CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DO PARANÁ
Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

DISSERTAÇÃO
apresentada ao CEFET-PR
para a obtenção do título de

Mestre em Ciências

por

FRANCISCO ADELL PÉRICAS

PROPOSTA DE UM MODELO DE INFORMAÇÃO TMN PARA A GERÊNCIA DINÂMICA DE REDE ATM BASEADO NA TECNOLOGIA JAVA JMX

Banca Examinadora

Presidente e Orientador:	
Professor Dr. Luiz Nacamura Júnior	CEFET-PR
Examinadores:	
Professor Dr. Carlos Becker Westphall	UFSC
Professor Dr. Manoel Camillo Penna	PUC-PR
Professor Dr. Walter Godoy Júnior (Suplente)	CEFET-PR

Curitiba, Setembro de 2000

FRANCISCO ADELL PÉRICAS

PROPOSTA DE UM MODELO DE INFORMAÇÃO TMN PARA A GERÊNCIA DINÂMICA DE REDE ATM BASEADO NA TECNOLOGIA JAVA JMX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciências.
Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Centro Federal de Educação Tecnológica do Paraná.

Ênfase: Telemática.
Orientador: Professor Dr. Luiz Nacamura Júnior

CURITIBA

2000

Agradecimentos

É com grande estima que agradeço a todas as pessoas que, direta ou indiretamente, auxiliaram no desenvolvimento deste trabalho.

Agradeço ao Professor Dr. Luiz Nacamura Júnior, pela confiança depositada, o apoio dedicado e, principalmente, pelo valoroso trabalho de orientação.

Agradeço a meus colegas do laboratório, em especial à Rafaela Vergès Mayrhofer e ao Daniel Puka, pelo convívio e pelo auxílio na realização do trabalho.

Agradeço à minha esposa Cynara Maria Reinert pela paciência e pelo apoio aos estudos durante todos estes anos.

Sumário

Agradecimentos i
Sumário ii
Lista de Ilustrações vi
Lista de Tabelas ix
Lista de Abreviaturas e Siglas x
Resumo xiii
Abstract xiv
1. Introdução 1
1.1. Motivações 1
1.2. Objetivo 6
1.3. Descrição de Trabalhos Existentes 7
1.4. Organização deste Documento 7
2. O TMN 9
2.1. Introdução 9
2.2. Estrutura do TMN 9
2.2.1. Padrão TMN 10
2.2.2. TMN e Gerência 11
2.2.3. Modelo de Informação de Gerência 11
2.3. Modelo Funcional do TMN 12
2.3.1. Blocos Funcionais do TMN 12
2.3.2. Funcionalidade OSI para o TMN 13
2.4. Modelo Lógico do TMN 14
2.5. Conclusão 16
3. Modelo de Informação para Gerência de Rede ATM 17
3.1. Introdução 17
3.2. A Tecnologia de Transmissão ATM 18
3.2.1. Conceitos Básicos do ATM 18
3.2.2. Arquitetura de uma Rede de Transporte ATM 23
3.2.3. Arquitetura de uma Rede de Gerência 29
3.2.4. Funções de Gerência de Rede 29
3.3. Entidades Gerenciadas 33
3.3.1. AlarmRecord 33
3.3.2. AlarmSeverityAssignmentProfile 35
3.3.3. AttributeValueChangeRecord 35
3.3.4. EventForwardingDiscriminator 36
3.3.5. LayerNetworkDomain 38
3.3.6. LinkConnection 40

3.3.7. Log ... 42
3.3.8. LogicalLinkTP ... 44
3.3.9. ManagedEntityCreationRecord ... 46
3.3.10. ManagedEntityDeletionRecord ... 47
3.3.11. Network ... 48
3.3.12. NetworkAccessProfile ... 49
3.3.13. NetworkCTP ... 50
3.3.14. NetworkTTP ... 52
3.3.15. RoutingProfile ... 53
3.3.16. StateChangeRecord ... 55
3.3.17. Subnetwork ... 55
3.3.18. SubnetworkConnection ... 58
3.3.19. TopologicalLink ... 61
3.3.20. TrafficDescriptorProfile ... 64
3.3.21. Trail ... 65
3.3.22. TrailRequest ... 67
3.3.23. VcLinkEnd ... 68
3.3.24. VpLinkEnd ... 70
3.4. Modelo de Informação ... 73
3.5. Conclusão ... 75
4. Metodologia de Implementação do Modelo de Informação ATM ... 76
4.1. Introdução ... 76
4.2. Formalização da Metodologia de Implementação: Ensemble ... 76
4.3. Ensembles ... 77
4.3.1. Inicialização do Sistema ... 79
4.3.2. Inicialização do Registro de Alarmes ... 81
4.3.3. Registro de uma Aplicação de Gerência para o Recebimento de Alarmes ... 82
4.3.4. Geração de Alarmes de Falha de Equipamentos ATM ... 84
4.3.5. Criação de uma Conexão de Sub-rede Simples ... 86
4.3.6. Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação ... 88
4.3.7. Criação de Sub-redes para o Particionamento de uma Sub-rede ... 90
4.3.8. Criação de uma Conexão de Enlace entre Sub-redes ... 91
4.3.9. Criação de uma Conexão de Sub-rede Composta ... 94
4.3.10. Criação de um Trail Servidor ... 97
4.3.11. Operação sobre um Trail Servidor por Agendamento ... 99
4.3.12. Criação de uma Conexão de Enlace entre Sub-redes através de um Trail Servidor 101
4.4. Conclusão ... 104
5. Modelo de Informação de Rede ATM para o JMX ... 105
5.1. Introdução ... 105
5.2. A Arquitetura JMX da Sun ... 106

5.2.1. A Estrutura do JMX 106
5.2.2. Componentes do Nível Instrumentação 109
5.2.3. Componentes do Nível Agente 115
5.2.4. Componentes do Nível Gerente 118
5.2.5. Componentes das APIs de Protocolos de Gerência Adicionais 119
5.3. Metodologia de Especificação das Interfaces para o JMX 119
5.3.1. Nome da Interface 119
5.3.2. Classes com Herança 119
5.3.3. Classes com Atributos 120
5.3.4. Classes com Métodos 121
5.3.5. Classes com Métodos de Envio de Eventos 121
5.4. Especificação das Interfaces JMX do Modelo de Informação 122
5.4.1. AlarmRecord 122
5.4.2. AlarmSeverityAssignmentProfile 122
5.4.3. AttributeValueChangeRecord 122
5.4.4. EventForwardingDiscriminator 123
5.4.5. LayerNetworkDomain 123
5.4.6. LinkConnection 123
5.4.7. Log 124
5.4.8. LogicalLinkTP 124
5.4.9. LogRecord 124
5.4.10. ManagedEntityCreationRecord 125
5.4.11. ManagedEntityDeletionRecord 125
5.4.12. Network 125
5.4.13. NetworkAccessProfile 125
5.4.14. NetworkCTP 125
5.4.15. NetworkTTP 126
5.4.16. RoutingProfile 126
5.4.17. StateChangeRecord 127
5.4.18. Subnetwork 127
5.4.19. SubnetworkConnection 127
5.4.20. TopologicalLink 128
5.4.21. TrafficDescriptorProfile 128
5.4.22. Trail 129
5.4.23. TrailRequest 129
5.4.24. VcLinkEnd 129
5.4.25. VpLinkEnd 130
5.5. Especificação das Classes de Suporte Utilizadas pelo Modelo nas Interfaces 131
5.6. Metodologia de Especificação das Classes para o JMX 137
5.6.1. Definição da Classe 138

5.6.2. Classes com Relacionamento Unidirecional 138
5.6.3. Classes com Relacionamento Bidirecional..... 139
5.6.4. Classes com Métodos de Envio de Eventos 141
5.6.5. Classes com Atributos e Métodos..... 141
5.7. Conclusão..... 142
6. Conclusões Finais 143
Referências Bibliográficas..... 145

Lista de Ilustrações

Figura 1: Relacionamento entre as normas de modelagem de rede 4
Figura 2: Arquitetura gerente/agente do JMX 5
Figura 3: O TMN e a Rede de Telecomunicações 10
Figura 4: Blocos Funcionais do TMN 12
Figura 5: Interação entre Gerente, Agente e Objetos Gerenciáveis 13
Figura 6: Decomposição da Funcionalidade de Gerência 14
Figura 7: Exemplo de Componentes TMN Distribuídos nas Camadas Lógicas 16
Figura 8: Cadeia de células ATM 19
Figura 9: Estrutura do cabeçalho das células ATM 20
Figura 10: Exemplo de um caminho virtual 21
Figura 11: Modelo de referência ATM 21
Figura 12: Gerência de alarmes ATM 22
Figura 13: Exemplo de particionamento de uma sub-rede 24
Figura 14: Camada de rede VP 27
Figura 15: Camada de rede VC 27
Figura 16: Exemplo de camadas de rede ATM 28
Figura 17: Entidade Gerenciada *AlarmRecord* 34
Figura 18: Entidade Gerenciada *AlarmSeverityAssignmentProfile* 35
Figura 19: Entidade Gerenciada *AttributeValueChangeRecord* 36
Figura 20: Entidade Gerenciada *EventForwardingDiscriminator* 37
Figura 21: *LayerNetworkDomain* para a camada VC ou VP 38
Figura 22: Entidade Gerenciada *LayerNetworkDomain* 40
Figura 23: *LinkConnection* para a camada VC ou VP 40
Figura 24: Entidade Gerenciada *LinkConnection* 42
Figura 25: Entidade Gerenciada *Log* 44
Figura 26: Entidade Gerenciada *LogicalLinkTP* 46
Figura 27: Entidade Gerenciada *ManagedEntityCreationRecord* 47
Figura 28: Entidade Gerenciada *ManagedEntityDeletionRecord* 48
Figura 29: Entidade Gerenciada *Network* 49
Figura 30: Entidade Gerenciada *NetworkAccessProfile* 50
Figura 31: Entidade Gerenciada *NetworkCTP* 52
Figura 32: Entidade Gerenciada *NetworkTTP* 53
Figura 33: Entidade Gerenciada *RoutingProfile* 54
Figura 34: Entidade Gerenciada *StateChangeRecord* 55
Figura 35: *Subetwork* para a camada VC ou VP 56
Figura 36: Entidade Gerenciada *Subnetwork* 58
Figura 37: *SubnetworkConnection* para a camada VC ou VP 59

Figura 38: Entidade Gerenciada *SubnetworkConnection* 61
Figura 39: Entidade Gerenciada *TopologicalLink* 64
Figura 40: Entidade Gerenciada *TrafficDescriptorProfile* 65
Figura 41: Entidade Gerenciada *Trail* 67
Figura 42: Entidade Gerenciada *TrailRequest* 68
Figura 43: Entidade Gerenciada *VcLinkEnd* 70
Figura 44: Entidade Gerenciada *VpLinkEnd* 73
Figura 45: Modelo de Informação para Gerência de Rede ATM 74
Figura 46: Arquitetura da gerência de rede 79
Figura 47: Arquitetura de transporte para inicialização do sistema 79
Figura 48: Diagrama de seqüência da inicialização do sistema 80
Figura 49: Diagrama de seqüência da inicialização do registro de alarmes 82
Figura 50: Diagrama de seqüência de registro de uma aplicação de gerência 83
Figura 51: Diagrama de seqüência da criação, registro e envio de alarmes 85
Figura 52: Conexão de sub-rede para uma sub-rede simples 86
Figura 53: Diagrama de seqüência de criação de uma conexão de sub-rede simples 87
Figura 54: Diagrama de seqüência de criação de uma conexão de sub-rede simples com TPs 89
Figura 55: Sub-redes de particionamento de uma sub-rede 90
Figura 56: Diagrama de seqüência de criação de uma sub-rede de particionamento 91
Figura 57: Conexão de enlace entre sub-redes 92
Figura 58: Diagrama de seqüência de criação de um enlace entre sub-redes 93
Figura 59: Conexão de sub-rede composta 94
Figura 60: Diagrama de seqüência de criação de uma conexão de sub-rede composta 95
Figura 61: Trail servidor 97
Figura 62: Diagrama de seqüência de criação de um trail servidor 98
Figura 63: Diagrama de seqüência de operação agendada sobre um trail 100
Figura 64: Conexão de enlace entre sub-redes através de um trail servidor 102
Figura 65: Diagrama de seqüência de criação de um enlace entre sub-redes através de um trail ... 103
Figura 66: Relacionamento entre os componentes da arquitetura do JMX 107
Figura 67: Definição da interface *MinhaClasseMBean* 112
Figura 68: Definição da interface *DynamicMBean* 113
Figura 69: Componentes do modelo de notificação do JMX 115
Figura 70: Propagação de uma operação sobre um MBean 116
Figura 71: Operação do serviço MLet 117
Figura 72: Exemplo de uma classe UML 119
Figura 73: Exemplo de uma classe UML com herança 120
Figura 74: Exemplo de uma classe UML com atributos 120
Figura 75: Exemplo de uma classe UML com métodos 121
Figura 76: Exemplo de uma classe UML com métodos de envio de eventos 121
Figura 77: Exemplo de uma classe UML 138

Figura 78: Exemplo de uma classe UML com relacionamento unidirecional 138
Figura 79: Exemplo de uma classe UML com relacionamento bidirecional 140
Figura 80: Exemplo de uma classe UML com métodos de envio de eventos 141
Figura 81: Exemplo de uma classe UML com atributos e métodos 142

Lista de Tabelas

Tabela 1: Descrição dos componentes funcionais do TMN 13
Tabela 2: Hierarquia de gerência do TMN 15
Tabela 3: Elementos da entidade gerenciada *AlarmRecord* 34
Tabela 4: Elementos da entidade gerenciada *AlarmSeverityAssignmentProfile* 35
Tabela 5: Elementos da entidade gerenciada *AttributeValueChangeRecord* 36
Tabela 6: Elementos da entidade gerenciada *EventForwardingDiscriminator* 37
Tabela 7: Elementos da entidade gerenciada *LayerNetworkDomain* 39
Tabela 8: Elementos da entidade gerenciada *LinkConnection* 42
Tabela 9: Elementos da entidade gerenciada *Log* 43
Tabela 10: Elementos da entidade gerenciada *LogicalLinkTP* 45
Tabela 11: Elementos da entidade gerenciada *ManagedEntityCreationRecord* 47
Tabela 12: Elementos da entidade gerenciada *ManagedEntityDeletionRecord* 47
Tabela 13: Elementos da entidade gerenciada *Network* 48
Tabela 14: Elementos da entidade gerenciada *NetworkAccessProfile* 49
Tabela 15: Elementos da entidade gerenciada *NetworkCTP* 51
Tabela 16: Elementos da entidade gerenciada *NetworkTTP* 53
Tabela 17: Elementos da entidade gerenciada *RoutingProfile* 54
Tabela 18: Elementos da entidade gerenciada *StateChangeRecord* 55
Tabela 19: Elementos da entidade gerenciada *Subnetwork* 57
Tabela 20: Elementos da entidade gerenciada *SubnetworkConnection* 60
Tabela 21: Elementos da entidade gerenciada *TopologicalLink* 63
Tabela 22: Elementos da entidade gerenciada *TrafficDescriptorProfile* 65
Tabela 23: Elementos da entidade gerenciada *Trail* 66
Tabela 24: Elementos da entidade gerenciada *TrailRequest* 68
Tabela 25: Elementos da entidade gerenciada *VcLinkEnd* 69
Tabela 26: Elementos da entidade gerenciada *VpLinkEnd* 72
Tabela 27: Entidades gerenciadas referenciadas pelos *ensembles* 78

Lista de Abreviaturas e Siglas

AAL	ATM Adaptation Layer
AIS	Alarm Indication Signal
AP	Access Point
API	Application Program Interface
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BLA	Business Level Agreements
CIM	Common Information Model
CMIP	Common Management Infomation Protocol
CMIS	Common Management Infomation Service
CORBA	Common Object Request Broker Architecture
CTP	Connection Termination Point
DCN	Data Communication Network
DMK	Dynamic Management Kit
DSL	Digital Subscriber Line
ETSI	European Telecommunication Standards Institute
GDMO	Guideline for Definition of Managed Objects
HTML	Hypertext Markup Language
IP	Internet Protocol
ISO	International Organization for Standardization
ITU	International Telecommunications Union
JDMK	Java Dynamic Management Kit
JMX	Java Management Extensions
MD	Mediation Device
MIB	Management Information Base
NE	Network Element
NNI	Network Node Interface
OAM	Operation, Administration and Maintenance
OMG	Object Management Group
OS	Operations System
OSI	Open Systems Interconnection
OSS	Operations Support Systems
PDH	Plesiochronous Digital Hierarchy
QA	Q Adapter
QoS	Quality of Service
RDI	Remote Defect Indication
RO	Read-Only

RW	Read-Write
SDH	Digital Hierarchy
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Network
TCP	Transmission Control Protocol
TM	TeleManagement
TMN	Telecommunications Management Network
TP	Termination Point
TTP	Trail Termination Point
UML	Unified Modeling Language
UNI	User Network Interface
URL	Universal Resource Locator
VC	Virtual Channel
VCCP	Virtual Channel Connection Point
VCI	Virtual Channel Identifier
VCLC	Virtual Channel Link Connection
VCNC	Virtual Channel Network Connection
VCSC	Virtual Channel Sub-network Connection
VCT	Virtual Channel Trail Termination
VP	Virtual Path
VPCP	Virtual Path Connection Point
VPI	Virtual Path Identifier
VPLC	Virtual Path Link Connection
VPNC	Virtual Path Network Connection
VPSC	Virtual Path Sub-network Connection
VPT	Virtual Path Trail Termination
WBEM	Web-Based Enterprise Management
WDM	Wavelength Division Multiplexing
WS	Worstation

Resumo

Este trabalho tem por objetivo especificar o modelo de informação para uma aplicação de gerência de falhas e de configuração para o gerenciamento TMN no nível de rede, para redes de transmissão baseadas na tecnologia ATM, de acordo com a especificação JMX da Sun. O JMX possibilita o desenvolvimento de sistemas de gerenciamento através da utilização de componentes de software em Java, os JavaBeans. Além disso, este trabalho apresenta uma proposta para a implementação deste modelo de informação de gerenciamento de redes utilizando-se o ambiente de desenvolvimento de aplicações de gerência da Sun, denominado Java DMK.

Palavras Chave

Modelo de Informação, Gerência de Rede, TMN, ATM, Java, JMX.

Área de Conhecimento

Sistemas de Gerência de Redes de Telecomunicações.

Abstract

This document objective is to specify an information model for a fault and configuration management application for the TMN network level management, for networks based on the ATM technology, in accordance to the Sun JMX specification. The JMX makes possible to develop management systems through the use of the Java based software components, the JavaBeans. Furthermore, this document presents a proposal for implementing this network management information model using the Sun development environment for management applications called Java DMK.

Keywords

Information Model, Network Management, TMN, ATM, Java, JMX.

Knowledge Area

Telecommunication Management Network Systems.

1. Introdução

1.1. Motivações

Arquitetura TMN

O TMN (Telecommunication Management Network) foi especificado pelo ITU-T (International Telecommunications Union) como sendo uma infra-estrutura capaz de suportar o gerenciamento e o desenvolvimento de serviços de gerência de redes de telecomunicações. Através do entendimento dos conceitos, da implementação e da implantação de aplicações de gerência utilizando esta infra-estrutura TMN [Vertel 1999], as empresas de telecomunicações podem maximizar o potencial dos seus sistemas de gerenciamento e, conseqüentemente, dos seus equipamentos atuais, buscando desta forma aumentar a sua eficiência e a sua integração para garantir a satisfação de seus clientes.

O segmento de sistemas de gerenciamento das indústrias de telecomunicações continua a evoluir em função da disponibilidade de novas opções de tecnologias, as quais podem ser utilizadas tanto pelos provedores de serviços quanto pelas empresas de implementação de sistemas integrados de gerência de redes de telecomunicações. Tem-se tornado uma necessidade competitiva para estas empresas incorporar, continuamente, estas novas tecnologias a seus produtos. É inevitável, portanto, que a arquitetura e os padrões atuais do TMN devam também evoluir para se adaptarem à nova realidade do mercado de telecomunicações caso não queiram desaparecer.

Para constatar a influência que o TMN tem sobre a indústria de telecomunicações, em 1997 foi feita uma pesquisa entre empresas européias provedoras de serviços de telecomunicações em relação ao grau de compatibilização dos seus sistemas de gerência em relação aos padrões TMN do ITU-T, CORBA (Common Object Request Broker Architecture) [OMG 1998] da OMG (Object Management Group) e do TM Fórum (TeleManagement Fórum) e chegou-se aos seguintes resultados [EURESCOM 1999]:

- 28% dos sistemas seguem totalmente o padrão TMN do ITU-T;
- 27% dos sistemas seguem parcialmente o padrão TMN do ITU-T;
- 16% dos sistemas seguem parcialmente o padrão CORBA da OMG;
- 14% dos sistemas seguem parcialmente os padrões do TM Fórum.

Estes percentuais indicam que o TMN ainda é um padrão influente e relevante entre as empresas de prestação de serviços de gerência de redes de telecomunicações.

O mundo ideal proposto pelo ITU-T, em que todos os sistemas de uma planta heterogênea poderiam ser gerenciados de forma integrada através do TMN, utilizando modelos de informação padrões e

abertos, não é a realidade atual do mercado[1]. Entre os motivos pelos quais o modelo e a arquitetura do TMN, da forma em que estão propostos pelo ITU-T, não atingiram as expectativas de ampla difusão, aceitação e implantação estabelecendo-se como um padrão pelo mercado de aplicações de gerência de redes de telecomunicações, pode-se destacar [Barillaud 1997] [CTIT 1999] [EURESCOM 1999] [IEEE 1995]:

- a complexidade dos modelos de informação (MIB – Management Information Base), baseados em GDMO (Guideline for Definition of Managed Objects) e ASN.1 (Abstract Syntax Notation One);
- a falta de modelos de informação padrões e específicos em contraste com vários modelos de informação proprietários;
- os esforços iniciais do ITU-T foram para padronizar modelos para gerência do nível de elemento de rede, tipicamente de interesse estratégico na competitividade da indústria, ao invés dos níveis de rede, de serviço e de negócio;
- a complexidade dos protocolos OSI (Open Systems Interconnection) e CMIP (Common Management Infomation Protocol) utilizados pelo TMN;
- os altos custos envolvidos na implementação e na manutenção de sistemas de gerência TMN;
- a impossibilidade de suportar modelos de informação de equipamentos e de serviços de forma dinâmica e incremental;
- as plataformas de gerência atuais são baseadas no paradigma de gerenciamento centralizado, onde um pequeno número de estações recebe grandes quantidades de dados para que sejam analisados;
- a falta de recursos humanos capacitados na tecnologia TMN para a implementação de sistemas de gerência;
- os preços proibitivos das poucas plataformas comerciais de desenvolvimento de aplicações de gerência TMN disponíveis no mercado.

Além destes motivos, a falta de padronização é ainda mais notória nas duas áreas no modelo de gerência TMN, descritas na recomendação M.3010 da ITU-T [ITU M3010], denominadas gerência do nível de rede e gerência do nível de serviços, para as quais as empresas de telecomunicações estão desenvolvendo sistemas de gerência proprietários específicos para as suas linhas de produtos e normalmente incompatíveis com outros sistemas. Esta incompatibilidade entre sistemas de gerência não é necessariamente conseqüência da utilização de diferentes protocolos, mas sim conseqüência da utilização de diferentes modelos de informação por parte das aplicações de gerenciamento destes sistemas, inviabilizando portanto a integração entre os de diferentes fornecedores.

[1] O que ocorre na prática são modelos de informação para sistemas de gerência de elemento de rede proprietários ou abertos, mas, neste último caso, com restrições contratuais de utilização do tipo “Non Disclosure Agreement”.

De qualquer forma, o mercado de gerência de redes de telecomunicações está convergindo para estabelecer o TMN como a plataforma de gerência para os níveis de gerência mais baixos, elemento de rede e principalmente rede, enquanto que os níveis de gerência de serviços e de negócios estão naturalmente migrando para outras plataformas de gerência em função da quantidade e da localização de informações requeridas por estes sistemas de gerenciamento [Ranc 2000].

A implementação de um sistema padrão de gerência implica necessariamente na implementação de um modelo de informação bem especificado, padrão e disponível no mercado. Neste contexto, torna-se necessário especificar um modelo de informações para viabilizar a implementação de sistemas de gerência do nível de rede, para redes de transporte, de acordo com os padrões estabelecidos pelo TMN. Esta especificação implica na adoção de um modelo de informação padrão para redes baseadas em uma determinada tecnologia de transmissão.

Gerência de Rede ATM

O ATM (Asynchronous Transfer Mode) é uma tecnologia de transmissão assíncrona de dados, com comutação de circuito e de célula, capaz de suportar qualquer tipo de aplicação, independentemente dos seus requisitos de largura de banda.

A gerência TMN para redes de transmissão baseadas na tecnologia ATM está definida e modelada pelo ITU-T nas recomendações:

- G.805 [ITU G805]: Generic Functional Architecture of Transport Networks, que define a notação e uma arquitetura genérica para a gerência do nível de rede de sistemas de transmissão;
- I.326 [ITU I326]: Functional Architecture of Transport Networks based on ATM, que especializa a arquitetura genérica para a tecnologia ATM de transmissão;

pelo ETSI (European Telecommunication Standards Institute) nas recomendações:

- ETR 269 [ETSI ETR 269]: Network Level Information Modeling, que define a notação e uma arquitetura genérica para a gerência do nível de rede de sistemas de transmissão;
- I-ETS 300 653 [ETSI ETS 653]: Generic Managed Object Class Library for Network Level View, que especializa a arquitetura genérica para a arquitetura de uma rede de transporte;

e pelo ATM Fórum nas recomendações:

- AF-NM-0058.001 [AF 0058]: M4 Interface Requirements and Logical MIB: ATM Network View, que define a notação e uma arquitetura genérica para a gerência de uma rede de transporte de sistemas de transmissão;
- AF-NM-0073.000 [AF 0073]: M4 Network View: CMIP MIB Specification, que especializa a arquitetura genérica para a tecnologia ATM de transmissão.

O relacionamento entre estas recomendações para redes de transmissão ATM está representado na Figura 1.

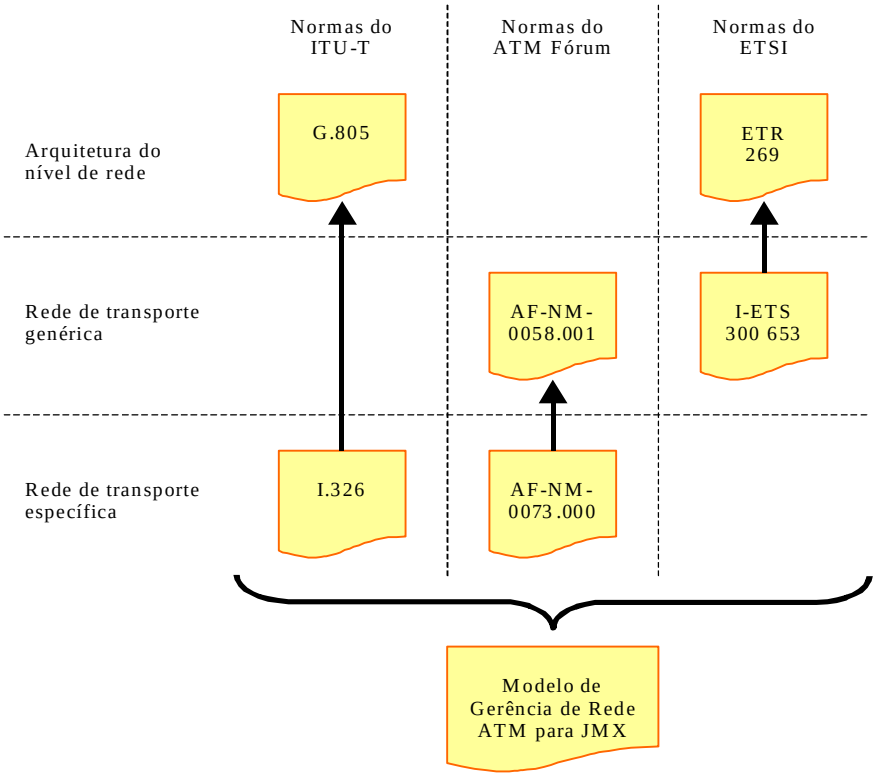

Figura 1: Relacionamento entre as normas de modelagem de rede

Estas recomendações foram desenvolvidas direcionadas basicamente ao protocolo de gerência CMIP e procuram ser o mais completas e abrangentes possível mas, talvez por isto, são extremamente grandes e complexas, praticamente inviabilizando a sua implementação ou pelo menos tornando-a proibitivamente cara. Por isso as empresas tem sido induzidas a especificarem e implementarem seus próprios modelos de informação para gerência de redes proprietários e, portanto, incompatíveis entre si.

Para especificar uma aplicação de gerência de configuração e de falhas de redes de telecomunicações, baseada na tecnologia ATM, é necessário que se desenvolva um modelo de informação para redes ATM. A melhor forma de se definir um modelo de informações é utilizar uma metodologia que permita obter um modelo independente da tecnologia ou protocolo de gerência [Pavlou 1998]. Isto implica em se utilizar uma notação neutra como o UML (Unified Modeling Language), de forma que o resultado se torne independente da plataforma de desenvolvimento a ser utilizada na implementação.

Tecnologia JMX

De uma maneira geral, a manutenção e as atualizações de aplicações de gerência são muito difíceis e caras de serem incorporadas nos ambientes de gerenciamento de redes baseados nas tecnologias

atualmente consagradas, tanto no CMIP quanto no SNMP (Simple Network Management Protocol), porque a sua natureza estática requer que os recursos e sistemas a serem gerenciados sejam previamente conhecidos pelos grupos de desenvolvimento dos sistemas de gerência de redes.

Recentemente, novas tecnologias de desenvolvimento de sistemas de gerenciamento de redes têm surgido no mercado, tipicamente na forma de mapeamento dos padrões de gerenciamento existentes em novos modelos de objetos, em especial no CORBA [Park 1999]. Infelizmente, as soluções baseadas nesta tecnologia têm demonstrado várias empecilhos na sua implantação principalmente em relação à enorme quantidade de código necessário para o seu desenvolvimento e às limitações e imperfeições do próprio mapeamento [Deri 1996] [Ranc 2000].

Por outro lado, novas plataformas, que seguem o modelo de orientação a objetos e flexibilizam a integração à Internet e o desenvolvimento de aplicações, têm se mostrado cada vez mais adequadas para o desenvolvimento de sistemas de gerenciamento. Estas plataformas são normalmente baseadas na linguagem Java da Sun. Portanto, pela tecnologia Java [Core Java V1] ser dinâmica, flexível e portável já a torna ideal para o desenvolvimento das novas gerações de soluções de gerência de sistemas [Barillaud 1997].

A especificação do JMX[2] (Java Management Extension), que visa suportar o desenvolvimento de aplicações dinâmicas de gerência baseados na tecnologia Java e integradas à Internet, incorpora os recursos inerentes à linguagem de programação Java ao paradigma gerente/agente proposto pela arquitetura do TMN. A proposta desta arquitetura, representada na Figura 2 e descrita detalhadamente no Capítulo 5.2, é viabilizada através da utilização dos componentes de software do Java: os JavaBeans[3] [Core Java V2].

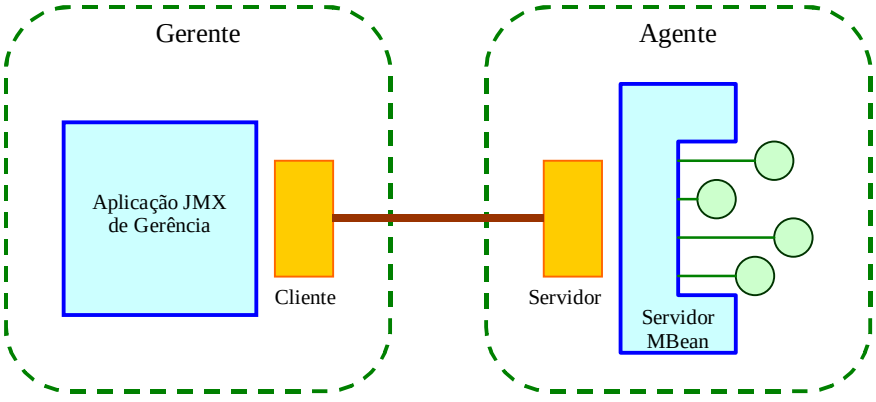

Figura 2: Arquitetura gerente/agente do JMX

[2] Marca registrada da Sun Microsystems.

[3] Os JavaBeans são os componentes de software da tecnologia Java que seguem um padrão de definição de sua interface e que externalizam suas propriedades internas (características e funcionalidades) para que possam ser incorporados e executados por qualquer aplicação em tempo de execução.

O JMX [JMXWP 1999] foi criado por um consórcio de empresas da área de sistemas de gerência para suprir os requisitos de mercado para o gerenciamento dinâmico de recursos e de sistemas através da tecnologia Java. O JMX oferece todo o suporte requerido pelos desenvolvedores de soluções de gerência.

Através do mapeamento de um modelo de informação de gerência especificado em UML para os JavaBeans utilizados pela arquitetura especificada pelo JMX, cria-se a infra-estrutura necessária para o desenvolvimento de um sistema de gerência TMN, para o nível de rede, baseado na tecnologia Java de acordo com a especificação do JMX.

A própria Sun desenvolveu uma plataforma de desenvolvimento de aplicações de gerência baseada na tecnologia especificada pelo JMX: o Java DMK (Dynamic Management Kit) [JDMKWP 1998] [JDMK 1999]. O Java DMK é uma solução integrada baseada na tecnologia Java para o desenvolvimento de soluções dinâmicas de gerência e para a distribuição natural do gerenciamento entre os dispositivos da rede. Esta plataforma de desenvolvimento baseia-se em componentes de software JavaBeans e possibilita, de forma eficiente e independente, a criação de gerentes e de agentes Java para sistemas, redes e serviços.

Outra empresa que possui uma linha de produtos de desenvolvimento de aplicações de gerência baseados na tecnologia especificada pelo JMX é a AdventNet [AGTTKT 2000]. A ferramenta de desenvolvimento de agentes da AdventNet permite que rapidamente se construam aplicações independentes de plataforma baseadas em componentes de software JavaBeans de acordo com o JMX.

1.2. Objetivo

O objetivo deste trabalho é especificar um sistema dinâmico de gerência de falhas e configuração para redes ATM baseado nos modelos de informação para o gerenciamento TMN no nível de rede propostos por organismos de padronização, tais como ITU-T, ETSI e ATM Fórum. Para tanto, será desenvolvido um modelo de informação, apresentado genericamente em UML e especificamente em JavaBeans, e uma metodologia de utilização para a sua implementação.

A modelagem de informações de gerência baseada em JavaBeans segue a proposta da Sun para a implementação de aplicações de gerência de redes TMN de acordo com a especificação JMX. Esta especificação utiliza componentes de software em Java, os JavaBeans, para representar dinamicamente as entidades a serem gerenciadas.

A implementação completa do modelo de informação de gerência de redes representado em JavaBeans, de acordo com a arquitetura JMX, através da plataforma de desenvolvimento Java DMK, é a etapa seguinte que fica proposta como complementação natural deste trabalho e que servirá como validação prática dos estudos, conceitos e resultados a serem apresentados.

1.3. Descrição de Trabalhos Existentes

A implementação de sistemas de gerência TMN para o nível de rede tem sido desenvolvido por algumas empresas, tais como Siemens, Alcatel e Lucent, mas a partir de modelos proprietários ao invés de estarem baseados exclusivamente nos modelos propostos pelos organismos internacionais de padronização.

Quanto ao uso da tecnologia Java, através da arquitetura JMX da Sun, para o desenvolvimento de aplicações de gerência TMN, existem alguns trabalhos sendo desenvolvidos pelo grupo *Munich Network Management Team* (http://wwwmnmteam.informatik.uni-muenchen.de) da Universidade de Tecnologia de Munique. O campo de pesquisa deste grupo é o gerenciamento integrado de ambientes de computação distribuída. O seu trabalho se baseia em experiências práticas e conhecimentos adquiridos a partir de trabalhos de cooperação com empresas fornecedoras de equipamentos de infra-estrutura de redes e com desenvolvedores de sistemas integrados de gerência.

Um projeto interessante, publicado por este grupo da Universidade de Tecnologia de Munique, é uma aplicação de gerência de um sistema de telefonia IP [Reiser 1999] [Knöchlein 1999] que se utiliza do ambiente de desenvolvimento Java DMK da Sun. O trabalho é o resultado da avaliação do Java DMK para o desenvolvimento de um protótipo feito em parceria com a Siemens AG de Munique para gerenciar o seu sistema de telefonia Hicom 300.

A principal motivação do grupo para avaliar o Java DMK foi a crescente demanda por sistemas escalonáveis e confiáveis e que permitam a extensão da funcionalidade de gerência dos agentes em tempo de execução. Como resultado, a experiência adquirida pelo grupo com o projeto permitiu avaliar a aplicabilidade do ambiente de desenvolvimento Java DMK para o desenvolvimento e suporte de soluções de gerenciamento de redes corporativas de larga escala.

1.4. Organização deste Documento

Este documento é composto basicamente pela apresentação teórica dos vários conceitos envolvidos neste trabalho e pelo desenvolvimento da especificação do modelo de informação em UML, de uma

metodologia para a sua utilização e de uma especialização deste modelo de acordo com a arquitetura JMX da Sun.

O primeiro capítulo será a introdução do trabalho apresentando principalmente o estado da arte atual dos sistemas de gerência TMN e a motivação que levou a propor o desenvolvimento do modelo de informação de rede genericamente em UML e especificamente de acordo com a arquitetura JMX.

O segundo capítulo apresentará os conceitos relativos à arquitetura TMN e os fundamentos dos modelos de informação para gerência do nível de rede especificados pelos organismos internacionais de padronização.

O terceiro capítulo conterá, primeiramente, uma introdução à tecnologia ATM, focando principalmente os conceitos básicos inerentes às redes de transmissão baseadas nesta arquitetura e os fundamentos da modelagem de informação para a sua gerência no nível de rede. Em seguida, será desenvolvido o modelo de informação de gerência TMN para o nível de rede para o gerenciamento de configuração e de falhas de uma rede de transmissão baseada nesta tecnologia de transmissão. Este modelo será apresentado em UML de forma que possa ser especializado para qualquer tecnologia de desenvolvimento de aplicações de gerência TMN.

O quarto capítulo apresentará uma metodologia para possibilitar a implementação do modelo de informação de gerência TMN de redes baseado na tecnologia ATM, descrito em UML, através da utilização de cenários e de diagramas de seqüência UML.

O quinto capítulo conterá uma compilação dos conceitos e técnicas envolvidos na utilização da especificação da arquitetura JMX da Sun e desenvolverá a especificação do modelo de informação de gerência de rede definido em UML no formato especificado pela arquitetura JMX e requerido pelo ambiente de desenvolvimento de sistemas de gerenciamento Java DMK.

O sexto capítulo apresentará a conclusão deste trabalho de desenvolvimento do modelo de informação TMN para um sistema de gerência de falhas e configuração para redes ATM, com uma especialização para a arquitetura JMX da Sun.

2. O TMN

2.1. Introdução

O TMN provê um ambiente de desenvolvimento que permite a interconectividade e a comunicação entre sistemas operacionais heterogêneos e redes de telecomunicações. TMN foi proposto pelo ITU-T como sendo uma infra-estrutura para suportar o gerenciamento e o desenvolvimento de serviços de gerência de redes de telecomunicações.

Para o gerenciamento de redes de telecomunicações, de serviços e de equipamentos, o ITU-T propôs a série M.3000 de recomendações definindo o TMN. Estas recomendações continuam sendo atualizadas para refletir as atuais necessidades do mercado e as suas tendências em relação a novas arquiteturas de desenvolvimento e de comunicação que estão sendo disponibilizadas. Atualmente a tecnologia TMN está sendo aplicada amplamente no gerenciamento de telefonia e de redes de transmissão SDH (Synchronous Digital Hierarchy), ATM e sem fio.

O objetivo deste capítulo é apresentar os conceitos do TMN de uma forma integrada, servindo assim como referência inicial para o entendimento dos padrões que compõe a sua especificação. Um estudo mais aprofundado do TMN está disponível em [Higa 1999] e uma comparação destes conceitos com os da gerência de redes SNMP está descrita em [Bertholdi 1999].

2.2. Estrutura do TMN

O TMN oferece uma arquitetura para a gerência de redes que é flexível, escalonável, confiável e facilmente ampliável, pois padroniza as formas de execução dos procedimentos de gerência de redes e de comunicação através de redes. O TMN permite a distribuição de processamento possibilitando a obtenção de níveis apropriados de desempenho e de eficiência de comunicação.

Os princípios do TMN devem ser incorporados numa rede de telecomunicações para a transmissão e a recepção de informações e para o gerenciamento de seus recursos [Vertel 1999]. Uma rede de telecomunicações é composta por vários equipamentos, tais como sistemas de comutação, circuitos de transmissão e terminais. Na terminologia do TMN, estes recursos são denominados elementos de rede, os Nes (Network Elements).

O TMN é por definição uma rede de computadores paralela à rede de equipamentos de telecomunicações, a qual permite a comunicação entre sistemas de suporte de operações (OSS) e elementos de rede (NE). Esta arquitetura está representada na Figura 3.

