        vpCTPBidirectionalPkg PACKAGE
                BEHAVIOUR vpCTPBidirectionalBeh;
                ATTRIBUTES
                        vpCTPId                 GET,
                        segmentEndPointDEFAULT VALUE
                                        AtmMIBMod.booleanFalseDefault
                                GET-REPLACE;;;


CONDITIONAL PACKAGES
 trafficDescriptorPkg
 PRESENT IF "supplied by the managing system. This package must be present at points where UPC/NPC
    functions are performed or when the upstreamConnectivityPointer and downstreamConnectivity Pointer
    attributes point to an instance of the vpTTPBidirectional object class",
 oamTrafficDescriptorPkg
 PRESENT IF "supplied by the managing system. This package must be present at points where UPC/NPC
functions are performed or when the upstreamConnectivityPointer and downstreamConnectivity Pointer
attributes point to an instance of the vpTTPBidirectional object class",
```

```
"ITU-T M.3100:1992":AdministrativeStatePkg
PRESENT IF "supported by the Network Element",
qosClassPkg
PRESENT IF "QOS Class Information is supplied by the managing system",
oamCellLoopbackPkg
PRESENT IF "the VPL termination point supports OAM cell Loopbacks";

REGISTERED AS {atmManagedObjectClass 4};

--------------------------------------------------------------------------
vpTTPBidirectional MANAGED OBJECT CLASS
        DERIVED FROM "ITU-T M.3100:1992":trailTerminationPointBidirectional;
        CHARACTERIZED BY
 Conditional Packages
oamCellLoopbackPkg,
-- ------------------------------------------------

 "Rec. X.721|ISO /IEC 10165-2":administrativeStatePackage,
 "ITU-T M.3100:1992":attributeValueChangeNotificationPackage,
 "ITU-T M.3100:1992":createDeleteNotificationPackage,
vpTTPBidirectionalPkg PACKAGE
        BEHAVIOUR vpTTPBidirectionalBeh;
        ATTRIBUTES
                vpTTPId                 GET;;;
 CONDITIONAL PACKAGES
 oamCellLoopbackPkg
         PRESENT IF "the VPC termination point supports OAM cell Loopbacks";
REGISTERED AS {atmManagedObjectClass 5};

-- ############ Packages templates ##########################

attributeValueChangeNotificationPackage  PACKAGE
        NOTIFICATIONS
                attributeValueChange;
REGISTERED AS {m3100Package 4};

createDeleteNotificationsPackage PACKAGE
        NOTIFICATIONS
                objectCreation,
                objectDeletion;
REGISTERED AS {m3100Package 10};

cellScramblingEnabledPkg PACKAGE
        ATTRIBUTES
                cellScramblingEnabled
                        GET-REPLACE;
REGISTERED AS {atmPackage 1};

oamCellLoopbackPkg PACKAGE
        ACTIONS
                loopbackOAMCell;
REGISTERED AS {atmPackage 2};

oamTrafficDescriptorPkg PACKAGE
        ATTRIBUTES
                oamIngressPeakCellRate
                        GET-REPLACE,
                oamEgressPeakCellRate
```

```
                    GET-REPLACE,
                oamIngressCDVTolerance
                    GET-REPLACE,
                oamEgressCDVTolerance
                    GET-REPLACE;
REGISTERED AS {atmPackage 3};

qosClassPkg PACKAGE
        ATTRIBUTES
                ingressQOSClass
                    GET,
                egressQOSClass
                    GET;
REGISTERED AS {atmPackage 4};

trafficDescriptorPkg PACKAGE
        ATTRIBUTES
                ingressPeakCellRate
                    GET-REPLACE
                    ADD-REMOVE,
                egressPeakCellRate
                    GET-REPLACE
                    ADD-REMOVE,
                ingressCDVTolerance
                    GET-REPLACE
                    ADD-REMOVE,
                egressCDVTolerance
                    GET-REPLACE
                    ADD-REMOVE,
                ingressSustainableCellRate
                    GET-REPLACE
                    ADD-REMOVE,
                egressSustainableCellRate
                    GET-REPLACE
                    ADD-REMOVE,
                ingressBurstTolerance
                    GET-REPLACE
                    ADD-REMOVE,
                egressBurstTolerance
                    GET-REPLACE
                    ADD-REMOVE;
REGISTERED AS {atmPackage 5};
-- ############ Attributes templates ##########################

cellScramblingEnabled ATTRIBUTE
        WITH ATTRIBUTE SYNTAX MetricModule.Boolean;
    MATCHES FOR EQUALITY;
        BEHAVIOUR cellScramblingEnabledBeh;
REGISTERED AS {atmAttributeID 1};

egressBurstTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.BurstTolerance;
        MATCHES FOR EQUALITY;
        BEHAVIOUR egressBurstToleranceBeh;
REGISTERED AS {atmAttributeID 2};

egressCDVTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.CDVTolerance;
```

