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

Gerenciamento Distribuído TMN: uma Experiência em Supervisão de Alarmes com CORBA

Vera Lúcia Lorenset

Florianópolis, Abril de 1998

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

Gerenciamento Distribuído TMN: uma Experiência em Supervisão de Alarmes com CORBA

Dissertação submetida à
Universidade Federal de Santa Catarina
para a obtenção do grau de Mestre
em Ciência da Computação

Vera Lúcia Lorenset

Prof. João Bosco Mangueira Sobral, Dr.
Orientador
Prof. Carlos Becker Westphall, Dr.
Co-Orientador

Florianópolis, Abril de 1998

Gerenciamento Distribuído TMN: uma Experiência em Supervisão de Alarmes com CORBA

Vera Lúcia Lorenset

Esta dissertação foi julgada adequada para a obtenção do título de

Mestre em Ciência da Computação

especialidade em **Sistemas de Computação**,

e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação.

Prof. João Bosco Mangueira Sobral, Dr.
Orientador

Prof. Carlos Becker Westphall, Dr.
Co-Orientador

Prof. Jorge Muniz Barreto, Dr.
Coordenador

Banca Examinadora:

Prof. João Bosco Mangueira Sobral, Dr.
Presidente

Prof. Carlos Becker Westphall, Dr.

Prof. Luis Fernando Friedrich, Dr.

Prof. Bernardo Gonçalves Riso, Dr.

Dedicatória

"O erro não se torna verdade
por se difundir e se multiplicar facilmente.
Do mesmo modo a verdade não se
torna erro pelo fato de
ninguém a ver."

Gandhi

Aos meus pais Helena Fiorentini Lorenset e Angelin Lorenset,
à minha irmã Bernardete e ao meu noivo Julinho.

Agradecimentos

Ao professor João Bosco Mangueira Sobral pela oportunidade, pelo trabalho de orientação e pelas imensas horas de dedicação.

Ao professor Carlos Becker Westphall pelo seu constante apoio, ânimo e compreensão, pelas suas novas idéias e pelos importantes *papers* que tornaram possível o desenvolvimento deste trabalho.

Aos professores Luis Fernando Friedrich e Bernardo Gonçalves Riso pela participação na banca examinadora, pelas críticas e comentários.

Ao Rafael Alves Chaves pela disponibilidade e grande auxílio na implementação deste trabalho, e pela pessoa amiga que és, sempre.

Aos amigos e integrantes do Laboratório de Redes e Gerência (LRG) pelas inúmeras contribuições e pelo constante ânimo no desenvolvimento das atividades.

Às queridas amigas, Verinha e Valdete, que não medem esforços para sempre nos ajudar.

Pelos administradores da rede, Sobral e Hoff, pela amizade, disponibilidade e ajuda.

À UFSC e ao CNPQ pelo suporte material e financeiro.

A todos os colegas e amigos de baia, de convivência, de festa, de conversas e de "família" aqui em Florianópolis, por nos suportarmos e pela querida e única amizade que apenas cada um de vocês sabem oferecer. Para vocês, um cantinho do meu coração, eternamente.

Aos meus amados pais, pela vida, pelo constante amor e apoio. Aos meus queridos irmãos e familiares, pelas orações, pela amizade, pelo apoio e incentivo, sempre. Amo a todos.

Ao meu eterno namorado Julinho, pelos seus anos de espera, pela sua compreensão e pelo grande amor que tens por mim.

A todos que me ajudaram...

A quem por tudo o que Foi, É e sempre Será possível: Deus e Jesus Cristo.

Sumário

Resumo

TMN (*Telecommunications Management Network*) é uma arquitetura de rede para gerenciamento de redes de telecomunicações. Esta arquitetura provê uma estrutura para operação, administração, manutenção e provisionamento das redes e serviços de telecomunicações em diferentes ambientes abertos. O principal requisito da TMN é suportar o processamento distribuído. Ela utiliza a estrutura de gerenciamento OSI (*Open Systems Interconection*) para construir um sistema de gerência distribuída.

Por outro lado, há uma tecnologia emergente, e de grande interesse atual, chamada CORBA (*Common Object Request Broker Architecture*), que estabelece um ambiente de computação distribuída orientado a objetos, com a qual um modelo de gerência distribuída pode ser desenvolvido.

As funções de gerenciamento em TMN com CORBA (por exemplo: o gerenciamento de falha e supervisão de alarmes) são realizadas por blocos de funções específicas, os quais têm capacidades de processamento distribuído. O gerenciamento distribuído do bloco de função é realizado utilizando-se o processamento distribuído da CORBA, onde cada elemento constituinte do bloco de função é visto como um objeto ORB (*Object Request Broker*), que provê o processamento distribuído dentro de cada bloco de função. Os ORBs se comunicam através do protocolo GIOP (*General Inter ORB Protocol*).