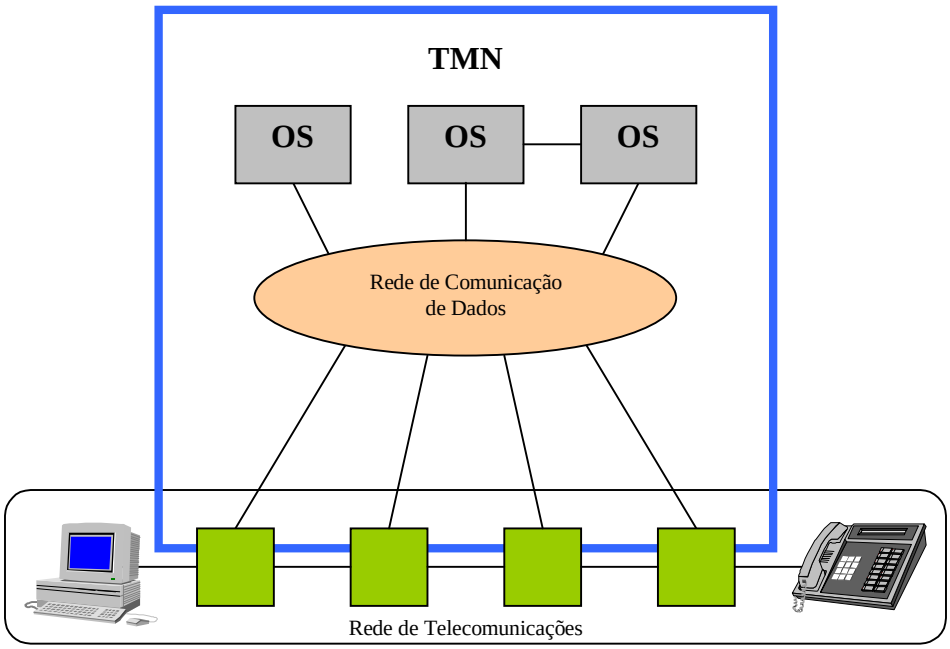

Figura 3: O TMN e a Rede de Telecomunicações

2.2.1. Padrão TMN

O TMN foi definido pelo ITU-T através da série de recomendações M.3000, e especialmente na recomendação M.3010 [ITU M3010]. Quando as redes de telecomunicações implementam as definições do TMN elas tornam-se inter-operáveis, mesmo quando interagem com redes e equipamentos de outros fornecedores e/ou provedores de serviços de telecomunicações.

Os princípios de orientação a objetos e as interfaces padrões são utilizados pelo TMN para definir a comunicação entre entidades gerenciáveis em uma rede. A interface de gerência padrão para o TMN é denominada interface Q.

A arquitetura e as interfaces do TMN foram baseadas nos padrões existentes da ISO (International Organization for Standardization). Estes padrões incluem:

- **CMIP** [ITU X711]: define os serviços de gerência compartilhados entre entidades gerenciadas;
- **GDMO** [ITU X722]: provê modelos para classificar e descrever os recursos gerenciáveis;
- **ASN.1** [ITU X208]: provê regras de sintaxe para representar os tipos de dados;
- **modelo de referência de sistemas abertos**: define as sete camadas OSI do modelo de referência.

Desde a sua publicação, os padrões do TMN têm sido utilizados por outros organismos de padronização, principalmente o TM Fórum, a Bellcore, o ATM Fórum e a ETSI [Vertel 1998]. Em

geral, os esforços do TM Fórum e da Bellcore são direcionados à implementação e à disponibilização de um ambiente genérico de estabelecimento de requisitos detalhados de gerência de redes de telecomunicações, enquanto que os esforços do ATM Fórum e do ETSI estão direcionados à especificação de interfaces de gerência TMN específicas.

2.2.2. TMN e Gerência

Os benefícios do TMN são importantes pois permitem empresas gerenciar redes e serviços complexos e dinâmicos, assim como permitem que estas mesmas empresas continuem expandindo seus serviços e mantendo a sua qualidade.

Através do TMN são descritas as redes para a gerência de telecomunicações utilizando visões específicas da rede fornecidas através do modelo lógico, do modelo de negócio ou do modelo funcional. Toda esta representação é realizada através de um conjunto de interfaces padrão.

O TMN foi no início exclusivamente baseado na infra-estrutura de gerenciamento especificado pela ISO (gerenciamento de sistemas OSI), onde as funções de gerência eram executadas exclusivamente por operações especificadas pelas primitivas do CMIS [ITU X710], que se utiliza da pilha OSI de comunicação. As novas revisões das recomendações relativas à especificação do TMN não estabelecem mais a obrigatoriedade de se manter integralmente a arquitetura da gerência OSI como base para o desenvolvimento de aplicação de gerenciamento de redes de telecomunicações. Já as informações de gerência dos recursos de rede continuam sendo modeladas como atributos de objetos gerenciados.

2.2.3. Modelo de Informação de Gerência

As informações de gerência de uma rede de telecomunicações, assim como as regras pelas quais estas informações são apresentadas e gerenciadas, é referenciada como sendo a base de informações de gerência (MIB), a qual contém o seu modelo de informação. Aplicações que gerenciam estas informações são denominados de entidades de gerenciamento. Uma entidade de gerenciamento pode assumir um dos seguintes dois papéis: gerente ou agente. Gerentes e agentes enviam e recebem requisições e notificações usando, por exemplo, o protocolo CMIP.

A implementação de um sistema padrão de gerência implica necessariamente na implementação de um modelo de informação bem especificado, padrão e disponível no mercado. Na falta de um modelo de informação, torna-se necessário especificá-lo e disponibilizá-lo para, desta forma, viabilizar a implementação de sistemas padrões de gerência, de acordo com as normas estabelecidos pelo TMN.

2.3. Modelo Funcional do TMN

O TMN permite aos provedores de serviços de telecomunicações viabilizar a interconectividade e a comunicação entre Oss e redes de telecomunicações. A interconectividade é conseguida através de interfaces padronizadas capazes de abstrair os recursos gerenciados na forma de objetos gerenciáveis.

2.3.1. Blocos Funcionais do TMN

O TMN é representada por vários blocos funcionais que provêm um encapsulamento de funcionalidades de gerenciamento e que podem ser apresentados de acordo com a Figura 4. A Tabela 1 lista e descreve cada um dos componentes funcionais definidos pelo TMN, com a sua respectiva funcionalidade.

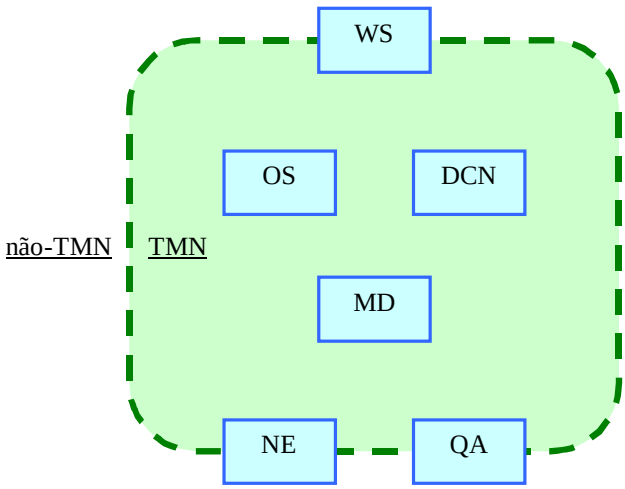

Figura 4: Blocos Funcionais do TMN

Bloco Funcional	Descrição
OS (Operations System)	Executa funções do sistema operacional, inclusive monitoração de operações e controle de funções de gerência de telecomunicações. Um OS pode prover também algumas funcionalidades de MD, QA, WS.
MD (Mediation Device)	Provê a mediação entre interfaces TMN locais e o modelo de informações do OS. A função de mediação pode ser necessária para garantir que informações, escopo e funcionalidades estejam presentes precisamente na forma especificada requerida pelo OS. As funções de mediação podem ser implementadas através de MDs em cascata.

QA (Q Adapter)	A QA permite a um TMN gerenciar Nes que não implementem interfaces TMN: é um tradutor entre interfaces TMN e interfaces não-TMN.
NE (Network Element)	No escopo do TMN, um NE contém informações gerenciáveis que são monitoradas e controladas por um OS. Para ser gerenciável dentro do escopo do TMN, um NE deve ter uma interface TMN padrão. Por outro lado, se um NE não possuir uma interface padrão, ele ainda pode ser gerenciado, mas através de um QA. Um NE provê ao OS uma representação das suas informações e funcionalidades gerenciáveis (MIB). Um NE pode prover também algumas funcionalidades de OS, QA, MD.
WS (Worstation)	A WS executa funções de estação de trabalho. Ele traduz informações do formato TMN e as disponibiliza num formato apresentável ao usuário.
DCN (Data Communication Network)	A DCN é a rede de comunicações dentro do TMN. Ela representa as camadas 1 à 3 da pilha OSI.

Tabela 1: Descrição dos componentes funcionais do TMN

2.3.2. Funcionalidade OSI para o TMN

Os blocos funcionais podem atuar no papel de um agente ou de um gerente. Os conceitos do TMN de agente e de gerente são os mesmos dos usados pela gerência OSI, os quais estão ilustrados na Figura 5.

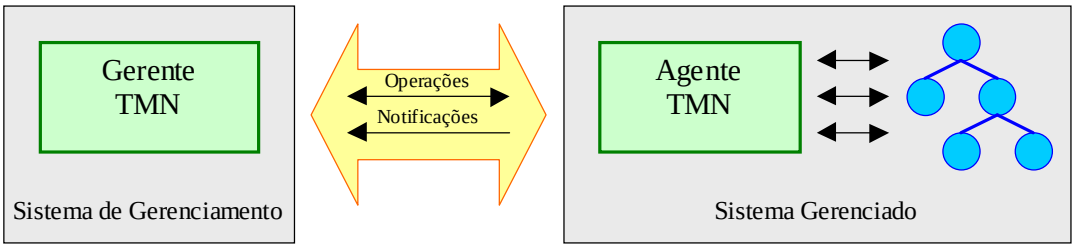

Figura 5: Interação entre Gerente, Agente e Objetos Gerenciáveis

Uma aplicação TMN no papel de gerente faz requisições de operações e recebe notificações, enquanto que no papel de agente processa operações, envia respostas e emite notificações. Esta aplicação TMN pode fazer ao mesmo tempo o papel de gerente para uma outra aplicação agente e o papel de agente para uma outra aplicação gerente.

Um objeto gerenciável é uma visão conceitual de um recurso que necessita ser monitorado e controlado para evitar falhas e degradação de desempenho em uma rede. Os objetos gerenciáveis com as mesmas propriedades são instâncias de uma classe de objetos. A MIB é por definição um repositório conceitual de instâncias de objetos gerenciáveis. Uma classe de objeto gerenciável é definida pelos:

- **atributos**: são elementos de dados e valores que caracterizam uma classe de objetos gerenciáveis;
- **operações de gerência**: são as operações que podem ser aplicadas às instâncias de objetos gerenciáveis;
- **comportamento**: é o que é apresentado por uma instância de objeto gerenciável baseado no recurso que ele representa;
- **notificações**: são as mensagens que instâncias de objetos gerenciáveis podem emitir espontaneamente.

2.4. Modelo Lógico do TMN

O TMN provê um modelo de camadas lógicas que define três níveis de gerência para funcionalidades específicas. Os mesmos tipos de funções podem ser implementados em várias outras camadas, desde um nível mais alto que gerencia assuntos corporativos até um nível mais baixo que pode ser definido por uma rede ou por um recurso de rede.

Para lidar com a complexidade inerente ao gerenciamento, a funcionalidade de gerência, em conjunto com sua informação associada, pode ser decomposta entre as camadas lógicas, conforme representado na Figura 6.

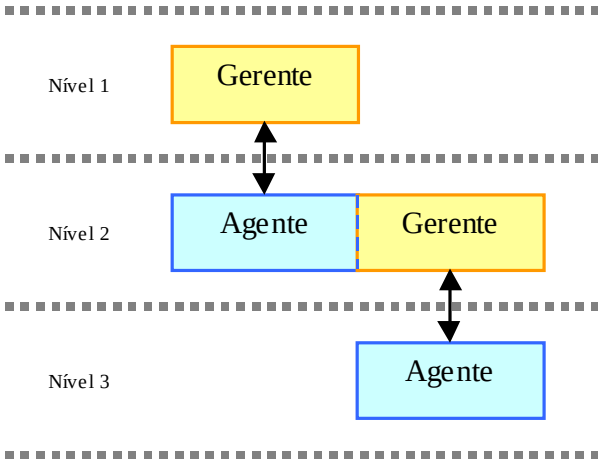

Figura 6: Decomposição da Funcionalidade de Gerência

A partir da sua camada inferior, a hierarquia inclui: NEs, gerência de elementos de rede, gerência de redes, gerência de serviços e gerência de negócios. Na Tabela 2 é apresentada a funcionalidade, do ponto de vista de gerência, de cada uma das camadas da hierarquia.

Camada	Responsabilidade
Gerência de negócios	Coordena o planejamento de alto nível, as cobranças, as estratégias, as decisões executivas, os contratos de negócios (BLA (Business Level Agreements)).
Gerência de serviços	Utiliza as informações disponibilizadas pela camada de gerência de rede para gerenciar contratos de serviço de clientes, desde o provisionamento e a qualidade de serviço, até a gerência de falhas. Esta gerência é também responsável pela interação com os provedores de serviços e com outros domínios administrativos, mantendo dados estatísticos para garantir a qualidade do serviço prestado.
Gerência de rede	Tem a visibilidade de toda a rede baseada em informações de NEs disponibilizadas pelos OSs da camada de gerência de elemento de rede. Esta gerência coordena todas as atividades de rede e suporta as requisições da camada de gerência de serviços.
Gerência de elemento de rede	Gerencia cada elemento de rede. Esta gerência possui OSs, cada um dos quais é responsável pelas informações gerenciáveis de NEs específicos. De uma forma geral, um gerente de elemento de rede é responsável por um subconjunto dos elementos de rede, gerenciando os seus dados, atividades e registros.
Elemento de rede	Apresenta as informações gerenciáveis TMN de cada um dos NEs individualmente. Esta gerência faz a interface entre a informação gerenciável proprietária e a infra-estrutura TMN.

Tabela 2: Hierarquia de gerência do TMN

Na Figura 7 é apresentado um exemplo de como componentes TMN de diferentes níveis lógicos de gerenciamento podem estar relacionados.

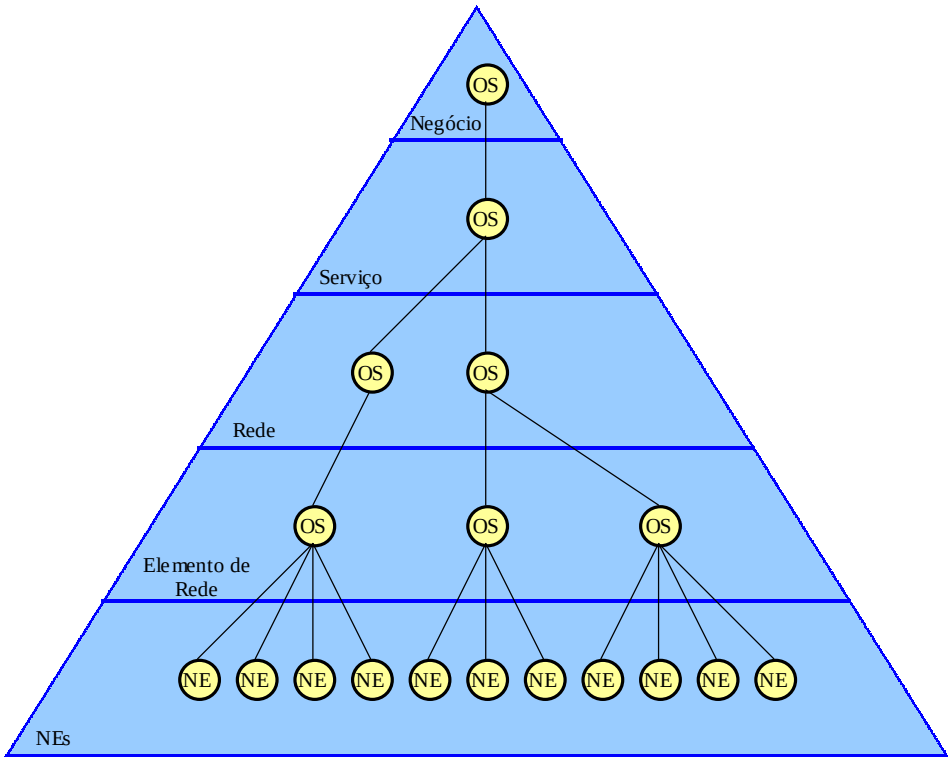

Figura 7: Exemplo de Componentes TMN Distribuídos nas Camadas Lógicas

Há sistemas de gerenciamento desenvolvidos atualmente pelo mercado em que mais de um nível lógico foram integrados e implementados em uma mesma aplicação.

2.5. Conclusão

O TMN, especificado e mantido pelo ITU-T, define uma arquitetura de interconectividade e de comunicação entre sistemas operacionais heterogêneos e redes de telecomunicações. Mas, devido a diversos fatores (principalmente os altos custos, a complexidade de implementação e de manutenção e a falta de padronização), a indústria de sistemas de gerência de telecomunicações tem pesquisado outras formas de padronizar os modelos de informação para a gerência de sistemas de telecomunicações e outras soluções para o desenvolvimento de aplicações TMN.

Este trabalho baseia-se no modelo lógico do TMN para desenvolver um modelo de informação para o gerenciamento de configuração e de falhas, para o nível de rede, de redes baseadas na tecnologia ATM. O modelo está especificado em UML, tornando-o desta forma independente da plataforma e da arquitetura que venha a ser implementado.

3. Modelo de Informação para Gerência de Rede ATM

3.1. Introdução

Nos últimos anos, diferentes redes de transmissão baseadas em diferentes tecnologias foram desenvolvidas de forma praticamente independente. Por isso, por um lado têm-se as redes de telefonia capazes de transmitir os sinais de voz e, com algumas restrições por causa de largura de banda, sinais de fax e de dados, e, por outro lado, têm-se as redes criadas especialmente para suprir os requisitos da transmissão de dados.

Baseado nesta realidade, surgiu o ATM, com o objetivo de prover uma tecnologia de transmissão de dados em uma rede de telecomunicações e capaz de suportar qualquer tipo de aplicação atual ou futura independente dos seus requisitos de largura de banda. O ATM permite, portanto, combinar as redes de transmissão de dados com as redes telefônicas em uma única tecnologia. A tecnologia do ATM apresenta as seguintes vantagens [Schultz 1999]:

- integração de vários serviços, tais como voz, imagem, vídeo, dados e multimídia, com adaptação para diferentes requisitos e configurações de tráfego;
- padronização das estruturas de redes e de seus componentes, diminuindo assim o seu custo de implantação e manutenção;
- provisionamento de larguras de banda para novas tecnologias, tais como Internet, ensino a distância, tele-medicina, vídeo por demanda;
- transmissão independente e transparente do meio físico utilizado (PDH (Plesiochronous Digital Hierarchy), SDH, SONET (Synchronous Optical Network), WDM (Wavelength Division Multiplexing));
- escalabilidade através da adaptação flexível da largura de banda para requisitos específicos de usuários;
- garantia da qualidade de serviço (QoS (Quality of Service)) de transmissão de acordo com requisitos mínimos de serviço requeridos pelos usuários;
- uso efetivo da mesma tecnologia de transmissão desde o usuário final até os serviços de transmissão de dados de longa distância.

Para especificar uma aplicação de gerência de configuração e de falhas de redes de telecomunicações, baseada na tecnologia ATM, é necessário que se desenvolva um modelo de informação para redes ATM. Este modelo de informação deve modelar a funcionalidade de gerenciamento de acordo com os requisitos mínimos apresentados pelo ATM Fórum para as gerências de configuração e de falhas. A recomendação [AF 0073] do ATM Fórum contém a especificação do modelo de informação para redes ATM descrita em GDMO, o que restringe o desenvolvimento de uma aplicação de gerência a uma plataforma CMIP.

A melhor forma de se definir um modelo de informações é utilizar uma metodologia que permita obter um modelo independente da tecnologia ou protocolo de gerência [Pavlou 1998]. Isto implica em se utilizar uma notação neutra como o UML, de forma que o resultado se torne independente da plataforma de desenvolvimento a ser utilizada na implementação.

O modelo de informação é constituído por um conjunto de entidades gerenciadas e pela forma com que elas se relacionam. As entidades gerenciadas descritas em UML representam a informação necessária para gerenciar os recursos de uma rede e os seus estados operacionais. Dependendo da plataforma e do protocolo em que serão implementadas, estas entidades não precisam necessariamente ser representadas na forma de classes de objetos, isto é, não há necessariamente um mapeamento um a um entre entidades gerenciadas e classes de objetos.

O objetivo deste capítulo é desenvolver o modelo de informação de gerência TMN para o nível de rede para o gerenciamento de configuração e de falhas de uma rede de transmissão baseada na tecnologia de transmissão ATM. Este modelo será apresentado em UML, através dos diagramas de classe, de forma que possa ser especializado para qualquer tecnologia de desenvolvimento de aplicações de gerência TMN. Para tanto, primeiramente serão apresentados os conceitos básicos envolvidos nesta tecnologia de transmissão, com ênfase especial nos componentes das redes ATM e na sua estruturação. Um estudo mais completo e abrangente da tecnologia ATM está disponível em [Puka 2000].

3.2. A Tecnologia de Transmissão ATM

3.2.1. Conceitos Básicos do ATM

Células ATM

O ATM é um método de transmissão de dados com comutação de circuito e comutação de célula, que utiliza células com um tamanho fixo de 53 bytes para transmitir tanto dados de usuário quanto informações de sinalização. Isto implica que o método é diferente dos sistemas de comutação de pacotes, pois estes usam pacotes de dados de comprimento variável.

A célula ATM é a menor unidade de informação padrão em uma rede ATM. Toda a informação de usuário e de sinalização tem de estar encapsulado por estas células. Dos 53 bytes que compõe uma célula, 5 bytes correspondem ao cabeçalho da célula e os 48 bytes restantes estão disponíveis para conter os dados de usuário ou as informações de sinalização. A informação contida no cabeçalho da célula é utilizada principalmente para direcioná -la ao longo da rede ATM.

As células ATM estão sincronizadas no tempo e, desta forma, compõe uma cadeia de células de dados contínua e constante no tempo, como representado na Figura 8.

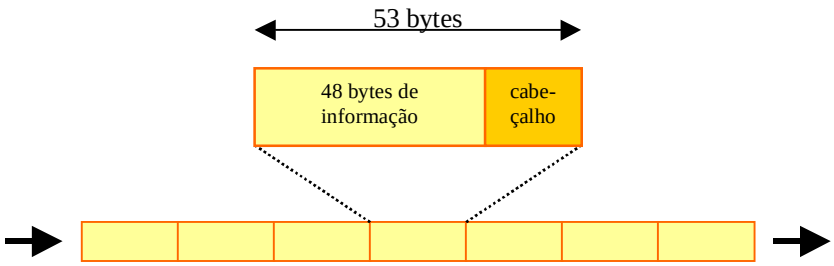

Figura 8: Cadeia de células ATM

Em comparação com sistemas síncronos de transmissão, cujos pacotes de dados têm posições fixas no tempo, as células ATM que trafegam por um equipamento terminal não têm uma posição predeterminada na cadeia de células. Desta forma, os requisitos de largura de banda são obtidos utilizando-se um número correspondente de células por unidade de tempo, tornando o sistema altamente flexível.

Uma cadeia constante de células ATM move-se do usuário para a rede. Caso não haja dado para ser transmitido durante um período, uma célula vazia é inserida na cadeia de células, mantendo-se desta forma a taxa de transmissão constante.

Há ainda células especiais utilizadas para a transmissão de informações específicas de sinalização, denominadas células OAM (Operation, Administration and Maintenance), as quais podem ser inseridas na cadeia de células sempre que for necessário. Este tipo de célula é utilizado no gerenciamento das redes ATM pois contêm informações para a monitoração de alarmes da rede ATM, para o controle dos seus elementos de rede e para a localização de falhas.

Interfaces do ATM

Para que haja uma uniformidade na especificação e na divisão de funcionalidades, são definidas as seguintes interfaces para uma rede ATM:

- UNI (User Network Interface): interface de comunicação entre o usuário e o equipamento terminal;
- NNI (Network Node Interface): interface de comunicação entre os equipamento da rede.

As redes privadas têm sua própria regulamentação para as interfaces NNI e UNI, as quais estão definidas pelo ATM Fórum. Os protocolos diferentes de sinalização para cada uma destas interfaces são definidos pelo ITU-T.

Comutação de Circuito ATM

O ATM é uma técnica de comunicação baseado em comutação de circuito, o que significa que a conexão através da rede deve ser estabelecida antes de que a informação possa ser transferida. A conexão ao longo de uma rede ATM é denominada virtual, pois não existe fisicamente, mas está presente apenas na forma de tabelas de roteamento nos roteadores ATM.

As células ATM são guiadas ao longo da rede através das informações contidas nos campos VCI (Virtual Channel Identifier) de 16 bits e VPI (Virtual Path Identifier) de 8 bits presentes no seu cabeçalho, como está representado na Figura 9.

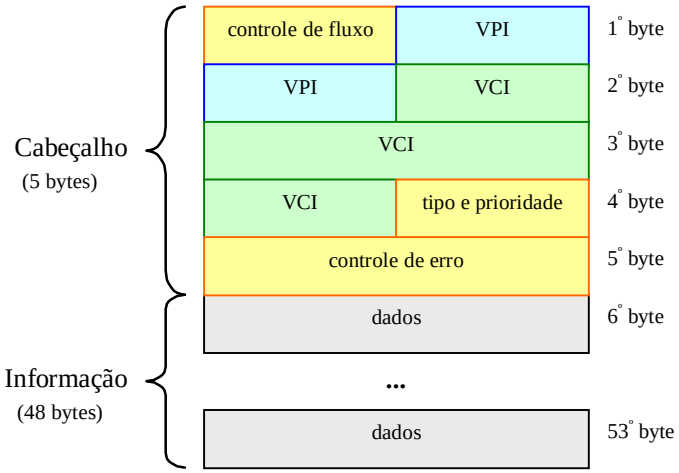

Figura 9: Estrutura do cabeçalho das células ATM

O campo VCI do cabeçalho da célula contém uma parte das instruções de endereçamento. As células pertencentes a um mesmo canal virtual terão o mesmo VCI. Cada VCI (como no exemplo da Figura 10) indica um caminho entre centros de comutação ou entre um centro de comutação e um terminal de usuário. O conjunto de VCIs identificam um caminho dentro da rede.

O campo VPI contém a segunda parte das instruções de endereçamento e é de prioridade maior que o VCI. O VPI aglutina vários canais virtuais permitindo o rápido direcionamento das células através de redes que contém equipamentos de *cross-connection* ATM, os quais são capazes de comutar cadeias de células em várias direções de acordo com o especificado no VPI (veja Figura 10).

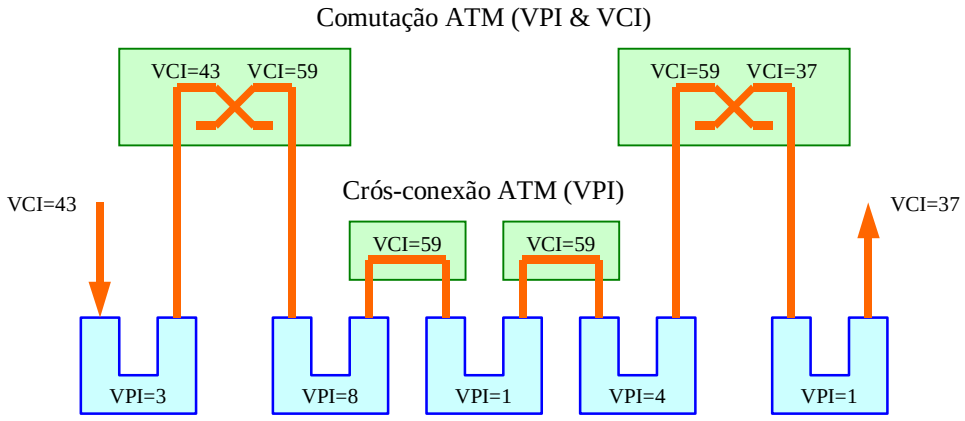

Figura 10: Exemplo de um caminho virtual

Os campos VPI e VCI são definidos pelos centros de comutação ATM assim que uma conexão é estabelecida e identificam univocamente todas as células pertencentes a uma conexão específica. Quando uma conexão é encerrada, estes valores são disponibilizados à rede para que possam ser novamente utilizados.

Como apresentado na Figura 10, os circuitos de *cross-connection* podem alterar o valor do VPI. A comutação de células através do gerenciamento das duas partes componentes da instrução de endereçamento é de responsabilidade exclusiva dos comutadores ATM.

Modelo de Referência do ATM

O modelo ATM é composto de 4 camadas baseadas no princípio do modelo de camadas OSI da ISO. A representação em camadas do ATM está apresentada na Figura 11. Para representar a tecnologia ATM, foram utilizadas 2 camadas específicas para o ATM: ATM e AAL (ATM Adaptation Layer).

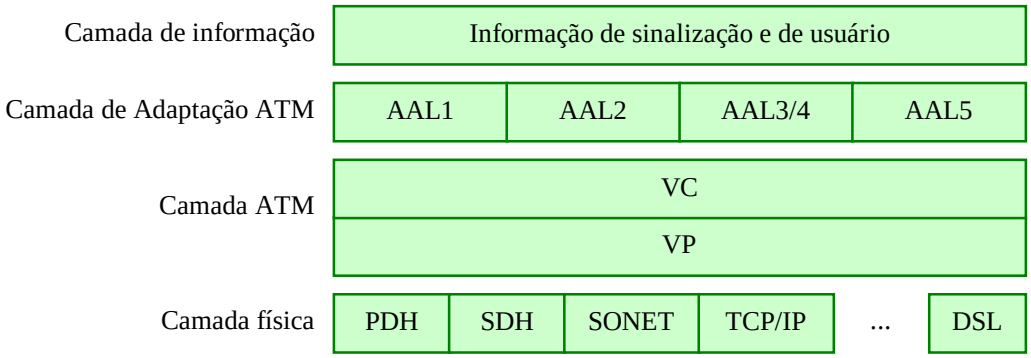

Figura 11: Modelo de referência ATM

A camada de adaptação (AAL) é responsável pela adaptação dos dados provenientes das camadas mais altas de informação ao formato do campo de informação das células ATM. Foram criados 4 tipos de serviços de adaptação, denominados respectivamente de AAL1, AAL2, AAL3/4, AAL5, cada qual com determinadas características de forma que possa prover os serviços de acordo com o requerido pelas tecnologias utilizadas na camada física.

A camada ATM é responsável principalmente pelo transporte e pela comutação das células ATM. Esta camada inclui o cabeçalho na célula recebida da camada de adaptação e é responsável pelo seu processamento, o que inclui a análise e o tratamento das informações VPI e VCI.

A camada física do modelo de referência do ATM não especifica um meio de transmissão específico, permitindo assim que várias tecnologias de transmissão possam ser utilizadas de acordo com as características ou requisitos de usuário da rede ATM.

Gerência de Redes ATM

As células OAM são utilizadas por uma rede ATM para que esta possa ser monitorada permitindo, desta forma, a detecção de erros, a monitoração da qualidade de uma conexão e a configuração de dados de desempenho para um determinado elemento desta rede. Estas células especiais utilizam-se do mesmo caminho através da rede ATM que as células comuns de transporte de informações de usuário.

Na gerência de falhas de uma rede ATM, as células OAM são utilizadas para enviar os alarmes para os terminais de usuário que contiverem aplicações de monitoração de falhas.

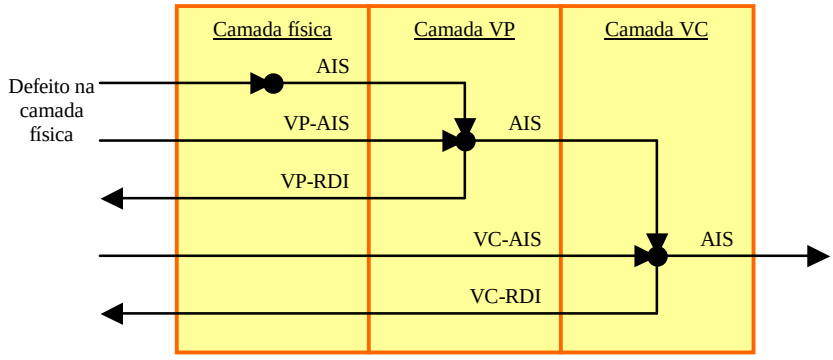

Figura 12: Gerência de alarmes ATM

Por exemplo, a Figura 12 apresenta os alarmes gerados a partir de um defeito na camada física. Este defeito é indicado para a camada VP (Virtual Path) através de um sinal de indicação de alarme AIS (Alarm Indication Signal) e para a camada VC (Virtual Channel) também através de um sinal AIS. Isto

causa um envio, em direção à origem do sinal, de um sinal de indicação de defeito remoto RDI (Remote Defect Indication) tanto da camada VP quanto da camada VC.

3.2.2. Arquitetura de uma Rede de Transporte ATM

A caracterização da arquitetura de uma rede de transporte ATM requer a identificação de alguns componentes funcionais básicos, definidos nas recomendações da ITU-T G.805 [ITU G805], com uma descrição funcional de uma rede de transporte genérica, e ITU-T I.326 [ITU I326], com uma descrição funcional de uma rede ATM.

Componentes Topológicos

Os componentes topológicos provêem o nível mais abstrato de descrição de uma rede, permitindo desta forma descrever sua topologia lógica. São definidos quatro componentes topológicos para uma rede ATM: a camada de rede, a sub-rede, o enlace e a porta.

A camada de rede é definida como sendo o conjunto completo de portas que podem estar associadas com o propósito de transferir informações. As informações transferidas por um nível da rede são ditas informações características. A informação característica é um sinal com uma taxa de transmissão específica e um formato predeterminado que é transferido entre as sub-redes e apresentado através de funções de adaptação para que seja acessado através de pontos de acesso.

As associações entre pontos de acesso podem ser criadas e removidas por uma aplicação de gerência de rede, alterando desta forma a conectividade da rede. Uma camada de rede é composta por sub-redes e por enlaces entre elas. Por sua vez, uma sub-rede pode ser particionada em sub-redes recursivas e enlaces interconectados, como está representado no exemplo na Figura 13. Uma sub-rede é definida como sendo o conjunto completo de portas que podem ser associadas para transferir uma determinada informação característica.

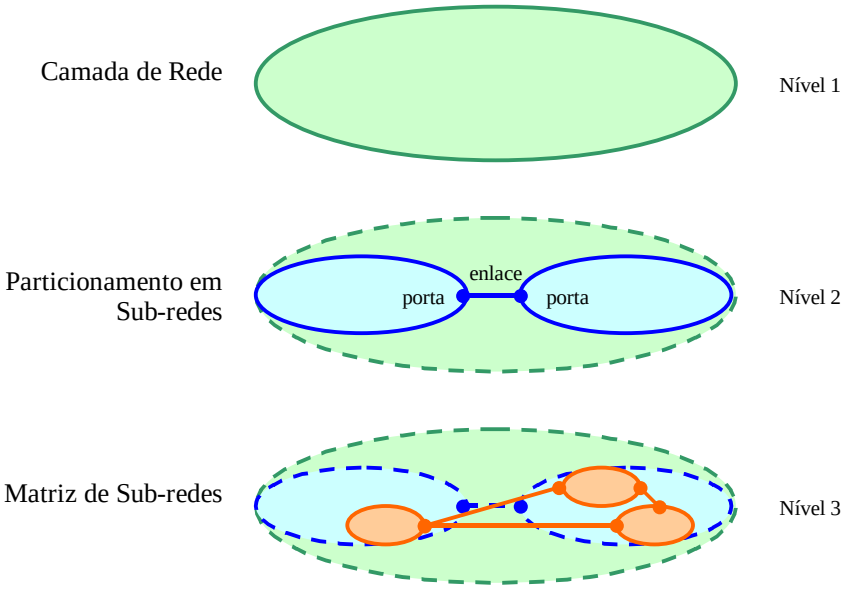

Figura 13: Exemplo de particionamento de uma sub-rede

Um enlace descreve um relacionamento fixo entre uma sub-rede e outra sub-rede ou grupo de acesso (conjunto de pontos de acesso). Cada enlace é definido como um subconjunto de pontos de conexão em uma sub-rede, ao qual está associado um outro subconjunto de pontos de conexão ou pontos de acesso em outra sub-rede ou grupo de acesso, que permite a transferência de informações características. Um enlace representa, portanto, um relacionamento topológico entre um par de sub-redes.

Uma porta corresponde a uma saída de uma origem de terminação de trail (entidade de transporte) ou de conexão de enlace unidirecional, ou a uma entrada do destino de terminação de trail ou de conexão de enlace unidirecional.

Entidades de Transporte

As entidades de transporte provêem a transferência de informações de forma transparente dentro da camada de rede. São definidos dois tipos de entidades de transporte de acordo com o fato da informação a ser transferida estar ou não sendo monitorada para garantir a sua integridade. Estas entidades são os trails e as conexões.

Os trails caracterizam-se pela sua direcionalidade e as conexões caracterizam-se tanto pela sua direcionalidade quanto pelo seu modo. As conexões ainda são divididas em conexões de sub-redes e conexões de enlaces, de acordo com o componente topológico ao qual elas fazem parte.

A direcionalidade de uma conexão ou de um trail indica quando a transmissão é unidirecional ou bidirecional. Na especificação do ATM, todas as conexões são bidirecionais, com diferentes larguras

de banda característica para cada direção. Isto implica que para representar uma conexão unidirecional basta representar uma conexão bidirecional com largura de banda nula na direção não utilizada.

O modo de uma conexão define o tipo de transmissão, podendo este ser ponto a ponto, ponto a multiponto, *multicast*, broadcast ou conferência.

O Trail

A entidade de transporte trail, na camada de rede servidora, é responsável pela integridade da transferência da informação característica de uma ou mais camadas de rede clientes entre as portas da camada servidora.

As funções de terminação de trail monitoram a integridade da transferência ao adicionar informações adaptando a informação característica da camada cliente para a camada servidora. Estas funções de terminação de trail são consideradas parte integrante dos trails.

Um trail contido em uma determinada camada servidora deve suportar um ou mais enlaces nas suas camadas de rede clientes.

A Conexão de Sub-rede

Uma conexão de sub-rede é responsável pela transmissão da informação característica através da sub-rede de forma transparente. Ela é delimitada por portas nas suas extremidades e representa a associação entre portas dentro de uma mesma sub-rede.

As conexões de sub-rede são constituídas geralmente por uma concatenação de conexões de enlace e de conexões de sub-rede de um nível inferior.

A Conexão de Enlace

Uma conexão de enlace é caracterizada por um enlace e seus pontos de terminação, denominados TPs (Termination Points), o qual é suportado por um trail em uma camada de rede servidora. Ela é responsável por transmitir a informação característica de forma transparente através do enlace entre dois pontos conexão ou, no caso de uma conexão de enlace na fronteira de uma camada de rede, entre um ponto de conexão e um ponto de conexão de trail.

Funções de Processamento de Transporte

Há duas funções de processamento de transporte na arquitetura de uma rede composta por camadas: a função de adaptação e a função de terminação de trail.

A função de adaptação é responsável pela adaptação de uma camada servidora às necessidades de uma camada de rede cliente. Por exemplo, é de responsabilidade da função de adaptação o processamento de:

- codificação e extração de células;
- alteração da taxa de transmissão;
- alinhamento, justificação e multiplexação de uma cadeia de células.

A função de terminação de trail é responsável pela geração da informação característica de uma camada de rede e pela garantia de sua integridade.

Estrutura de Camada da Arquitetura de Rede ATM

As redes ATM podem ser decompostas em duas camadas independentes com uma associação cliente/servidor entre elas. Estas camadas são a camada de VP (Virtual Path) e a camada de VC (Virtual Channel).

A camada de VP é a servidora da camada de VC e permite o transporte das células ATM através de um trail VP entre seus pontos de acesso. Uma camada de rede VP, representada na Figura 14, contém as seguintes funções de transporte e entidades de transporte:

- VPT (Virtual Path Trail Termination): gera e termina células de OAM;
- VPNC (Virtual Path Network Connection): transporta informação característica através de uma conexão de rede;
- VPLC (Virtual Path Link Connection): transporta informação característica através de uma conexão entre sub-redes;
- VPSC (Virtual Path Sub-network Connection): transporta informação característica através de uma sub-rede;
- VPCP (Virtual Path Connection Point): liga conexões de enlace e conexões de sub-rede;
- AP (Access Point): liga funções de terminação de trail e funções de adaptação.

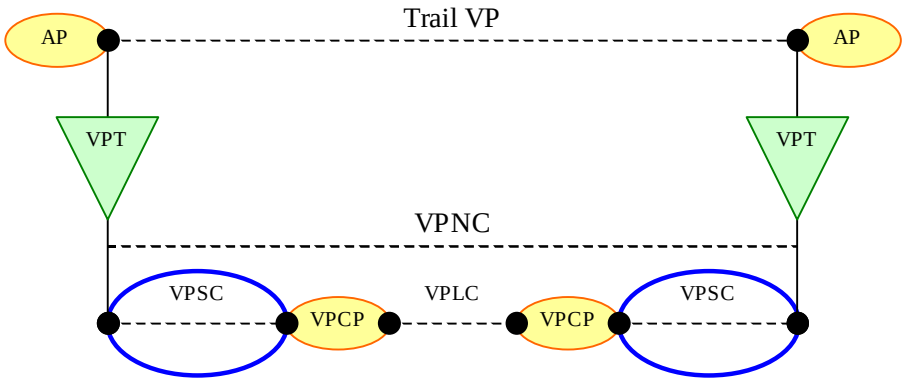

Figura 14: Camada de rede VP

A camada de rede VC permite o transporte de células ATM através de um trail VC entre pontos de acesso. Uma camada de rede VC, representada na Figura 15, contém as seguintes funções de transporte e entidades de transporte:

- VCT (Virtual Channel Trail Termination): gera e termina células de OAM;
- VCNC (Virtual Channel Network Connection): transporta informação característica através de uma conexão de rede;
- VCLC (Virtual Channel Link Connection): transporta informação característica através de uma conexão entre sub-redes;
- VCSC (Virtual Channel Sub-network Connection): transporta informação característica através de uma sub-rede;
- VCCP (Virtual Channel Connection Point): liga conexões de enlace e conexões de sub-rede;
- AP (Access Point): liga funções de terminação de trail e funções de adaptação.