```
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR egressCDVToleranceBeh;
REGISTERED AS {atmAttributeID 3};

egressPeakCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.PeakCellRate;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR egressPeakCellRateBeh;
REGISTERED AS {atmAttributeID 4};

egressQOSClass ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.QosClass;
        MATCHES FOR EQUALITY;
        BEHAVIOUR egressQOSClassBeh;
REGISTERED AS {atmAttributeID 5};

egressSustainableCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.SustainableCellRate;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR egressSustainableCellRateBeh;
REGISTERED AS {atmAttributeID 6};

ingressBurstTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.BurstTolerance;
        MATCHES FOR EQUALITY;
        BEHAVIOUR ingressBurstToleranceBeh;
REGISTERED AS {atmAttributeID 7};

ingressCDVTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.CDVTolerance;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR ingressCDVToleranceBeh;
REGISTERED AS {atmAttributeID 8};

ingressPeakCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.PeakCellRate;
    MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR ingressPeakCellRateBeh;
REGISTERED AS {atmAttributeID 9};

ingressQOSClass ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.QosClass;
        MATCHES FOR EQUALITY;
        BEHAVIOUR ingressQOSClassBeh;
REGISTERED AS {atmAttributeID 10};

ingressSustainableCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.SustainableCellRate;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR ingressSustainableCellRateBeh;
REGISTERED AS {atmAttributeID 11};

oamEgressCDVTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1TypeModule.OAMCDVTolerance;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR oamEgressCDVToleranceBeh;
REGISTERED AS {atmAttributeID 14};
```

```
oamEgressPeakCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1TypeModule.OAMPeakCellRate;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR oamEgressPeakCellRateBeh;
REGISTERED AS {atmAttributeID 15};

oamIngressCDVTolerance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1TypeModule.OAMCDVTolerance;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR oamIngressCDVToleranceBeh;
REGISTERED AS {atmAttributeID 12};

oamIngressPeakCellRate ATTRIBUTE
        WITH ATTRIBUTE SYNTAX ASN1TypeModule.OAMPeakCellRate;
        MATCHES FOR EQUALITY, ORDERING;
        BEHAVIOUR oamIngressPeakCellRateBeh;
REGISTERED AS {atmAttributeID 13};

segmentEndPoint ATTRIBUTE
        WITH ATTRIBUTE SYNTAX AtmMIBMod.Boolean;
        MATCHES FOR EQUALITY;
        BEHAVIOUR segmentEndPointBeh;
REGISTERED AS {atmAttributeID 16};

tcTTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
        MATCHES FOR EQUALITY;
        BEHAVIOUR tcTTPIdBeh;
REGISTERED AS {atmAttributeID 17};

vcCTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
    MATCHES FOR EQUALITY;
        BEHAVIOUR vcCTPIdBeh;
REGISTERED AS {atmAttributeID 18};

vcTTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
    MATCHES FOR EQUALITY;
        BEHAVIOUR vcTTPIdBeh;
REGISTERED AS {atmAttributeID 19};

vpCTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
    MATCHES FOR EQUALITY;
        BEHAVIOUR vpCTPIdBeh;
REGISTERED AS {atmAttributeID 20};

vpTTPId ATTRIBUTE
        WITH ATTRIBUTE SYNTAX UCLAttribute-ASN1Module.SimpleNameType;
        MATCHES FOR EQUALITY;
        BEHAVIOUR vpTTPIdBeh;
REGISTERED AS {atmAttributeID 21};

-- ############ Notifications and actions templates ##########################
attributeValueChange NOTIFICATION
  BEHAVIOUR             attributeValueChangeBehaviour;
  WITH INFORMATION SYNTAX    Notification-ASN1Module.AttrValChangeInfo
```

```
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attrValChangeDefinition attrValChangeDefinition;
  REGISTERED AS          { smi2Notification 1 }

loopbackOAMCell ACTION
        BEHAVIOUR loopbackOAMCellBeh;
        MODE CONFIRMED;
        WITH INFORMATION SYNTAX AtmMIBMod.LoopbackOAMCellInfo;
        WITH REPLY SYNTAX AtmMIBMod.LoopbackOAMCellReply;
REGISTERED AS {atmAction 1};

objectCreation NOTIFICATION
  BEHAVIOUR              objectCreationBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attributeList        attributeList;
  REGISTERED AS          { smi2Notification 6 };

objectDeletion NOTIFICATION
  BEHAVIOUR              objectDeletionBehaviour;
  WITH INFORMATION SYNTAX   Notification-ASN1Module.ObjectInfo
  AND ATTRIBUTE IDS
    sourceIndicator      sourceIndicator,
    attributeList        attributeList;
  REGISTERED AS          { smi2Notification 7 };