O presente trabalho apresenta o potencial da arquitetura CORBA utilizado para permitir o gerenciamento distribuído em TMN, mais especificamente, em supervisão de alarmes. Este trabalho apresenta a implementação de objetos gerenciados no contexto CORBA, em contraste à definição desses objetos no modelo OSI e faz uma comparação entre estes padrões. Também, é apresentado um exemplo para a utilização da supervisão de alarmes com CORBA: a especificação IDL (*Interface Definition Language*) de um elemento de rede (comutador) ATM - *Asynchronous Transfer Mode*.

Abstract

TMN (Telecommunication Management Network) is a network architecture for the management of telecommunication networks which provides a framework for operation, administration, maintenance and provision of networks and telecommunication services in different open environments. The essential TMN requirement is supporting distributed processing. It uses an OSI (Open Systems Interconnection) management structure to form a distributed management system.

On the other hand, there is an emerging technology, called CORBA (Common Object Request Broker Architecture), that establishes a distributed object oriented computer environment, with which a distributed management model can be developed.

The management functions in TMN with CORBA, for example: fault management - alarm surveillance, are performed by specific function blocks, which have distributed processing capabilities. The function block distributed management is performed, using the CORBA distributed processing, where each function block element is known as an ORB (Object Request Broker) object. The ORB provides the distributed processing inside each function block. The inter ORBs communications occur through the GIOP (General Inter ORB Protocol), where the q, x and f interface operations are encapsulated inside GIOP messages.

In this context, this work investigates the potential of the CORBA architecture in the sense of allowing the distributed management in TMN, more specifically, distributed objects are specified in alarm surveillance service. This work proposes the implementation of managed and supported objects in the CORBA context, in contrast to the definition of these objects in the OSI model and does a comparison between this standards. It is presented an IDL (Interface Definition Language) specification of ATM (Asynchronous Transfer Mode) network element - an ATM switch, as an example.

Lista de Siglas

ACE	*Application Construction Environment*
ACSE	*Association Control Service Element*
AE	*Application Entity*
AIS	*Alarm Indication Signal*
ANSAWare	*Advanced Networked Systems Architecture Ware*
ANS.1	*Abstract Syntax Notation One*
AP	*Application Process*
ATM	*Asynchronous Transfer Mode*
BER	*Basic Encoding Rules*
BOA	*Basic Object Adapter*
CASC	*Current Alarm Summary Control*
CCITT	*Consultative Committee for International Telegraph and Telephone*
CDV	*Cell Delay Variation*
CDR	*Common Data Representation*
CMIP	*Common Management Information Protocol*
CMIS	*Common Management Information Service*
CORBA	*Commom Object Request Broker Architecture*
COSS	*Common Object Services Specifications*
DAP	*Directory Access Protocol*
DCE	*Distributed Computing Environment*
DCN	*Data Communication Network*
DIB	*Directory Information Base*
DII	*Dynamic Invocation Interface*
DIT	*Directory Information Tree*
DME	*Distributed Management Environmet*
DMI	*Desktop Management Interface*
DN	*Distinguished Name*
DSA	*Directory Service Agent*

DSI	*Dynamic Skeleton Interface*
DCOM	*Distributed Component Object Model*
DSP	*Directory Service Protocol*
DUA	*Directory User Agent*
ESIOP	*Environment-Specific Inter-ORB Protocol*
FTP	*File Transfer Protocol*
GDMO	*Guideline for Definitions of Managed Objects*
GIOP	*General Inter ORB Protocol*
GMI	*Generic Management Information*
HTTP	*Hypertext Transfer Protocol*
IDL	*Interface Definition Language*
IIOP	*Internet Inter-ORB Protocol*
IPC	*Inter-Process Communication*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunications Union - Telecommunications Standardization Sector*
JIDL	*IDL Compiler for Java*
JIDM	*Joint Inter-Domain Management*
LAN	*Local Area Network*
LLA	*Logical Layered Architecture*
LME	*Layer Management Entity*
LP	Linhas Dedicadas
MD	*Mediation Device*
MIB	*Management Information Base*
MIM	*Management Information Model*
MO	*Managed* Object
MOC	*Managed Object Class*
MOS	*Management Operations Schedule*
NE	*Network Element*
NEML	Camada de Gerenciamento de Elemento de Rede
NEL	*Network Element Layer*

NM Forum	*Network Management Forum*
NML	*Network Management Layer*
NSAP	*Network Service Access Point*
OAM	*Operation, Administration and Maintenance*
ODMA	*Open Distributed Management Architecture*
ODP	Processamento Distribuído Aberto
OMA	*Object Management Architecture*
OMG	*Object Management Group*
ORB	*Object Request Broker*
OSI	*Open Systems Interconnection*
OS	*Operations Systems*
OSF	*Operation System Function*
PDU's	*Protocol Data Units*
PSTN	Rede Telefônica Pública Comutada
RDI	*Remote Defect Indicator*
RDN	*Relative Distinguished Name*
RDSI	Rede Digital de Serviços Integrados
RFP	*Request For Proposals*
ROSE	*Remote Operations Service Element*
RM-ODP	*Reference Model - Open Distributed Processing*
RPC	*Remote Procedure Call*
SDU's	*Service Data Units*
SMAE	*System Management Application Entity*
SMI	*Structure of Management Information*
SMK	*Shared Management Knowledge*
SNMP	*Simple Network Management Protocol*
SO	*Support Operations*
TCP/IP	*Transport Control Protocol / Internet Protocol*
TINA	*Telecommunications Information Networking Architecture*
TMN	*Telecommunications Management Network*
TSIG	*Telecom Special Interest Group*