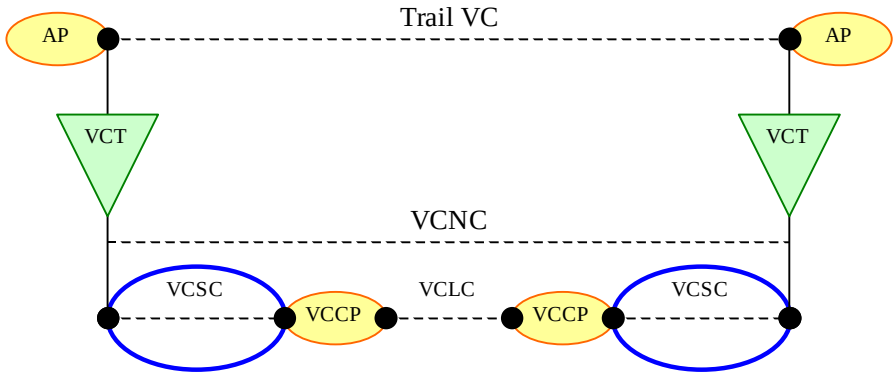

Figura 15: Camada de rede VC

Adaptações da Camada de Rede ATM

O relacionamento completo e genérico das camadas de rede de uma rede ATM está representado na Figura 16.

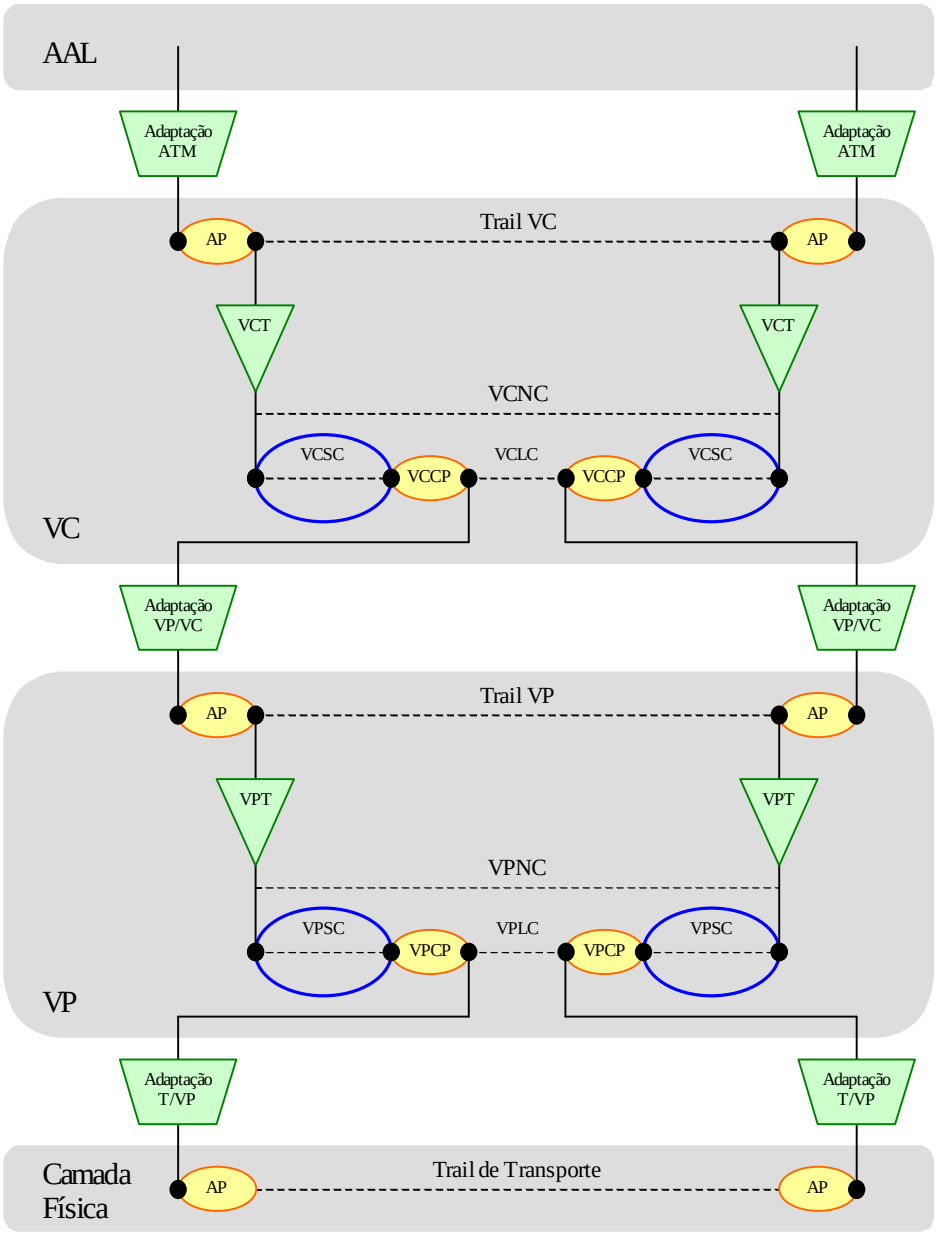

Figura 16: Exemplo de camadas de rede ATM

Uma conexão de enlace VP utiliza-se de um trail da camada física de transporte da rede. Uma conexão de enlace VC utiliza-se de um trail da camada de rede VP. Um trail VP é utilizado para transportar a informação característica de um serviço específico.

O nível de rede da arquitetura da rede de transporte ATM suporta a decomposição da rede em camadas independentes, com um relacionamento cliente/servidor entre camadas adjacentes. Desta forma, cada camada pode ser descrita separadamente e gerenciada independentemente.

3.2.3. Arquitetura de uma Rede de Gerência

O ATM Fórum define na sua arquitetura de gerência ATM cinco interfaces de gerenciamento, interfaces estas denominadas M1 (entre uma gerência de rede privada e um usuário final), M2 (entre uma gerência de rede privada e uma rede privada), M3 (entre uma gerência de rede pública e uma gerência de rede privada), M4 (entre uma gerência de rede pública e uma rede pública) e M5 (entre uma gerência de rede pública e uma gerência de rede pública). A interface de gerência que interessa para este estudo é a M4, a qual é responsável pelo gerenciamento das redes públicas ATM.

A interface M4 foi especificada a partir da necessidade de se integrar dois níveis lógicos de gerenciamento TMN: o nível de rede e o de elemento de rede. O nível de elemento de rede trata o gerenciamento de cada um dos NEs ATM individualmente, enquanto que o nível de rede trata do gerenciamento de NEs agregados em uma ou mais sub-redes.

Com o propósito de operar sobre as entidades gerenciáveis específicas do nível de rede, as funções de gerência de rede atuam no papel de gerente enquanto que as funções de gerência subordinadas atuam como papel de agente, de acordo com o paradigma gerente/agente proposto pela arquitetura funcional do TMN.

3.2.4. Funções de Gerência de Rede

O ATM Fórum apresenta um conjunto de requisitos mandatários considerados mínimos para caracterizar uma aplicação de gerência de redes ATM. Os requisitos funcionais, definidos para a interface M4, requerem informações provenientes do nível de rede e, às vezes, informações provenientes do nível de NE. Esta lista de funcionalidades tem como propósito identificar um conjunto adequado de entidades de gerenciamento capazes de gerenciar uma rede ATM.

Os requisitos funcionais estão separados de acordo com o seu propósito de gerenciamento específico para a camada de rede sob diferentes perspectivas: provedores de infra-estrutura ATM, provedores de serviços e clientes.

Os sistemas de gerência que suprirem os requisitos destes usuários, podem estar inter-relacionados de forma compartilhada ou hierárquica, mas nem todas as informações devem estar disponibilizadas para todos eles.

Gerência de Configuração para a Interface M4

Gerência da Rede

Uma sub-rede pode ser criada automaticamente durante o processo de instalação do sistema de gerência de rede (ou de sub-rede) ou posteriormente através do próprio sistema de gerência de configuração. Para gerenciar a criação de cada sub-rede de uma camada de rede, a interface de gerência deve:

1. suportar pedidos de informações sobre os componentes da sub-rede;
2. suportar a geração automática de notificações da existência ou da mudança de configuração de uma sub-rede;
3. suportar a geração automática de notificações de alterações em parâmetros de configuração de sub-redes;
4. suportar pedidos de informações sobre a configuração atual dos componentes de uma sub-rede;
5. disponibilizar informações suficientes para que sistemas de gerência de níveis superiores possam incluir a sub-rede nos seus sistemas;
6. suportar pedidos de leitura e de atualização da identificação das sub-redes;
7. permitir que uma sub-rede seja criada com ou sem pontos de terminação;
8. permitir que recursos de transporte sejam compartilhados entre várias sub-redes;
9. permitir que uma sub-rede seja configurada com ou sem um conjunto inicial de pontos de terminação;
10. suportar pedidos de criação e remoção de sub-redes;
11. permitir a associação de um ponto de terminação de enlace com mais de uma sub -rede;
12. disponibilizar informações que identifiquem os pontos de terminação de enlace componentes de uma sub-rede.

Gerência de Enlaces de Sub-rede

A interface de gerência da camada de rede suporta os pedidos de configuração de rede para criação, alteração e remoção de enlaces de sub-redes. A partir destes pedidos de configuração desde um sistemas de gerência, a rede passa a ser responsável por alocar os recursos necessários, ativar as conexões requeridas e alterar ou remover recursos alocados.

O modelo da camada de rede permite que os enlaces sejam representados como enlaces unários, enlaces compostos (formados a partir da agregação de outros enlaces) e enlaces particionados (formados pela alocação parcial da capacidade de um outro enlace).

Um enlace ATM conecta duas sub-redes ATM e representa a capacidade de utilização de um trail servidor de nível de transporte inferior. Um enlace é terminado por dois pontos de terminação de enlace.

Para prover as funcionalidades associadas aos enlaces de uma sub-rede de uma camada de rede, a interface de gerência deve:

1. suportar pedidos de criação e de configuração de enlaces ATM entre duas sub-redes VP;
2. relacionar dois pontos de terminação de enlace com o enlace ATM;
3. suportar pedidos de criação e de configuração de enlaces ATM entre duas sub-redes VC quando os trails VP a serem utilizados já existirem;
4. suportar pedidos de criação e de configuração de enlaces ATM entre duas sub-redes VC quando os trails VP a serem utilizados ainda não existirem;
5. suportar pedidos de alteração da configuração de largura de banda de um enlace ATM entre duas sub-redes;
6. suportar pedidos de remoção de enlaces ATM entre sub-redes contidas em uma sub-rede composta (uma sub-rede que pode ser decomposta em sub-redes) e de liberação de recursos utilizados;
7. suportar pedidos de informação sobre a largura de banda alocada e disponível em um enlace ATM;
8. suportar pedidos de configuração do modo de restauração de um enlace ATM;
9. suportar pedidos de informação e de alteração da ponderação associada com um enlace ATM entre duas sub-redes;
10. suportar pedidos de informação e de alteração da identificação de cliente associada com um enlace ATM privado entre duas sub-redes;
11. suportar pedidos de informação e de alteração da identificação de usuário associada com um enlace ATM entre duas sub-redes;
12. suportar pedidos de informação e de alteração da identificação de usuário de uma terminação de enlace ATM;
13. permitir a mudança de estado administrativo de uma terminação de enlace;
14. associar pontos de terminação de um trail servidor de uma camada de transporte inferior com os pontos de terminação de um enlace;
15. suportar pedidos de informação e de alteração do tipo de interface de uma terminação de enlace ATM;
16. disponibilizar informações descrevendo o elemento e a identificação da porta utilizados por cada uma das terminações de enlace ATM;
17. suportar pedidos de informação e de alteração da identificação de localização de retorno associada a uma terminação de enlace ATM;
18. suportar pedidos de informação e de alteração da lista de pontos de terminação de conexão suportados por uma terminação de enlace ATM;
19. suportar o pedido de criação de uma instância que represente a interface para uma rede externa para terminar um enlace que não é visível através da interface de gerência;
20. suportar pedidos de informação e de alteração da identificação de usuário de uma terminação de enlace lógico ATM;
21. suportar pedidos de informação e de alteração da lista de pontos de terminação de conexão suportados por uma terminação de enlace lógico ATM;

22. suportar o pedido de criação de uma nova instância que represente a interface de conexão para uma rede externa para terminar um enlace lógico que não é visível através da interface de gerência.

Gerência de Conexões de Sub-rede

As entidades gerenciadas pelo nível de rede suportam os pedidos de gerenciamento de sub-redes para criar, reservar, remover e modificar conexões de sub-redes, conexões de enlaces e trails. A partir dos pedidos de configuração desde um sistemas de gerência, a rede passa a ser responsável por alocar os recursos necessários, ativar as conexões requeridas e alterar ou remover recursos alocados.

Para gerenciar as conexões de uma sub-rede de uma camada de rede, a interface de gerência deve:

1. suportar pedidos de agendamento de reserva, cancelamento, ativação e desativação de conexões de sub-redes;
2. suportar pedidos de informações de VP, VC, VPI ou VCI associados com pontos de terminação VP e VC configurados a uma sub-rede ATM;
3. suportar os parâmetros requeridos pelo descritor de tráfico de uma sub-rede ATM;
4. suportar a criação, configuração e remoção de conexões de uma sub-rede ATM.

Gerência de Estados de Sub-rede

Para gerenciar os estados de uma sub-rede de uma camada de rede, a interface de gerência deve:

1. suportar a geração automática de notificações de mudança de estado operacional das conexões de uma sub-rede;
2. suportar a geração automática de notificações de mudança de estado operacional de quaisquer dos componentes de uma sub-rede;
3. suportar a geração automática de notificações de chaveamento de proteção entre conexões de uma sub-rede;
4. suportar pedidos de informação sobre a disponibilidade de conexões de sub-rede.

Gerência de Falhas para a Interface M4

A aplicação de gerência de falhas de uma rede ATM visa a monitoração do estado de alarme das entidades constituintes da rede. A interface de gerência de falhas deve:

1. suportar a recuperação e o armazenamento dos alarmes de nível de rede ocorridos em uma sub-rede ATM;
2. suportar a notificação assíncrona de falhas detectadas em uma sub-rede ATM;

3. suportar o pedido de execução e a apresentação de resultados de procedimentos de teste ao longo de uma sub-rede ATM.

3.3. Entidades Gerenciadas

Para a gerência ATM no nível de rede, que é o propósito deste trabalho, as seguintes entidades gerenciadas são necessárias [AF 0020] [AF 0058]:

- AlarmRecord
- AlarmSeverityAssignmentProfile
- AttributeValueChangeRecord
- EventForwardingDiscriminator
- LayerNetworkDomain
- LinkConnection
- Log
- LogicalLinkTP
- ManagedEntityCreationRecord
- ManagedEntityDeletionRecord
- Network
- NetworkAccessProfile
- NetworkCTP
- NetworkTTP
- RoutingProfile
- StateChangeRecord
- Subnetwork
- SubnetworkConnection
- TopologicalLink
- TrafficDescriptorProfile
- Trail
- TrailRequest
- VcLinkEnd
- VpLinkEnd

Estas entidades gerenciadas são os objetos resultantes da modelagem da informação necessária para a gerência de falhas e de configuração de uma rede baseada na tecnologia ATM, especificamente para o gerenciamento do nível de rede.

Nas próximas seções será apresentada cada uma das entidades gerenciadas com a descrição do seu conteúdo (atributos, notificações, operações), do seu relacionamento com outras entidades gerenciadas e do seu comportamento.

3.3.1. AlarmRecord

Esta entidade gerenciada é utilizada para representar as informações armazenadas resultantes da geração de uma notificação de alarme de elementos de rede ATM. Uma instância desta entidade deve ser criada automaticamente para cada notificação de alarme gerada pela rede. Os seus elementos estão listados na Tabela 3.

Atributo	Perm.[4]	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada
loggingTime	RO	Identifica o horário em que a instância foi criada
managedEntity	RO	Identifica o tipo e a instância da entidade gerenciada que gerou a notificação
genericTroubleDescription	RO	Contém o tipo de alarme gerado
specificProblems	RO	Contém detalhes mais específicos do tipo de alarme gerado
severity	RO	Identifica a severidade do alarme
backupStatus	RO	Indica quando foi feito o backup da entidade que gerou o alarme de forma que os serviços prestados não foram interrompidos
backupEntity	RO	Identifica a entidade gerenciada de backup que está provendo os serviços da entidade que gerou o alarme
AdditionalText	RO	Permite incluir informações adicionais referentes à notificação de alarme que foi gerada
proposedRepairActions	RO	Sugere uma ou mais soluções para o problema reportado pelo alarme gerado

Tabela 3: Elementos da entidade gerenciada *AlarmRecord*

A classe *LogRecord* é uma classe virtual que contém as informações básicas necessárias para qualquer tipo de registro de alarme que esteja sendo utilizado ou que venha a ser incluído no sistema. A entidade gerenciada *AlarmRecord* está modelada em UML de acordo com a Figura 17.

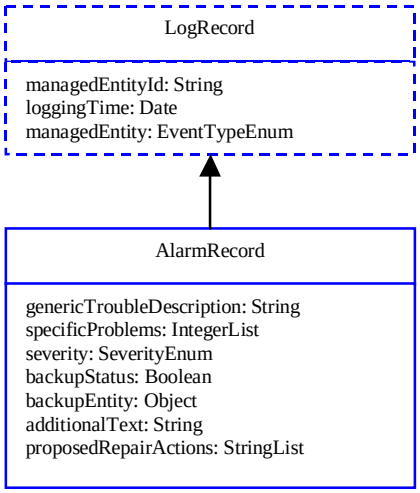

Figura 17: Entidade Gerenciada *AlarmRecord*

[4] Permissão de acesso ao atributo: RO (Read-Only): atributo somente de leitura; RW (Read-Write): atributo de leitura e de gravação.

3.3.2. AlarmSeverityAssignmentProfile

Esta entidade gerenciada é utilizada para representar atribuições de severidades de alarme para os relatórios de alarmes das entidades gerenciadas. Podem haver várias instâncias desta entidade para cada elemento da rede ATM. Os seus elementos estão listados na Tabela 4.

Atributo	**Perm.**	**Descrição**
managedEntityId	RO	Contém uma identificação única da instância criada
alarmSeverityAssignmentList	RW	Identifica um ou mais pares de alarme e sua respectiva severidade
Notificação	**Descrição**	
AttributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
ManagedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
ManagedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	

Tabela 4: Elementos da entidade gerenciada *AlarmSeverityAssignmentProfile*

A entidade gerenciada *AlarmSeverityAssignmentProfile* está modelada em UML de acordo com a Figura 18.

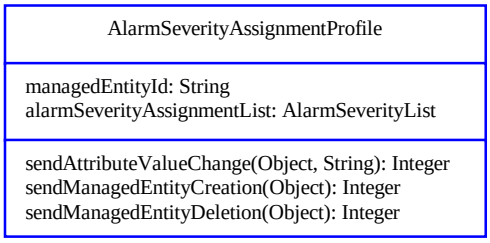

Figura 18: Entidade Gerenciada *AlarmSeverityAssignmentProfile*

3.3.3. AttributeValueChangeRecord

Esta entidade gerenciada é utilizada para representar as informações armazenadas resultantes da geração de uma notificação de alteração do valor de um atributo. Uma instância desta entidade deve

ser criada automaticamente para cada notificação de alteração de valor de atributo gerada pelos elementos da rede. Os seus elementos estão listados na Tabela 5.

Atributo	Perm.	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada
loggingTime	RO	Identifica o horário em que a instância foi criada
managedEntity	RO	Identifica o tipo e a instância da entidade gerenciada que gerou a notificação
attributeType	RO	Identifica o tipo do atributo cujo valor foi alterado
oldAttributeValue	RO	Contém o valor prévio do atributo
newAttributeValue	RO	Contém o valor novo do atributo

Tabela 5: Elementos da entidade gerenciada *AttributeValueChangeRecord*

A classe *LogRecord* é uma classe virtual que contém as informações básicas necessárias para qualquer tipo de registro de alarme que esteja sendo utilizado ou que venha a ser incluído no sistema. A entidade gerenciada *AttributeValueChangeRecord* está modelada em UML de acordo com a Figura 19.

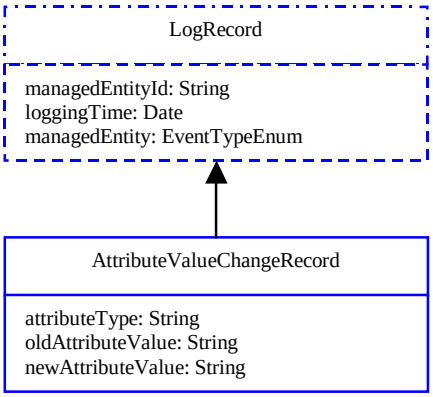

Figura 19: Entidade Gerenciada *AttributeValueChangeRecord*

3.3.4. EventForwardingDiscriminator

Esta entidade gerenciada é utilizada para definir as condições que devem ser satisfeitas pelas notificações geradas pelas entidades gerenciadas para que sejam enviadas aos sistemas de gerência destinatários. Instâncias desta entidade devem ser criadas explicitamente pelos sistemas de gerenciamento. Os seus elementos estão listados na Tabela 6.

Atributo	Perm.	Descrição
ManagedEntityId	RO	Contém uma identificação única da instância criada
discriminatorConstruct	RW	Especifica os condições a serem testadas (filtro) nas notificações para avaliar se devem ou não serem enviadas aos sistemas de gerência
destination	RW	Identifica o endereço do sistema de gerenciamento destinatário a quem as notificações filtradas devem ser enviadas
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
operationalState	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar as suas funcionalidades
Notificação	**Descrição**	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	

Tabela 6: Elementos da entidade gerenciada *EventForwardingDiscriminator*

A entidade gerenciada *EventForwardingDiscriminator* está modelada em UML de acordo com a Figura 20.

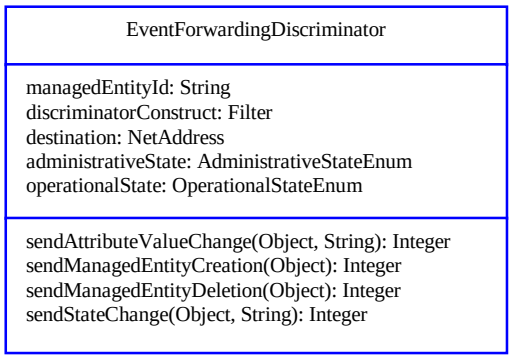

Figura 20: Entidade Gerenciada *EventForwardingDiscriminator*

3.3.5. LayerNetworkDomain

Esta entidade gerenciada é definida para viabilizar o gerenciamento independente das camadas, tanto da camada VC quanto da camada VP.

A entidade gerenciada *LayerNetworkDomain* (veja Figura 21) representa a parte da tecnologia ATM disponibilizada para um sistema de gerenciamento que contém apenas as entidades gerenciadas de uma única camada ATM (VC ou VP). Esta camada ATM é composta por uma única sub-rede que pode, por sua vez, ser decomposta.

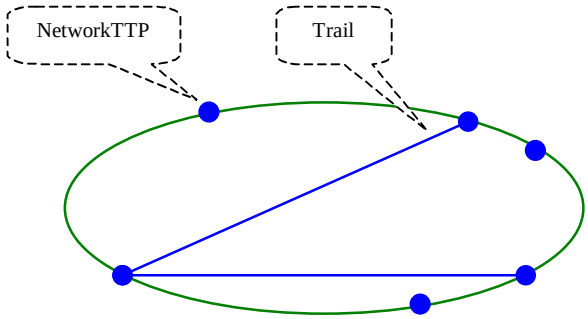

Figura 21: *LayerNetworkDomain* para a camada VC ou VP

Esta entidade gerenciada é criada automaticamente durante a instalação da entidade gerenciada *Network* e a sua criação deve ser comunicada aos sistemas conectados de gerenciamento. Os seus elementos estão listados na Tabela 7.

Atributo	**Perm.**	**Descrição**
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryDelimitingNetworkTTPs	Usada para obter os identificadores das terminação de trails delimitadores	

queryExistingTrails	Usada para obter os identificadores dos trails agrupados
queryComponentSubnetwork	Usada para obter o identificador da sub-rede básica constituinte do domínio da camada de rede
setupTrail	Usada para criar um trail entre dois pontos de terminação de trail não conectados, identificados explicitamente ou através da informação característica requerida
setupTrailRequest	Usada para criar um pedido de alteração de trail
addTPsToMultipointTrail	Permite adicionar uma terminação de trail a um trail multiponto, identificado explicitamente ou através da informação característica requerida
releaseTrail	Usada para remover um trail entre dois pontos de terminação de trail
makeExternalLinkEnd	Usada para criar uma instância da entidade gerenciada *VcLinkEnd* ou *VpLinkEnd*, a qual representa uma interface com uma rede externa, ou seja, a qual termina um enlace que não é visível pelo sistema de gerência
removeExternalLinkEnd	Usada para remover uma instância da entidade gerenciada *VcLinkEnd* ou *VpLinkEnd*
setupTopologicalLink	Permite criar um enlace ponto a ponto entre duas sub-redes VC na camada VC. Este enlace VC pode conectar diretamente duas sub-redes VP ou pode ser suportado por uma conexão VP que atravessa sub-redes VP
releaseTopologicalLink	Permite remover um enlace ponto a ponto entre dois *VcLinkEnd* ou *VpLinkEnd* ou *LogicalLinkTP* conectados
Relacionamento	**Descrição**
NetworkTTP	Várias destas instâncias podem ser utilizadas para delimitar a entidade gerenciada *LayerNetworkDomain*
Trail	Várias destas instâncias podem estar agrupadas na entidade gerenciada *LayerNetworkDomain*
Subnetwork	Várias destas instâncias podem estar particionando a entidade gerenciada *LayerNetworkDomain*
TrailRequest	Várias destas instâncias podem estar modificando os trails contidos na entidade gerenciada *LayerNetworkDomain*
TopologicalLink	Várias destas instâncias podem estar contidas numa entidade gerenciada *LayerNetworkDomain* conectando suas sub-redes

Tabela 7: Elementos da entidade gerenciada *LayerNetworkDomain*

A entidade gerenciada *LayerNetworkDomain* está modelada em UML de acordo com a Figura 22.

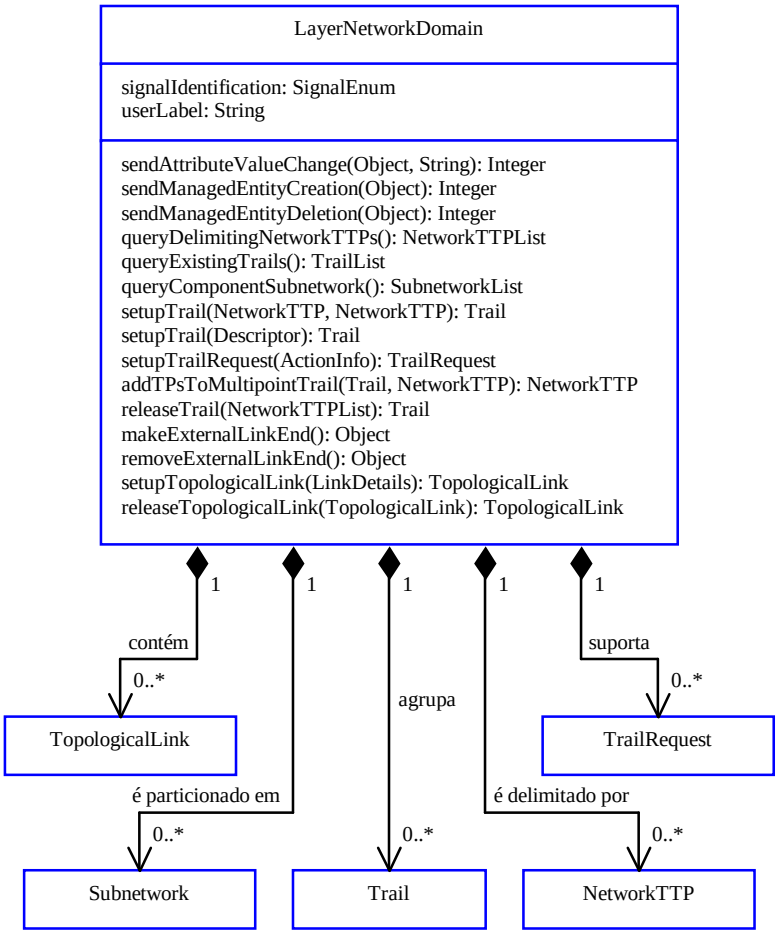

Figura 22: Entidade Gerenciada *LayerNetworkDomain*

3.3.6. LinkConnection

A entidade gerenciada *LinkConnection* (veja Figura 23) representa uma conexão de enlace que é uma entidade de transporte responsável pela transferência de informações entre portas de um enlace ATM.

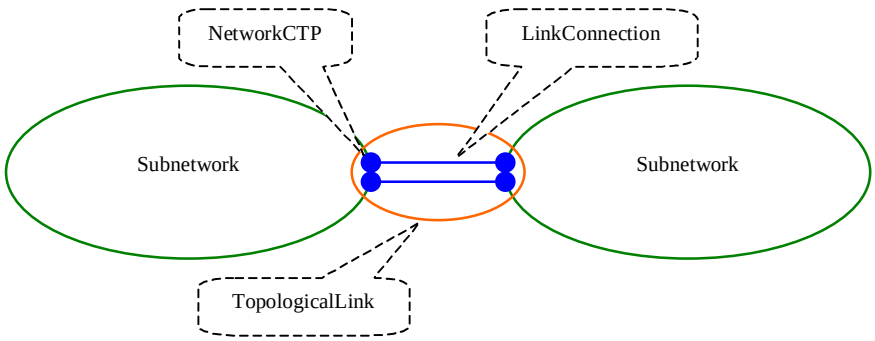

Figura 23: *LinkConnection* para a camada VC ou VP

Esta entidade gerenciada só pode ser criada explicitamente por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 8.

Atributo	Perm.	Descrição
linkConnectionId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
directionality	RO	Define a direção dos dados da rede de transporte ATM: fixo em bidirecional
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
retainedResource	RW	Indica se a entidade gerenciada precisa ser reservada quando compor uma conexão composta (*LinkConnection*, *SubnetworkConnection*) ou quando suportar um trail (*LinkConnection*)
Notificação	**Descrição**	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
Operação	**Descrição**	
queryTopologicalLink	Usada para obter o identificador da entidade gerenciada *TopologicalLink* que inclui esta conexão de enlace	
queryNetworkCTPs	Usada para obter os dois identificadores das terminação de enlace que terminam a conexão	
Relacionamento	**Descrição**	
TopologicalLink	Várias destas instâncias podem estar contidas numa entidade gerenciada *LayerNetworkDomain* conectando suas sub-redes	
NetworkCTP	Duas destas instâncias terminam uma entidade gerenciada	

	LinkConnection, uma para cada sub-rede que está sendo conectada

Tabela 8: Elementos da entidade gerenciada *LinkConnection*

A entidade gerenciada *LinkConnection* está modelada em UML de acordo com a Figura 24.

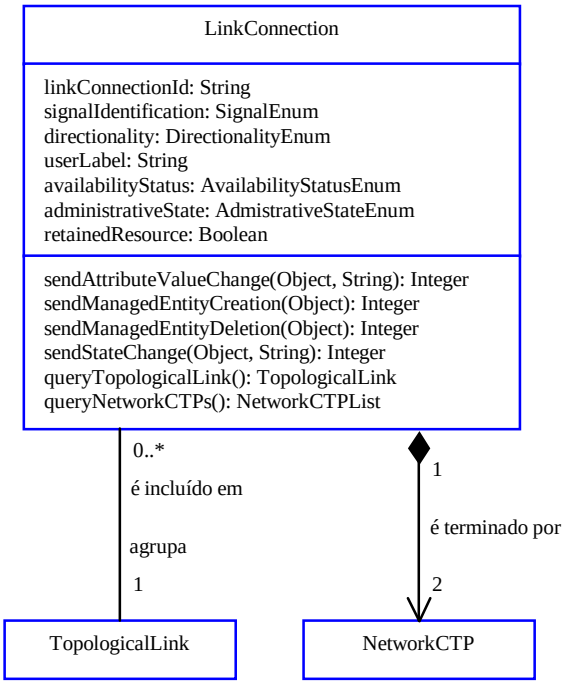

Figura 24: Entidade Gerenciada *LinkConnection*

3.3.7. Log

Esta entidade gerenciada é utilizada para agrupar várias instâncias de *StateChangeRecord*, *AttributeValueChangeRecord*, *ManagedEntityCreationRecord*, *ManagedEntityDeletionRecord* e/ou *AlarmRecord* para gerar o registro de notificações. Instâncias desta entidade devem ser criadas automaticamente após a inicialização dos elementos da rede ATM. Os seus elementos estão listados na Tabela 9.

Atributo	Perm.	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
logRecordTypes	RO	Identifica os tipos de registro que serão agrupados nesta entidade gerenciada: *ManagedEntityDeletionRecord*,

		ManagedEntityCreationRecord, *StateChangeRecord*, *AlarmRecord* e/ou *AttributeValueChangeRecord*
logFullAction	RW	Identifica a ação a ser tomada pela entidade gerenciada quando não houver mais espaço disponível para os registros
operationalState	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar as suas funcionalidades
Notificação	**Descrição**	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	
Relacionamento	**Descrição**	
ManagedEntityCreationRecord	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Log*, desde que permitido pelo atributo *logRecordTypes* de identificação dos tipos de registro	
ManagedEntityDeletionRecord	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Log*, desde que permitido pelo atributo *logRecordTypes* de identificação dos tipos de registro	
AttributeValueChangeRecord	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Log*, desde que permitido pelo atributo *logRecordTypes* de identificação dos tipos de registro	
StateChangeRecord	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Log*, desde que permitido pelo atributo *logRecordTypes* de identificação dos tipos de registro	
AlarmRecord	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Log*, desde que permitido pelo atributo *logRecordTypes* de identificação dos tipos de registro	

Tabela 9: Elementos da entidade gerenciada *Log*

A entidade gerenciada *Log* está modelada em UML de acordo com a Figura 25.

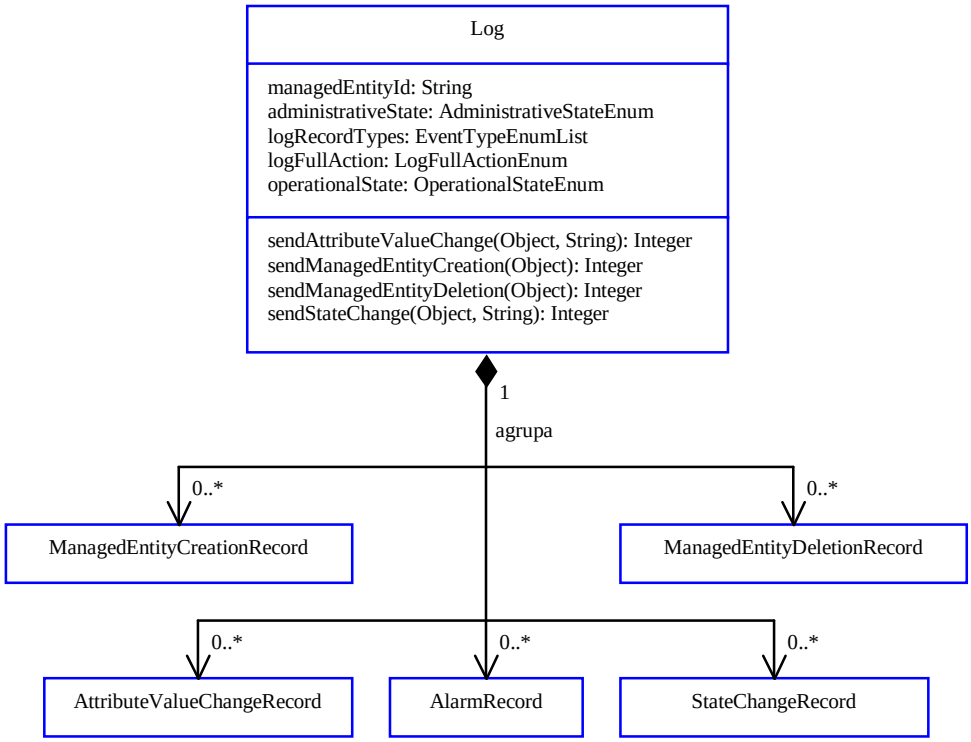

Figura 25: Entidade Gerenciada *Log*

3.3.8. LogicalLinkTP

Esta entidade gerenciada é utilizada para representar a terminação de um *TopologicalLink* de uma mesma camada VC ou VP, armazenando suas informações de configuração. Esta entidade só pode ser criada por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 10.

Atributo	Perm.	Descrição
logicalLinkTPId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
egressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da saída
ingressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da entrada
egressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da saída
ingressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da entrada
rangeIdentification	RW	Identifica os limites de valores de VCI/VPI que podem ser

		utilizados por este enlace lógico
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryTopologicalLink	Usada para obter o identificador da entidade gerenciada *TopologicalLink* que termina esta conexão de enlace	
querySubnetwork	Usada para obter as sub-redes delimitadas por este *LogicalLinkTP*	
queryVcLinkEnd	Usada para obter os *VcLinkEnds* associados com este *LogicalLinkTP* quando da camada VC	
associateVcLinkEnd	Usada para associar um novo *VcLinkEnd* à este *LogicalLinkTP* quando da camada VC	
queryVpLinkEnd	Usada para obter os *VpLinkEnds* associados com este *LogicalLinkTP* quando da camada VP	
associateVpLinkEnd	Usada para associar um novo *VpLinkEnd* à este *LogicalLinkTP* quando da camada VP	
traceLink	Usada para identificar as conexões de sub-rede que terminam neste *LogicalLinkTP*	
Relacionamento	**Descrição**	
TopologicalLink	Cada uma destas instâncias pode ser terminada por uma entidade gerenciada *LogicalLinkTP*	
VcLinkEnd	Uma ou mais destas instâncias suportam a entidade gerenciada *LogicalLinkTP* da camada VC	
VpLinkEnd	Uma ou mais destas instâncias suportam a entidade gerenciada *LogicalLinkTP* da camada VP	
Subnetwork	Cada entidade gerenciada *LogicalLinkTP* está associada a pelo menos uma destas instâncias	
NetworkAccessProfile	Uma destas instâncias pode ser usada por uma entidade gerenciada *LogicalLinkTP*	
NetworkCTP	Duas destas instâncias terminam uma entidade gerenciada *LogicalLinkTP*, uma para cada sub-rede que está sendo conectada	

Tabela 10: Elementos da entidade gerenciada *LogicalLinkTP*

A entidade gerenciada *LogicalLinkTP* está modelada em UML de acordo com a Figura 26.

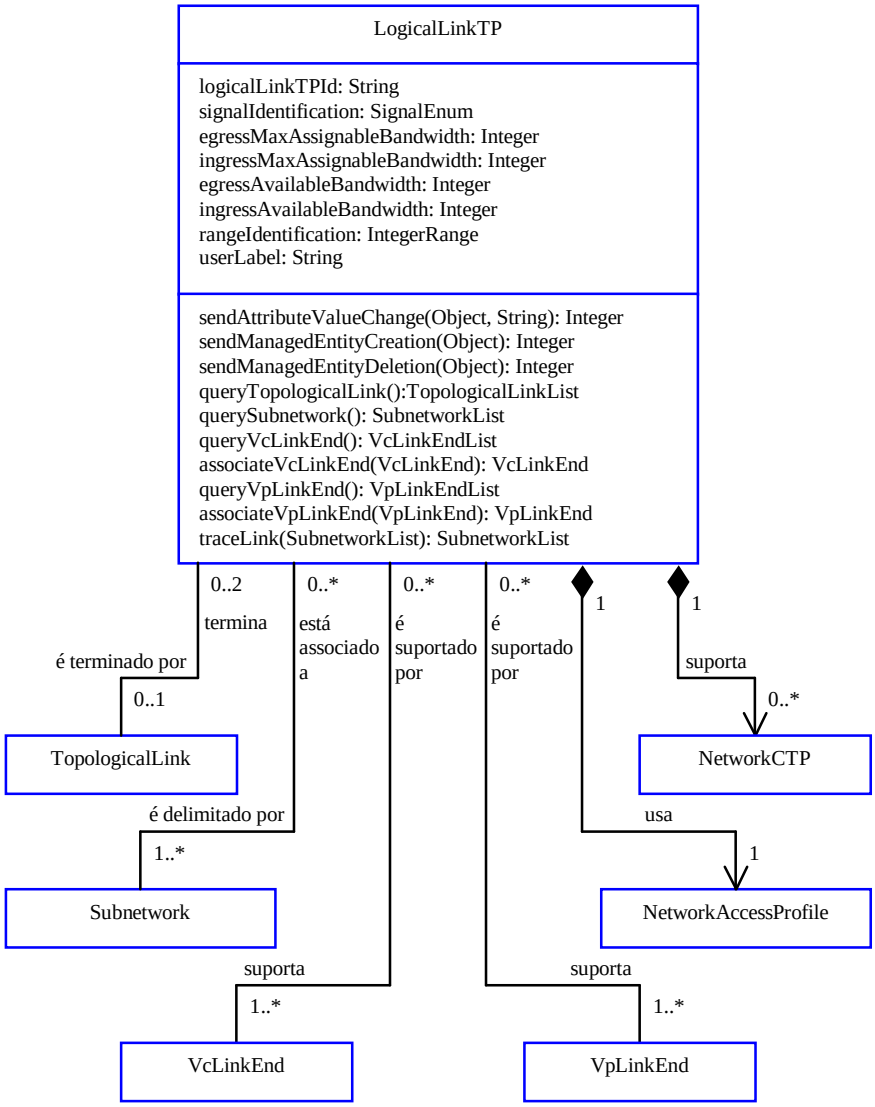

Figura 26: Entidade Gerenciada *LogicalLinkTP*

3.3.9. ManagedEntityCreationRecord

Esta entidade gerenciada é utilizada para representar as informações armazenadas resultantes da geração de uma notificação de criação de uma entidade. Uma instância desta entidade deve ser criada automaticamente para cada notificação de criação gerada pelos elementos da rede. Os seus elementos estão listados na Tabela 11.