-- ############## Name Binding Templates ##############

tcAdaptorTTPBidirectional-managedElementR1 NAME BINDING
  SUBORDINATE OBJECT CLASS   tcAdaptorTTPBidirectional AND SUBCLASSES;
  NAMED BY
  SUPERIOR OBJECT CLASS      "ITU-T M.3100:1992":managedElementR1 AND SUBCLASSES;
  WITH ATTRIBUTE          tcTTPId;
  REGISTERED AS           {atmNameBindings 1};

vcCTPBidirectional-vpTTPBidirectional NAME BINDING
  SUBORDINATE OBJECT CLASS   vcCTPBidirectional AND SUBCLASSES;
  NAMED BY
  SUPERIOR OBJECT CLASS   vpTTPBidirectional AND SUBCLASSES;
  WITH ATTRIBUTE          vcCTPId;
  CREATE                     WITH-AUTOMATIC-INSTANCE-NAMING;
  DELETE                     ONLY-IF-NO-CONTAINED-OBJECTS;
  REGISTERED AS           {atmNameBindings 2};

vcTTPBidirectional-managedElementR1   NAME BINDING
  SUBORDINATE OBJECT CLASS       vcTTPBidirectional AND SUBCLASSES;
  NAMED BY
  SUPERIOR OBJECT CLASS   "ITU-T M.3100:1992":managedElementR1 AND SUBCLASSES;
  WITH ATTRIBUTE          vcTTPId;
  CREATE                     WITH-AUTOMATIC-INSTANCE-NAMING;
  DELETE                     ONLY-IF-NO-CONTAINED-OBJECTS;
  REGISTERED AS              {atmNameBindings 3};

vpCTPBidirectional-tcAdaptorTTPBidirectional NAME BINDING
  SUBORDINATE OBJECT CLASS       vpCTPBidirectional AND SUBCLASSES;
  NAMED BY
```

```
  SUPERIOR OBJECT CLASS    tcAdaptorTTPBidirectional AND SUBCLASSES;
  WITH ATTRIBUTE           vpCTPId;
  CREATE                   WITH-AUTOMATIC-INSTANCE-NAMING;
  DELETE                   DELETES-CONTAINED-OBJECTS;
  REGISTERED AS          {atmNameBindings 4};

vpTTPBidirectional-managedElementR1 NAME BINDING
  SUBORDINATE OBJECT CLASS   vpTTPBidirectional AND SUBCLASSES;
  NAMED BY
  SUPERIOR OBJECT CLASS     "ITU-T M.3100:1992":managedElementR1 AND SUBCLASSES;
  WITH ATTRIBUTE          vpTTPId;
  CREATE                    WITH-AUTOMATIC-INSTANCE-NAMING;
  DELETE                    ONLY-IF-NO-CONTAINED-OBJECTS;
  REGISTERED AS           {atmNameBindings 5};
```

10.4. Implementação de objetos da recomendação G.atmm

10.4.1 Arquivo vcCTPBidirectional.inc.h

```
protected:
        static int _VCCTPId;
        inline static int GetNewVCCTPId() { return(++ _VCCTPId); }
public:
        int createRR (AVA*&, void* = NULLVD, int = -1);
```

10.5. Arquivo vcCTPBidirectional.inc.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <MonitorSntx.h>

int vcCTPBidirectional::_VCCTPId = 1;

// The following function is called to create the RDN to a new
// object instace.

RDN vcCTPBidirectional::makeRdn (MO*)
{
  // build the RDN
        char buf[64];
        sprintf(buf, "vcCTPId=%d",  GetNewVCCTPId());
        return str2rdn(buf);
}

int vcCTPBidirectional::createRR (AVA*& err, void*, int)
{
  static Bool classInitialised;

   // call the parent class createRR method -
   // in this case it will do nothing but for correct OO style...

        if (Top::createRR(err) == NOTOK)
        return NOTOK;

        if (!classInitialised)
        classInitialised = True;

  if (MO::getClassInfo() -> getFirst() -> getClassNext())
        {       _classInfo->setCreate(vcCTPBidirectional::create); }
return OK;
}
```

10.5.1 Arquivo switch.h

```
#include <stdlib.h>
/* Numero maximos de VPL e VCL permitidos em um no */
#define MAXVPL                  256
#define MAXVCL                  256
/* Numero maximo de switchs em um no */
#define MAXSWITCHESINNODE       10
```

```
/* Descritores de Trafego */
#define atmNoTrafficDescriptor    0
#define atmNoClpNoScr             1
#define atmClpNoTaggingNoScr      2
#define atmClpTaggingNoScr        3
#define atmNoClpScr               4
#define atmClpNoTaggingScr        5
#defme atmClpTaggingScr           6

/* Indices para busca de parametros de trafego */
#defme type               0
#define param1            1
#define param2            2
#define param3            3
#define param4            4
#define param5            5
#define QoSClass          6
#define send              0
#define receive           1

class TrafficDescriptor {
public:
  int atmTrafficDescrType;
  int atmTrafficDescrParam1;
  int atmTrafficDescrParam2;

  int atmTrafficDescrParam3;

  int atmTrafficDescrParam4;

  int atmTrafficDescrParam5;
  int atmQoSClass;
  inline TrafficDescriptor(void)  {
    atmTrafficDescrType=-1;
    atmTrafficDescrParam1=-1;
    atmTrafficDescrParam2=-1;

    atmTrafficDescrParam3=-1;

    atmTrafficDescrParam4=-1;

    atmTrafficDescrParam5=-1;
    atmQoSClass=-1;
   }; };

class AtmLink {
public:
  char MyIpAddress[16];
  char MyNeighborIpAddress[16];
  int Vpi;
  int Vci;
  int MyInterfaceIndex;
  int ReceiveTrafficDescriptorIndex;
  int SendTrafficDescriptorIndex;
  int OperStatus;
  int CrossConnectIdentifier;
  TrafficDescriptor receiveParameters;
```

```
 TrafficDescriptor sendParameters;
 inline AtmLink(void) {
  MyNeighborIpAddress[0]='\0';
  MyIpAddress[0]='\0';
  Vpi=-1;
  Vci=-1;
  ReceiveTrafficDescriptorIndex=-1;
  SendTrafficDescriptorIndex=-1;
  OperStatus=-1;
  CrossConnectIdentifier=-1; };};

class TMNTrafficAttribute{
 public:
 int egressCLP0;
 int egressCLP0Plus1;
 int ingressCLP0;
 int ingressCLP0Plus1;
 inline TMNTrafficAttribute(void)
 {  egressCLP0=-1;
  egressCLP0Plus1=-1;
  ingressCLP0=-1;
  ingressCLP0Plus1=-1;  };};

class TMNLink {
 public:
  int Vpi;
  int Vci;
  TMNTrafficAttribute PeakCellRate;
  TMNTrafficAttribute QoSClassTMN;
  TMNTrafficAttribute SustainedCellRate;
  TMNTrafficAttribute MaxBurstSize;
  TMNLink *next;
  TMNLink *prev;
  inline TMNLink(void)   {
   Vpi = -1;
   Vci= -1;
   next = NULL;
   prev = NULL;  };};

class TMNLinkList{
 public:
 int posAtual;
 TMNLink *inicio;
 TMNLink *fim;
 TMNLink * get(int posicao);
 int add(TMNLink *Novo);
 inline TMNLinkList(void) {
  inicio = NULL;
  fim   = NULL;
  posAtual = -1; };
  ~TMNLinkList(void);
 inline int startGetNext(void){ posAtual = -1;};
 TMNLink *getNext(void); };


class Switch{
public:
 int numVpl;
```

```
 int numVcl;
 AtmLink *vpls[MAXVPL];
 AtmLink *vcls[MAXVCL];
 int numSwitchesInNode;
 char MyIpAddresses[MAXSWITCHESINNODE][16];

 Switch(char *IpAddresses);
 ~Switch(void);
 int getTrafficParameter(TrafficDescriptor * trafficDescriptor, int parameter);
 int iniciaLinks(void);
 int eMembro(char * ip);

 AtmLink * getVcl(char * ip, int Vpi, int Vci);
 int getVclTrafficParameter(char * ip, int Vpi, int Vci, int direction, int parameter);
 int getVclNeighborIpAddress(char * ip, int Vpi, int Vci, char * IpAddress);
 int getVclOperStatus(char * ip, int Vpi, int Vci);
 int getVclCrossConnectIdentifier(char * ip, int Vpi, int Vci);
 AtmLink * getVpl(char * ip, int Vpi);
 int getVplTrafficParameter(char * ip, int Vpi, int direction, int parameter);
 int getVplNeighborIpAddress(char * ip, int Vpi, char * IpAddress);
 int getVplOperStatus(char * ip, int Vpi);
 int getVplCrossConnectIdentifier(char * ip, int Vpi);
 int setTrafficParameter(char * ip, int index, int param, int value);
 int setInterfaceParameter(char * ip, int index, char * ipDest);
 TMNLinkList * getVcConnections(void);
 TMNLinkList * getVpConnections(void);
 int setTMNLink(TMNLink * tmnlink, AtmLink * atmlink);
};
```