Atributo	Perm.	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada

loggingTime	RO	Identifica o horário em que a instância foi criada
managedEntity	RO	Identifica o tipo e a instância da entidade gerenciada que gerou a notificação

Tabela 11: Elementos da entidade gerenciada *ManagedEntityCreationRecord*

A classe *LogRecord* é uma classe virtual que contém as informações básicas necessárias para qualquer tipo de registro de alarme que esteja sendo utilizado ou que venha a ser incluído no sistema. A entidade gerenciada *ManagedEntityCreationRecord* está modelada em UML de acordo com a Figura 27.

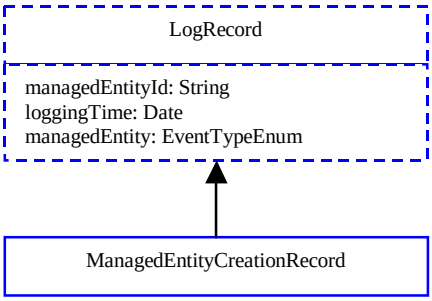

Figura 27: Entidade Gerenciada *ManagedEntityCreationRecord*

3.3.10. ManagedEntityDeletionRecord

Esta entidade gerenciada é utilizada para representar as informações armazenadas resultantes da geração de uma notificação de remoção de uma entidade. Uma instância desta entidade deve ser criada automaticamente para cada notificação de remoção gerada pelos elementos da rede. Os seus elementos estão listados na Tabela 12.

Atributo	Perm.	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada
loggingTime	RO	Identifica o horário em que a instância foi criada
managedEntity	RO	Identifica o tipo e a instância da entidade gerenciada que gerou a notificação

Tabela 12: Elementos da entidade gerenciada *ManagedEntityDeletionRecord*

A classe *LogRecord* é uma classe virtual que contém as informações básicas necessárias para qualquer tipo de registro de alarme que esteja sendo utilizado ou que venha a ser incluído no sistema. A entidade gerenciada *ManagedEntityDeletionRecord* está modelada em UML de acordo com a Figura 28.

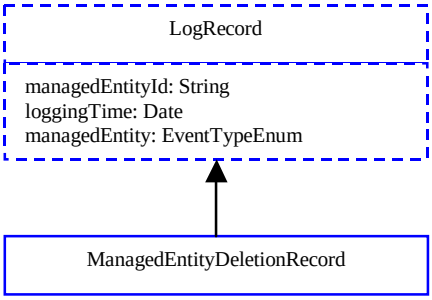

Figura 28: Entidade Gerenciada *ManagedEntityDeletionRecord*

3.3.11. Network

Esta entidade gerenciada agrupa todas as entidades gerenciadas da gerência do nível de rede. Uma instância desta entidade é criada automaticamente após a inicialização do sistema. Os seus elementos estão listados na Tabela 13.

Atributo	**Perm.**	**Descrição**
networkId	RO	Contém uma identificação única da instância criada
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
Relacionamento	**Descrição**	
AlarmSeverityAssignmentProfile	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Network*	
EventForwardingDiscriminator	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Network*	
LayerNetworkDomain	Duas destas instâncias, uma para a camada VC e outra para a camada VP, devem estar contidas na entidade gerenciada *Network*	
Log	Várias destas instâncias podem estar agrupadas pela entidade gerenciada *Network*	

Tabela 13: Elementos da entidade gerenciada *Network*

A entidade gerenciada *Network* está modelada em UML de acordo com a Figura 29.

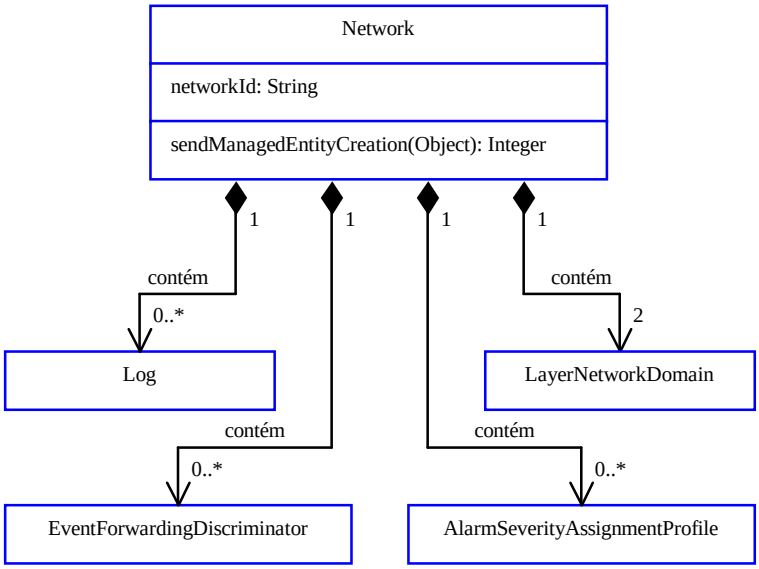

Figura 29: Entidade Gerenciada *Network*

3.3.12. NetworkAccessProfile

Esta entidade gerenciada contém informações que descrevem a capacidade máxima de entrada e de saída da largura de banda para os VCI ou os VPI. Esta entidade é criada por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 14.

Atributo	Perm.	Descrição
networkAccessProfileId	RO	Contém uma identificação única da instância criada
totalEgressBandwidth	RW	Identifica a largura total de banda saída sendo utilizada por um enlace ou uma terminação de enlace
totalIngressBandwidth	RW	Identifica a largura total de banda entrada sendo utilizada por um enlace ou uma terminação de enlace
maxActiveConnectionAllowed	RW	Identifica o número máximo de conexões VC ou VP ativas que um enlace pode suportar
rangeIdentification	RW	Identifica os limites de valores de VCI/VPI que podem ser utilizados por conexões associadas com um enlace
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	

Tabela 14: Elementos da entidade gerenciada *NetworkAccessProfile*

A entidade gerenciada *NetworkAccessProfile* está modelada em UML de acordo com a Figura 30.

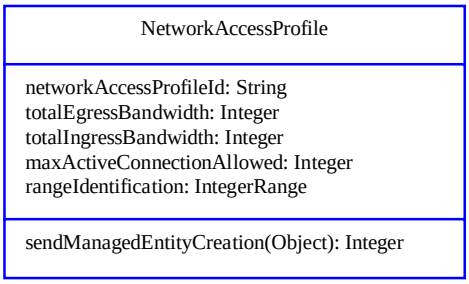

Figura 30: Entidade Gerenciada *NetworkAccessProfile*

3.3.13. NetworkCTP

Esta entidade gerenciada é utilizada para representar a terminação de conexões VC em uma sub-rede ATM (veja Figura 23). Em uma conexão ponto a ponto, uma instância da entidade gerenciada *SubnetworkConnection* ou *LinkConnection* pode ser utilizada para relacionar duas instâncias de *NetworkCTP*.

Esta entidade gerenciada pode ser criada e removida automaticamente como resultado da criação e remoção de uma conexão de enlace ou de sub-rede ou explicitamente por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 15.

Atributo	Perm.	Descrição
cTPId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
vpVcIdValue	RO	Identifica o valor VPI ou VCI associado à conexão VC sendo terminada
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
segmentEndpoint	RW	Indica quando esta entidade gerenciada foi configurada como um ponto de terminação de uma conexão VC
ingressTaggingIndicator	RW	Especifica se está sendo utilizada identificação no lado de recepção de uma conexão VC
egressTaggingIndicator	RW	Especifica se está sendo utilizada identificação no lado de transmissão de uma conexão VC
Notificação	**Descrição**	
alarm	Usada para notificar ao sistema de gerenciamento quando uma falha foi detectada ou corrigida	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta	

	entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada
Operação	**Descrição**
queryNetworkTTP	Usada para obter os *NetworkTTPs* associados com este *NetworkCTP*
associateNetworkTTP	Usada para associar um novo *NetworkTTP* à este *NetworkCTP*
querySubnetworkConnection	Usada para obter as conexões de sub-rede associadas a este *NetworkCTP*
loopbackTrail	Usada para inserir uma célula OAM, verificar seu retorno e reportar os resultados para o sistema de gerenciamento com propósito de teste
Relacionamento	**Descrição**
NetworkTTP	Uma destas instâncias pode existir para uma entidade gerenciada *NetworkCTP*
SubnetworkConnection	Zero ou mais entidades gerenciadas *NetworkCTP* podem existir para cada uma destas instâncias
TrafficDescriptorProfile	Uma destas instâncias pode ser usada para caracterizar uma entidade gerenciada *NetworkCTP*

Tabela 15: Elementos da entidade gerenciada *NetworkCTP*

A entidade gerenciada *NetworkCTP* está modelada em UML de acordo com a Figura 31.

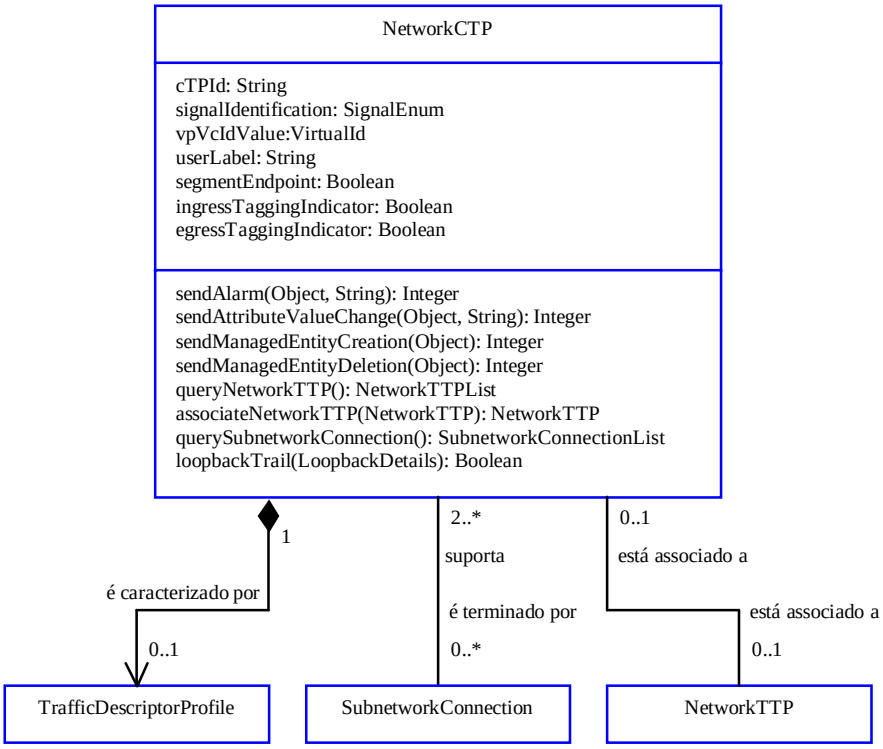

Figura 31: Entidade Gerenciada *NetworkCTP*

3.3.14. NetworkTTP

Esta entidade gerenciada é utilizada para representar onde os trails VC ou VP são terminados/originados em uma sub-rede ATM (veja Figura 21). Sistemas de gerenciamento podem configurar/remover terminações de trail VC ou VP criando/removendo instâncias desta entidade gerenciada. Os seus elementos estão listados na Tabela 16.

Atributo	**Perm.**	**Descrição**
tTPId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta	

	entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo
Operação	**Descrição**
queryNetworkCTP	Usada para obter os *NetworkCTPs* associados com este *NetworkTTP*
queryTrail	Usada para obter os trails associados a este *NetworkTTP*
loopbackTrail	Usada para inserir uma célula OAM, verificar seu retorno e reportar os resultados para o sistema de gerenciamento com propósito de teste
Relacionamento	**Descrição**
NetworkCTP	Uma destas instâncias pode existir para cada entidade gerenciada *NetworkTTP*
Trail	Uma destas entidades gerenciadas é terminada por duas entidades gerenciadas *NetworkTTP*

Tabela 16: Elementos da entidade gerenciada *NetworkTTP*

A entidade gerenciada *NetworkTTP* está modelada em UML de acordo com a Figura 32.

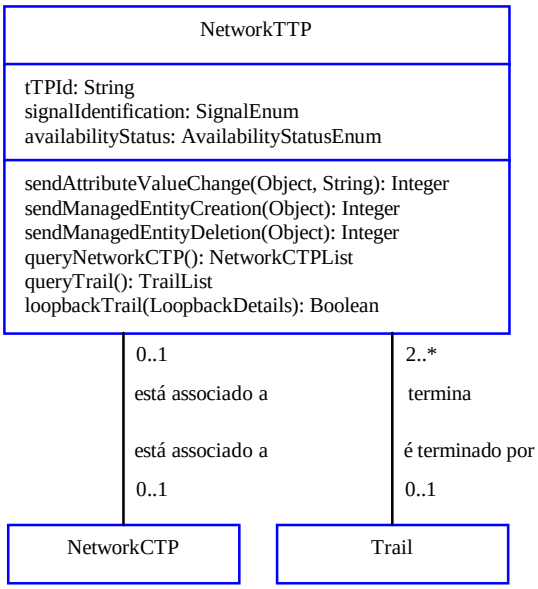

Figura 32: Entidade Gerenciada *NetworkTTP*

3.3.15. RoutingProfile

Esta entidade gerenciada fornece um conjunto de diretivas que podem ser aplicadas na criação de novas conexões ou trails. Esta entidade pode ser criada automaticamente baseado na descrição de

uma rota ou por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 17.

Atributo	Perm.	Descrição
routingProfileId	RO	Define um nome único para esta entidade gerenciada
connectionTypeSupport	RW	Representa o tipo de conexão suportada
routeDescriptionList	RW	Contém uma lista de objetos (enlaces, sub-redes, conexões) e o seu uso na criação de uma rota
maxHops	RW	Contém o número máximo de intervalos entre nós que uma nova conexão pode atravessar
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
Operação	**Descrição**	
setupRoutingProfile	Usada para configurar uma entidade *RoutingProfile*	
Relacionamento	**Descrição**	
Subnetwork	Cada entidade gerenciada *RoutingProfile* está associada a uma destas instâncias	

Tabela 17: Elementos da entidade gerenciada *RoutingProfile*

A entidade gerenciada *RoutingProfile* está modelada em UML de acordo com a Figura 33.

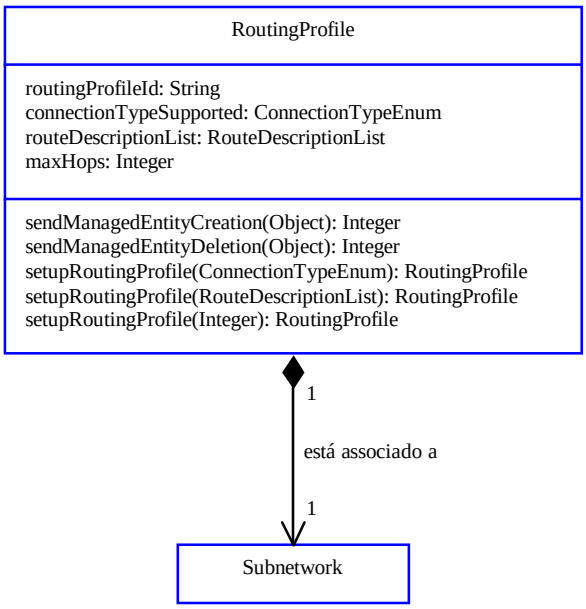

Figura 33: Entidade Gerenciada *RoutingProfile*

3.3.16. StateChangeRecord

Esta entidade gerenciada é utilizada para representar as informações armazenadas resultantes da geração de uma notificação de alteração do estado de uma entidade. Uma instância desta entidade deve ser criada automaticamente para cada notificação de alteração de estado gerada pelos elementos da rede. Os seus elementos estão listados na Tabela 18.

Atributo	Perm.	Descrição
managedEntityId	RO	Contém uma identificação única da instância criada
loggingTime	RO	Identifica o horário em que a instância foi criada
managedEntity	RO	Identifica o tipo e a instância da entidade gerenciada que gerou a notificação
stateAttributeType	RO	Identifica o tipo do atributo de estado cujo valor foi alterado
oldStateAttributeValue	RO	Contém o valor prévio de estado do atributo
newStateAttributeValue	RO	Contém o valor novo de estado do atributo

Tabela 18: Elementos da entidade gerenciada *StateChangeRecord*

A classe *LogRecord* é uma classe virtual que contém as informações básicas necessárias para qualquer tipo de registro de alarme que esteja sendo utilizado ou que venha a ser incluído no sistema. A entidade gerenciada *StateChangeRecord* está modelada em UML de acordo com a Figura 34.

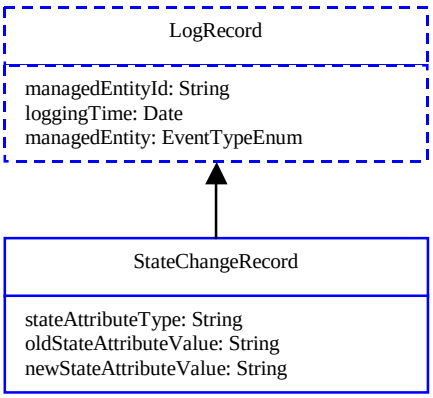

Figura 34: Entidade Gerenciada *StateChangeRecord*

3.3.17. Subnetwork

Esta entidade gerenciada é um componente topológico usado para transportar uma determinada informação característica através das células ATM. As sub-redes são delimitadas por entidades

gerenciadas do tipo VcLinkEnd (no caso de uma sub-rede VC), VpLinkEnd (no caso de uma sub-rede VP) e LogicalLinkTP, como está representado na Figura 35.

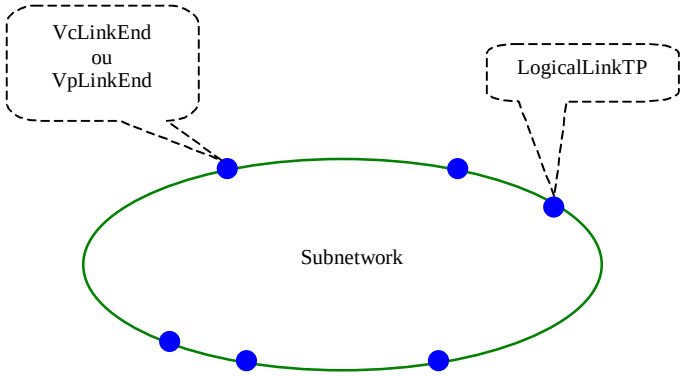

Figura 35: *Subetwork* para a camada VC ou VP

Os elementos constituintes desta entidade gerenciada estão listados na Tabela 19.

Atributo	**Perm.**	**Descrição**
SubnetworkId	RO	Define um nome único para esta entidade gerenciada
SignalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
supportedByObjectList	RW	Aponta para as entidades gerenciadas que suportam a sub-rede
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
querySubnetworkConnection	Usada para obter os *SubnetworkConnections* contidos nesta *Subnetwork*	
querySubnetwork	Usada para obter as *Subnetworks* contidas nesta *Subnetwork*	
queryTopologicalLink	Usada para obter os *TopologicalLinks* contidos nesta *Subnetwork*	

queryLinkEndAndTP	Usada para obter os *VcLinkEnds* ou *VpLinkEnds* e os *LogicalLinkTPs* que delimitam esta *Subnetwork*
setupSubnetworkConnection	Usada para criar uma nova conexão ponto a ponto entre dois *NetworkCTPs* desta *Subnetwork*
modifySubnetworkConnection	Usada para modificar uma conexão entre dois *NetworkCTPs* desta *Subnetwork*
addTpToSubnetworkConnection	Usada para adicionar um ponto de terminação a uma conexão multiponto existente nesta *Subnetwork*
releaseSubnetworkConnection	Usada para remover uma conexão entre dois *NetworkCTPs* desta *Subnetwork*
Relacionamento	**Descrição**
SubnetworkConnection	A entidade gerenciada *Subnetwork* contém zero ou mais destas entidades gerenciadas
Subnetwork	A entidade gerenciada *Subnetwork* pode ser particionada em uma ou mais destas entidades gerenciadas
TopologicalLink	Uma sub-rede composta contém estas entidades gerenciadas entre as entidades gerenciadas *Subnetwork* componentes
LogicalLinkTP	Uma entidade gerenciada *Subnetwork* é delimitada por zero ou mais destas destas entidades gerenciadas
VcLinkEnd	Uma entidade gerenciada *Subnetwork* da camada VC é delimitada por zero ou mais destas destas entidades gerenciadas
VpLinkEnd	Uma entidade gerenciada *Subnetwork* da camada VP é delimitada por zero ou mais destas destas entidades gerenciadas

Tabela 19: Elementos da entidade gerenciada *Subnetwork*

A entidade gerenciada *Subnetwork* está modelada em UML de acordo com a Figura 36.

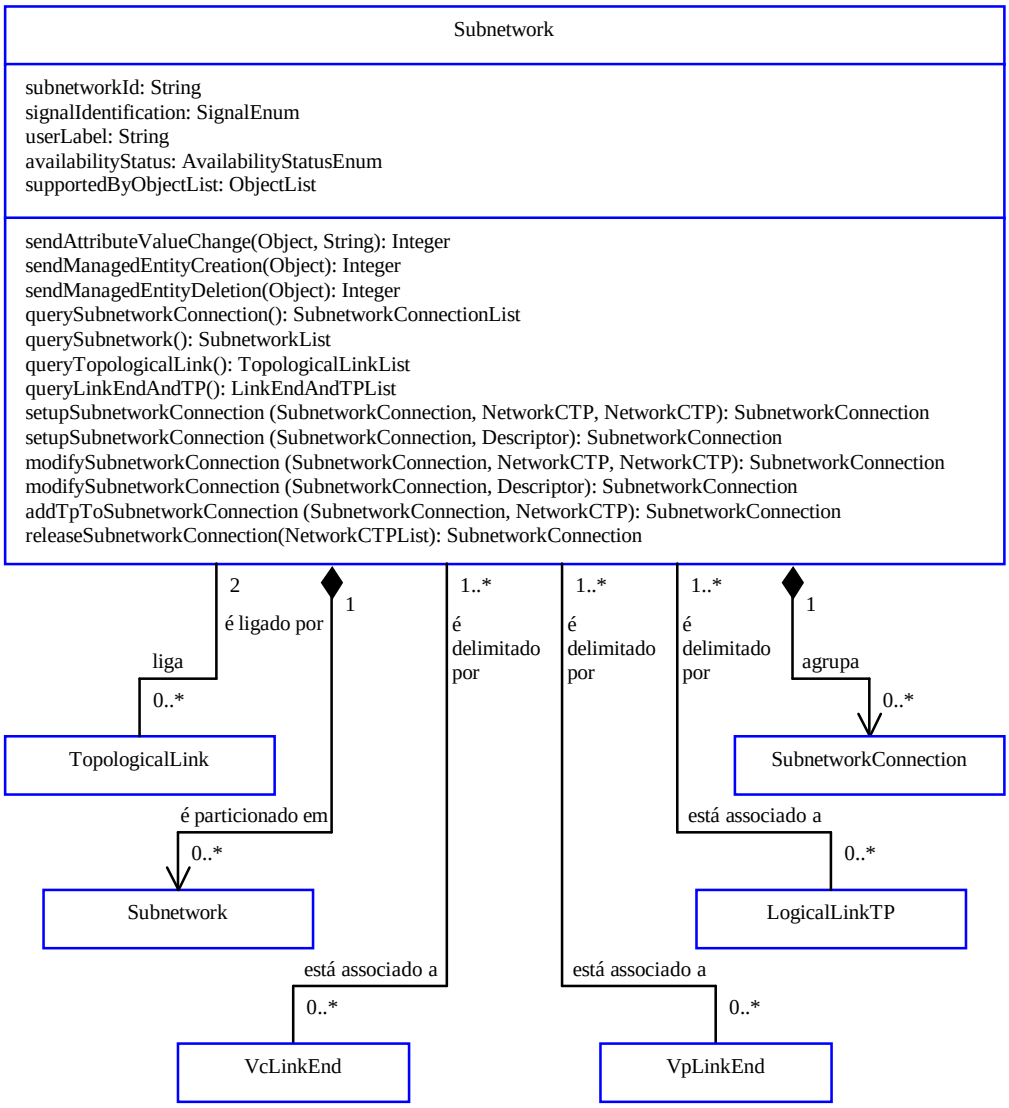

Figura 36: Entidade Gerenciada *Subnetwork*

3.3.18. SubnetworkConnection

Esta entidade gerenciada representa a entidade de transporte responsável pela transferência da informação característica ao longo de uma sub-rede, a qual é formada pela associação de portas existentes nos seus limites, conforme representado na Figura 37.

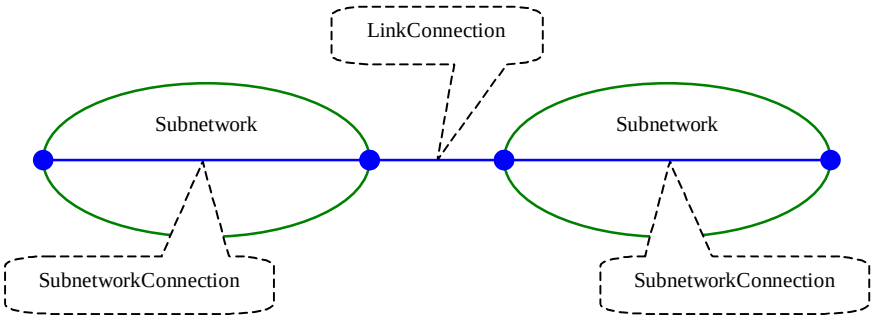

Figura 37: *SubnetworkConnection* para a camada VC ou VP

Esta entidade é criada explicitamente por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 20.

Atributo	Perm.	Descrição
subnetworkConnectionId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
directionality	RO	Define a direção dos dados da rede de transporte ATM: fixo em bidirecional
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
restorableIndicator	RW	Configura a conexão como sendo recuperável ou não
retainedResource	RW	Indica se a entidade gerenciada precisa ser reservada quando compor uma conexão composta (*LinkConnection*, *SubnetworkConnection*) ou quando suportar um trail (*LinkConnection*)
provisionType	RW	Indica quando a rota composta pela associação de *SubnetworkConnections* foi especificada manualmente ou automaticamente por uma sistema de gerenciamento
Notificação	**Descrição**	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	

managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo
Operação	**Descrição**
queryNetworkCTP	Usada para obter os *NetworkCTPs* que terminam esta *SubnetworkConnection*
querySubnetworkConnection	Usada para obter os *SubnetworkConnections* contidos em uma *SubnetworkConnection* composta
traceConnection	Determina o caminho de uma *SubnetworkConnection* retornando os VPI/VCI do nível mais baixo possível suportado pelo sistema de gerenciamento
Relacionamento	**Descrição**
NetworkCTP	A entidade gerenciada *SubnetworkConnection* contém pelo menos duas destas entidades gerenciadas
LinkConnection	Uma entidade gerenciada *SubnetworkConnection* composta é formada por várias destas entidades gerenciadas (no mínimo uma)
SubnetworkConnection	Uma entidade gerenciada *SubnetworkConnection* composta é formada por várias destas entidades gerenciadas (no mínimo duas)
RoutingProfile	Uma entidade gerenciada *SubnetworkConnection* pode ser definida por esta entidade gerenciada

Tabela 20: Elementos da entidade gerenciada *SubnetworkConnection*

A entidade gerenciada *SubnetworkConnection* está modelada em UML de acordo com a Figura 38.

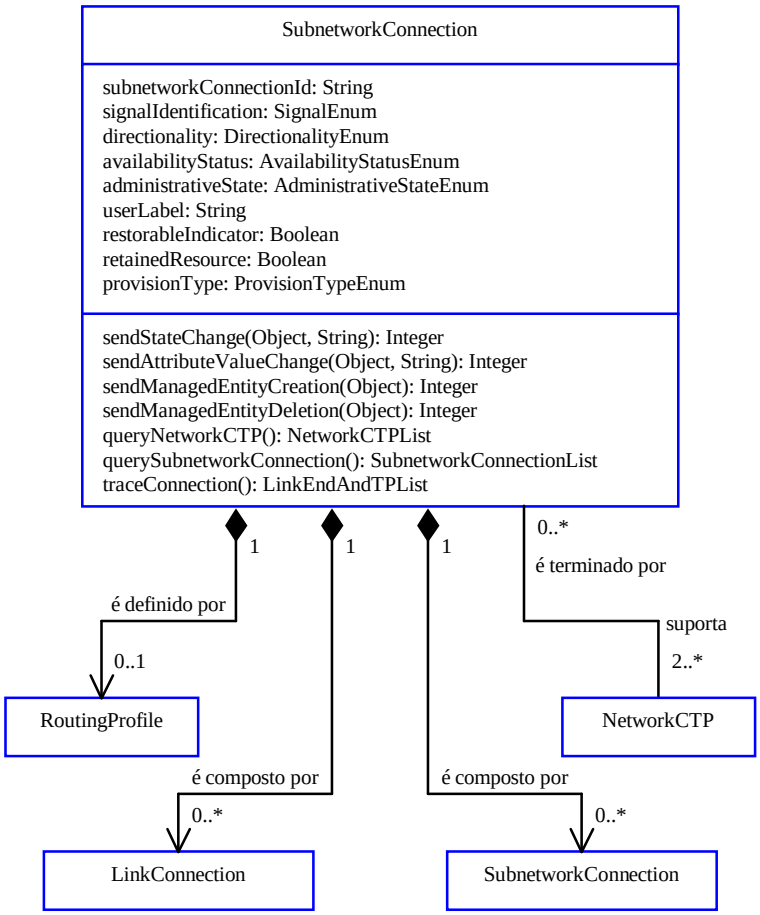

Figura 38: Entidade Gerenciada *SubnetworkConnection*

3.3.19. TopologicalLink

Esta entidade gerenciada é o enlace entre duas sub-redes, representado na Figura 23. Um enlace unário, terminado por *VcLinkEnds* ou *VpLinkEnds*, é um enlace topológico que corresponde diretamente a um trail servidor. Um enlace terminado por *LogicalLinkTPs* representa tanto um enlace composto (um grupo de enlaces unários) quanto um enlace particionado (uma parte de um enlace unário).

Os elementos constituintes desta entidade gerenciada estão listados na Tabela 21.

Atributo	Perm.	Descrição
topologicalLinkId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
directionality	RO	Define a direção dos dados da rede de transporte ATM: fixo em bidirecional

operationalState	RO	Identifica o estado operacional da terminação de trail representada por esta entidade gerenciada
provisionedBandwidth	RW	Identifica a quantidade máxima de largura de banda configurada para o enlace
availabelBandwidth	RO	Identifica a quantidade de largura de banda ainda disponível no enlace
restorationMode	RW	Configura o modo de restauração do enlace
customerIdentification	RW	Identifica o cliente que pode usar o enlace, tornando-o privativo
weight	RW	Define a prioridade para o uso do enlace
Notificação	**Descrição**	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryLinkConnection	Usada para obter os *LinkConnections* que terminam este *TopologicalLink*	
queryLinkEndOrTP	Usada para obter os *VcLinkEnds* ou *VpLinkEnds* ou os *LogicalLinkTPs* que delimitam este *TopologicalLink*	
querySubnetwork	Usada para obter as *Subnetworks* delimitados por este *TopologicalLink*	
setupLinkConnection	Usada para criar uma nova conexão entre dois *NetworkCTPs* de duas *Subnetworks*	
modifyLinkConnection	Usada para modificar uma conexão entre dois *NetworkCTPs* de duas *Subnetworks*	
releaseLinkConnection	Usada para remover uma conexão entre dois *NetworkCTPs* de duas *Subnetworks*	
traceLink	Usada para identificar as conexões de sub-rede que definidas por este *TopologicalLink*	
Relacionamento	**Descrição**	
LinkConnection	Uma entidade gerenciada *TopologicalLink* é um grupo destas entidades gerenciadas que compartilham as mesmas terminações	

LogicalLinkTP	Uma entidade gerenciada *TopologicalLink* é terminada por duas destas entidades gerenciadas, uma em cada sub-rede que estiver ligando
VcLinkEnd	Uma entidade gerenciada *TopologicalLink* da camada VC é terminada por duas destas entidades gerenciadas, sendo uma em cada sub-rede que faça parte do enlace
VpLinkEnd	Uma entidade gerenciada *TopologicalLink* da camada VP é terminada por duas destas entidades gerenciadas, sendo uma em cada sub-rede que faça parte do enlace
Subnetwork	Uma entidade gerenciada *TopologicalLink* relaciona-se com duas destas entidades gerenciadas as quais fazem parte do enlace
NetworkAccessProfile	Uma entidade gerenciada *TopologicalLink* é definida por esta entidade gerenciada

Tabela 21: Elementos da entidade gerenciada *TopologicalLink*

A entidade gerenciada *TopologicalLink* está modelada em UML de acordo com a Figura 39.

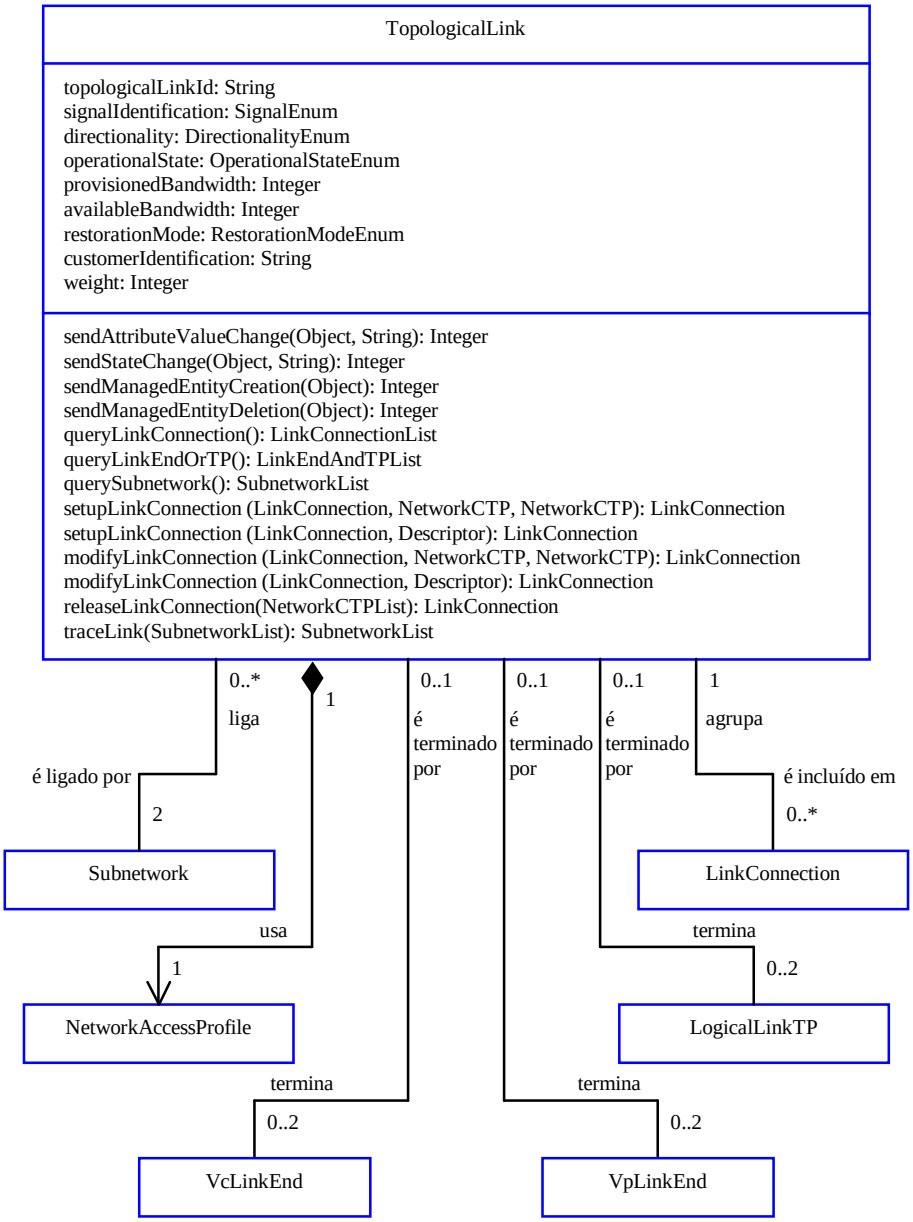

Figura 39: Entidade Gerenciada *TopologicalLink*

3.3.20. TrafficDescriptorProfile

Esta entidade gerenciada fornece um conjunto de diretivas que podem ser aplicadas na criação de novas conexões ou trails. Esta entidade pode ser criada automaticamente baseado na descrição de uma rota ou por uma função de gerenciamento de rede. Os seus elementos estão listados na Tabela 22.

Atributo	Perm.	Descrição
trafficDescriptorProfileId	RO	Define um nome único para esta entidade gerenciada

ingressPeakCellRate	RW	Contém a taxa máxima de células ATM de entrada
egressPeakCellRate	RW	Contém a taxa máxima de células ATM de saída
ingressSustainableCellRate	RW	Contém a taxa máxima de células ATM de entrada que o sistema pode suportar
egressSustainableCellRate	RW	Contém a taxa máxima de células ATM de saída que o sistema pode suportar
ingressMaximumBurstSize	RW	Contém a taxa máxima de rajada de células ATM de entrada que o sistema pode suportar
egressMaximumBurstSize	RW	Contém a taxa máxima de rajada de células ATM de saída que o sistema pode suportar
Notificação	**Descrição**	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	

Tabela 22: Elementos da entidade gerenciada *TrafficDescriptorProfile*

A entidade gerenciada *TrafficDescriptorProfile* está modelada em UML de acordo com a Figura 40.

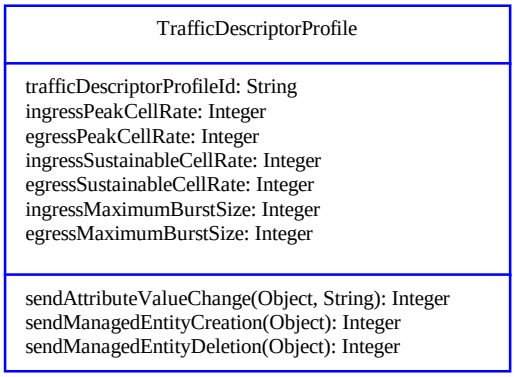

Figura 40: Entidade Gerenciada *TrafficDescriptorProfile*

3.3.21. Trail

Esta entidade gerenciada representa um trail VC/VP, terminado por *NetworkTTPs* (veja Figura 21). Esta entidade é criada pelo sistema de gerenciamento. Os seus elementos estão listados na Tabela 23.

Atributo	Perm.	Descrição
trailId	RO	Define um nome único para esta entidade gerenciada
signalIdentification	RO	Define a camada que esta entidade gerenciada está representando: VC ou VP
directionality	RO	Define a direção dos dados da rede de transporte ATM: fixo em bidirecional
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
restorableIndicator	RW	Configura a conexão como sendo recuperável ou não
retainedResource	RW	Indica se a entidade gerenciada precisa ser reservada quando compor uma conexão composta (*LinkConnection*, *SubnetworkConnection*) ou quando suportar um trail (*LinkConnection*)
Notificação	**Descrição**	
stateChange	Usada para reportar alterações no estado desta entidade, informando qual atributo de estado foi alterado, e o seu valor antigo e o seu valor novo	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryNetworkTTP	Usada para obter os *NetworkTTPs* que terminam este *Trail*	
traceConnection	Determina o caminho de uma *Trail* retornando os VPI/VCI do nível mais baixo possível suportado pelo sistema de gerenciamento	
Relacionamento	**Descrição**	
NetworkTTP	A entidade gerenciada *Trail* é terminada por pelo menos duas destas entidades gerenciadas	

Tabela 23: Elementos da entidade gerenciada *Trail*

A entidade gerenciada *Trail* está modelada em UML de acordo com a Figura 41.

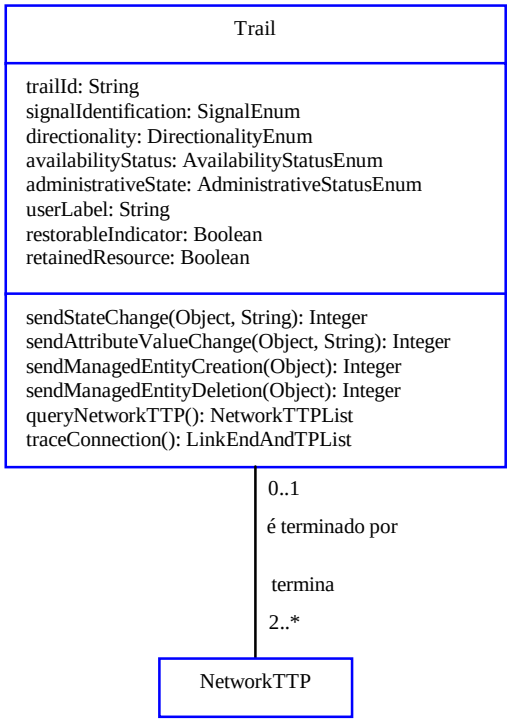

Figura 41: Entidade Gerenciada *Trail*

3.3.22. TrailRequest

Esta entidade gerenciada representa um pedido de alteração a ser executado pela camada de rede sobre um de seus trails. Os seus elementos estão listados na Tabela 24.

Atributo	Perm.	Descrição
trailRequestId	RO	Define um nome único para esta entidade gerenciada
requestStatus	RO	Representa o estado desta entidade gerenciada
requestType	RO	Descreve o tipo do pedido: configurar, modificar, remover, adicionar pontos de terminação, remover pontos de terminação
requestCommittedTime	RO	Especifica quando o NML deve executar a ação desta entidade gerenciada
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	

Relacionamento	Descrição
Trail	Cada entidade gerenciada *TrailRequest* está associada a uma destas instâncias

Tabela 24: Elementos da entidade gerenciada *TrailRequest*

A entidade gerenciada *TrailRequest* está modelada em UML de acordo com a Figura 42.