10.5.2 Arquivo switch.cc

```
#include "switch.h"
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

/* Lista para retorno */

TMNLinkList::~TMNLinkList(void)
{
 TMNLink * atual = inicio;
 while (atual != NULL)
 {
  inicio = atual->next;
  delete atual;
  atual = inicio;
 };
};

int TMNLinkList::add(TMNLink * novo)
{
 int result = -1;
 if ((inicio == NULL) && (novo != NULL))
 {
  inicio = novo;
  fim = novo;
  novo->next = NULL;
```

```
  result = 0;
 }
 else if (novo != NULL)
 {
  result = 0;
  fim->next = novo;
  novo->next = NULL;
  fim = novo;
 };
 return(result);
};

TMNLink * TMNLinkList::get(int posicao)
{
 TMNLink * result = NULL;
 TMNLink * atual = inicio;
 int contador = 0;
 while ((atual != NULL) && (contador <= posicao))
 {
  if (contador == posicao)
  {
   result = atual;
  };
  atual = atual->next;
  contador++;
 };
 return(result);
};

TMNLink * TMNLinkList::getNext(void)
{
 posAtual++;
 return(get(posAtual));
};



/* Construtor da Classe
  Recebe como parametros os numeros IP dos switches
  do no' separados por ;. A string deve tambem ser
  terminada por ;.
*/

Switch::Switch(char *IpAddresses)
{
 int contador=0;
 int posIp=0;
 numSwitchesInNode = 0;
 numVpl = 0;
 numVcl = 0;

 while ((IpAddresses[contador] != '\0') &&
       (numSwitchesInNode < MAXSWITCHESINNODE))
 {
  if (IpAddresses[contador] !=';')
  {
   MyIpAddresses[numSwitchesInNode][posIp] = IpAddresses[contador];
   contador++;
```

```
    posIp++;
   }
   else
   {
    MyIpAddresses[numSwitchesInNode][posIp] = '\0';
    posIp = 0;
    contador++;
    numSwitchesInNode++;
   };
  };
  iniciaLinks();
};

/* Destrutor da Classe */
Switch::~Switch(void)
{
 int contador=0;
 while (contador<numVpl)
 {
  delete vpls[contador];
  contador++;
 };
 contador=0;
 while (contador<numVcl)
 {
  delete vcls[contador];
  contador++;
 };
};

Switch::iniciaLinks(void)
{
 int contador = 0;
 int presentIndex = -999;
 int paramId;
 char linha[100];
 FILE * arquivo;
 while (contador < numSwitchesInNode)
 {
  arquivo = fopen(MyIpAddresses[contador],"r");
  if (arquivo != NULL)
  {
   while (fscanf(arquivo,"%s\n",linha) != EOF)
   {

    /* Leitura de VPLs */
    if (!strcmp(linha,"AtmVplEntry.atmVplVpi"))
    {
     numVpl++;
     vpls[numVpl-1]=new AtmLink;
     strcpy(vpls[numVpl-1]->MyIpAddress,MyIpAddresses[contador]);
     if (fscanf(arquivo,"%s\n",linha) != EOF)
     {
      vpls[numVpl-1]->Vpi = atoi(linha);
     };
    }
    else if (!strcmp(linha,"AtmVplEntry.interfaceIndex"))
    {
```

```
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vpls[numVpl-1]->MyInterfaceIndex = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVplEntry.atmVplOperStatus"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vpls[numVpl-1]->OperStatus = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVplEntry.atmVplReceiveTrafficDescrIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vpls[numVpl-1]->ReceiveTrafficDescriptorIndex = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVplEntry.atmVplTransmitTrafficDescrIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vpls[numVpl-1]->SendTrafficDescriptorIndex = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVplEntry.atmVplCrossConnectIdentifier"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vpls[numVpl-1]->CrossConnectIdentifier = atoi(linha);
 };
}
/* Leitura de VCLs */
else if (!strcmp(linha,"AtmVclEntry.atmVclVpi"))
{
 numVcl++;
 vcls[numVcl-1]=new AtmLink;
 strcpy(vcls[numVcl-1]->MyIpAddress,MyIpAddresses[contador]);
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->Vpi = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVclEntry.atmVclVci"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->Vci = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVclEntry.interfaceIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->MyInterfaceIndex = atoi(linha);
 };
}
```

```