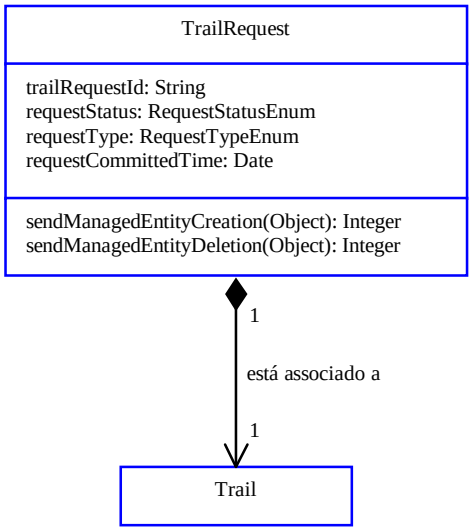

Figura 42: Entidade Gerenciada *TrailRequest*

3.3.23. VcLinkEnd

Esta entidade gerenciada é utilizada para representar a terminação de um *TopologicalLink* na camada VC e é associado a um *NetworkTTP* da camada servidora VP (veja Figura 35). O *VcLinkEnd* sempre é criado por um sistema de gerenciamento. Os seus elementos estão listados na Tabela 25.

Atributo	Perm.	Descrição
linkEndId	RO	Define um nome único para esta entidade gerenciada
administrativeState	RW	Usado para ativar ou desativar as funções da entidade gerenciada
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
egressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da saída
ingressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da entrada

egressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da saída
ingressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da entrada
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
linkTPType	RW	Descreve o tipo de interface suportado pela conexão: UNI, inter-NNI, intra-NNI, unconfigured
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryTopologicalLink	Usada para obter o identificador da entidade gerenciada *TopologicalLink* que é terminado por este *VcLinkEnd*	
querySubnetwork	Usada para obter as sub-redes delimitadas por este *VcLinkEnd*	
queryNetworkTTP	Usada para obter os *NetworkTTPs* associados com este *VcLinkEnd*	
associateNetworkTTP	Usada para associar um novo *NetworkTTP* à este *VcLinkEnd*	
traceLink	Usada para identificar as conexões de sub-rede que terminam neste *VcLinkEnd*	
Relacionamento	**Descrição**	
TopologicalLink	Cada uma destas instâncias pode ser terminada por uma entidade gerenciada *VcLinkEnd*	
LogicalLinkTP	Cada uma destas instâncias pode ser suportada por uma ou mais entidades gerenciadas *VcLinkEnd*	
Subnetwork	Cada entidade gerenciada *VcLinkEnd* está associada a pelo menos uma destas instâncias	
NetworkAccessProfile	Uma destas instâncias pode ser usada por uma entidade gerenciada *VcLinkEnd*	
NetworkTTP	Cada entidade gerenciada *VcLinkEnd* pode estar associada a uma instância de terminação de trail da camada servidora	
NetworkCTP	Cada entidade gerenciada *VcLinkEnd* suporta estas instâncias presentes na mesma camada VC	

Tabela 25: Elementos da entidade gerenciada *VcLinkEnd*

A entidade gerenciada *VcLinkEnd* está modelada em UML de acordo com a Figura 43.

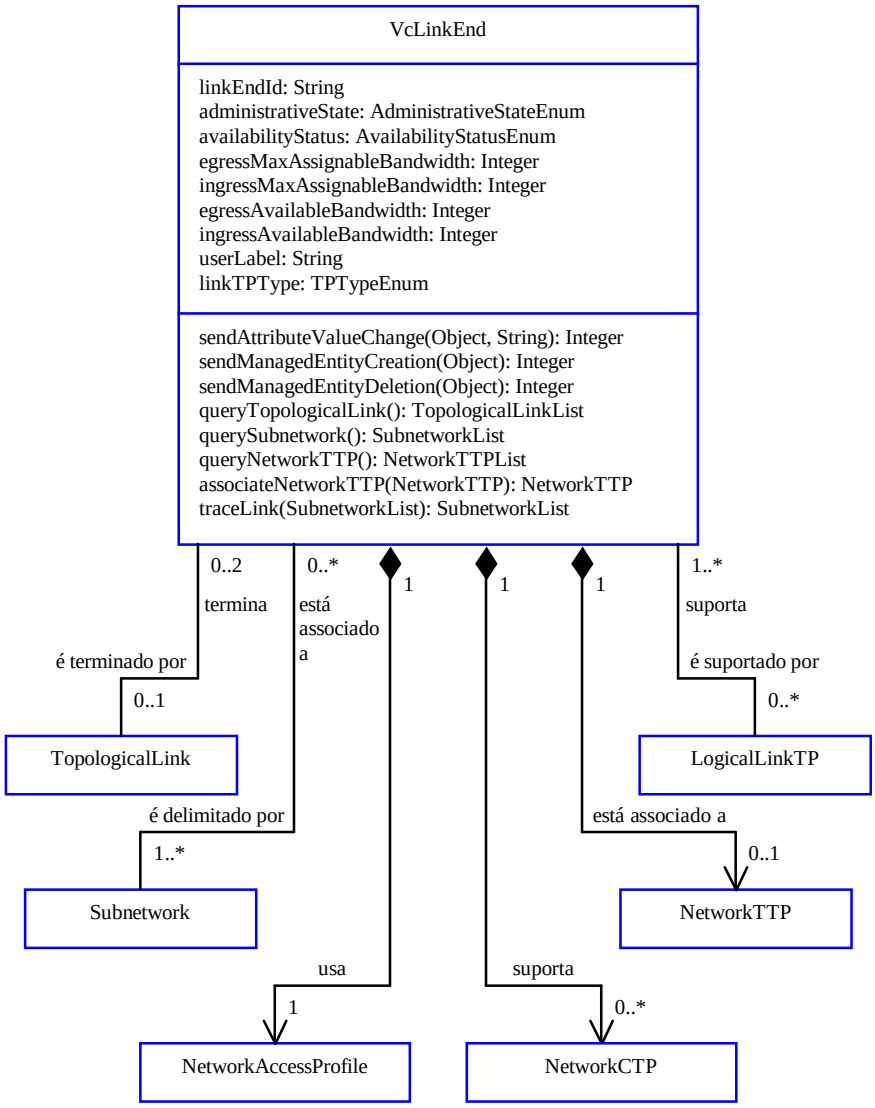

Figura 43: Entidade Gerenciada *VcLinkEnd*

3.3.24. VpLinkEnd

Esta entidade gerenciada é utilizada para representar a terminação de um *TopologicalLink* na camada VP e é associado a um ponto de terminação de trail de uma camada servidora (veja Figura 35). Além disto, pode ser usada para representar a informação de terminação de trail servidora. Os seus elementos estão listados na Tabela 26.

Atributo	Perm.	Descrição
linkEndId	RO	Define um nome único para esta entidade gerenciada
administrativeState	RW	Usado para ativar ou desativar as funções da entidade

		gerenciada
availabilityStatus	RO	Identifica quando a entidade gerenciada encontra-se capaz de executar suas funções normais
egressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da saída
ingressMaxAssignableBandwith	RO	Identifica a máxima largura de banda que pode ser associada a esta conexão na direção da entrada
egressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da saída
ingressAvailableBandwith	RO	Identifica a largura de banda ainda disponível para ser associada a esta conexão na direção da entrada
userLabel	RW	Permite aos sistemas de gerência incluir informações adicionais para esta camada ATM
linkTPType	RW	Descreve o tipo de interface suportado pela conexão: UNI, inter-NNI, intra-NNI, unconfigured
loopbackLocationIdentifier	RW	Utilizado para identificar quando células OAM devem ser retornadas com propósito de teste
iLMIVirtualIdentifier	RW	Identifica o valor VPI/VCI utilizado pelo UNI
supportingNELocation	RW	Identifica a localização do NE representada por esta entidade gerenciada
supportingCircuitPackLocation	RW	Identifica a localização do circuito representada por esta entidade gerenciada
serverTTPName	RW	Define um nome descritivo para representar a terminação de trail suportada por esta entidade gerenciada
serverTTPCharacteristicInfo	RW	Identifica o tipo da terminação de trail representada por esta entidade gerenciada
serverTTPPortId	RW	Identifica a porta da terminação de trail representada por esta entidade gerenciada
serverTTPOperationalState	RW	Identifica o estado operacional da terminação de trail representada por esta entidade gerenciada
serverTTPAdditionalInformation	RW	Contém informações específicas adicionais em relação à terminação de trail representada por esta entidade gerenciada
cellScramblingEnable	RW	Permite ativar ou desativar o embaralhamento de células OAM
subscriberAddress	RW	Contém o endereço do assinante associado a esta entidade gerenciada
preferedCarrier	RW	Contém a informação de qual provedor é o preferido quando selecionado diretamente por esta entidade

		gerenciada
Notificação	**Descrição**	
managedEntityCreation	Usada para reportar a criação de uma nova instância desta entidade gerenciada	
managedEntityDeletion	Usada para reportar a remoção de uma instância desta entidade gerenciada	
attributeValueChange	Usada para reportar alterações no conteúdo dos atributos desta entidade, informando qual atributo foi alterado, e o seu valor antigo e o seu valor novo	
Operação	**Descrição**	
queryTopologicalLink	Usada para obter o identificador da entidade gerenciada *TopologicalLink* que é terminado por este *VpLinkEnd*	
querySubnetwork	Usada para obter as sub-redes delimitadas por este *VpLinkEnd*	
queryNetworkTTP	Usada para obter os *NetworkTTPs* associados com este *VpLinkEnd*	
associateNetworkTTP	Usada para associar um novo *NetworkTTP* à este *VpLinkEnd*	
traceLink	Usada para identificar as conexões de sub-rede que terminam neste *VpLinkEnd*	
Relacionamento	**Descrição**	
TopologicalLink	Cada uma destas instâncias pode ser terminada por uma entidade gerenciada *VpLinkEnd*	
LogicalLinkTP	Cada uma destas instâncias pode ser suportada por uma ou mais entidades gerenciadas *VpLinkEnd*	
Subnetwork	Cada entidade gerenciada *VpLinkEnd* está associada a pelo menos uma destas instâncias	
NetworkAccessProfile	Uma destas instâncias pode ser usada por uma entidade gerenciada *VpLinkEnd*	
NetworkTTP	Cada entidade gerenciada *VpLinkEnd* pode estar associada a uma instância de terminação de trail da camada servidora	
NetworkCTP	Cada entidade gerenciada *VpLinkEnd* suporta estas instâncias presentes na mesma camada VC	

Tabela 26: Elementos da entidade gerenciada *VpLinkEnd*

A entidade gerenciada *VpLinkEnd* está modelada em UML de acordo com a Figura 44.

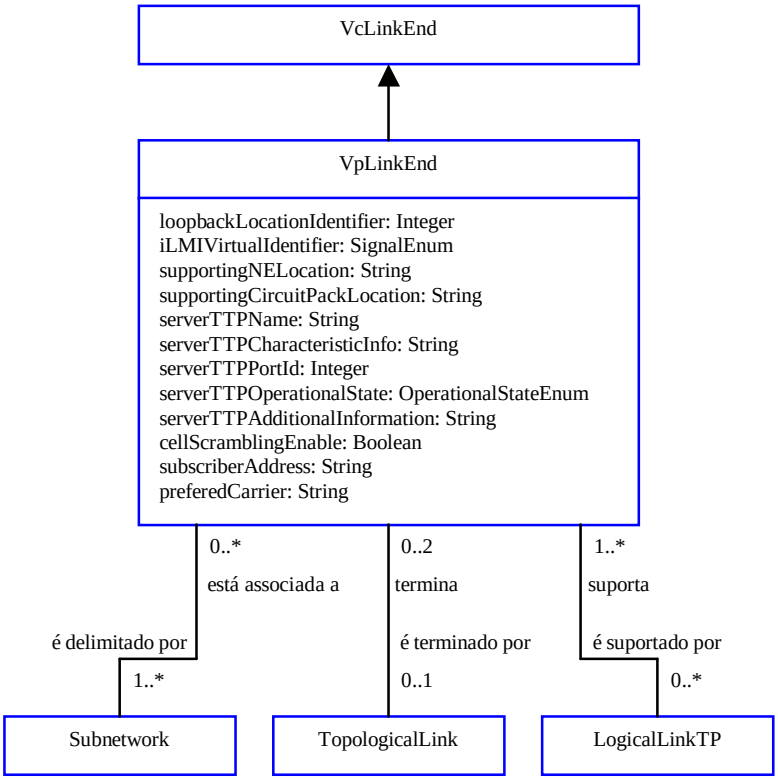

Figura 44: Entidade Gerenciada *VpLinkEnd*

3.4. Modelo de Informação

O modelo de informação completo, apresentado na Figura 45, visa basicamente modelar o VP e o VC de uma rede ATM como a representada esquematicamente na Figura 16. Esta modelagem implica na utilização de objetos auxiliares para representar as sub-redes, os enlaces, os trails e os perfis necessários, assim como de objetos responsáveis pelo armazenamento e pelo encaminhamento de eventos gerados pelo sistema.

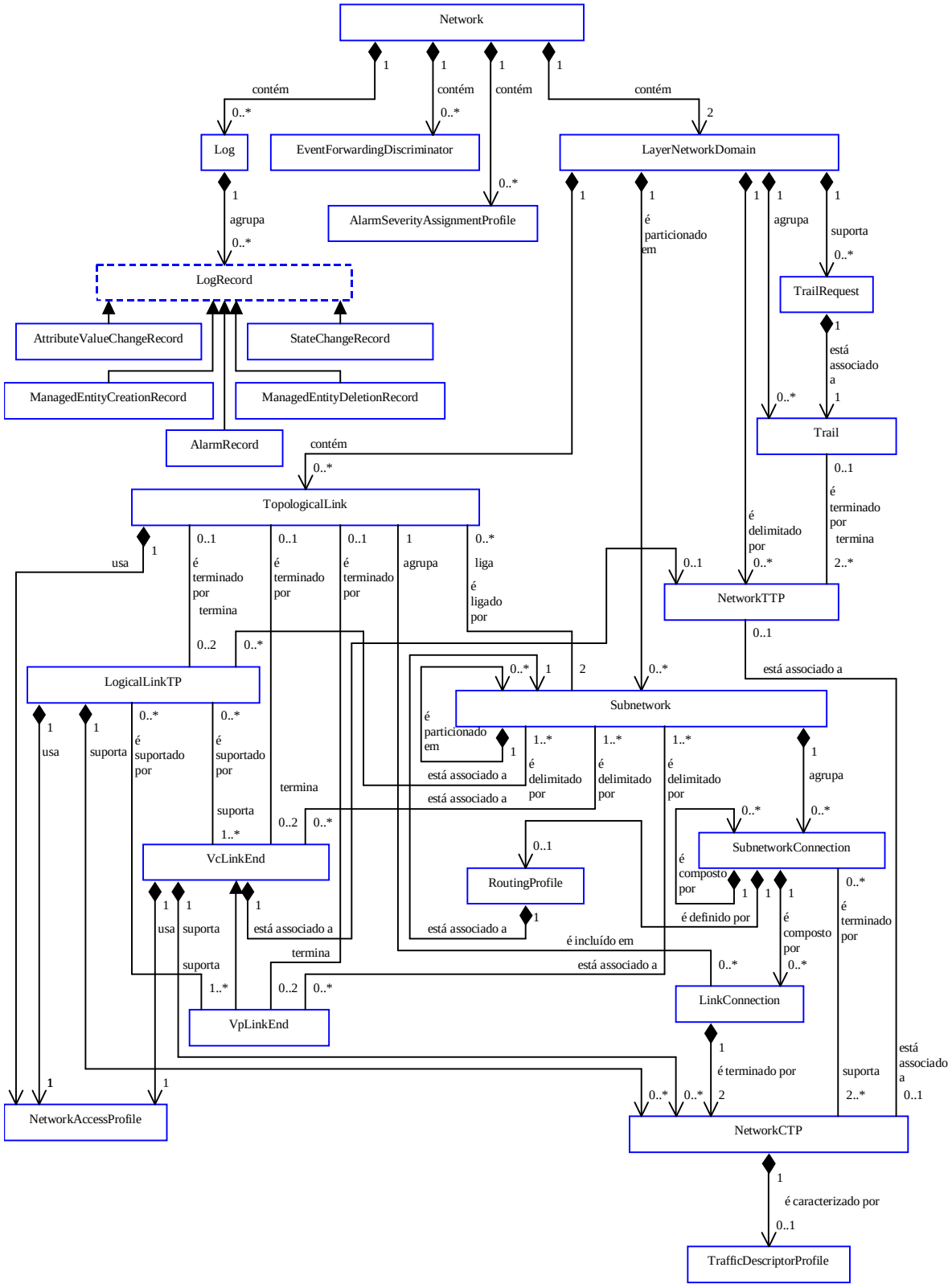

Figura 45: Modelo de Informação para Gerência de Rede ATM

3.5. Conclusão

Para especificar uma aplicação de gerência de configuração e de falhas de redes de telecomunicações, baseada na tecnologia ATM, é necessário que se desenvolva um modelo de informação para redes ATM. O modelo de informação desenvolvido neste capítulo modela a funcionalidade de gerenciamento de acordo com os requisitos apresentados pelo ATM Fórum para as gerências de configuração e de falhas.

Neste capítulo foi desenvolvido o modelo de informação de gerência TMN para o nível de rede para o gerenciamento de configuração e de falhas de uma rede de transmissão baseada na tecnologia de transmissão ATM. Para que o modelo de informação se tornasse independente da plataforma de desenvolvimento a ser utilizada na implementação, utilizou-se a notação neutra UML para a modelagem do sistema de gerenciamento. Este modelo está apresentado através dos diagramas de classe do UML.

4. Metodologia de Implementação do Modelo de Informação ATM

4.1. Introdução

A especificação do modelo de informação de gerência do nível de rede para redes ATM foi feita utilizando-se os diagramas de classe do UML, os quais representam exclusivamente a informação sintática dos objetos a serem instanciados. Para uma especificação completa de um modelo de informação é necessário que se provenha também uma caracterização da forma com que estes objetos interagem em tempo de execução.

A interação dinâmica dos objetos descritos pelos diagramas de classe será apresentada utilizando-se os diagramas de seqüência do UML, aplicados a cenários de operação bem definidos. Estes cenários de operação devem ser abrangentes o suficiente para que qualquer operação de gerência sobre o modelo de informação seja um caso específico dos cenários apresentados.

Para a apresentação destes cenários de operação utilizou-se a formalização e a metodologia definidas pelo TM Fórum [NMF 025]: os *ensembles*. Os *ensembles* constituem, basicamente, uma metodologia de descrição do comportamento de entidades gerenciadas através de um formulário de especificação de um contexto de gerência e dos correspondentes procedimentos envolvidos.

Este capítulo de apresentação da metodologia de implementação do modelo de rede ATM está composto basicamente por duas seções: a primeira apresenta os conceitos e a formalização da metodologia baseada em *ensembles* utilizada neste trabalho e a segunda contém os cenários de operação, na forma de *ensembles*, que melhor caracterizam o comportamento das entidades gerenciadas do modelo de informação de rede ATM.

4.2. Formalização da Metodologia de Implementação: Ensemble

A metodologia de implementação do modelo de informação ATM será apresentado na forma de cenários, os quais serão especificados e descritos formalmente utilizando-se os *ensembles*[5] de acordo com os requisitos do TM Fórum.

Um *ensemble* simplificado, como o proposto e utilizado pelo ATM Fórum, é constituído pelos seguintes cinco itens:

[5] Este trabalho utiliza a metodologia e a formalização simplificadas dos *ensembles*, da mesma forma com que foi feito pelo ATM Fórum nas especificações de cenários.

- **Contexto de Gerência de Rede**: contém uma descrição textual da arquitetura de gerenciamento envolvida no cenário de operação;
- **Arquitetura de Transporte**: contém uma descrição textual da arquitetura de transporte a ser gerenciada no cenário de operação;
- **Requisitos Funcionais**: contém uma descrição textual do conjunto das funções de gerência executadas no cenário de operação;
- **Entidades Gerenciadas**: contém uma lista e uma descrição gráfica, na forma de diagramas de seqüência do UML, das entidades gerenciadas envolvidas no cenário de operação e das operações de gerência executadas. Os objetos envolvidos no cenário foram nomeados (nomes criados como exemplo) de forma a serem facilmente referenciados e as operações foram numeradas para serem descritas na seção Cenário do *ensemble*;
- **Cenário**: contém uma descrição textual dos passos seguidos pela função de gerência (numerados no diagrama de seqüência do *ensemble*) e das operações utilizadas para executar estes passos.

Os *ensembles* são desenvolvidos para cenários específicos bem caracterizados de forma que possam ser utilizados por outros *ensembles* mais abrangentes e, desta forma, permitindo constituir cenários cada vez mais abrangentes e complexos.

4.3. Ensembles

A interação dinâmica das entidades gerenciadas constituintes do modelo de informação do nível de rede para o gerenciamento de redes ATM é a forma com que estas entidades se comportam durante os procedimentos usuais de operação de um sistema de gerenciamento. Para analisar este comportamento, identificaram-se alguns cenários representativos das operações de gerenciamento que podem ser executadas sobre os objetos constituintes deste modelo de informação ATM.

Os cenários deste modelo de informação foram agrupados em *ensembles* de três tipos de contextos: de inicialização do sistema, de supervisão de alarmes e de configuração do sistema. Nas próximas seções foram desenvolvidos os *ensembles* para os seguintes cenários:

- *Ensembles* de inicialização do sistema:
  1. Inicialização do Sistema;
  2. Inicialização do Registro de Alarmes;
- *Ensembles* de supervisão de alarmes:
  3. Registro de uma Aplicação de Gerência para o Recebimento de Alarmes;
  4. Geração de Alarmes de Falha de Equipamentos ATM;
- *Ensembles* de configuração do sistema:
  5. Criação de uma Conexão de Sub-rede Simples;
  6. Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação;

7. Criação de Sub-redes para o Particionamento de uma Sub-rede;
8. Criação de uma Conexão de Enlace entre Sub-redes;
9. Criação de uma Conexão de Sub-rede Composta;
10. Criação de um Trail Servidor;
11. Operação sobre um Trail Servidor por Agendamento;
12. Criação de uma Conexão de Enlace entre Sub-redes através de um Trail Servidor.

Cada uma das entidades gerenciadas constituintes do modelo de informação ATM foi referenciada por pelo menos um dos cenários dos *ensembles* apresentados nas próximas seções, de acordo com a Tabela 27.

Entidade Gerenciada	***Ensembles***
AlarmRecord	4.3.4
AlarmSeverityAssignmentProfile	4.3.1
AttributeValueChangeRecord	4.3.4
EventForwardingDiscriminator	4.3.3, 4.3.4
LayerNetworkDomain	4.3.1, 4.3.8, 4.3.9, 4.3.10, 4.3.11, 4.3.12
LinkConnection	4.3.8, 4.3.9, 4.3.12
Log	4.3.2, 4.3.4
LogicalLinkTP	4.3.8, 4.3.9
ManagedEntityCreationRecord	4.3.4
ManagedEntityDeletionRecord	4.3.4
Network	4.3.1, 4.3.2, 4.3.3
NetworkAccessProfile	4.3.8, 4.3.9, 4.3.12
NetworkCTP	4.3.4, 4.3.5, 4.3.6, 4.3.8, 4.3.9, 4.3.10, 4.3.12
NetworkTTP	4.3.10, 4.3.11, 4.3.12
RoutingProfile	4.3.6, 4.3.9
StateChangeRecord	4.3.4
Subnetwork	4.3.1, 4.3.5, 4.3.6, 4.3.7, 4.3.8, 4.3.9, 4.3.12
SubnetworkConnection	4.3.5, 4.3.6, 4.3.9
TopologicalLink	4.3.8, 4.3.9, 4.3.12
TrafficDescriptorProfile	4.3.5
Trail	4.3.10, 4.3.11, 4.3.12
TrailRequest	4.3.11
VcLinkEnd	4.3.12
VpLinkEnd	4.3.12

Tabela 27: Entidades gerenciadas referenciadas pelos *ensembles*

4.3.1. Inicialização do Sistema

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

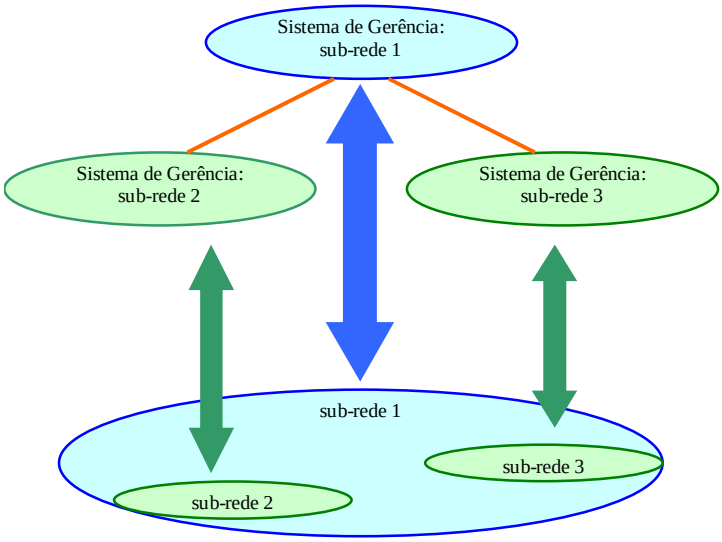

Figura 46: Arquitetura da gerência de rede

Arquitetura de Transporte

A arquitetura de transporte consiste em uma única sub-rede, tanto para a camada VC quanto para a camada VP, não contendo nenhum ponto de terminação pré-definido, como está representado na Figura 47.

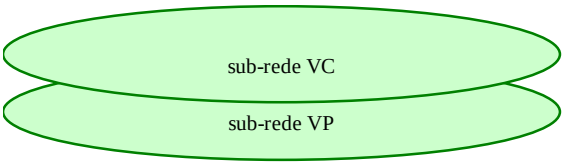

Figura 47: Arquitetura de transporte para inicialização do sistema

O objetivo deste *ensemble* é demonstrar o procedimento de inicialização da aplicação agente do sistema de gerência.

Requisitos Funcionais

Por tratar-se da inicialização da aplicação agente do sistema de gerência de rede, este *ensemble* não implementa nenhuma das funções básicas de gerência de rede.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* de inicialização do sistema são:

- AlarmSeverityAssignmentProfile;
- LayerNetworkDomain;
- Network;
- Subnetwork.

O diagrama de seqüência do UML apresentado na Figura 48 representa a interação destas entidades gerenciadas durante a inicialização do sistema.

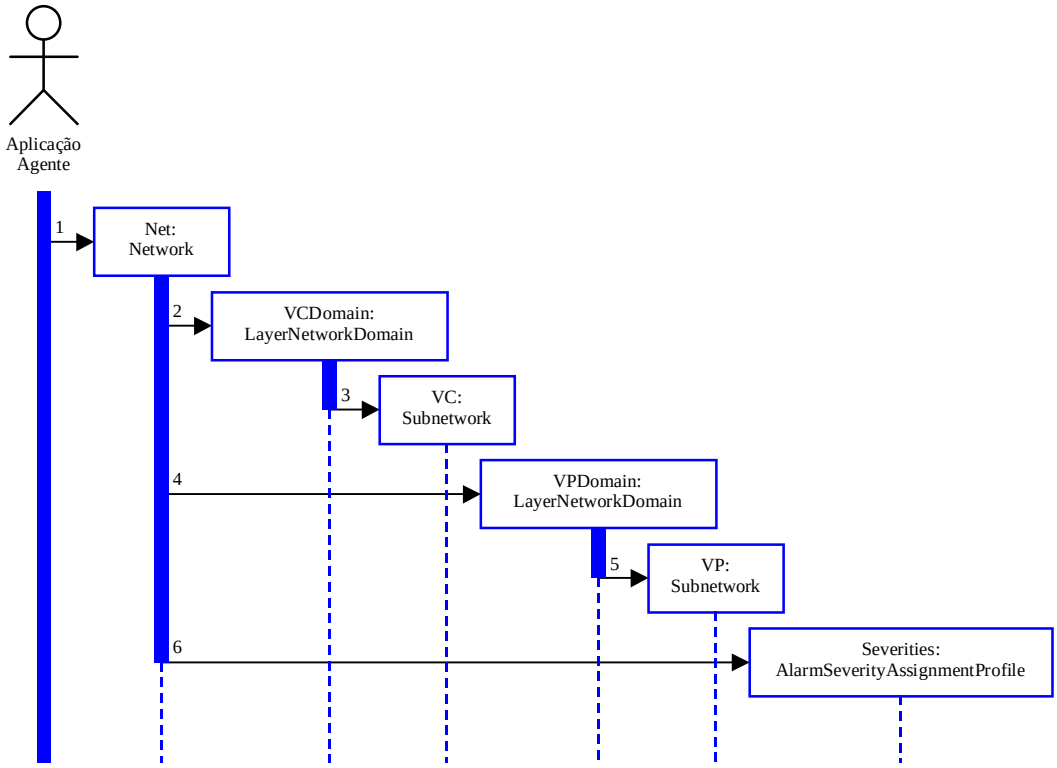

Figura 48: Diagrama de seqüência da inicialização do sistema

Cenário

O processo de inicialização do sistema descrito na Figura 48 apresenta a interação entre as entidades gerenciadas envolvidas na inicialização da aplicação agente. Os passos componentes desta inicialização são:

1. A aplicação agente cria uma instância da classe *Network*, denominada *Net*, que é a raiz de todo o modelo de informação de rede ATM (veja Figura 45);
2. O objeto *Net* cria uma instância da classe *LayerNetworkDomain* para a camada VC, denominada *VCDomain*;
3. O objeto *VCDomain* cria uma instância da classe *Subnetwork*, denominada *VC*;
4. O objeto *Net* cria uma instância da classe *LayerNetworkDomain* para a camada VP, denominada *VPDomain*;
5. O objeto *VPDomain* cria uma instância da classe *Subnetwork*, denominada *VP*;
6. O objeto *Net* cria uma instância da classe *AlarmSeverityAssignmentProfile*, denominada *Severities*, contendo a lista das falhas que podem ser alarmadas pelo sistema com as suas respectivas severidades.

4.3.2. Inicialização do Registro de Alarmes

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de inicialização do registro de alarmes da aplicação agente, independentemente de uma arquitetura específica de transporte.

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Falhas), em relação ao registro de alarmes.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- Log;
- Network.

O diagrama de seqüência do UML apresentado na Figura 49 representa a interação destas entidades gerenciadas durante a inicialização do registro de alarmes do sistema.

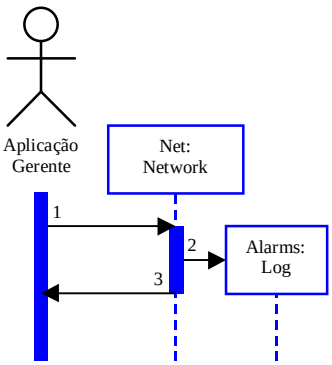

Figura 49: Diagrama de seqüência da inicialização do registro de alarmes

Cenário

O processo de inicialização do registro de alarmes do sistema descrito na Figura 49 apresenta a interação entre as entidades gerenciadas envolvidas na sua inicialização. Os passos componentes desta inicialização são:

1. A aplicação de gerência pede ao objeto *Net* que inicialize o registro de alarmes, especificando um conjunto específico de notificações a serem armazenadas;
2. O objeto *Net* cria uma instância da classe *Log*, denominada *Alarms*, inicializada com o conjunto de alarmes a serem armazenados;
3. O objeto *Net* retorna para a aplicação de gerência o resultado da inicialização do registro de alarmes.

4.3.3. Registro de uma Aplicação de Gerência para o Recebimento de Alarmes

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de registro de uma aplicação de gerência para o recebimento de alarmes da aplicação agente, independentemente de uma arquitetura específica de transporte.

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Falhas), em relação ao envio de alarmes.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- EventForwardingDiscriminator;
- Network.

O diagrama de seqüência do UML apresentado na Figura 50 representa a interação destas entidades gerenciadas durante o registro de uma aplicação de gerência para o recebimento de alarmes.

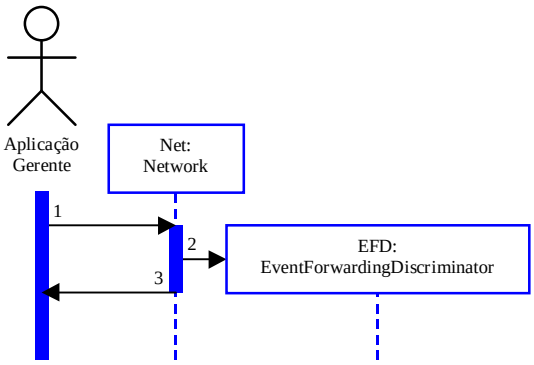

Figura 50: Diagrama de seqüência de registro de uma aplicação de gerência

Cenário

O processo de registro de uma aplicação de gerência para o recebimento de alarmes está descrito na Figura 50. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *Net* que crie um canal para o envio de alarmes, especificando o endereço da aplicação de gerência destinatária e características específicas dos alarmes que deseja receber;

2. O objeto *Net* cria uma instância da classe *EventForwardingDiscriminator*, denominada *EFD*, inicializada com o endereço da aplicação de gerência e o filtro contendo a especificação dos alarmes a serem enviados;
3. O objeto *Net* retorna para a aplicação de gerência o resultado do registro.

4.3.4. Geração de Alarmes de Falha de Equipamentos ATM

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar os procedimentos envolvidos na geração espontânea de um alarme de equipamento de rede ATM, independentemente de uma arquitetura específica de transporte. Este *ensemble* é o mesmo tanto para a camada VC quanto para a VP, assim como independe do tipo de notificação: AttributeValueChange, StateChange, ManagedEntityCreation, ManagedEntityDeletion, Alarm.

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à monitoração dos estados de sub-redes, e na seção 0 (Gerência de Falhas), em relação à geração, ao registro e ao envio de alarmes,.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- AlarmRecord ou AttributeValueChangeRecord ou ManagedEntityCreationRecord ou ManagedEntityDeletionRecord ou StateChangeRecord;
- EventForwardingDiscriminator;
- Log;
- NetworkCTP.

O diagrama de seqüência do UML apresentado na Figura 51 representa a interação destas entidades gerenciadas para a geração, o registro e o envio de alarmes.

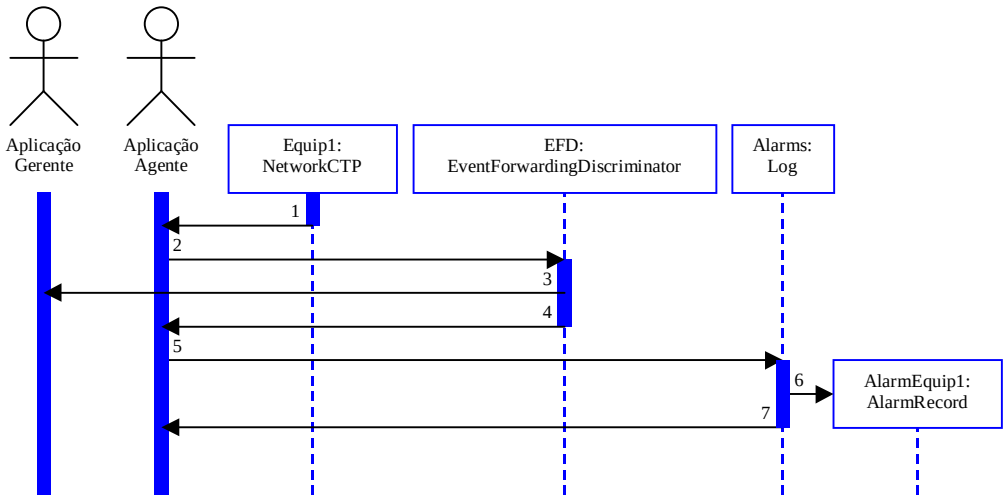

Figura 51: Diagrama de seqüência da criação, registro e envio de alarmes

Cenário

O processo de criação, registro e envio de um alarme para uma aplicação de gerência está descrito na Figura 51. Os passos componentes deste processo são (para qualquer tipo de notificação ocorrido em qualquer tipo de classe, o processo é idêntico):

1. Uma falha de um equipamento é identificada espontaneamente pelo objeto *Equip1*, instância de uma classe *NetworkCTP*, e enviada na forma de uma notificação ao controle da aplicação agente para que seja tratada;
2. O núcleo da aplicação agente envia a notificação para as instâncias da classe *EventForwardingDiscriminator*, neste caso apenas para o objeto *EFD*, para que seja processada;
3. O objeto *EFD* verifica se a notificação está de acordo com as características definidas no seu filtro e, neste caso, envia-a para o endereço da aplicação de gerência;
4. O objeto *EFD* retorna para a aplicação agente o resultado da filtragem e do envio da notificação;
5. O núcleo da aplicação agente envia a notificação para as instâncias da classe *Log*, neste caso apenas para o objeto *Alarms*, para que seja processada;
6. O objeto *Alarms* verifica se a notificação é do tipo a selecionado para que seja armazenada e, neste caso, cria uma instância da classe *AlarmRecord*, denominada *AlarmEquip1*, incluindo-a na sua lista de registros de alarmes armazenados;
7. O objeto *Alarms* retorna para a aplicação agente o resultado da filtragem e do registro da notificação.

4.3.5. Criação de uma Conexão de Sub-rede Simples

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de uma conexão de sub-rede simples (Figura 52) contida em um único domínio: tanto VC quanto VP. Este processo supõe que a sub-rede já exista e que os *NetworkCTPs* requeridos ainda não existam.

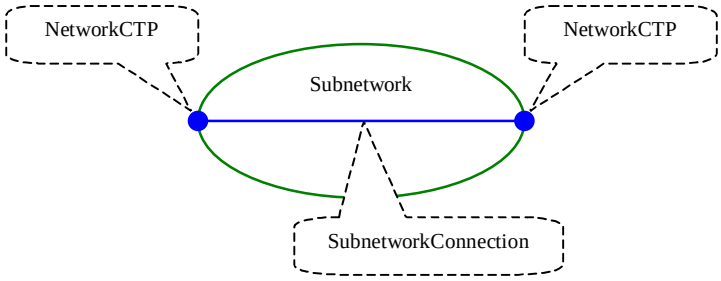

Figura 52: Conexão de sub-rede para uma sub-rede simples

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das conexões de sub-redes, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- NetworkCTP;
- Subnetwork;
- SubnetworkConnection;
- TrafficDescriptorProfile.

O diagrama de seqüência do UML apresentado na Figura 53 representa a interação destas entidades gerenciadas durante a criação de uma conexão de sub-rede simples.

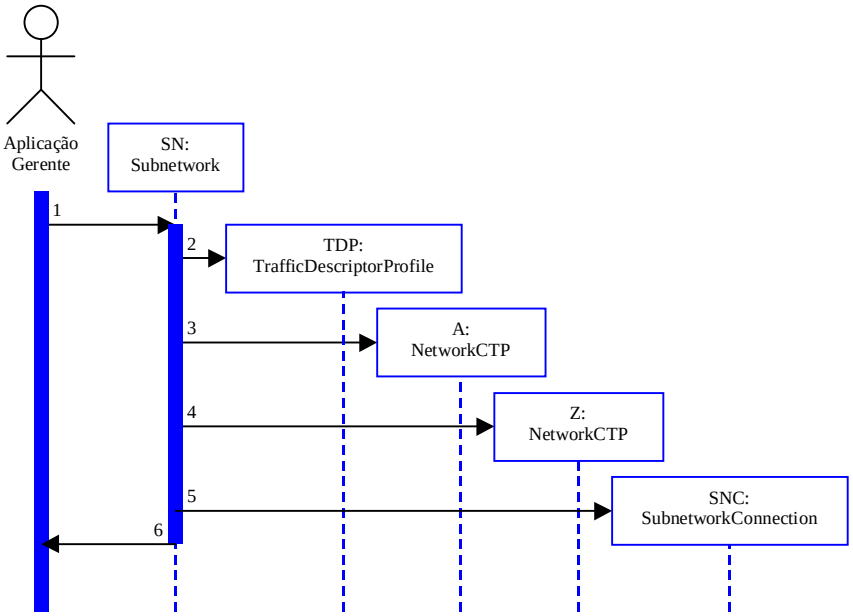

Figura 53: Diagrama de seqüência de criação de uma conexão de sub-rede simples

Cenário

O processo de criação de uma conexão de sub-rede em uma sub-rede simples contida em uma única camada (VC ou VP) está descrito na Figura 53. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *SN* que crie uma conexão na sub-rede, especificando as características do tráfego que deve suportar;
2. O objeto *SN* cria uma instância da classe *TrafficDescriptorProfile*, denominada *TDP*, contendo as características do tráfego a ser suportado pela conexão;
3. O objeto *SN* cria uma instância da classe *NetworkCTP*, denominada *A*, contendo um novo valor de VCI ou de VPI, uma referência ao objeto *TDP* e a inclui na sua lista de pontos de terminação de conexões;
4. O objeto *SN* cria uma segunda instância da classe *NetworkCTP*, denominada *Z*, contendo o valor de VCI ou de VPI, uma referência ao objeto *TDP* e a inclui na sua lista de pontos de terminação de conexões;
5. O objeto *SN* cria uma instância da classe *SubnetworkConnection*, denominada *SNC*, e a inclui na sua lista de conexões de sub-redes;
6. O objeto *SN* retorna para a aplicação de gerência o resultado da criação da conexão de sub-rede.

4.3.6. Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de uma conexão de sub-rede simples (Figura 52) contida em um único domínio: tanto VC quanto VP. Este processo supõe que a sub-rede já exista e que os *NetworkCTPs* requeridos já existam e sejam fornecidos pelo sistema de gerência.

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das conexões de sub-redes, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- NetworkCTP;
- RoutingProfile;
- Subnetwork;
- SubnetworkConnection.

O diagrama de seqüência do UML apresentado na Figura 54 representa a interação destas entidades gerenciadas durante a criação de uma conexão de sub-rede simples com pontos de terminação selecionados pela aplicação de gerência.