else if (!strcmp(linha,"AtmVclEntry.atmVclOperStatus"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->OperStatus = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVclEntry.atmVclReceiveTrafficDescrIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->ReceiveTrafficDescriptorIndex = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVclEntry.atmVclTransmitTrafficDescrIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->SendTrafficDescriptorIndex = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmVclEntry.atmVclCrossConnectIdentifier"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  vcls[numVcl-1]->CrossConnectIdentifier = atoi(linha);
 };
}
else if (!strcmp(linha,"AtmTrafficDescrParamEntry.atmTrafficDescrParamIndex"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  presentIndex = atoi(linha);
 };
}
else if (!strncmp(linha,"AtmTrafficDescrParamEntry.atmTrafficDescrParam",46))
{
 paramId = atoi(&linha[46]);
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  setTrafficParameter(MyIpAddresses[contador], presentIndex, paramId, atoi(linha));
 };
}
else if (!strcmp(linha,"AtmTrafficDescrParamEntry.atmTrafficDescrType"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  setTrafficParameter(MyIpAddresses[contador], presentIndex, type, atoi(linha));
 };
}
else if (!strcmp(linha,"AtmTrafficDescrParamEntry.atmTrafficQoSClass"))
{
 if (fscanf(arquivo,"%s\n",linha) != EOF)
 {
  setTrafficParameter(MyIpAddresses[contador], presentIndex, QoSClass, atoi(linha));
 };
}
else if (!strcmp(linha,"AtmInterface.index"))
```

```
      {
       if (fscanf(arquivo,"%s\n",linha) != EOF)
       {
        presentIndex = atoi(linha);
       };
      }
      else if (!strcmp(linha,"AtmInterface.neighborIpAddress"))
      {
       if (fscanf(arquivo,"%s\n",linha) != EOF)
       {
        setInterfaceParameter(MyIpAddresses[contador], presentIndex, linha);
       };
      };
     };
    };
    contador++;
  };
};

int Switch::eMembro(char * ip)
{
 int contador=0;
 int result = 0;
 while ((contador < numSwitchesInNode) && (!result))
 {
  result = !strcmp(ip, MyIpAddresses[contador]);
  contador++;
 };
 return(result);
};

/* Retorna determinado parametro de trafego */
int Switch::getTrafficParameter(TrafficDescriptor * trafficDescriptor, int parameter)
{
 int result=-1;
 switch(parameter) {
  case type:    result = trafficDescriptor->atmTrafficDescrType;
                break;
  case param1:  result = trafficDescriptor->atmTrafficDescrParam1;
                break;
  case param2:  result = trafficDescriptor->atmTrafficDescrParam2;
                break;
  case param3:  result = trafficDescriptor->atmTrafficDescrParam3;
                break;
  case param4:  result = trafficDescriptor->atmTrafficDescrParam4;
                break;
  case param5:  result = trafficDescriptor->atmTrafficDescrParam5;
                break;
  case QoSClass: result = trafficDescriptor->atmQoSClass;
                break;
 };
 return(result);
};

int Switch::setTrafficParameter(char * ip, int index, int parameter, int value)
{
 int contador;
 int result=0;
```

```
TrafficDescriptor * trafficDescriptor = NULL;

contador = 0;
while ((contador < numVpl) && (trafficDescriptor == NULL))
{
 if (!(strcmp(vpls[contador]->MyIpAddress,ip)))
 {
  if (index == vpls[contador]->ReceiveTrafficDescriptorIndex)
  {
   trafficDescriptor = &vpls[contador]->receiveParameters;
  }
  else if (index == vpls[contador]->SendTrafficDescriptorIndex)
  {
   trafficDescriptor = &vpls[contador]->sendParameters;
  };
 };
 contador++;
};
contador = 0;
while ((contador < numVcl) && (trafficDescriptor == NULL))
{
 if (!(strcmp(vcls[contador]->MyIpAddress,ip)))
 {
  if (index == vcls[contador]->ReceiveTrafficDescriptorIndex)
  {
   trafficDescriptor = &vcls[contador]->receiveParameters;
  }
  else if (index == vcls[contador]->SendTrafficDescriptorIndex)
  {
   trafficDescriptor = &vcls[contador]->sendParameters;
  };
 };
 contador++;
};
if (trafficDescriptor != NULL)
{
 result = 1;
 switch(parameter) {
  case type:    trafficDescriptor->atmTrafficDescrType=value;
                break;