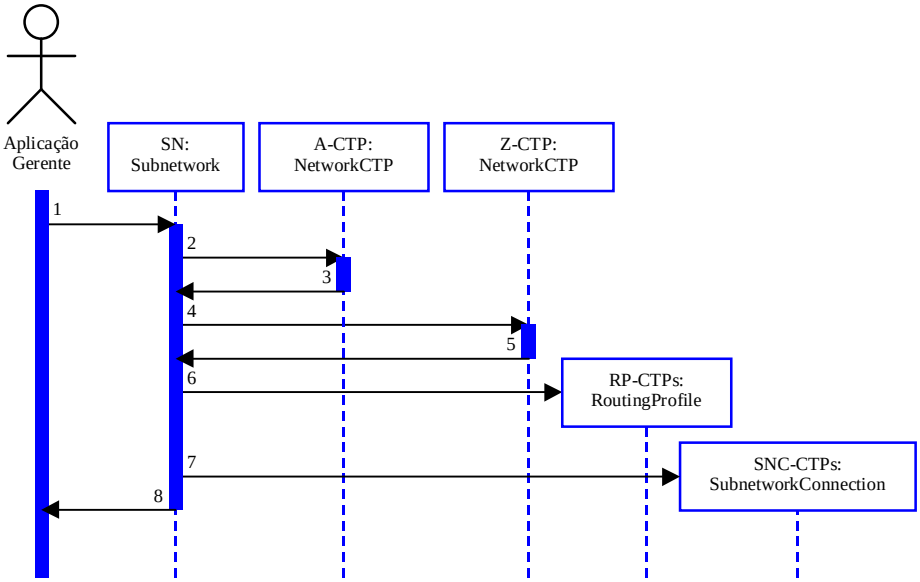

Figura 54: Diagrama de seqüência de criação de uma conexão de sub-rede simples com TPs

Cenário

O processo de criação de uma conexão de sub-rede em uma sub-rede simples contida em uma única camada (VC ou VP) está descrito na Figura 54. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *SN* que crie uma conexão na sub-rede, especificando as características do tráfego que deve suportar assim como os *NetworkCTPs A-CTP* e *Z-CTP* a serem conectados;
2. O objeto *SN* verifica se o ponto de terminação representado pelo objeto *A-CTP* suporta as características do tráfego requeridas pela conexão;
3. O objeto *A-CTP* retorna ao objeto *SN* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;
4. O objeto *SN* verifica se o ponto de terminação representado pelo objeto *Z-CTP* suporta as características do tráfego requeridas pela conexão;
5. O objeto *Z-CTP* retorna ao objeto *SN* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;
6. O objeto *SN* cria uma instância da classe *RoutingProfile*, denominada *RP-CTPs*, contendo as características do tráfego a ser suportado pela conexão e incluindo uma referência à sub-rede *SN* à qual está associada;
7. O objeto *SN* cria uma instância da classe *SubnetworkConnection*, denominada *SNC-CTPs*, contendo uma referência ao objeto *RP-CTPs* e a inclui na sua lista de conexões de sub-redes;
8. O objeto *SN* retorna para a aplicação de gerência o resultado da criação da conexão de sub-rede.

4.3.7. Criação de Sub-redes para o Particionamento de uma Sub-rede

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de sub-redes para o particionamento de uma sub-rede (Figura 55) contidas em um único domínio: tanto VC quanto VP.

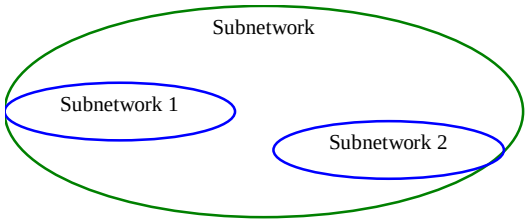

Figura 55: Sub-redes de particionamento de uma sub-rede

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das sub-redes, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- Subnetwork.

O diagrama de seqüência do UML apresentado na Figura 56 representa a interação destas entidades gerenciadas durante a criação de uma sub-rede de particionamento.

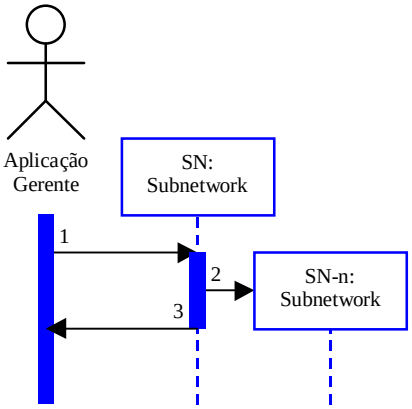

Figura 56: Diagrama de seqüência de criação de uma sub-rede de particionamento

Cenário

O processo de criação de sub-redes para o particionamento de uma sub-rede simples contida em uma única camada (VC ou VP) está descrito na Figura 56. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *SN* que crie uma sub-rede para o seu particionamento;
2. O objeto *SN* cria uma instância da classe *Subnetwork*, denominada *SN-n*, e a inclui na sua lista de particionamento;
3. objeto *SN* retorna para a aplicação de gerência o resultado da criação da sub-rede.

4.3.8. Criação de uma Conexão de Enlace entre Sub-redes

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de uma conexão de enlace, que não necessita de uma camada servidora, entre sub-redes (Figura 57) contidas em um único domínio: tanto VC quanto VP.

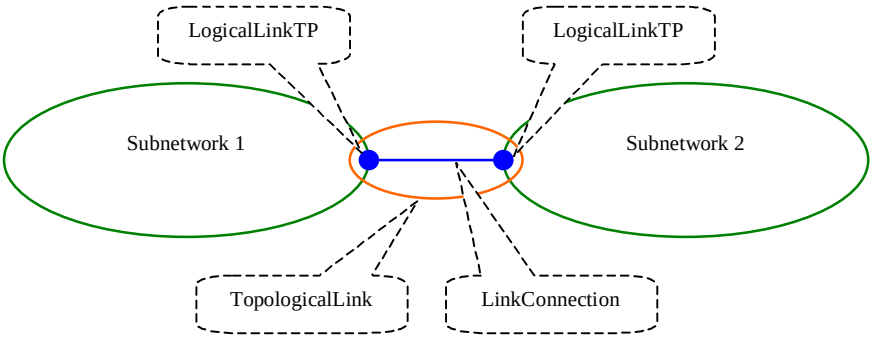

Figura 57: Conexão de enlace entre sub-redes

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das conexões de enlaces, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- LayerNetworkDomain;
- LinkConnection;
- LogicalLinkTP;
- NetworkAccessProfile;
- NetworkCTP;
- Subnetwork;
- TopologicalLink.

O diagrama de seqüência do UML apresentado na Figura 58 representa a interação destas entidades gerenciadas durante a criação de um enlace entre sub-redes que não necessita de uma camada servidora. Esta operação de criação pressupõe que as sub-redes já existam, cada qual com os pontos de terminação *NetworkCTP* disponíveis e fornecidos pelo sistema de gerenciamento.

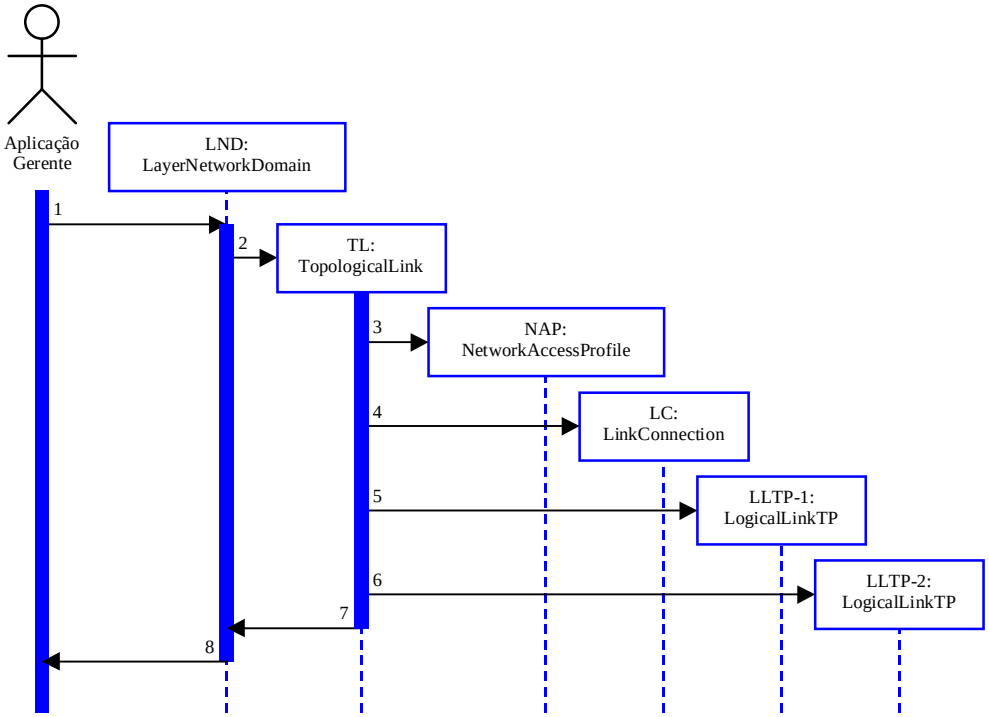

Figura 58: Diagrama de seqüência de criação de um enlace entre sub-redes

Cenário

O processo de criação de uma conexão de enlace entre sub-redes contida em uma única camada (VC ou VP) está descrito na Figura 58. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *LND* que crie uma conexão de enlace entre as sub-redes *SN-1* e *SN-2*, utilizando os seus pontos de terminação respectivos *NCTP-1* e *NCTP-2*, especificando as características do tráfego que deve suportar;
2. O objeto *LND* cria uma instância da classe *TopologicalLink*, denominada *TL*, com referências às sub-redes *SN-1* e *SN-2*, e pede que crie uma conexão de enlace entre as sub-redes utilizando os seus pontos de terminação respectivos *NCTP-1* e *NCTP-2*, especificando as características do tráfego que deve suportar;
3. O objeto *TL* cria uma instância da classe *NetworkAccessProfile*, denominada *NAP*, contendo as características do tráfego a ser suportado pela conexão;
4. O objeto *TL* cria uma instância da classe *LinkConnection*, denominada *LC*, com uma referência aos objetos *NCTP-1* e *NCTP-2*, e a inclui na sua lista de conexões de enlace;
5. O objeto *TL* cria uma instância da classe *LogicalLinkTP*, denominada *LLTP-1*, com referências aos objetos *NAP*, *SN-1* e *NCTP-1*, e a inclui na sua lista de pontos de terminação de conexões de enlaces lógicos;
6. O objeto *TL* cria uma segunda instância da classe *LogicalLinkTP*, denominada *LLTP-2*, com referências aos objetos *NAP*, *SN-2* e *NCTP-2*, e a inclui na sua lista de pontos de terminação de conexões de enlaces lógicos;

7. O objeto *TL* retorna para a o objeto *LND* o resultado da criação dos elementos da conexão de enlace;
8. O objeto *LND* retorna para a aplicação de gerência o resultado da criação da conexão de enlace.

4.3.9. Criação de uma Conexão de Sub-rede Composta

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de uma conexão de sub-rede composta, que não necessita de uma camada servidora, formada por sub-redes (Figura 52, Figura 57 e Figura 59) contidas em um único domínio: tanto VC quanto VP.

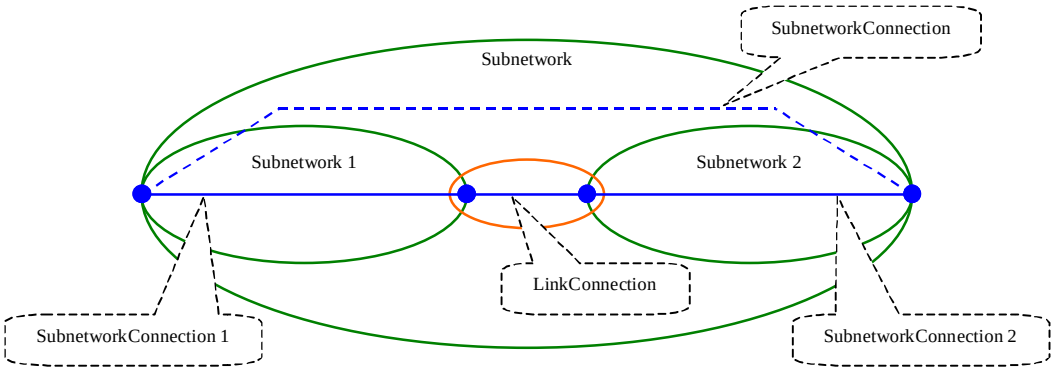

Figura 59: Conexão de sub-rede composta

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das conexões de enlaces e de sub-rede, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- LayerNetworkDomain;
- LinkConnection;
- LogicalLinkTP;
- NetworkAccessProfile;
- NetworkCTP;
- RoutingProfile;
- Subnetwork;
- SubnetworkConnection;
- TopologicalLink.

O diagrama de seqüência do UML apresentado na Figura 60 representa a interação destas entidades gerenciadas durante a criação de uma conexão de sub-rede composta que não necessita de uma camada servidora. Esta operação de criação pressupõe que as sub-redes já existam, cada qual com os pontos de terminação *NetworkCTP* disponíveis e fornecidos pelo sistema de gerenciamento.

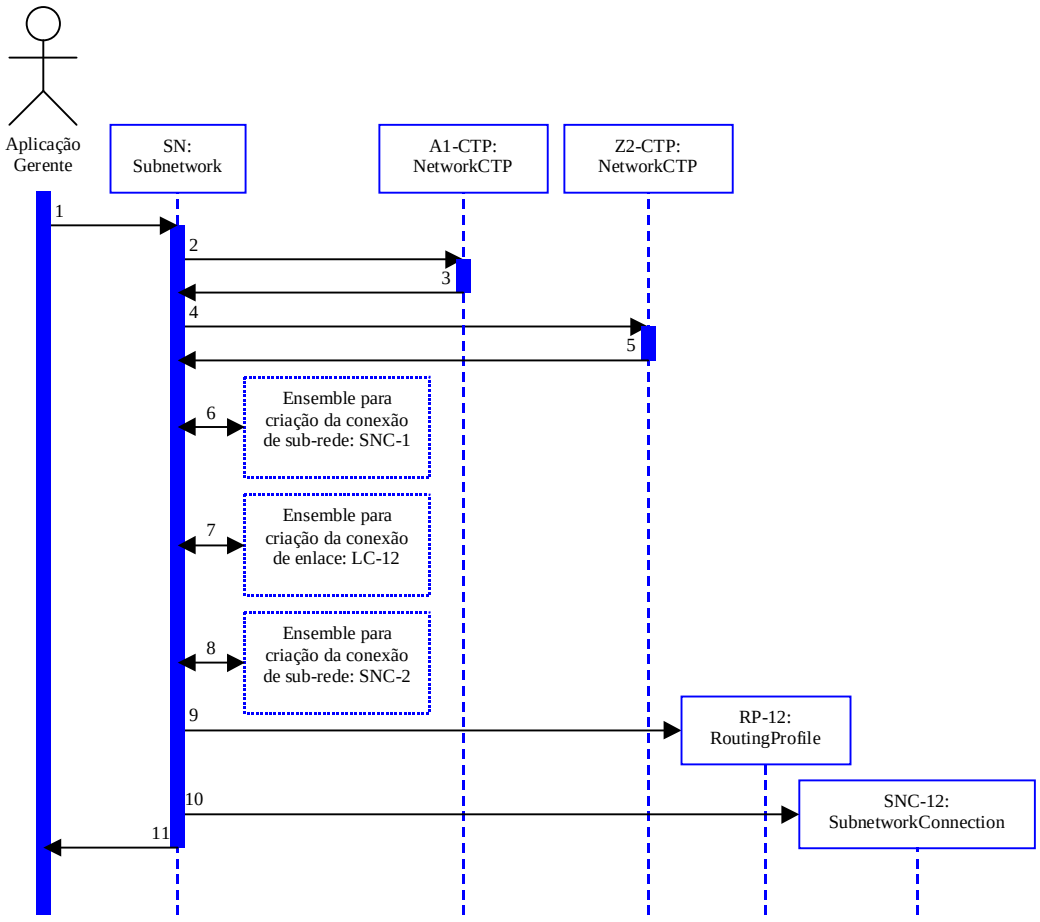

Figura 60: Diagrama de seqüência de criação de uma conexão de sub-rede composta

Cenário

O processo de criação de uma conexão de sub-rede em uma sub-rede composta contida em uma única camada (VC ou VP) está descrito na Figura 60. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *SN* que crie uma conexão na sub-rede *SN* composta pelas sub-redes *SN-1* e *SN-2*, especificando as características do tráfego que deve suportar assim como os *NetworkCTPs A1-CTP* e *Z1-CTP* da sub-rede *SN-1* e *A2-CTP* e *Z2-CTP* da sub-rede *SN-2* a serem conectados;
2. O objeto *SN* verifica se o ponto de terminação representado pelo objeto *A1-CTP* suporta as características do tráfego requeridas pela conexão;
3. O objeto *A1-CTP* retorna ao objeto *SN* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;
4. O objeto *SN* verifica se o ponto de terminação representado pelo objeto *Z2-CTP* suporta as características do tráfego requeridas pela conexão;
5. O objeto *Z2-CTP* retorna ao objeto *SN* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;
6. Cria-se a conexão de sub-rede denominada *SNC-1* através do *ensemble* 4.3.6 (Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação) passando os pontos de terminação *A1-CTP* e *Z1-CTP* da sub-rede *SN-1* e as características do tráfego requeridas pela conexão;
7. Cria-se a conexão de enlace denominada *LC* através do *ensemble* 4.3.6 (Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação) passando os pontos de terminação *Z1-CTP* da sub-rede *SN-1* e A2-*CTP* da sub-rede *SN-2* e as características do tráfego requeridas pela conexão;
8. Cria-se a conexão de sub-rede denominada *SNC-2* através do *ensemble* 4.3.6 (Criação de uma Conexão de Sub-rede Simples com Pontos de Terminação) passando os pontos de terminação *A2-CTP* e *Z2-CTP* da sub-rede *SN-2* e as características do tráfego requeridas pela conexão;
9. O objeto *SN* cria uma instância da classe *RoutingProfile*, denominada *RP-12*, contendo as características do tráfego a ser suportado pela conexão e incluindo uma referência à sub-rede *SN* à qual está associada;
10. O objeto *SN* cria uma instância da classe *SubnetworkConnection*, denominada *SNC-12*, contendo uma referência ao objeto *RP-12*, referências para as conexões de sub-rede *SNC-1* e *SNC-2* que a compõe, referência para a conexão de enlace *LC* que a compõe e a inclui na sua lista de conexões de sub-redes;
11. O objeto *SN* retorna para a aplicação de gerência o resultado da criação da conexão de sub-rede.

4.3.10. Criação de um Trail Servidor

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de um trail servidor (Figura 61) contido em um único domínio: tanto VC quanto VP. Este processo supõe que os *NetworkCTPs* requeridos já existam.

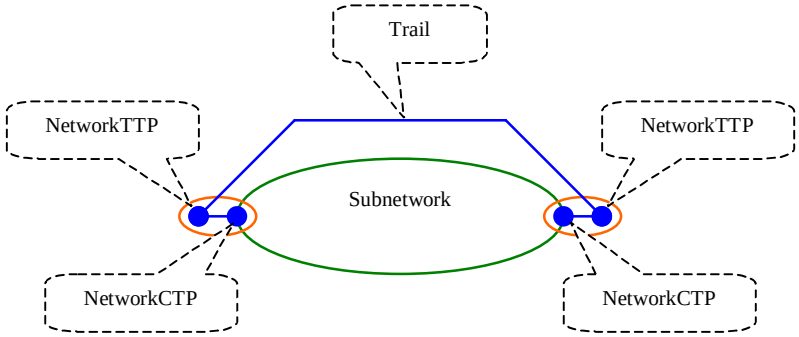

Figura 61: Trail servidor

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção da rede e das conexões de sub-redes, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- LayerNetworkDomain;
- NetworkCTP;
- NetworkTTP;

- Trail.

O diagrama de seqüência do UML apresentado na Figura 62 representa a interação destas entidades gerenciadas durante a criação de um trail servidor.

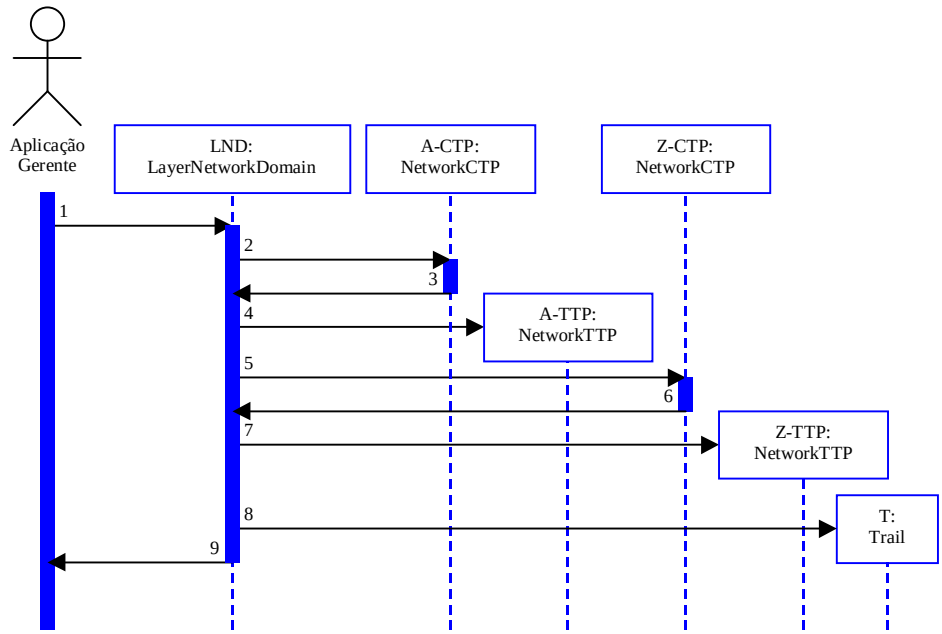

Figura 62: Diagrama de seqüência de criação de um trail servidor

Cenário

O processo de criação de um trail servidor contido em uma única camada (VC ou VP) está descrito na Figura 62. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *LND* que crie um trail entre os *NetworkCTPs A-CTP* e *Z-CTP* de uma camada servidora, especificando as características do tráfego que deve suportar;
2. O objeto *LND* verifica se o ponto de terminação representado pelo objeto *A-CTP* suporta as características do tráfego requeridas pela conexão;
3. O objeto *A-CTP* retorna ao objeto *LND* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;
4. O objeto *LND* cria uma instância da classe *NetworkTTP*, denominada *A-TTP*, incluindo uma referência ao ponto de terminação *A-CTP* ao qual está associada e uma referência inversa deste objeto;
5. O objeto *LND* verifica se o ponto de terminação representado pelo objeto *Z-CTP* suporta as características do tráfego requeridas pela conexão;
6. O objeto *Z-CTP* retorna ao objeto *LND* o resultado da verificação, o qual interrompe a criação da conexão caso a verificação retorne falsa;

7. O objeto *LND* cria uma instância da classe *NetworkTTP*, denominada *Z-TTP*, incluindo uma referência ao ponto de terminação *Z-CTP* ao qual está associada e uma referência inversa deste objeto;
8. O objeto *LND* cria uma instância da classe *Trail*, denominada *T*, incluindo uma referência aos pontos de terminação *A-TTP* e *Z-TTP* aos quais está associada e uma referência inversa em cada um destes objetos;
9. O objeto *LND* retorna para a aplicação de gerência o resultado da criação do trail servidor.

4.3.11. Operação sobre um Trail Servidor por Agendamento

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de especificação de uma operação a ser efetuada sobre um trail servidor (Figura 61) contido em um único domínio: tanto VC quanto VP. Este processo supõe que o *Trail* já exista.

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção da rede e das conexões de sub-redes, tanto para a camada VC quanto para a VP.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- LayerNetworkDomain;
- NetworkTTP;
- Trail;
- TrailRequest.

O diagrama de seqüência do UML apresentado na Figura 63 representa a interação destas entidades gerenciadas durante uma operação agendada sobre um trail servidor.

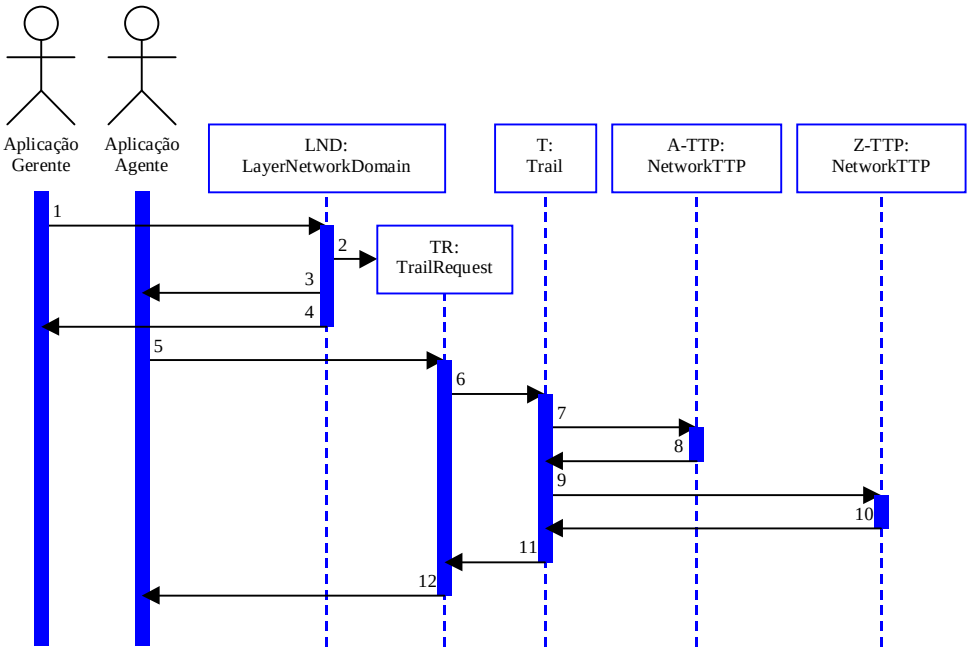

Figura 63: Diagrama de seqüência de operação agendada sobre um trail

Cenário

O processo de operação agendada sobre um trail servidor contido em uma única camada (VC ou VP) está descrito na Figura 63. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *LND* que crie um pedido de operação (alteração das características de tráfego a ser suportado, remoção do trail, inclusão ou remoção de pontos de terminação de trail) sobre um dado trail *T* a ser executado em uma data preestabelecida;
2. O objeto *LND* cria uma instância da classe *TrailRequest*, denominada *TR*, com a identificação do tipo de operação a ser executada, as informações necessárias para que esta operação seja efetuada com sucesso, a data a ser agendada para a execução da operação e uma referência ao trail *T* sobre o qual a operação deverá ser executada;
3. O objeto *LND* registra em um temporizador da aplicação agente o objeto *TR* para que seja agendada a execução da operação sobre o trail *T*;
4. O objeto *LND* retorna para a aplicação de gerência o resultado da criação do agendamento de uma operação sobre o trail;
5. Assim que o temporizador da aplicação agente verificar que a operação agendada sobre um trail deve ser executada, o objeto *TR* é ativado para que execute o operação;
6. O objeto *TR* chama o objeto *T* para que execute a operação sobre o trail com as informações necessárias;

7. O objeto *T* verifica se o ponto de terminação representado pelo objeto *A-TTP* suporta a operação a ser executada (através dos seus respectivos pontos de terminação de conexão);
8. O objeto *A-TTP* retorna ao objeto *T* o resultado da verificação, o qual interrompe a execução da operação caso a verificação retorne falsa;
9. O objeto *T* verifica se o ponto de terminação representado pelo objeto *Z-TTP* suporta a operação a ser executada (através dos seus respectivos pontos de terminação de conexão);
10. O objeto *Z-TTP* retorna ao objeto *T* o resultado da verificação, o qual interrompe a execução da operação caso a verificação retorne falsa;
11. O objeto *T* conclui a execução da operação que estava agendada e retorna ao objeto *TR* o seu resultado;
12. O objeto *TR* retorna para a aplicação agente o resultado da operação sobre o trail. A aplicação de gerência é notificada do resultado da operação através do recebimento de alarmes espontâneos de mudança de valores de atributos ou de mudança de estado de objetos.

4.3.12. Criação de uma Conexão de Enlace entre Sub-redes através de um Trail Servidor

Contexto de Gerência de Rede

O contexto de gerência é o gerenciamento do nível de rede, isto é, a visibilidade de gerenciamento se restringe a uma rede ou sub-rede de um domínio. Desta forma, a integração da gerência de sub-redes de domínios diferentes em uma única sub-rede implica necessariamente na utilização de um sistema de gerência de mais alto nível que integre os sistemas de gerência destas sub-redes, como está representado na Figura 46.

Arquitetura de Transporte

O objetivo deste *ensemble* é demonstrar o procedimento de criação de uma conexão de enlace através de um trail servidor, ou seja, que necessita de uma camada servidora, entre sub-redes (Figura 57, Figura 62, Figura 64) contidas na camada VC.

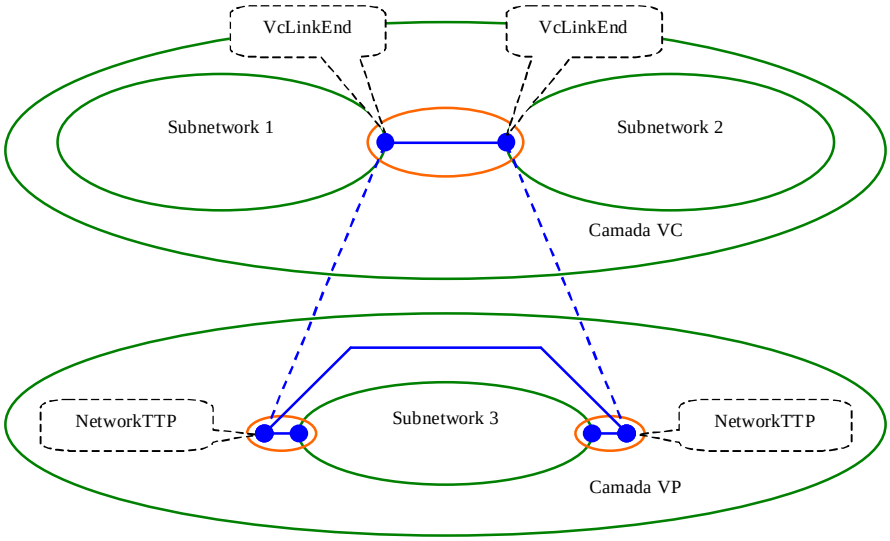

Figura 64: Conexão de enlace entre sub-redes através de um trail servidor

Requisitos Funcionais

Este *ensemble* implementa as funções básicas de gerência de rede, descritas na seção 0 (Gerência de Configuração), em relação à manutenção das conexões de enlaces, para a camada VC.

Entidades Gerenciadas

As entidades gerenciadas envolvidas neste *ensemble* são:

- LayerNetworkDomain;
- LinkConnection;
- NetworkAccessProfile;
- NetworkCTP;
- NetworkTTP;
- Subnetwork;
- TopologicalLink;
- Trail;
- VcLinkEnd;
- VpLinkEnd.

O diagrama de seqüência do UML apresentado na Figura 65 representa a interação destas entidades gerenciadas durante a criação de um enlace entre sub-redes da camada VC que necessita de um trail servidor da camada VP. Esta operação de criação pressupõe que as sub-redes de ambas camadas já existam, cada qual com os pontos de terminação *NetworkCTP* disponíveis e fornecidos pelo sistema de gerenciamento.

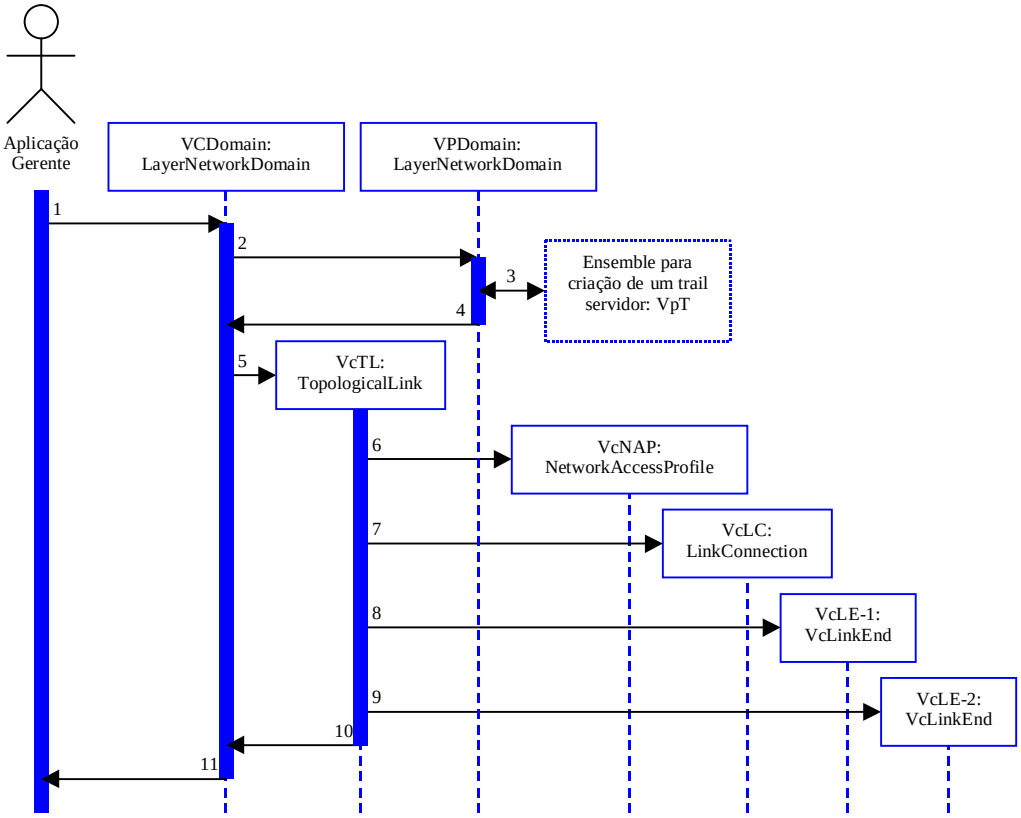

Figura 65: Diagrama de seqüência de criação de um enlace entre sub-redes através de um trail

Cenário

O processo de criação de uma conexão de enlace entre sub-redes contida da camada VC através de um trail servidor da camada VP está descrito na Figura 65. Os passos componentes deste processo são:

1. A aplicação de gerência pede ao objeto *VCDomain* que crie uma conexão de enlace entre as sub-redes *VcSN-1* e *VcSN-2*, utilizando os seus pontos de terminação respectivos *VcNCTP-1* e *VcNCTP-2* e especificando as características do tráfego que deve suportar, através de um trail da camada servidora VP utilizando os pontos de terminação *VpNCTP-1* e *VpNCTP-2* da sub-rede *VpSN-3*;
2. O objeto *VCDomain* pede ao objeto *VPDomain* que crie um trail servidor utilizando os pontos de terminação *VpNCTP-1* e *VpNCTP-2* da sub-rede *VpSN-3*, especificando as características do tráfego que deve suportar;
3. Cria-se o trail denominada *VpT* através do *ensemble* 4.3.10 (Criação de um Trail Servidor) passando os pontos de terminação *VpNCTP-1* e *VpNCTP-2* da sub-rede *VpSN-3* e as características do tráfego requeridas pela conexão;
4. O objeto *VPDomain* retorna ao objeto *VCDomain* o resultado da criação do trail servidor *VpT* com os correspondentes pontos de terminação de trail *VpNTTP-1* e *VpNTTP-2*;

5. O objeto *VCDomain* cria uma instância da classe *TopologicalLink*, denominada *VcTL*, com referências às sub-redes *VcSN-1* e *VcSN-2*, e pede que crie uma conexão de enlace entre as sub-redes utilizando os seus pontos de terminação respectivos *VcNCTP-1* e *VcNCTP-2*, especificando as características do tráfego que deve suportar;
6. O objeto *VcTL* cria uma instância da classe *NetworkAccessProfile*, denominada *VcNAP*, contendo as características do tráfego a ser suportado pela conexão;
7. O objeto *VcTL* cria uma instância da classe *LinkConnection*, denominada *VcLC*, com uma referência aos objetos *VcNCTP-1* e *VcNCTP-2*, e a inclui na sua lista de conexões de enlace;
8. O objeto *VcTL* cria uma instância da classe *VcLinkEnd*, denominada *VcLE-1*, com referências aos objetos *VcNAP*, *VcSN-1*, *VcNCTP-1* e o correspondente *VpNTTP-1*, e a inclui na sua lista de pontos de terminação de conexões de enlaces lógicos;
9. O objeto *VcTL* cria uma segunda instância da classe *VcLinkEnd*, denominada *VcLE-2*, com referências aos objetos *VcNAP*, *VcSN-2*, *VcNCTP-2* e o correspondente *VpNTTP-2*, e a inclui na sua lista de pontos de terminação de conexões de enlaces lógicos;
10. O objeto *VcTL* retorna para a o objeto *VCDomain* o resultado da criação dos elementos da conexão de enlace através do trail servidor da camada VP;
11. O objeto *VCDomain* retorna para a aplicação de gerência o resultado da criação da conexão de enlace.

4.4. Conclusão

A metodologia de implementação do modelo de informação de gerência de rede para redes ATM foi descrito através dos diagramas de seqüência do UML para representar a interação dinâmica dos objetos descritos pelos diagramas de classe. A representação destes diagramas de seqüência seguiu uma metodologia de especificação que se utiliza de uma estrutura de documentação padrão denominada *ensemble*, a qual foi definida pelo TM Fórum e cuja utilização foi simplificada pelo ATM Fórum.

O processo de especificação através desta metodologia requer que se definam cenários bem característicos de forma que se obtenham situações de operação o mais abrangentes possíveis, procurando-se assim cobrir a maior parte das interações entre as entidades gerenciadas do modelo de informação.

Este capítulo apresentou uma série de cenários, na forma de *ensembles*, que procuraram cobrir as principais funções de gerência e as funcionalidades básicas envolvidas nos processos de criação de todos os elementos constituintes da arquitetura da tecnologia ATM modelados pelas classes que compõe o seu modelo de informação.

5. Modelo de Informação de Rede ATM para o JMX

5.1. Introdução

O modelo de informação de gerência do nível de rede para redes ATM foi especificado utilizando-se a linguagem de modelagem de informação UML, tornando-o independente da plataforma em que será implementado. Os diagramas de classe deste modelo de informação representam a informação sintática dos objetos a serem instanciados.

Para o desenvolvimento de aplicações de gerenciamento que sejam dinâmicas, flexíveis e portáteis, plataformas de desenvolvimento baseadas na linguagem Java da Sun têm se mostrado ideais [Barillaud 1997].

A Sun desenvolveu uma especificação que estende a linguagem Java especificamente para suportar o desenvolvimento de aplicações de gerência de sistemas. As extensões de gerência Java (JMX) [JMX 1999] definem a arquitetura, os padrões de especificação, as APIs (Application Program Interfaces) e serviços para aplicações de gerenciamento de redes baseadas na tecnologia Java.

O JMX provê aos desenvolvedores suporte para a criação de agentes Java e para a implementação de aplicações distribuídas de gerência, assim como APIs de integração destas soluções com tecnologias de gerência padrões atualmente utilizados pelo mercado.

As APIs de integração com outras tecnologias de gerência permitem aos desenvolvedores de aplicações de gerenciamento abstrair a complexidade envolvida em cada uma destas tecnologias. Algumas destas APIs de integração já foram criadas pela Sun:

- API de gerência SNMP (já está especificada [JMXSNMP 1999]);
- API de gerência CIM/WBEM (Common Information Model/Web-Based Enterprise Management) (já está especificada [JMXWBEM 1999]);
- API de gerência CORBA (em fase final de especificação);
- API de gerência TMN.

A tecnologia JMX se baseia em um objeto gerenciável servidor que atua como uma agente de gerenciamento e que pode ser executado em qualquer dispositivo que esteja habilitado para a linguagem Java. O JMX especifica uma forma padrão de, através deste objeto gerenciável servidor, habilitar qualquer dispositivo, serviço ou aplicação Java a capacidade de tornar-se gerenciável.

Qualquer serviço de gerência JMX é um módulo independente (componente) que pode ser carregado no agente de acordo com a necessidade. Esta arquitetura de gerência baseada em componentes implica que soluções desenvolvidas de acordo com o JMX são escalonáveis de acordo com as

necessidades de utilização do agente e dos requisitos de gerenciamento do sistema. Portanto, todos estes serviços podem ser carregados, descarregados ou atualizados dinamicamente pelo sistema.

Agentes JMX são capazes de serem gerenciados por navegadores HTML (Hypertext Markup Language) ou por protocolos de gerência tais como SNMP e WBEM. A especificação do JMX inclui a definição da API de um gerente SNMP e também de um cliente WBEM.

Para implementar o modelo de informação desenvolvido neste trabalho utilizando a plataforma JMX é necessário que se faça um mapeamento dos diagramas de classe do modelo descrito em UML para as interfaces requeridas na especificação do JMX. O conjunto de interfaces é que compõe a MIB do agente JMX. O mapeamento está detalhado na forma de uma metodologia de especificação, o que tornou o processo genérico para qualquer modelo de informação.

Para implementar os MBeans requeridos pelo agente JMX, desenvolveu-se uma metodologia de implementação das classes (MBeans) componentes do modelo de informação a partir das suas interfaces, de forma que o processo se torne genérico para qualquer modelo de informação que se queira implementar de acordo com a especificação do JMX.

A seqüência com que os MBeans devem ser instanciados e a forma com que eles devem se comportar e interagir durante o processo de gerenciamento do sistema estão especificadas na metodologia de implementação descrita pelos diagramas de seqüência do modelo de informação em UML.

5.2. A Arquitetura JMX da Sun

5.2.1. A Estrutura do JMX

A arquitetura do JMX é composta de três níveis: nível gerente, nível agente e nível instrumentação. Esta composição da arquitetura de um sistema de gerenciamento baseado no JMX é semelhante à proposta pelo modelo OSI de gerenciamento de objetos utilizado pelo TMN. Esta semelhança pode ser constatada comparando-se a Figura 5, que apresenta a interação dos componentes do modelo OSI, com a Figura 66, que apresenta a forma com que os componentes da arquitetura JMX se relacionam entre si.

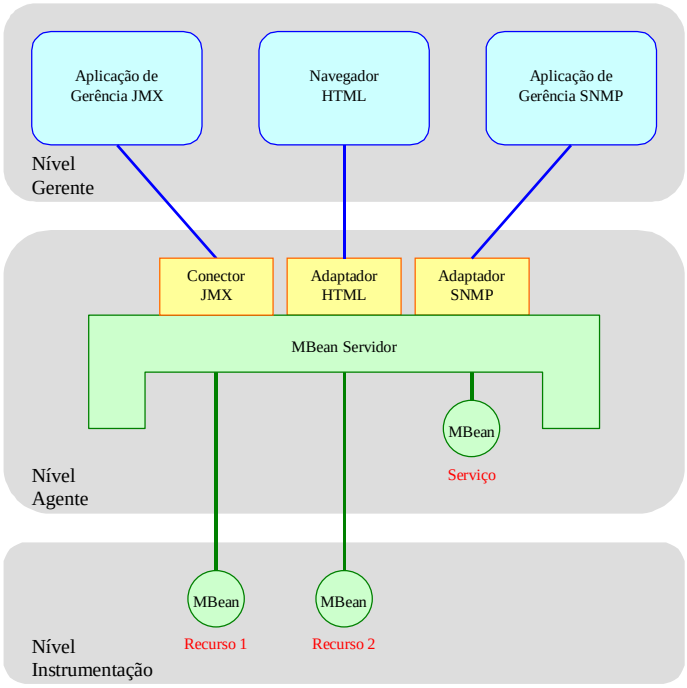

Figura 66: Relacionamento entre os componentes da arquitetura do JMX

Nível Instrumentação

O nível instrumentação corresponde a uma especificação da forma com que os recursos gerenciáveis devem ser implementados usando os JavaBeans. Os recursos gerenciáveis podem ser uma aplicação, um serviço, um dispositivo ou um usuário, desde que implementados de acordo com a especificação do JMX.

A instrumentação de um determinado recurso é provida por um ou mais Java Beans de gerência, os MBeans, que são gerenciados através do nível agente. Os MBeans são projetados para serem flexíveis, simples e fáceis de serem implementados. Desenvolvedores de aplicações, de serviços ou até mesmo de dispositivos, através desta metodologia podem tornar seus produtos gerenciáveis de uma forma padrão sem que para isto tenham que investir e dominar sistemas de gerenciamento complexos.

O nível instrumentação também define um mecanismo de notificação, o qual permite que os MBeans gerem e propaguem eventos de notificações para componentes de outros níveis.

Nível Agente

O nível agente é uma especificação da implementação dos agentes JMX. Os agentes JMX são responsáveis pelo controle direto dos recursos e por tornarem estes recursos disponíveis para as

aplicações de gerenciamento remotas. Este nível usa o nível instrumentação para definir os recursos e serviços que o agente JMX fornecerá ao sistema de gerenciamento. Um agente JMX consiste de um MBean servidor, um conjunto de serviços básicos de gerência e pelo menos um adaptador de comunicação ou um conector JMX.

As aplicações de gerência acessam MBeans de um agente através de um adaptador de comunicação ou de um conector JMX e usam os seus serviços. Por outro lado, um agente JMX não precisa ter conhecimento da aplicação de gerência que o está gerenciando.

A arquitetura do JMX permite que uma aplicação de gerência execute as seguintes operações sobre um agente JMX:

- gerencie MBeans já disponíveis, recuperando ou alterando os valores dos seus atributos;
- receba notificações emitidas por MBeans;
- carregue novos MBeans para que possam ser instanciados e registrados;
- instancie e registre novos MBeans já carregados.

Agentes JMX são implementados por desenvolvedores de sistemas de gerenciamento os quais podem construir os seus produtos de uma forma padrão sem que para isto tenham que entender a semântica dos recursos gerenciados ou as funcionalidades das aplicações de gerência envolvidas no sistema.

Nível Gerente

O nível gerente corresponde a uma especificação da implementação dos gerentes JMX. Este nível define as interfaces de gerência e os componentes que podem operar sobre os agentes JMX ou, até mesmo, sobre hierarquias destes agentes.

A combinação do nível gerente com os níveis agente e instrumentação provê uma arquitetura completa para o projeto e o desenvolvimento de soluções de gerenciamento de sistemas. A tecnologia JMX traz várias facilidades para estas soluções: portabilidade, desenvolvimento de funcionalidades de gerência sob demanda, serviços de gerenciamento dinâmicos e móveis, segurança.

APIs de Protocolos de Gerência Adicionais

As APIs de protocolos de gerência adicionais, que compõe a interface de comunicação entre o nível agente e o nível gerente, provêem uma especificação para permitir que agentes JMX possam interagir com ambientes de gerenciamento já disponíveis no mercado. Já estão completamente

especificadas pela Sun as interfaces com sistemas de gerenciamento que se utilizam dos seguintes protocolos padrões de gerência:

- SNMP [JMXSNMP 1999]: representa e manipula objetos SNMP como classes Java em agentes JMX para que estes possam ser gerenciados por aplicações de gerência SNMP;
- CIM/WBEM [JMXWBEM 1999]: representa e manipula objetos CIM como classes Java em agentes JMX para que estes possam ser gerenciados por aplicações de gerência WBEM.

Desenvolvedores de aplicações de gerência podem utilizar-se destas APIs para interagir com ambientes de gerenciamento baseados nestes protocolos padrões, possivelmente encapsulando esta interação em um recurso gerenciável JMX. Estas APIs Java auxiliam desenvolvedores de sistemas de gerência a construir aplicações independentes de plataforma para os protocolos padrões de gerenciamento mais comuns da indústria. Desta forma, novas soluções de gerência podem integrar-se com infra-estruturas já existentes e sistemas de gerenciamento já existentes podem tirar vantagem de aplicações de gerenciamento baseadas na tecnologia Java.

5.2.2. Componentes do Nível Instrumentação

Os componentes chave do nível instrumentação são os MBeans, JavaBeans de gerenciamento, e o modelo de notificação.

MBeans

Os objetos Java que implementam as características e funcionalidades de um determinado recurso são denominados JavaBeans de gerenciamento, ou abreviadamente, MBeans. Os MBeans devem seguir rigorosamente os padrões de especificação e as interfaces definidas pela especificação do JMX, de forma que se possam gerenciar os recursos de uma maneira padrão por qualquer agente JMX.

Um MBean é uma classes Java não abstrata que contém:

- um construtor público;
- a implementação da interface MBean correspondente ou a interface de um MBean dinâmico (*DynamicMBean*);
- opcionalmente, a implementação da interface *NotificationBroadcaster*.

As classes que implementam a sua própria interface MBean são denominadas MBeans padrões. O MBean padrão é o tipo mais simples de MBean disponível na especificação do JMX.

As classes que implementam a interface *DynamicMBean* são denominada MBeans dinâmicos. Estas classes permitem que algumas das suas características e funcionalidades internas sejam controladas em tempo de execução.

Os agentes JMX permitem que se manipulem ambos os tipos de MBeans de forma transparente, ou seja, independente do tipo das aplicações de gerenciamento. Desta forma, o tipo de interface que o MBean implementa determina como ele será desenvolvido e não como ele será gerenciado.

Quando se desenvolve uma classe Java através de uma interface MBean padrão, os recursos a serem gerenciados são acessados diretamente através dos seus atributos e operações. Os atributos são entidades internas disponibilizadas através de métodos de recuperação (*get*) e de alteração (*set*). As operações são os outros métodos da classe que estão disponíveis para a aplicação de gerenciamento. Todos estes métodos são definidos estaticamente na interface do MBean e são visíveis (externalizados) aos agentes através da característica de introspecção do Java. Esta é a forma mais direta de se desenvolver o gerenciamento de um novo recurso.

Quando se desenvolve uma classe Java através de uma interface MBean dinâmico, os atributos e as operações são indiretamente acessados através de chamadas de métodos. Ao invés da introspecção, os agentes JMX devem chamar um método que encontre o nome e a natureza dos atributos e das operações do MBean. Quando o agente JMX for acessar um atributo ou operação, ele inicialmente chama um método genérico cujos argumentos contém o nome do método ou da operação. Através dos MBeans dinâmicos é possível rapidamente gerenciar recursos ou aplicações já existentes, desde que estes sigam a especificação do JMX.

Todos os métodos de um MBean devem ser implementados de forma que a classe do MBean possa ser instanciada (não pode ser uma classe abstrata) e que esta instância possa ser gerenciada por uma aplicação de gerenciamento.

A classes Java de um MBean devem ter pelo menos um construtor público, podendo ter sido declarado explicitamente ou ter sido criado automaticamente pelo compilador (quando nenhum construtor tiver sido definido). No exemplo de código de definição de uma classe a seguir é apresentada uma classe que pode ser um MBean:

```
public class Simples {
    private Integer estado = new Integer(0);

    protected Simples() {                    // Construtor acessível somente pelas sub-classes
    }
    public Simples(Integer e) {              // Construtor público: esta classe poderia ser um MBean
        estado = e;
    }
    ...
}
```

MBeans Padrões

Um MBean padrão define explicitamente a interface de gerenciamento para permitir a gerência de um recurso através do agente JMX. A interface de um MBean é constituída pelos métodos que permitem a leitura e gravação dos seus atributos e pelas operações que podem ser invocadas.

Os MBeans padrões baseiam-se em regras de nomeação que devem ser observadas ao se definir a interface dos seus objetos Java. Estas regras de nomeação definem os atributos e operações de acordo com o modelo de componentes JavaBeans da tecnologia Java.

Os atributos de um MBean são os seus campos ou propriedades os quais encontram-se especificados na sua interface através de métodos de leitura (prefixo *get*) e/ou gravação (prefixo *set*) dos seus conteúdos. Desta forma:

- um atributo somente de leitura é identificado na especificação na interface do MBean por um método *getAtributo()*;
- um atributo somente de gravação é identificado na especificação na interface do MBean por um método *setAtributo()*;
- um atributo de leitura e de gravação é identificado na especificação na interface do MBean por um método *getAtributo()* e por um método *setAtributo().*

As operações de um MBean são os métodos Java especificados na sua interface e implementados na sua classe. Qualquer método da interface do MBean que não identifique um atributo (não está de acordo com o padrão de especificação de um atributo) é considerado uma operação.

O processo de inspeção da interface de um MBean e de verificação da aplicação das regras de nomeação é denominado de introspecção. O agente JMX usa a introspecção para verificar se os métodos e superclasses de uma classe estão de acordo com os padrões de especificação de um MBean, reconhecendo desta forma os nomes tanto dos seus atributos quanto das suas operações.

A classe Java de um MBean deve implementar a interface Java que a define, a qual especifica a assinatura completa dos métodos que compõe a classe e que são externalizados para gerenciamento (é o caso dos métodos públicos contidos na interface). O nome da interface de um MBean Java é formado pelo acréscimo do sufixo *MBean* ao nome da sua classe.

Uma classe MBean pode definir quaisquer outros métodos, públicos ou protegidos, que não apareçam na especificação da interface do MBean, a qual especifica tanto seus próprios métodos explicitamente quanto os herdados de interfaces pais implicitamente.

A seguir está sendo apresentado um exemplo de definição da interface em Java e em UML (Figura 67) de um MBean denominado *MinhaClasse*:

```
public interface MinhaClasseMBean {
    public Integer getEstado();
    public void setEstado(Integer e);
    public void reset();
}
```

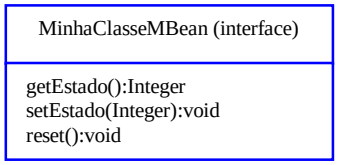

Figura 67: Definição da interface *MinhaClasseMBean*

A seguir é apresentado o código da classe do MBean exemplo denominado *MinhaClasse*:

```
public class MinhaClasse implements MinhaClasseMBean {
    private Integer estado = null;
    private String oculto = null;

    public Integer getEstado() {
        return(estado);
    }
    public void setEstado(Integer e) {
        estado = e;
    }
    public String getOculto() {
        return(oculto);
    }
    public void setOculto(String o) {
        oculto = o;
    }
    public void reset() {
        estado = null;
        oculto = null;
    }
}
```

No exemplo apresentado acima, os métodos *getOculto* e *setOculto* da classe do MBean *MinhaClasse* não farão parte da interface de gerenciamento porque não aparecem na especificação da sua interface.

MBeans Dinâmicos

Os MBeans padrões são ideais para serem utilizados em estruturas de dados de gerenciamento bem definidas e estáveis pois provêem a forma mais fácil e rápida para implementar o gerenciamento de recursos através do JMX. Por outro lado, quando a estrutura dos dados de gerenciamento são passíveis de atualizações constantes, o nível instrumentação disponibiliza uma forma mais flexível de implementar o gerenciamento de recursos através de um mecanismo dinâmico e em tempo de

execução de tratamento dos recursos gerenciados. Os MBeans dinâmicos é que acrescentam à especificação do JMX esta capacidade de adaptação dinâmica disponibilizando uma forma alternativa de tratamento dos recursos gerenciados.

Os MBeans dinâmicos são recursos gerenciados que são monitorados através de uma interface predefinida a qual externaliza os atributos e as operações em tempo de execução: a interface *DynamicMBean*. Ao invés de externalizá-los diretamente através dos nomes dos métodos, os MBeans dinâmicos implementam um método que retorna as assinaturas (identificações unívocas) de todos os atributos e operações disponibilizadas pelas suas interfaces.

Como os nomes e operações são determinados dinamicamente, os MBeans dinâmicos provêem uma grande flexibilidade quando estão gerenciando recursos já disponíveis que não foram implementados pelos padrões de especificação de interface de um MBean padrão. Isto acontece porque, ao invés da introspecção, os agentes JMX chamam o método dos MBeans que retornam as suas especificações de interface.

Quando gerenciados através de um agente JMX, os MBeans dinâmicos oferecem a mesma capacidade de gerenciamento disponibilizada pelos MBeans padrões, o que implica que aplicações de gerência podem manipular quaisquer MBeans exatamente da mesma forma independentemente do seu tipo.

A interface *DynamicMBean* está definida em UML na Figura 68. O método *getMBeanInfo* retorna toda a estrutura da definição da interface do MBean dinâmico que o implementa. A descrição detalhada desta interface e de cada um dos seus métodos é apresentada na especificação do JMX ([JMX 1999]).

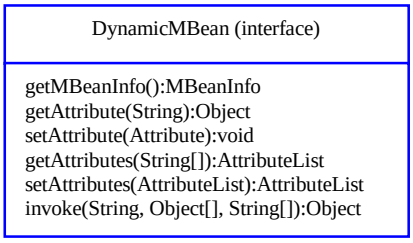

Figura 68: Definição da interface *DynamicMBean*

A identificação do tipo de um MBean segue uma regra geral em que o tipo de interface de gerência é definido pela classe MBean ou por uma de suas classes pais a qual implementa ou sua própria interface MBean ou a interface *DynamicMBean*, não podendo implementar ambas interfaces ao mesmo tempo.

Modelo de Notificação

A interface de gerência de um MBean permite que o agente JMX monitore e tenha o seu controle através de operações de configuração. Esta é somente uma parte da funcionalidade requerida para o gerenciamento de sistemas complexos e distribuídos, pois, normalmente, as aplicações de gerência precisam reagir a alterações de estado, de valor de atributo ou de alguma condição específica que ocorra nos recursos que estão sendo gerenciados. Estas alterações são reportadas pelos agentes na forma de notificações espontâneas.

O modelo de notificação do JMX permite que aplicações que necessitem receber notificações espontâneas se registrem em um MBean de broadcast, que faz parte do modelo de notificação do JMX. Este modelo é composto pelos seguintes componentes:

- um tipo de evento genérico, denominado *Notification*, que representa qualquer tipo de notificação de gerência e é composto pela:
  - identificação do tipo da notificação;
  - identificação da instância;
  - identificação do instante de ocorrência;
  - mensagem da notificação;
  - dados adicionais a serem passados do gerador para o consumidor da notificação;
- a interface *NotificationListener*, que precisa ser implementada pelos objetos que desejam receber notificações geradas por MBeans (consumidores de notificações);
- a interface *NotificationFilter*, que precisa ser implementada pelos objetos que atuarão como filtros de notificações;
- a interface *NotificationBroadcaster*, que precisa ser implementada pelos MBeans que estiverem monitorando recursos que possam gerar informações espontâneas (geradores de notificações).

O modelo de notificação do JMX está representado na Figura 69, a qual apresenta os passos envolvidos no processo de registro e de envio de uma notificação desde um MBean gerador até um MBean consumidor.

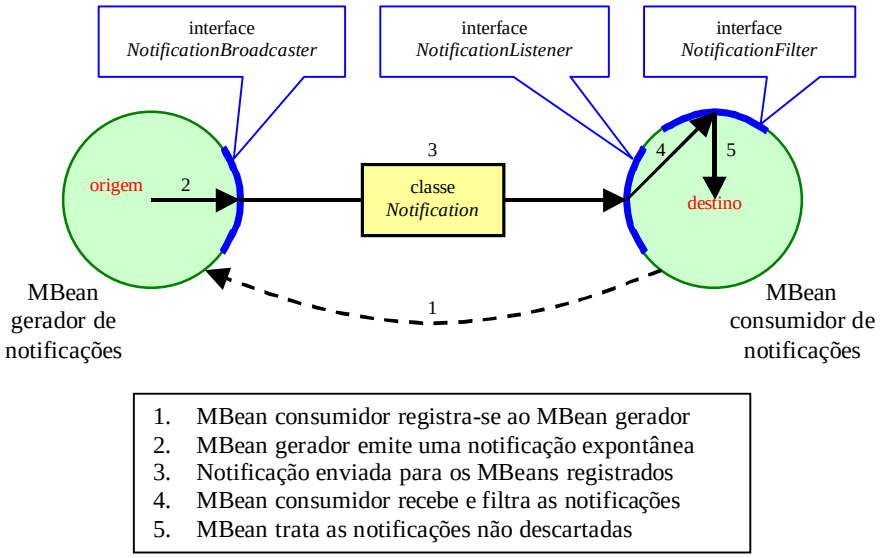

Figura 69: Componentes do modelo de notificação do JMX

Usando tipos de eventos genéricos, o modelo de notificação permite que qualquer consumidor de notificações receba todos os tipos de eventos de um gerador de notificações. O filtro de notificações é utilizado por um consumidor para especificar quais os tipos de notificações que ele precisa.

Qualquer tipo de MBean, padrão ou dinâmico, pode ser tanto um consumidor de notificações, um gerador de notificações ou ambos ao mesmo tempo.

5.2.3. Componentes do Nível Agente

Um agente JMX é uma entidade gerenciável Java que atua como intermediário entre os MBeans e as aplicações de gerência. Um agente JMX é constituído de um MBean servidor, um conjunto de MBeans representando os recursos gerenciados, um número mínimo de serviços de gerenciamento implementados como MBeans e pelo menos um adaptador de protocolo ou um conector JMX (veja a Figura 66).

Os conectores JMX são utilizados para conectar um agente a uma aplicação de gerência desenvolvida usando a plataforma do JMX. Por outro lado, os adaptadores de protocolo são utilizados para conectar um agente a aplicação de gerência desenvolvida sobre um protocolo de gerenciamento específico.

Os componentes chave do nível agente são o MBean servidor, responsável pelo registro dos objetos do nível instrumentação, e os serviços padrões, responsáveis pelas funcionalidades básicas do agente.

MBean Servidor

O MBean servidor é o núcleo do agente JMX e é responsável pelo registro dos MBeans no agente e por prover os serviços necessários à sua manipulação. Todas as operações de gerência executadas sobre os MBeans são feitas através das interfaces do MBean servidor (*MBeanServer*).

A Figura 70 representa como uma operação de gerenciamento é propagada desde uma aplicação de gerência até um MBean registrado no agente JMX. O exemplo ilustra a propagação de um método para recuperar o conteúdo do atributo *estado* de um MBean desde uma aplicação de gerência, tanto através de uma interface de invocação estática quanto através de uma interface de invocação dinâmica.

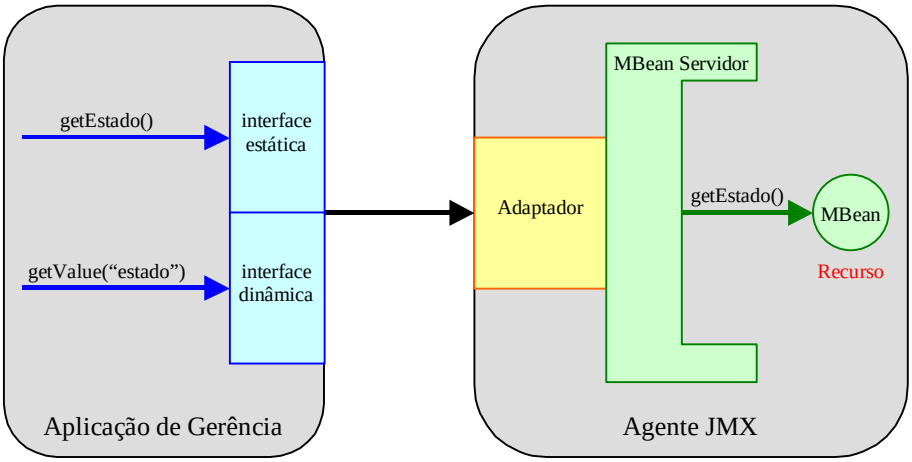

Figura 70: Propagação de uma operação sobre um MBean

Os MBeans podem ser registrados em um MBean servidor tanto por aplicações de gerência quanto por outros MBeans. Os seguintes tipos de MBeans podem ser registrados:

- MBeans que representem recursos gerenciados, do tipo aplicações, sistemas ou recursos de rede, e que tenham sido desenvolvidos de acordo com a especificação do JMX;
- MBeans que acrescentem funcionalidades de gerência ao agente JMX;
- MBeans de implementação dos adaptadores de protocolo ou dos conectores JMX.

Consultas e Filtragem

As expressões de consulta são utilizadas na forma de filtros e permitem que se recupere MBeans de acordo com o valor de seus atributos em um MBean servidor. Estas expressões de consulta são compostas por um escopo e por um filtro.

As classes componentes deste serviço estão enumeradas e descritas na especificação do JMX ([JMX 1999])

Carregamento Dinâmico

O serviço de carregamento dinâmico é representado pelo serviço MLet (programa Java de gerenciamento para ser inserido em uma página da Internet) o qual é utilizado para instanciar MBeans obtidos de uma URL (Universal Resource Locator) remota.

Um serviço MLet permite que se instancie e registre em um MBean servidor um ou mais MBeans vindos através da rede. Isto é feito através do carregamento de um arquivo texto contento a especificação do MLet o qual contém as informações de cada um dos MBeans a serem carregados desde o local especificado pela URL.

A Figura 71 descreve a operação de carregamento, registro e instanciação de um MBean desde uma URL remota. Neste exemplo, a classe do MBean *obj1* está disponível na máquina local onde o agente JMX está sendo executado, enquanto que o MBean *obj2* foi carregado por um navegador Internet, fazendo o papel de uma aplicação de gerência, através de uma página HTML que contém a descrição e a fonte do MBean a ser instanciado no agente.

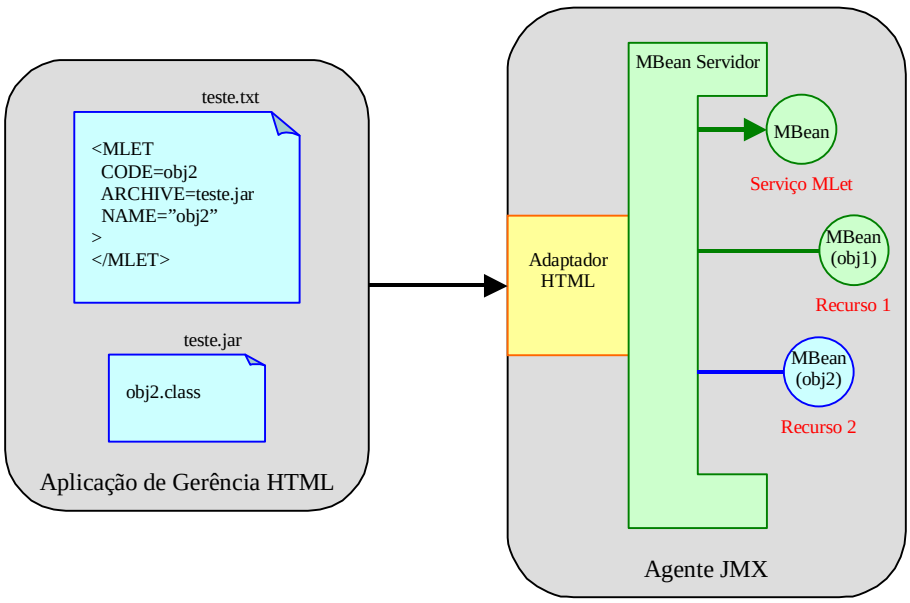

Figura 71: Operação do serviço MLet

O serviço MLet é implementado como um MBean e registrado no MBean servidor de forma que ele pode ser utilizado tanto por aplicações de gerenciamento quanto por outros MBeans registrados no agente JMX.

Monitores

Através da utilização de MBeans de monitoração, um MBean pode ter seus atributos monitorados a intervalos específicos definidos por um período de granularidade. Um tipo específico de notificação é emitido pelo MBean monitor quando o valor da medida satisfaz um conjunto de condições preestabelecidas durante a sua inicialização.

As classes componentes deste serviço estão enumeradas e descritas na especificação do JMX ([JMX 1999])

Temporizadores

O serviço de temporização permite que se enviem notificações em determinadas datas e horários. As notificações são enviadas para todos os MBeans registrados para recebê-las. A classe *Timer* que implementa este serviço gerencia uma lista de notificações datadas. Um método desta classe permite que se adicionem ou removam notificações desta lista.

O serviço de temporização pode ser interrompido e reiniciado, através de métodos disponíveis especificamente para este fim, e pode gerenciar notificações de duas formas distintas:

- notificações que ocorrem apenas uma vez;
- notificações que se repetem com uma certa periodicidade e/ou um determinado número de ocorrências.

5.2.4. Componentes do Nível Gerente

Os componentes do nível gerente devem cooperar entre si ao longo da rede para prover funcionalidades de gerenciamento distribuídas e escalonáveis.

Estes componentes de um sistema de gerência JMX devem:

- Prover uma interface para que aplicações de gerenciamento interajam com um agente JMX através de conectores JMX;
- Distribuir informações de gerenciamento para agentes JMX de vários níveis;
- Consolidar informações de gerenciamento provenientes de agentes JMX de vários níveis em perspectivas lógicas específicas para que se tornem relevantes para as operações e expectativas de um determinado usuário;
- Prover mecanismos que garantam a segurança do sistema de gerenciamento como um todo.

5.2.5. Componentes das APIs de Protocolos de Gerência Adicionais

Os componentes adicionais das APIs de protocolos de gerência já especificadas pela Sun são a API de gerência SNMP e a API de gerência CIM/WBEM.

5.3. Metodologia de Especificação das Interfaces para o JMX

Para especificar as interfaces requeridas pelo JMX a partir dos diagramas de classe do UML desenvolvidos neste trabalho, devem-se considerar determinadas informações do diagrama de classes. Estas informações a serem consideradas estão descritas nas seções a seguir.

5.3.1. Nome da Interface

Considere o exemplo, na Figura 72, de uma classe definida em UML.

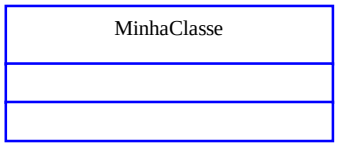

Figura 72: Exemplo de uma classe UML

De acordo com a especificação do JMX, o nome de uma interface de uma classe é o nome da classes acrescido do sufixo *MBean*, como está apresentado no código exemplo a seguir:

```
public interface MinhaClasseMBean {
}
```

5.3.2. Classes com Herança

Considere o exemplo, na Figura 73, de uma classe definida em UML que herda informações de uma interface pai.

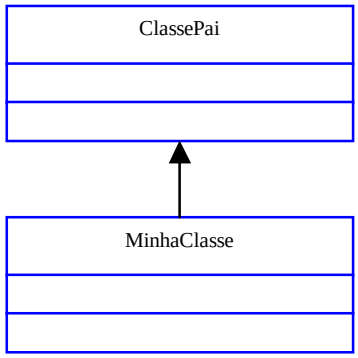

Figura 73: Exemplo de uma classe UML com herança

A seguir é apresentado o código da interface do MBean exemplo que herda informações de uma interface pai:

```
public interface MinhaClasseMBean extends ClassePaiMBean {
}
```

5.3.3. Classes com Atributos

Considere o exemplo, na Figura 74, de uma classe definida em UML que contém atributos com permissões de leitura e de gravação específicos.

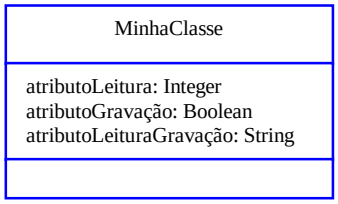

Figura 74: Exemplo de uma classe UML com atributos

De acordo com a especificação do JMX, os atributos de um MBean encontram-se especificados na sua interface através de métodos de leitura (prefixo *get*) e/ou gravação (prefixo *set*) dos seus conteúdos, como está apresentado no código exemplo a seguir:

```
public interface MinhaClasseMBean {
    public Integer getAtributoLeitura();
    public void setAtributoGravação(Boolean x);
    public String getAtributoLeituraGravação();
    public void setAtributoLeituraGravação(Str ing x);
}
```

5.3.4. Classes com Métodos

Considere o exemplo, na Figura 75, de uma classe definida em UML que contém métodos.

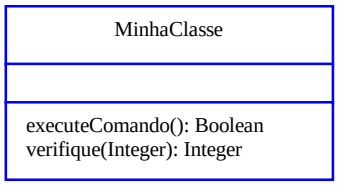

Figura 75: Exemplo de uma classe UML com méto dos

De acordo com a especificação do JMX, os métodos são mapeados diretamente na sua interface, como está apresentado no código exemplo a seguir:

```
public interface MinhaClasseMBean {
    public Boolean executeComando();
    public Integer verifique(Integer x);
}
```

5.3.5. Classes com Métodos de Envio de Eventos

Considere o exemplo, na Figura 76, de uma classe definida em UML que contém métodos de envio de eventos.

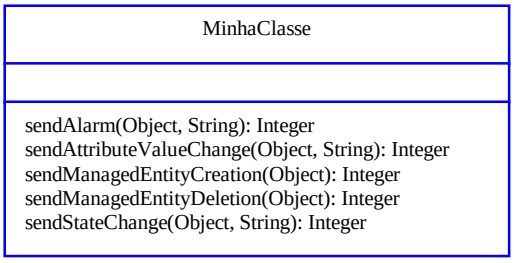

Figura 76: Exemplo de uma classe UML com métodos de envio de eventos

De acordo com a especificação do JMX, as classes que geram e enviam eventos devem implementar a interface *NotificationBroadcaster* (veja Figura 69) e não precisam externalizar os métodos na interface do MBean, pois a sua geração é interna e espontânea. Por isso esses métodos não são mapeados diretamente na sua interface, como está apresentado no código exemplo a seguir:

```
public interface MinhaClasseMBean {
}
```

5.4. Especificação das Interfaces JMX do Modelo de Informação

De acordo com a metodologia de especificação das interfaces dos MBeans utilizados pelo JMX, uma vez definido o modelo de informação de rede ATM para a gerência de configuração e de falhas em UML, obtém-se as seguintes interfaces componentes da MIB do agente JMX.

5.4.1. AlarmRecord

A interface do MBean *AlarmRecord* é definida como:

```
public interface AlarmRecordMBean extends LogRecordMBean {
    public String getGenericTroubleDescription();
    public Integer[] getSpecificProblems();
    public SeverityEnum getSeverity();
    public Boolean getBackupStatus();
    public Object getBackupEntity();
    public String getAdditionalText();
    public String[] getProposedRepairActions();
}
```

5.4.2. AlarmSeverityAssignmentProfile

A interface do MBean *AlarmSeverityAssignmentProfile* é definida como:

```
public interface AlarmSeverityAssignmentProfileMBean {
    public String getManagedEntityId();
    public AlarmSeverity[] getAlarmSeverityAssignmentList();
    public void setAlarmSeverityAssignmentList (AlarmSeverity[] as);
}
```

5.4.3. AttributeValueChangeRecord

A interface do MBean *AttributeValueChangeRecord* é definida como:

```
public interface AttributeValueChangeRecordMBean extends LogRecordMBean {
    public String getAttributeType();
    public String getOldAttributeValue();
    public String getNewAttributeValue();
}
```

5.4.4. EventForwardingDiscriminator

O MBean *EventForwardingDiscriminator* não é implementado pois, de acordo com a especificação do JMX, a funcionalidade de filtragem e envio de eventos do agente JMX para aplicações de gerência é um serviço interno suprido por MBeans que implementam as interfaces *NotificationListener* e *NotificationFilter* (veja o modelo de notificação do JMX descrito na Figura 69).

5.4.5. LayerNetworkDomain

A interface do MBean *LayerNetworkDomain* é definida como:

```
public interface LayerNetworkDomainMBean {
    public SignalEnum getSignalIdentification();
    public String getUserLabel();
    public void setUserLabel(String ul);

    public NetworkTTP[] queryDelimitingNetworkTTPs();
    public Trail[] queryExistingTrails();
    public Subnetwork[] queryComponentSubnetwork();
    public Trail setupTrail(NetworkTTP ttp1, NetworkTTP ttp2);
    public Trail setupTrail(Descriptor d);
    public TrailRequest setupTrailRequest(ActionInfo ai);
    public NetworkTTP addTPsToMultipointTrail(Trail t, NetworkTTP ttp);
    public Trail releaseTrail(NetworkTTP[] ttps);
    public Object makeExternalLinkEnd();
    public Object removeExternalLinkEnd();
    public TopologicalLink setupTopologicalLink(LinkDetails ld);
    public TopologicalLink releaseTopologicalLink(TopologicalLink tl);
}
```

5.4.6. LinkConnection

A interface do MBean *LinkConnection* é definida como:

```
public interface LinkConnectionMBean {
    public String getLinkConnectionId();
    public SignalEnum getSignalIdentification();
    public DirectionalityEnum getDirectionality();
    public String getUserLabel();
    public void setUserLabel(String ul);
    public AvailabilityStatusEnum getAvailabilityStatus();
    public AdministrativeStateEnum getAdministrativeState();
    public void setAdministrativeState(AdministrativeStateEnum as);
    public Boolean getRetainedResource();
    public void setRetainedResource(Boolean rr);

    public TopologicalLink queryTopologicalLink();
    public NetworkCTP[] queryNetworkCTPs();
}
```

5.4.7. Log

A interface do MBean *Log* é definida como:

```
public interface LogMBean {
    public String getManagedEntityId();
    public AdministrativeStateEnum getAdministrativeState();
    public void setAdministrativeState(AdministrativeStateEnum as);
    public EventTypeEnum[] getLogRecordTypes();
    public LogFullActionEnum getLogFullAction();
    public void setLogFullAtion(LogFullActionEnum lfa);
    public OperationalStateEnum getOperationalState();
}
```

5.4.8. LogicalLinkTP

A interface do MBean *LogicalLinkTP* é definida como:

```
public interface LogicalLinkTPMBean {
    public String getLogicalLinkTPId();
    public SignalEnum getSignalIdentification();
    public Integer getEgressMaxAssignableBandwidth();
    public Integer getIngressMaxAssignableBandwidth();
    public Integer getEgressAvailableBandwidth();
    public Integer getIngressAvailableBandwidth();
    public IntegerRange getRangeIdentification();
    public void setRangeIdentification(IntegerRange ir);
    public String getUserLabel();
    public void setUserLabel(String ul);

    public TopologicalLink[] queryTopologicalLink();
    public Subnetwork[] querySubnetwork();
    public VcLinkEnd[] queryVcLinkEnd();
    public VcLinkEnd associateVcLinkEnd(VcLinkEnd vle);
    public VpLinkEnd[] queryVpLinkEnd();
    public VpLinkEnd associateVpLinkEnd(VpLinkEnd vle);
    public Sobnetwork[] traceLink(Subnetwork[] s);
}
```

5.4.9. LogRecord

A interface do MBean *LogRecord* é definida como:

```
public interface LogRecordMBean {
    public String getManagedEntityId();
    public Date getLoggingTime();
    public EventTypeEnum getManagedEntity();
}
```

5.4.10. ManagedEntityCreationRecord

A interface do MBean *ManagedEntityCreationRecord* é definida como:

```
public interface ManagedEntityCreationRecordMBean extends LogRecordMBean {
}
```

5.4.11. ManagedEntityDeletionRecord

A interface do MBean *ManagedEntityDeletionRecord* é definida como:

```
public interface ManagedEntityDeletionRecordMBean extends LogRecordMBean {
}
```

5.4.12. Network

A interface do MBean *Network* é definida como:

```
public interface NetworkMBean {
    public String getNetworkId();
}
```

5.4.13. NetworkAccessProfile

A interface do MBean *NetworkAccessProfile* é definida como:

```
public interface NetworkAccessProfileMBean {
    public String getNetworkAccessProfileId();
    public Integer getTotalEgressBandwidth();
    public void setTotalEgressBandwidth(Integer teb);
    public Integer getTotalIngressBandwidth();
    public void setTotalIngressBandwidth(Integer tib);
    public Integer getMaxActiveConnectionAllowed();
    public void setMaxActiveConnectionAllowed(Integer maca);
    public IntegerRange getRangeIdentification();
    public void setRangeIdentification(IntegerRange ri);
}
```

5.4.14. NetworkCTP

A interface do MBean *NetworkCTP* é definida como:

```
public interface NetworkCTPMBean {
    public String getCTPId();
    public SignalEnum getSignalIdentification();
    public VirtualId getVpVcIdValue();
    public String getUserLabel();
    public void setUserLabel(String ul);
    public Boolean getSegment Endpoint();
    public void setSegmentEndpoint(Boolean se);
    public Boolean getIngressTaggingIndicator();
    public void setIngressTaggingIndicator(Boolean iti);
    public Boolean getEgressTaggingIndicator();
    public void setEgressTaggingIndicator( Boolean eti);

    public NetworkTTP[] queryNetworkTTP();
    public NetworkTTP associateNetworkTTP(NetworkTTP ttp);
    public SubnetworkConnection[] querySubnetworkConnection();
    public Boolean loopbackTrail(LoopbackDetails ld);
}
```

5.4.15. NetworkTTP

A interface do MBean *NetworkTTP* é definida como:

```
public interface NetworkTTPMBean {
    public String getTTPId();
    public SignalEnum getSignalIdentification();
    public AvailabilityStatusEnum getAvailabilityStatus();

    public NetworkCTP[] queryNetworkCTP();
    public Trail[] queryTrail();
    public Boolean loopbackTrail(LoopbackDetails ld);
}
```

5.4.16. RoutingProfile

A interface do MBean *RoutingProfile* é definida como:

```
public interface RoutingProfileMBean {
    public String getRoutingProfileId();
    public ConnectionTypeEnum getConnectionTypeSupported();
    public void setConnectionTypeSupported(ConnectionTypeEnum cts);
    public RouteDescription[] getRouteDescriptionList();
    public void setRouteDescriptionList(RouteDescription[] rdl);
    public Integer getMaxHops();
    public void setMaxHops(Integer mh);

    public RoutingProfile setupRoutingProfile(ConnectionTypeEnum cts);
    public RoutingProfile setupRoutingProfile(RouteDescription[] rdl);
    public RoutingProfile setupRoutingProfile(Integer mh);
}
```

5.4.17. StateChangeRecord

A interface do MBean *StateChangeRecord* é definida como:

```
public interface StateChangeRecordMBean extends LogRecordMBean {
    public String getStateAttributeType();
    public String getOldStateAttributeValue();
    public String getNewStateAttributeValue();
}
```

5.4.18. Subnetwork

A interface do MBean *Subnetwork* é definida como:

```
public interface SubnetworkMBean {
    public String getSubnetworkId();
    public SignalEnum getSignalIdentification();
    public String getUserLabel();
    public void setUserLabel(String ul);
    public AvailabilityStatusEnum getAvailabilityStatus();
    public Object[] getSupportedByObjectList();
    public void setSupportedByObjectList(Object[] ol);

    public SubnetworkConnection[] querySubnetworkConnection();
    public Subnetwork[] querySubnetwork();
    public TopologicalLink[] queryTopologicalLink();
    public LinkEndAndTPList queryLinkEndAndTp();
    public SubnetworkConnection setupSubnetworkConnection(SubnetworkConnection sc,
NetworkCTP ctp1, NetworkCTP ctp2);
    public SubnetworkConnection setupSubnetworkConnection(SubnetworkConnection sc,
Descriptor d);
    public SubnetworkConnection modifySubnetworkConnection(SubnetworkConnection sc,
NetworkCTP ctp1, NetworkCTP ctp2);
    public SubnetworkConnection modifySubnetworkConnection(SubnetworkConnection sc,
Descriptor d);
    public SubnetworkConnection addTpToSubnetworkConnection(SubnetworkConnection sc,
NetworkCTP ctp);
    public SubnetworkConnection releaseSubnetworkConnection(NetworkCTP[] ctps);
}
```

5.4.19. SubnetworkConnection

A interface do MBean *SubnetworkConnection* é definida como:

```
public interface SubnetworkConnectionMBean {
    public String getSubnetworkConnectionId();
    public SignalEnum getSignalIdentification();
    public DirectionalityEnum getDirectionality();
    public AvailabilityStatusEnum getAvailabilityStatus();
    public AdministrativeStateEnum getAdministrativeState();
    public void setAdministrativeState(AdministrativeStateEnum as);
```

```
    public String getUserLabel();
    public void setUserLabel(String ul);
    public Boolean getRestorableIndicator();
    public void setRestorableIndicator(Boolean ri);
    public Boolean getRetainedResource();
    public void setRetainedResource(Boolean rr);
    public ProvisionTypeEnum getProvisionType();
    public void setProvisionType(ProvisionTypeEnum pt);

    public NetworkCTP[] queryNetworkCTP();
    public SubnetworkConnection[] querySubnetworkConnection();
    public LinkEndAndTPList traceConnection();
}
```