  case param1:  trafficDescriptor->atmTrafficDescrParam1=value;
                break;
  case param2:  trafficDescriptor->atmTrafficDescrParam2=value;
                break;
  case param3:  trafficDescriptor->atmTrafficDescrParam3=value;
                break;
  case param4:  trafficDescriptor->atmTrafficDescrParam4=value;
                break;
  case param5:  trafficDescriptor->atmTrafficDescrParam5=value;
                break;
  case QoSClass: trafficDescriptor->atmQoSClass=value;
                break;
 };
};
 return(result);
};

int Switch::setInterfaceParameter(char * ip, int index, char * ipDest)
```

```
{
 int contador;
 int result=0;

 contador = 0;
 while (contador < numVpl)
 {
  if (!(strcmp(vpls[contador]->MyIpAddress,ip)))
  {
   if (index == vpls[contador]->MyInterfaceIndex)
   {
    strcpy(vpls[contador]->MyNeighborIpAddress, ipDest);
    result = result +1;
   }
  };
  contador++;
 };
 contador = 0;
 while (contador < numVcl)
 {
  if (!(strcmp(vcls[contador]->MyIpAddress,ip)))
  {
   if (index == vcls[contador]->MyInterfaceIndex)
   {
    strcpy(vcls[contador]->MyNeighborIpAddress, ipDest);
    result = result +1;
   }
  };
  contador++;
 };
 return(result);
};

/* Retorna um ponteiro para a estrutura do VCL desejado */
AtmLink * Switch::getVcl(char * ip, int Vpi, int Vci)
{
 AtmLink * result = NULL;
 int posicao=0;
 while ((posicao < numVcl) && (result==NULL))
 {
  if ((vcls[posicao]->Vpi == Vpi) && (vcls[posicao]->Vci == Vci) &&
     (!strcmp(vcls[posicao]->MyIpAddress, ip)))
  {
   result = vcls[posicao];
  };
  posicao++;
 };
 return(result);
};

/* Retorna um ponteiro para a estrutura do VPL desejado */
AtmLink * Switch::getVpl(char * ip, int Vpi)
{
 AtmLink * result = NULL;
 int posicao=0;
 while ((posicao < numVpl) && (result == NULL))
 {
  if ((vpls[posicao]->Vpi == Vpi) &&
```

```
      (!strcmp(vpls[posicao]->MyIpAddress, ip)))
    {
     result = vpls[posicao];
    };
   posicao++;
  };
  return(result);
};

int Switch::getVclTrafficParameter(char * ip, int Vpi, int Vci, int direction, int parameter)
{
  TrafficDescriptor * trafficParameter = NULL;
  AtmLink * Vcl = NULL;
  int result = -1;
  Vcl = getVcl(ip, Vpi, Vci);
  if (Vcl != NULL)
  {
   switch(direction)
    {
     case send:    trafficParameter = &Vcl->sendParameters;
                   break;
     case receive: trafficParameter = &Vcl->receiveParameters;
                   break;
    };
    if (trafficParameter != NULL)
    {
     result = getTrafficParameter(trafficParameter, parameter);
    };
  };
  return(result);
};

int Switch::getVplTrafficParameter(char * ip, int Vpi, int direction, int parameter)
{
  TrafficDescriptor * trafficParameter = NULL;
  AtmLink * Vpl = NULL;
  int result = -1;
  Vpl = getVpl(ip, Vpi);
  if (Vpl != NULL)
  {
   switch(direction)
    {
     case send:    trafficParameter = &Vpl->sendParameters;
                   break;
     case receive: trafficParameter = &Vpl->receiveParameters;
                   break;
    };
    if (trafficParameter != NULL)
    {
     result = getTrafficParameter(trafficParameter, parameter);
    };
  };
  return(result);
};

int Switch::getVclNeighborIpAddress(char * ip, int Vpi, int Vci, char * IpAddress)
{
  AtmLink * Vcl = NULL;
```

```
 int result = -1;

 IpAddress[0]='\0';
 Vcl = getVcl(ip, Vpi, Vci);
 if (Vcl != NULL)
 {
  strcpy(IpAddress, Vcl->MyNeighborIpAddress);
  result = 0;
 }
 return(result);
};

int Switch::getVplNeighborIpAddress(char * ip, int Vpi, char * IpAddress)
{
 AtmLink * Vpl = NULL;
 int result = -1;

 IpAddress[0]='\0';
 Vpl = getVpl(ip, Vpi);
 if (Vpl != NULL)
 {
  strcpy(IpAddress, Vpl->MyNeighborIpAddress);
  result = 0;
 }
 return(result);
};

int Switch::getVclOperStatus(char * ip, int Vpi, int Vci)
{
 AtmLink * Vcl = NULL;
 int result = -1;
 Vcl = getVcl(ip, Vpi, Vci);
 if (Vcl != NULL)
 {
  result = Vcl->OperStatus;
 };
 return(result);
};

int Switch::getVplOperStatus(char * ip, int Vpi)
{
 AtmLink * Vpl = NULL;
 int result = -1;
 Vpl = getVpl(ip, Vpi);
 if (Vpl != NULL)
 {
  result = Vpl->OperStatus;
 };
 return(result);
};

int Switch::getVclCrossConnectIdentifier(char * ip, int Vpi, int Vci)
{
 AtmLink * Vcl = NULL;
 int result = -1;
 Vcl = getVcl(ip, Vpi, Vci);
 if (Vcl != NULL)
 {
```

```
  result = Vcl->CrossConnectIdentifier;
 };
 return(result);
};

int Switch::getVplCrossConnectIdentifier(char * ip, int Vpi)
{
 AtmLink * Vpl = NULL;
 int result = -1;
 Vpl = getVpl(ip, Vpi);
 if (Vpl != NULL)
 {
  result = Vpl->CrossConnectIdentifier;
 };
 return(result);
};

int Switch::setTMNLink(TMNLink * tmnlink, AtmLink * atmlink)
{
 tmnlink->Vpi = atmlink->Vpi;
 tmnlink->Vci = atmlink->Vci;
 tmnlink->QoSClassTMN.ingressCLP0Plus1 = atmlink->receiveParameters.atmQoSClass;
 tmnlink->QoSClassTMN.ingressCLP0      = atmlink->receiveParameters.atmQoSClass;
 tmnlink->QoSClassTMN.egressCLP0Plus1  = atmlink->sendParameters.atmQoSClass;
 tmnlink->QoSClassTMN.egressCLP0       = atmlink->sendParameters.atmQoSClass;
 switch(atmlink->receiveParameters.atmTrafficDescrType)
 {
  case atmNoClpNoScr:
   tmnlink->PeakCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam1;
   break;
  case atmClpNoTaggingNoScr:
   tmnlink->PeakCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam1;
   tmnlink->PeakCellRate.ingressCLP0 = atmlink->receiveParameters.atmTrafficDescrParam2;
   break;
  case atmClpTaggingNoScr:
   tmnlink->PeakCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam1;
   tmnlink->PeakCellRate.ingressCLP0 = atmlink->receiveParameters.atmTrafficDescrParam2;
   break;
  case atmNoClpScr:
   tmnlink->PeakCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam1;
   tmnlink->SustainedCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam2;
   tmnlink->MaxBurstSize.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam3;
   break;
  case atmClpNoTaggingScr:
   tmnlink->PeakCellRate.ingressCLP0Plus1 = atmlink->receiveParameters.atmTrafficDescrParam1;
   tmnlink->SustainedCellRate.ingressCLP0 = atmlink->receiveParameters.atmTrafficDescrParam2;
   tmnlink->MaxBurstSize.ingressCLP0 = atmlink->receiveParameters.atmTrafficDescrParam3;
   break;
 };
 switch(atmlink->sendParameters.atmTrafficDescrType)
 {
  case atmNoClpNoScr:
   tmnlink->PeakCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam1;
   break;
  case atmClpNoTaggingNoScr:
   tmnlink->PeakCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam1;
   tmnlink->PeakCellRate.egressCLP0 = atmlink->sendParameters.atmTrafficDescrParam2;
   break;
```

```
   case atmClpTaggingNoScr:
    tmnlink->PeakCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam1;
    tmnlink->PeakCellRate.egressCLP0 = atmlink->sendParameters.atmTrafficDescrParam2;
    break;
   case atmNoClpScr:
    tmnlink->PeakCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam1;
    tmnlink->SustainedCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam2;
    tmnlink->MaxBurstSize.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam3;
    break;
   case atmClpNoTaggingScr:
    tmnlink->PeakCellRate.egressCLP0Plus1 = atmlink->sendParameters.atmTrafficDescrParam1;
    tmnlink->SustainedCellRate.egressCLP0 = atmlink->sendParameters.atmTrafficDescrParam2;
    tmnlink->MaxBurstSize.egressCLP0 = atmlink->sendParameters.atmTrafficDescrParam3;
    break;
  };
  return(1);
};

TMNLinkList * Switch::getVcConnections(void)
{
  TMNLinkList * lista = new TMNLinkList;
  TMNLink * vcl = NULL;
  int contador = 0;
  while (contador < numVcl)
  {
   if (!eMembro(vcls[contador]->MyNeighborIpAddress))
   {
    vcl = new TMNLink;
    setTMNLink(vcl, vcls[contador]);
    lista->add(vcl);
   };
   contador++;
  };
  return(lista);
};

TMNLinkList * Switch::getVpConnections(void)
{
  TMNLinkList * lista = new TMNLinkList;
  TMNLink * vpl = NULL;
  int contador = 0;
  while (contador < numVpl)
  {
   if (!eMembro(vpls[contador]->MyNeighborIpAddress))
   {
    vpl = new TMNLink;
    setTMNLink(vpl, vpls[contador]);
    lista->add(vpl);
   };
   contador++;
  };
  return(lista);
};
```