5.4.20. TopologicalLink

A interface do MBean *TopologicalLink* é definida como:

```
public interface TopologicalLinkMBean {
    public String getTopologicalLinkId();
    public SignalEnum getSignalIdentification();
    public DirectionalityEnum getDirectionality();
    public OperationalStateEnum getOperationalState();
    public Integer getProvisionedBandwidth();
    public void setProvisionedBandwidth(Integer pb);
    public Integer getAvailableBandwidth();
    public RestorationModeEnum getRestorationMode();
    public void setRestorationMode(RestorationModeEnum rm);
    public String getCustomerIdentification();
    public void setCustomerIdentification(String ci);
    public Integer getWeight();
    public void setWeight(Integer w);

    public LinkConnection[] queryLinkConnection();
    public LinkEndAndTPList queryLinkEndOrTP();
    public Subnetwork[] querySubnetwork();
    public LinkConnection setupLinkConnection(LinkConnection lc, NetworkCTP ctp1, NetworkCTP
ctp2);
    public LinkConnection setupLinkConnection(LinkConnection lc, Descriptor d);
    public LinkConnection modifyLinkConnection(LinkConnection lc, NetworkCTP ctp1, NetworkCTP
ctp2);
    public LinkConnection modifyLinkConnection(LinkConnection lc, Descriptor d);
    public LinkConnection releaseLinkConnection(NetworkCTP[] ctps);
    public Subnetwork[] traceLink(Subnetwork[] sl);
}
```

5.4.21. TrafficDescriptorProfile

A interface do MBean *TrafficDescriptorProfile* é definida como:

```
public interface TrafficDescriptorProfileMBean {
    public String getTrafficDescriptorProfileId();
    public Integer getIngressPeakCellRate();
    public void setIngressPeakCellRate(Integer ipcr);
    public Integer getEgressPeakCellRate();
```

```
    public void setEgressPeakCellRate(Integer epcr);
    public Integer getIngressSustainableCellRate();
    public void setIngressSust ainableCellRate(Integer iscr);
    public Integer getEgressSustainableCellRate();
    public void setEgressSustainableCellRate(Integer escr);
    public Integer getIngressMaximumBurstSize();
    public void setIngressMaximumBurstSize(Integer imbs);
    public Integer getEgressMaximumBurstSize();
    public void setEgressMaximumBurstSize(Integer embs);
}
```

5.4.22. Trail

A interface do MBean *Trail* é definida como:

```
public interface TrailMBean {
    public String getTrailId();
    public SignalEnum getSignalIdentification();
    public DirectionalityEnum getDirectionality();
    public AvailabilityStatusEnum getAvailabilityStatus();
    public AdministrativeStateEnum getAdministrativeState();
    public void setAdministrativeState(AdministrativeStateEnum as);
    public String getUserLabel();
    public void setUserLabel(String ul);
    public Boolean getRestorableIndicator();
    public void setRestorableIndicator(Boolean ri);
    public Boolean getRetainedResource();
    public void setRetainedResource(Boolean rr);

    public NetworkTTP[] queryNetworkTTP();
    public LinkEndAndTPList traceConnection();
}
```

5.4.23. TrailRequest

A interface do MBean *TrailRequest* é definida como:

```
public interface TrailRequestMBean {
    public String getTrailRequestId();
    public RequestStatusEnum getRequestStatus();
    public RequestTypeEnum getRequestType();
    public Date getRequestCommittedTime();
}
```

5.4.24. VcLinkEnd

A interface do MBean *VcLinkEnd* é definida como:

```
public interface VcLinkEndMBean {
    public String getLinkEndId();
    public AdministrativeStateEnum getAdministrativeState();
    public void setAdministrativeState(AdministrativeStateEnum as);
    public AvailabilityStatusEnum getAvailabilityStatus();
    public Integer getEgressMaxAssignableBandwidth();
    public Integer getIngressMaxAssignableBandwidth();
    public Integer getEgressAvailableBandwidth();
    public Integer getIngressAvailableBandwidth();
    public String getUserLabel();
    public void setUserLabel(String ul);
    public TPTypeEnum getLinkTPType();
    public void setLinkTPType(TPTypeEnum tpt);

    public TopologicalLink[] queryTopologicalLink();
    public Subnetwork[] querySubnetwork();
    public NetworkTTP[] queryNetworkTTP();
    public NetworkTTP associateNetworkTTP(NetworkTTP ttp);
    public Subnetwork[] traceLink(Subnetwork[] sl);
}
```

5.4.25. VpLinkEnd

A interface do MBean *VpLinkEnd* é definida como:

```
public interface VpLinkEndMBean extends VcLinkEndMBean {
    public Integer getLoopbackLocationIdentifier();
    public void setLoopbackLocationIdentifier (Integer lli);
    public SignalEnum getILMIVirtualIdentifier();
    public void setILMIVirtualIdentifier (SignalEnum vi);
    public String getSupportingNELocation();
    public void setSupportingNELocation (String ne);
    public String getSupportingCircuitPa ckLocation();
    public void setSupportingCircuitPackLocation (String cp);
    public String getServerTTPName();
    public void setServerTTPName(String ttp);
    public String getServerTTPCharacteristicInfo();
    public void setServerTTPCharacteristicIn fo(String ttp);
    public Integer getServerTTPPortId();
    public void setServerTTPPortId(Integer ttp);
    public OperationalStateEnum getServerTTPOperationalState();
    public void setServerTTPOperationalState(OperationalStateEnum ttp);
    public String getServerTTPAdditionalInformation();
    public void setServerTTPAdditionalInformation(String ttp);
    public Boolean getCellScramblingEnable();
    public void setCellScramblingEnable(Boolean cs);
    public String getSubscriberAddress();
    public void setSubscriberAddress(String as);
    public String getPreferedCarrier();
    public void setPreferedCarrier(String pc);
}
```

5.5. Especificação das Classes de Suporte Utilizadas pelo Modelo nas Interfaces

Na especificação das interfaces do JMX para o modelo de informação de rede ATM foram utilizados vários tipos de dados não primitivos do Java que devem ser especificados para que possam ser referenciados pelas interfaces dos MBeans. Estes tipos específicos de dados estão definidos no código a seguir:

```
public class ActionInfo {
   private Trail trailRef;
   private Date committedTime;
   private RequestTypeEnum requestType;

   public RouteDescription(Trail ref, Date t, RequestTypeEnum req) {
      trailRef = ref;
      committedTime = t;
      requestType = req;
   }
   public Trail getTrail() { return trailRef; }
   public Date getCommittedTime() { return committedTime; }
   public RequestTypeEnum getRequestType() { return requestType; }
   public void setTrail(Trail ref) { trailRef = ref; }
   public void setCommittedTime(Date t) { committedTime = t; }
   public void setRequestType(RequestTypeEnum req) { requestType = req; }
}

public class AdministrativeStateEnum {
   private final String name;

   private AdministrativeStateEnum(String n) { name = n; }
   public String toString() { return name; }
   public static final AdministrativeStateEnum locked = new AdministrativeStateEnum("locked");
   public static final AdministrativeStateEnum unlocked = new
AdministrativeStateEnum("unlocked");
   public static final AdministrativeStateEnum shuttingDown = new
AdministrativeStateEnum("shuttingDown");
}

public class AlarmSeverity {
   private Integer problem;
   private AlarmSeverityEnum severity;

   public AlarmSeverity(Integer p, Alarm SeverityEnum s) {
      problem = p;
      severity = s;
   }
   public Integer getProblem() { return problem; }
   public AlarmSeverityEnum getSeverity() { return severity; }
   public void setAlarmSeverity(Integer p, AlarmSeverityEnum s) {
      problem = p;
      severity = s;
   }
}

public class AlarmSeverityEnum {
   private final String name;

   private AlarmSeverityEnum(String n) { name = n; }
   public String toString() { return name; }
   public static final AlarmSeverityEnum nonAlarmed = new AlarmSeverityEnum("nonAlarmed");
```

```
    public static final AlarmSeverityEnum minor = new AlarmSeverityEnum("minor");
    public static final AlarmSeverityEnum major = new AlarmSeverityEnum("major");
    public static final AlarmSeverityEnum critical = new AlarmSeverityEnum("critical");
    public static final AlarmSeverityEnum warning = new AlarmSeverityEnum("warning");
}

public class AvailabilityStatusEnum {
    private final String name;

    private AvailabilityStatusEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final AvailabilityStatusEnum inTest = new AvailabilityStatusEnum("inTest");
    public static final AvailabilityStatusEnum failed = new AvailabilityStatusEnum("failed");
    public static final AvailabilityStatusEnum powerOff = new AvailabilityStatusEnum("powerOff");
    public static final AvailabilityStatusEnum offLine = new AvailabilityStatusEnum("offLine");
    public static final AvailabilityStatusEnum offDuty = new AvailabilityStatusEnum("offDuty");
    public static final AvailabilityStatusEnum dependency = new
AvailabilityStatusEnum("dependency");
public static final AvailabilityStatusEnum degraded = new AvailabilityStatusEnum("degraded");
    public static final AvailabilityStatusEnum notInstalled = new
AvailabilityStatusEnum("notInstalled");
    public static final AvailabilityStatusEnum logFull = new AvailabilityStatusEnum("logFull");
}

public class CellTypeEnum {
    private final String name;

    private CellTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final CellTypeEnum segment = new CellTypeEnum("segment");
    public static final CellTypeEnum endToEnd = new CellTypeEnum("endToEnd");
}

public class ConnectionTypeEnum {
    private final String name;

    private ConnectionTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final ConnectionTypeEnum broadcast = new ConnectionTypeEnum("broadcast");
    public static final ConnectionTypeEnum merge = new ConnectionTypeEnum("merge");
    public static final ConnectionTypeEnum composite = new ConnectionTypeEnum("composite");
    public static final ConnectionTypeEnum multipoint = new ConnectionTypeEnum("multipoint");
    public static final ConnectionTypeEnum pointToPoint = new
ConnectionTypeEnum("pointToPoint");
}

public class DirectionalityEnum {
    private final String name;

    private DirectionalityEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final DirectionalityEnum unidirectional = new DirectionalityEnum("unidirectional");
    public static final DirectionalityEnum bidirectional = new DirectionalityEnum("bidirectional");
}

public class EventTypeEnum {
    private final String name;

    private EventTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final EventTypeEnum alarm = new EventTypeEnum("alarm");
    public static final EventTypeEnum attributeValueChange = new
EventTypeEnum("attributeValueChange");
    public static final EventTypeEnum stateChange = new EventTypeEnum("stateChange");
    public static final EventTypeEnum managedEntityCreation = new
EventTypeEnum("managedEntityCreation");
```

```
    public static final EventTypeEnum managedEntityDeletion = new
EventTypeEnum("managedEntityDeletion");
}

public class Descriptor {
    private NetworkTTP interfaceId;
    private Boolean vciVpiPresent = False;
    private Integer vpi;
    private Integer vci;
    private Boolean cdvTolerancePresent = False;
    private Integer egressCdvTolerance;
    private Integer ingressCdvTolerance;
    private Boolean maxBurstSizePresent = False;
    private Integer egressMaxBurstSize;
    private Integer ingressMaxBurstSize;
    private Boolean cellRatePresent = False;
    private Integer egressPeakCellRate;
    private Integer ingressPeakCellRate;
    private Integer egressSustainableCellRate;
    private Integer ingressSustainableCellRate;
    private Boolean qosClassPresent = False;
    private QosClassEnum qosClass;

    public Descriptor(NetworkTTP ttp) { interfaceId = ttp; }
    public NetworkTTP getInterfaceId() { return interfaceId; }
    public Boolean isVciVpiPresent() { return vciVpiPresent; }
    public Integer getVpi() { return (vciVpiPresent)? vpi : 0; }
    public Integer getVci() { return (vciVpiPresent)? vci : 0; }
    public Boolean isCdvTolerancePresent() { return cdvTolerancePresent; }
    public Integer getEgressCdvTolerance() { return (cdvTolerancePresent)? egressCdvTolerance :
0; }
    public Integer getIngressCdvTolerance() { return (cdvTolerancePresent)? ingressCdvTolerance :
0; }
    public Boolean isMaxBurstSizePresent() { return maxBurstSizePresent; }
    public Integer getEgressMaxBurstSize() { return (maxBurstSizePresent)? egressMaxBurstSize :
0; }
    public Integer getIngressMaxBurstSize() { return (maxBurstSizePresent)? ingressMaxBurstSize :
0; }
    public Boolean isCellRatePresent() { return cellRatePresent; }
    public Integer getEgressPeakCellRate() { return (cellRatePresent)? egressPeakCellRate : 0; }
    public Integer getIngressPeakCellRate() { return (cellRatePresent)? ingressPeakCellRate : 0; }
    public Integer getEgressSustainableCellRate() { return (cellRatePresent)?
egressSustainableCellRate : 0; }
    public Integer getIngressSustainableCellRate() { return (cellRatePresent)?
ingressSustainableCellRate : 0; }
    public Boolean isQosClassPresent() { return qosClassPresent; }
    public QosClassEnum getQosClass() { return (qosClassPresent)? qosClass :
QosClassEnum.class0; }
    public void setInterfaceId(NetworkTTP ttp) { interfaceId = ttp; }
    public void setVciVpi(Integer c, Integer p) {
        vciVpiPresent = True;
        vpi = p;
        vci = c;
    }
    public void setCdvTolerance(Integer e, Integer i) {
        cdvTolerancePresent = True;
        egressCdvTolerance = e;
        ingressCdvTolerance = i;
    }
    public void setMaxBurstSize(Integer e, Integer i) {
        maxBurstSizePresent = True;
        egressMaxBurstSize = e;
        ingressMaxBurstSize = i;
    }
    public void setMaxBurstSize(Integer ePeak, Integer iPeak, Integer eSustainable, Integer
iSustainable) {
```

```
        cellRatePresent = True;
        egressPeakCellRate = ePeak;
        ingressPeakCellRate = iPeak;
        egressSustainableCellRate = eSustainable;
        ingressSustainableCellRate = iSustainable;
    }
    public void setQosClass(QosClassEnum qos) {
        qosClassPresent = True;
        qosClass = qos;
    }
}

public class IntegerRange {
    private Integer lowInt;
    private Integer hightInt;

    public IntegerRange(Integer l, Integer h) {
        lowInt = l;
        hightInt = h;
    }
    public Integer getLowInt() { return lowInt; }
    public Integer getHightInt() { return hightInt; }
    public void setRange(Integer l, In teger h) {
        lowInt = l;
        hightInt = h;
    }
}

public class LinkDetails {
    private NetworkTTP interfaceId;
    private Integer maxNumActiveVPCAllowed;
    private Integer maxNumActiveVCCAllowed;
    private Integer egressBandwidth;
    private Integer ingressBandwidth;
    private Integer maxEgressBandwidth;
    private Integer maxIngressBandwidth;
    private IntegerRange vpiOrVciRange;

    public LinkDetails(NetworkTTP ttp, Integer vpc, Integer vcc, Integer e, Integer i, Integer maxE,
Integer maxI, IntegerRange range) {
        interfaceId = ttp;
        maxNumActiveVPCAllowed = vpc;
        maxNumActiveVCCAllowed = vcc;
        egressBandwidth = e;
        ingressBandwidth = i;
        maxEgressBandwidth = maxE;
        maxIngressBandwidth = maxI;
        vpiOrVciRange = range;
    }
    public NetworkTTP getInterfaceId() { return interfaceId; }
    public Integer getMaxNumActiveVPCAllowed() { return maxNumActiveVPCAllowed; }
    public Integer getMaxNumActiveVCCAllowed() { return maxNumActiveVCCAllowed; }
    public Integer getEgressBandwidth() { return egressBandwidth; }
    public Integer getIngressBandwidth() { return ingressBandwidth; }
    public Integer getMaxEgressBandwidth() { return maxEgressBandwidth; }
    public Integer getMaxIngressBandwidth() { return maxIngressBandwidth; }
    public IntegerRange getVpiOrVciRange() { return vpiOrVciRange; }
    public void setInterfaceId(NetworkTTP ttp) { interfaceId = ttp; }
    public void setMaxNumActiveCAllowed(Integer vpc, Integer vcc) {
        maxNumActiveVPCAllowed = vpc;
        maxNumActiveVCCAllowed = vcc;
    }
    public void setBandwidth(Integer e, Integer i, Integer maxE, Integer maxI) {
        egressBandwidth = e;
        ingressBandwidth = i;
        maxEgressBandwidth = maxE;
```

```
        maxIngressBandwidth = maxI;
    }
    public void setVpiOrVciRange(IntegerRange range) { vpiOrVciRange = range; }
}

public class LinkEndAndTPList {
    private VcLinkEnd[] vcLinkEnds;
    private VpLinkEnd[] vpLinkEnds;
    private LogicalLinkTP[] logicalLinkTPs;

    public LinkEndAndTPList () { }
    public VcLinkEnd[] getVcLinkEnds() { return vcLinkEnds; }
    public VpLinkEnd[] getVpLinkEnds() { return vpLinkEnds; }
    public LogicalLinkTP[] getLogicalLinkTPs() { return logicalLinkTPs; }
    public void setVcLinkEnds(VcLinkEnd[] vc) { vcLinkEnds = vc; }
    public void setVpLinkEnds(VpLinkEnd[] vp) { vpLinkEnds = vp; }
    public void setLogicalLinkTPs(LogicalLinkTP[] tp) { logicalLinkTPs = tp; }
}

public class LogFullActionEnum {
    private final String name;

    private LogFullActionEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final LogFullActionEnum wrap = new LogFullActionEnum("wrap");
    public static final LogFullActionEnum halt = new LogFullActionEnum("halt");
}

public class LoopbackDetails {
    private Boolean endPoint;
    private CellTypeEnum cellType;

    public LoopbackDetails(Boolean ep, CellTypeEnum c) {
        endPoint = ep;
        cellType = c;
    }
    public Boolean getEndPoint() { return endPoint; }
    public CellTypeEnum getCellType() { return cellType; }
    public void setEndPoint(Boolean ep) { endPoint = ep; }
    public void setCellType(CellTypeEnum c) { cellType = c; }
}

public class OperationalStateEnum {
    private final String name;

    private OperationalStateEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final OperationalStateEnum disabled = new OperationalStateEnum("disabled");
    public static final OperationalStateEnum enabled = new OperationalStateEnum("enabled");
}

public class ProvisionTypeEnum {
    private final String name;

    private ProvisionTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final ProvisionTypeEnum manual = new ProvisionTypeEnum("manual");
    public static final ProvisionTypeEnum automatic = new ProvisionTypeEnum("automatic");
}

public class QosClassEnum {
    private final String name;

    private QosClassEnum(String n) { name = n ; }
    public String toString() { return name; }
    public static final QosClassEnum class0 = new QosClassEnum("class0");
```

```
    public static final QosClassEnum class1 = new QosClassEnum("class1");
    public static final QosClassEnum class2 = new QosClassEnum("class2");
    public static final QosClassEnum class3 = new QosClassEnum("class3");
    public static final QosClassEnum class4 = new QosClassEnum("class4");
}

public class RequestStatusEnum {
    private final String name;

    private RequestStatusEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final RequestStatusEnum notScheduled = new
RequestStatusEnum("notScheduled");
    public static final RequestStatusEnum scheduled = new RequestStatusEnum("scheduled");
    public static final RequestStatusEnum suspended = new RequestStatusEnum("suspended");
    public static final RequestStatusEnum userCanceled = new
RequestStatusEnum("userCanceled");
    public static final RequestStatusEnum beingHandled = new
RequestStatusEnum("beingHandled");
    public static final RequestStatusEnum completed = new RequestStatusEnum("completed");
}

public class RequestTypeEnum {
    private final String name;

    private RequestTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final RequestTypeEnum setup = new RequestTypeEnum("setup");
    public static final RequestTypeEnum modify = new RequestTypeEnum("modify");
    public static final RequestTypeEnum release = new RequestTypeEnum("release");
    public static final RequestTypeEnum addTPs = new RequestTypeEnum("addTPs");
    public static final RequestTypeEnum removeTPs = new RequestTypeEnum("removeTPs");
}

public class RestorationModeEnum {
    private final String name;

    private RestorationModeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final RestorationModeEnum unavailable = new
RestorationModeEnum("unavailable");
    public static final RestorationModeEnum availRoutingOnly = new
RestorationModeEnum("availRoutingOnly");
    public static final RestorationModeEnum availReRoutingOnly = new
RestorationModeEnum("availReRoutingOnly");
    public static final RestorationModeEnum availRoutingAndReRouting = new
RestorationModeEnum("availRoutingAndReRouting");
}

public class RouteDescription {
    private Object referenceObject;
    private RoutingOptionEnum routingOption;

    public RouteDescription(Object obj, RoutingOptionEnum opt) {
        referenceObject = obj;
        routingOption = opt;
    }
    public Object getReferenceObject() { return referenceObject; }
    public RoutingOptionEnum getRoutingOption() { return routingOption; }
    public void setReferenceObject(Object obj) { referenceObject = obj; }
    public void setRoutingOption(RoutingOptionEnum opt) { routingOption = opt; }
}

public class RoutingOptionEnum {
    private final String name;
```

```
    private RoutingOptionEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final RoutingOptionEnum mandatory = new RoutingOptionEnum("mandatory");
    public static final RoutingOptionEnum preferred = new RoutingOptionEnum("preferred");
    public static final RoutingOptionEnum exclude = new RoutingOptionEnum("exclude");
    public static final RoutingOptionEnum sameRoute = new RoutingOptionEnum("sameRoute");
    public static final RoutingOptionEnum diverseRoute = new RoutingOptionEnum("diverseRoute");
}

public class SeverityEnum {
    private final String name;

    private SeverityEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final SeverityEnum indeterminate = new SeverityEnum("indeterminate");
    public static final SeverityEnum critical = new SeverityEnum("critical");
    public static final SeverityEnum major = new SeverityEnum("major");
    public static final SeverityEnum minor = new SeverityEnum("minor");
    public static final SeverityEnum warning = new SeverityEnum("warning");
    public static final SeverityEnum cleared = new SeverityEnum("cleared");
}

public class SignalEnum {
    private final String name;

    private SignalEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final SignalEnum vc = new SignalEnum("vc");
    public static final SignalEnum vp = new SignalEnum("vp");
}

public class TPTypeEnum {
    private final String name;

    private TPTypeEnum(String n) { name = n; }
    public String toString() { return name; }
    public static final TPTypeEnum UNI = new TPTypeEnum("UNI");
    public static final TPTypeEnum interNNI = new TPTypeEnum("interNNI");
    public static final TPTypeEnum intraNNI = new TPTypeEnum("intraNNI");
    public static final TPTypeEnum unconfigured = new TPTypeEnum("unconfigured");
}

public class VirtualId {
    private Integer vpi;
    private Integer vci;

    public VirtualId(Integer p, Integer c) {
        vpi = p;
        vci = c;
    }
    public Integer getVpi() { return vpi; }
    public Integer getVci() { return vci; }
    public void setVpi(Integer p) { vpi = p; }
    public void setVci(Integer c) { vci = c; }
}
```

5.6. Metodologia de Especificação das Classes para o JMX

Para especificar as classes (MBeans) requeridas pelo JMX a partir dos diagramas de classe do UML, devem-se considerar, além da especificação dos MBeans de acordo com o JMX, determinadas

características de relacionamento entre as classes do modelo de informação. Estas características a serem observadas estão descritas nas seções a seguir.

5.6.1. Definição da Classe

Considere o exemplo, na Figura 77, de uma classe definida em UML.

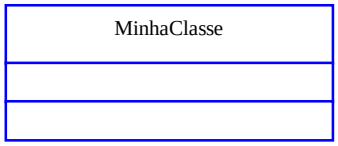

Figura 77: Exemplo de uma classe UML

De acordo com a especificação do JMX, a classe a ser implementada para constituir o MBean do agente JMX deve possuir pelo menos um construtor público e implementar a sua interface, como está apresentado no código exemplo a seguir:

```
public class MinhaClasse implements MinhaClasseMBean {
    public MinhaClasse() { ... }
    ...
}
```

5.6.2. Classes com Relacionamento Unidirecional

Considere o exemplo, na Figura 78, de classes definidas em UML que contêm um relacionamento unidirecional.

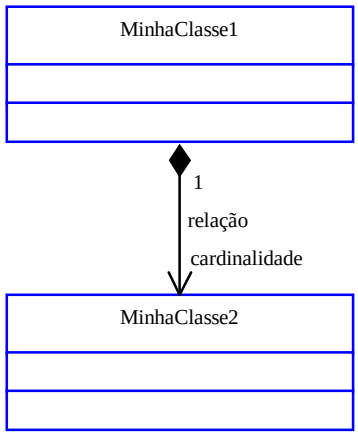

Figura 78: Exemplo de uma classe UML com relacionamento unidirecional

Este tipo de relacionamento unidirecional significa que a classe *MinhaClasse1* contém uma referência para a classe *MinhaClasse2*. Dependendo da cardinalidade, pode-se implementar o relacionamento de 2 formas:

- para a cardinalidade 1 ou 1..0, implementa-se o relacionamento através de um ponteiro para a classe destino, como apresentado no código das classes exemplo a seguir:

```
public class MinhaClasse1 implements MinhaClasse1MBean {
   private MinhaClasse2 relaçãoMi nhaClasse2;

   public MinhaClasse2 getRelaçãoMinhaClasse2() { return relaçãoMinhaClasse2; }
   public void setRelaçãoMinhaClasse2(MinhaClasse2 rel) { relaçãoMinhaClasse2 = rel; }
   ....
}

public class MinhaClasse2 implements MinhaClasse2MBean {
   ...
}
```

- para as demais cardinalidades, implementa-se o relacionamento através de uma lista de ponteiros para a classe destino, como apresentado no código das classes exemplo a seguir:

```
public class MinhaClasse1 implements MinhaClasse1MBean {
   private MinhaClasse2 relaçãoMinhaClasse2[CARDINALIDADE_MÁXIMA];

   public MinhaClasse2[] getRelaçãoMinhaClasse2() { return relaçãoMinhaClasse2; }
   public void setRelaçãoMinhaClasse2(MinhaClasse2[] rel) { relaçãoMinhaClasse2 = rel; }
   ....
}

public class MinhaClasse2 implements MinhaClasse2MBean {
   ...
}
```

5.6.3. Classes com Relacionamento Bidirecional

Considere o exemplo, na Figura 79, de classes definidas em UML que contêm um relacionamento bidirecional.

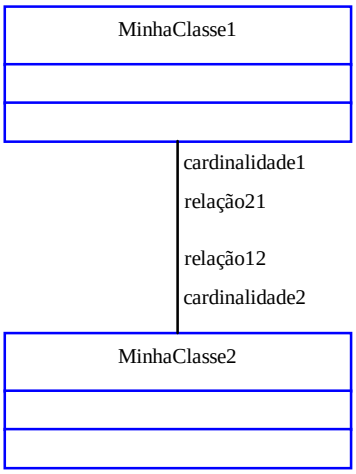

Figura 79: Exemplo de uma classe UML com relacionamento bidirecional

Este tipo de relacionamento bidirecional significa que a classe *MinhaClasse1* contém uma referência para a classe *MinhaClasse2*, e esta uma referência para a classe *MinhaClasse1*. Dependendo da cardinalidade, pode-se implementar o relacionamento de 2 formas:

- para cardinalidade 1 ou 0..1, implementa-se o relacionamento através de um ponteiro para a classe destino, como apresentado no código das classes exemplo a seguir:

```
public class MinhaClasse1 implements MinhaClasse1MBean {
   private MinhaClasse2 relação12MinhaClasse2;

   public MinhaClasse2 getRelação12MinhaClasse2() { return relação12MinhaClasse2; }
   public void setRelação12MinhaClasse2(MinhaClass e2 rel) { relação12MinhaClasse2 = rel; }
   ....
}

public class MinhaClasse2 implements MinhaClasse2MBean {
   private MinhaClasse1 relação21MinhaClasse1;

   public MinhaClasse1 getRelação21MinhaClasse1() { return relação21MinhaClasse1; }
   public void setRelação21MinhaClasse1(MinhaClasse1 rel) { relação21MinhaClasse1 = rel; }
   ....
}
```

- para as demais cardinalidades, implementa-se o relacionamento através de uma lista de ponteiros para a classe destino, como apresentado no código das classes exemplo a seguir:

```
public class MinhaClasse1 implements MinhaClasse1MBean {
   private MinhaClasse2 relação12MinhaClasse2[CARDINALIDADE_MÁXIMA];

   public MinhaClasse2[] getRelação12MinhaClasse2() { return relação12MinhaClasse2; }
   public void setRelação12 MinhaClasse2(MinhaClasse2[] rel) { relação12MinhaClasse2 = rel; }
   ....
}

public class MinhaClasse2 implements MinhaClasse2MBean {
   private MinhaClasse1 relação21MinhaClasse1[CARDINALIDADE_MÁXIMA];
```

```
    public MinhaClasse1[] getRelação21MinhaClasse1() { return relação21MinhaClasse1; }
    public void setRelação21MinhaClasse1(MinhaClasse1[] rel) { relação21MinhaClasse1 = rel; }
    ....
}
```

5.6.4. Classes com Métodos de Envio de Eventos

Considere o exemplo, na Figura 80, de uma classe definida em UML que contém métodos de envio de eventos.

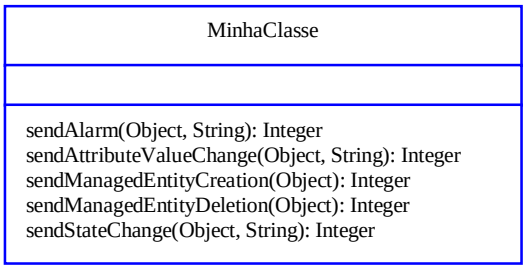

Figura 80: Exemplo de uma classe UML com métodos de envio de eventos

De acordo com a especificação do JMX, as classes que geram e enviam eventos devem implementar a interface *NotificationBroadcaster* (veja Figura 69). Por isso os métodos de envio de eventos não são mapeados diretamente na sua interface mas podem ser implementados pela classe para encapsular os procedimentos de geração de eventos, como está apresentado no código exemplo a seguir:

```
public class MinhaClasse implements MinhaClasseMBean, NotificationBroadcaster {
    Integer sendAlarm(Object obj, String alarm) { ... }
    Integer sendAttributeValueChange(Object obj, String attributeName) { ... }
    Integer sendEntityCreation(Object obj) { ... }
    Integer sendEntityDeletion(Object obj) { ... }
    Integer sendStateChange(Object obj, String stateName) { ... }
    ...
}
```

5.6.5. Classes com Atributos e Métodos

Considere o exemplo, na Figura 81, de uma classe definida em UML que contém a definição de atributos e de métodos.

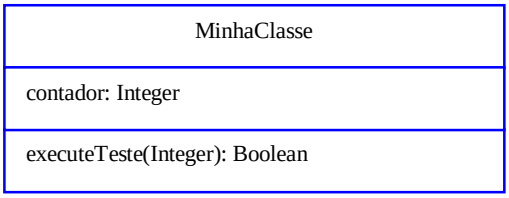

Figura 81: Exemplo de uma classe UML com atributos e métodos

De acordo com a especificação do JMX, os atributos de um MBean encontram-se especificados na sua interface através de métodos de leitura (prefixo *get*) e/ou gravação (prefixo *set*) dos seus conteúdos, enquanto os métodos estão mapeados diretamente na sua interface.

A implementação de um MBean (classe para o JMX) implica na implementação dos métodos especificados na definição de sua interface, incluindo-se adicionalmente a definição dos atributos propriamente ditos, como está apresentado na definição da interface e no código exemplo a seguir:

```
public interface MinhaClasseMBean {
    public Integer getContador();
    public void setContador(Integer c);
    public Boolean executeTeste(Integer numTeste);
}

public class MinhaClasse implements MinhaClasseMBean {
    private Integer contador;

    public Integer getContador() { return contador; }
    public void setContador(Integer c) { contador = c; }
    public Boolean executeTeste(Integer numTeste) { ... }
    ...
}
```

5.7. Conclusão

Este capítulo apresentou o modelo de informação de gerência do nível de rede para redes ATM, assim como a sua forma de implementação, de acordo com a especificação do JMX.

Para implementar um modelo de informação em uma plataforma JMX, desenvolveu-se um mapeamento dos diagramas de classe de um modelo descrito em UML para as interfaces requeridas na especificação do JMX. Para implementar os MBeans de um agente JMX, desenvolveu-se uma metodologia de implementação das classes componentes de um modelo de informação a partir das suas interfaces.

Estas metodologias tornam o processo genérico para qualquer modelo de informação descrito em UML que se queira implementar em uma plataforma que siga a especificação do JMX.

6. Conclusões Finais

Diferentes redes baseadas em diferentes tecnologias têm sido desenvolvidas de forma praticamente independente. Neste cenário surgiu o ATM, com o objetivo de prover uma tecnologia padronizada de transmissão de dados em uma rede de telecomunicações capaz de suportar qualquer tipo de aplicação atual ou futura independente dos seus requisitos de largura de banda.

A tecnologia ATM apresenta uma série de vantagens em relação a várias outras tecnologias de transmissão, principalmente em relação a sua padronização, flexibilidade e capacidade de integração de vários serviços, tais como voz, imagem, vídeo, dados e multimídia. Além disto, em função dos requisitos atuais do mercado, esta tecnologia foi desenvolvida com a preocupação de facilitar o seu gerenciamento, tanto dos seus equipamentos quanto da sua arquitetura funcional.

O gerenciamento dos sistemas ATM baseia-se em um modelo de informação padrão, padronizado por organismos internacionais e desenvolvido de acordo com a arquitetura de gerência proposta pelo TMN, visando especificamente o protocolo de gerência CMIP.

O TMN, especificado e mantido pelo ITU-T, define uma arquitetura de interconectividade e de comunicação entre sistemas operacionais heterogêneos e redes de telecomunicações. Mas, devido a diversos fatores (principalmente os altos custos, a complexidade de implementação e de manutenção e a falta de padronização), a indústria de sistemas de gerência de telecomunicações não o adotou da forma com que foi proposto, pesquisando outras formas de padronizar os modelos de informação para a gerência de sistemas de telecomunicações e outras soluções para o desenvolvimento de aplicações TMN.

Este trabalho apresentou o desenvolvimento do modelo de informação para o gerenciamento de configuração e de falhas, para o nível de rede, de redes baseadas na tecnologia ATM, utilizando-se do UML, tornando-o desta forma independente da plataforma e da arquitetura que venha a ser utilizada na sua implementação. Este modelo de informação em UML está apresentado na forma de diagramas de classe e de uma metodologia de implementação baseada em diagramas de seqüência de cenários de operação.

Quanto a plataformas de desenvolvimento TMN, o ambiente de desenvolvimento de aplicações de gerência JMX da Sun surgiu exatamente para suprir a falta de opções de plataformas alternativas baseadas na tecnologia dinâmica, flexível e portável do Java, através dos seus componentes de software, os JavaBeans.

Este trabalho desenvolveu uma metodologia para o mapeamento de um modelo de informação de gerência em UML para as interfaces e as classes requeridas pela especificação do JMX. Esta

metodologia foi utilizada para obter o conjunto de interfaces do modelo de informação para a gerência de rede ATM e para direcionar o desenvolvimento das suas classes.

O resultado final deste trabalho é um modelo de informação de gerência de redes baseado na tecnologia de transmissão ATM, descrito em UML, e uma metodologia de implantação deste modelo, para que possa ser utilizado no desenvolvimento de uma aplicação de gerência ATM em qualquer plataforma de desenvolvimento ou de operação. Como aprofundamento do estudo do modelo de informação, desenvolveu-se uma metodologia para portá-lo para a plataforma JMX de desenvolvimento de aplicações de gerência baseado em Java.

A implementação deste modelo de informação de redes ATM para o gerenciamento de falhas e de configuração, através de uma plataforma de desenvolvimento de aplicações de gerência baseada na tecnologia JMX, é a continuação natural deste trabalho.

Referências Bibliográficas

[AF 0020] ATM Forum. **Recommendation AF-NM-0020.000: M4 Interface Requirements and Logical MIB**. October 1994.

[AF 0058] ATM Forum. **Recommendation AF-NM-0058.001: M4 Interface Requirements and Logical MIB: ATM Network View**. May 1999.

[AF 0073] ATM Forum**. Recommendation AF-NM-0073.000: M4 Network View: CMIP MIB Specification**. January 1997.

[AGTTKT 2000] AdventNet, Inc. **AdventNet Agent Toolkit Release 4.0**. May 2000.

[Barillaud 1997] F. Barillaud, L. Deri, M. Feridum. **Network Management using Internet Technologies**. IBM Research Division, La Gaude Research Laboratory. La Gaude, France. 1997.

[Bertholdi 1999] C. E. Bertholdi. **Arquitetura Baseada em CORBA para Gerenciamento Integrado da Infra-estrutura de Telecomunicações**. Pontifícia Universidade Católica do Paraná. Curitiba. 1999.

[Core Java V1] C. S. Horstmann, G. Cornell. **Core Java 2, Volume 1: Fundamentals**. Sun Microsystems, Inc. Prentice Hall. 1999.

[Core Java V2] C. S. Horstmann, G. Cornell. **Core Java 2, Volume 2: Advanced Features**. Sun Microsystems, Inc. Prentice Hall. 2000.

[CTIT 1999] CTIT. **Introduction to TMN**. CTIT Technical Report 99-09. University of Twente, The Netherlands. April 1999.

[Deri 1996] L. Deri. **Network Management fot the 90s**. IBM Research Division, Zurich Research Laboratory. Zurich, Switzerland. 1996.

[ETSI ETR 269] ETSI. **Transmission and Multiplexing; Network Level Information Modelling**. April 1996.

[ETSI ETS 653] ETSI. **Telecommunications Management Network; Generic Managed Object Class Library for the Network Level View**. May 1996.

[EURESCOM 1999] EURESCOM. **TMN Evolution – Service Providers' Needs for the Next Millennium**. EURESCOM Project 812-GI Main Report. January 1999.

[Higa 1999] F. S. Higa. **Estudo do Uso de CORBA para Supervisão de Alarmes em Sistemas de Suporte à Operação de Telecomunicações**. Centro Federal de Educação Tecnológica do Paraná. Curitiba. Dezembro de 1999.

[IEEE 1995] IEEE. **Telecommunications Management NetworkVision vs. Reality**. IEEE Communications Magazine. March 1995.

[ITU G805] ITU-T. **Recommendation G.805: Generic Functional Architecture of Transport Networks**. November 1995.

[ITU I326] ITU-T. **Recommendation I.326: Functional Architecture of Transport Networks based on ATM**. November 1995.

[ITU M3010] ITU-T. **Recommendation M.3010: Principles for a Telecommunications**

Managememt Network. 1996.

[ITU X208] ITU-T. **Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1)**. 1993.

[ITU X710] ITU-T. **Recommendation X.710: Common Management Information Services Specification**. 1991.

[ITU X711] ITU-T. **Recommendation X.711: Common Management Information Protocol Specification**. 1991.

[ITU X722] ITU-T. **Recommendation X.722: Structure of Management Information: Guidelines for the Definition of Managed Objects**. 1992.

[JDMK 1999] Sun Microsystems, Inc. **Java Dynamic Management Kit. Programming Guide**. March 1999.

[JDMKWP 1998] Sun Microsystems, Inc. **Java Dynamic Management Kit. A White Paper**. February 1998.

[JMX 1999] Sun Microsystems, Inc. **Java Management Extensions**. June 1999.

[JMXWP 1999] Sun Microsystems, Inc. **Java Management Extensions White Paper**. August 1999.

[JMXWBEM 1999] Sun Microsystems, Inc. **Java Management Extensions. CIM/WBEM APIs**. August 1999.

[JMXSNMP 1999] Sun Microsystems, Inc. **Java Management Extensions. SNMP Manager API**. August 1999.

[Knöchlein 1999] H. Knöchlein. **Management eines Internet Telefonie Servers mittels JDMK**. Technische Universität München. Munich, Germany. February 1999.

[NMF 025] TM Forum. **The "Ensemble" Concepts and Format**. 1995.

[OMG 1998] Object Management Group. **The Common Object Request Broker: Architecture and Specification**. 1998.

[Park 1999] J.-T. Park, M.-S. Jeong, S.-B. Kim. **A Platform Architecture for the Integration of CORBA Technology within TMN Framework**. IEICE Trans. Commun., Vol. E82-B, No. 11. November 1999.

[Pavlou 1998] G. Pavlou, O. Festor. **Management Information Model Engineering**. Journal of Network and System Management (JNSM), Special Issue on Information Modelling, Vol. 6, No. 3, pages 239-243. Plenum Publishing. September 1998.

[Puka 2000] D. Puka. **Definição de Acordos de Nível de Serviço (SLAs) para a Gerência de QoS em Conexões Virtuais ATM**. Centro Federal de Educação Tecnológica do Paraná. Curitiba. Janeiro de 2000.

[Ranc 2000] D. Ranc, G. Pavlou, D. Griffin, J. S. Horra. **Issues and Experiences of CORBA-Based Management Agents**. Institut National des Télécommunications. Evry, France. 2000.

[Reiser 1999] H. Reiser, A. Keller. **Dynamic Management of Internet Telephony Servers: A Case Study based on JavaBeans and JDMK**. Proceedings of the Third

International Enterprise Distributed Object Computing Conference: EDOC'99. Mannheim, Germany. September 1999.

[Schultz 1999] S. Schultz. **Pocket Guide for Asynchronous Transfer Mode and ATM Testing**. Wandel & Goltermann GmbH & Co. Eningen, Germany. February 1999.

[Vertel 1998] Vertel Corporation. **Introduction to TMN**. Vertel Corporation home page http://www.vertel.com/tmn_prod.htm. 1998.

[Vertel 1999] Vertel Corporation. **Telecommunications Management Network Tutorial**. Vertel Web ProForum home page http://www.webproforum.com/tmn/. April 1999.