URL	*Universal Resource Locator*
VC	*Virtual Channel*
VCC	*Virtual Channel Connection*
VCI	*Virtual Channel Identifier*
VP	*Virtual Path*
VPC	*Virtual Path Connection*
WS	*WorkStation*
WSF	*WorkStation Function*

Lista de Figuras

Lista de Tabelas

1. Introdução

Este trabalho apresenta uma experiência em relação ao uso da arquitetura CORBA [COR 95], como meio de implementação do sistema *Telecomunication Network Management* (TMN) [M.3010].

O trabalho é baseado na arquitetura proposta pelo *Telecom Special Interest Group* (TSIG) e o *Object Management Group* (OMG) para TMN baseada em CORBA [TEL 96]. Um objetivo no desenvolvimento desta nova arquitetura de implementação é reutilizar o conhecimento e a experiência adquirida com os padrões ITU-T/OSI (*International Telecommunications Union - Telecommunications*), nos quais a TMN se baseia.

Por mais de uma década, a indústria de telecomunicações tem se voltado para o estabelecimento de padrões para Operação, Administração, Manutenção e Provisão (OAM&P) de redes de telecomunicações baseados sobre o conceito TMN.

O conceito básico de TMN é prover uma arquitetura organizada para interconexão dos vários tipos de Sistemas de Operações (OS's) e equipamentos de telecomunicações, para troca de informação de gerenciamento usando interfaces padronizadas. Este trabalho tenta abordar tal interconexão, do ponto de vista da arquitetura CORBA, ao invés das interfaces padronizadas ITU-T/OSI, e mostra como esta visão alternativa pode ser vista como um sistema distribuído através de CORBA.

No sentido de limitar o trabalho, foi escolhida a parte da TMN voltada para o serviço de Supervisão de Alarmes [Q.821] e dentro deste limite foi estudado o gerenciamento de um recurso, um elemento de rede como um comutador ATM, através de CORBA.

Discussões relacionadas a *Object Broker Request* (ORBs) nos grupos de telecomunicações têm ocorrido sobre duas áreas:

- ORBs facilitando a monitoração e o controle sobre elementos de rede de telecomunicações, redes e serviços;
- ORBs usados para manipular a distribuição dentro da rede física e seus elementos de redes.

Enquanto a última é considerada uma valiosa área de estudo e um pouco complementar ao primeiro, este trabalho se concentra na projeção da primeira, buscando no módulo de

referência proposto pelo TSIG e OMG. Dentro das áreas funcionais de gerenciamento OSI, este trabalho aborda o Gerenciamento de Falhas e se utiliza, portanto, de Funções de Gerenciamento definidas pela ISO, tais como as Funções da Supervisão de Alarmes: relatório de alarmes; relatório de sumarização de alarmes; critério de evento de alarmes; gerenciamento de indicação de alarmes; e, controle de *log*.

Este trabalho mostra que é possível desenvolver um sistema de gerenciamento distribuído aberto na TMN, através da emergente tecnologia de processamento distribuído CORBA. No que segue, é mostrada uma visão histórica da pesquisa na área.

1.1 Visão Histórica da Pesquisa

Queremos agora mostrar, resumidamente, a visão histórica da pesquisa existente e de alguns trabalhos, principalmente do ano 1994 até a época atual, com relação a Redes de Gerenciamento de Telecomunicações e a arquitetura CORBA, envolvendo o paradigma orientado a objetos em sistemas distribuídos abertos.

Um suporte de gerenciamento e uma rede de gerenciamento de telecomunicações devem fornecer um conjunto de capacidades para permitir a troca e o processamento de informações de gerenciamento para planejar, provisionar, instalar, manter, operar e administrar as redes e os serviços de telecomunicações. Devido a isso, o ITU-T publicou uma série de Recomendações conhecidas como a série M.3000 (TMN - *Telecomunications Management Network*), M.3010, M.3020, M.3180, M.3200, M.3300 e a M.3400.

O estudo de sistemas distribuídos abertos é recente e tem-se diversos padrões e modelos para o seu desenvolvimento. Um dos modelos de sistema distribuído aberto e orientado a objeto é a arquitetura CORBA (*Common Object Request Broker Architecture*), resultado do trabalho de diversas companhias, integrantes do OMG (*Object Management Group*), desde 1987. Seu objetivo é propor um padrão para permitir a integração de diferentes sistemas de programação, máquinas, objetos e sistemas operacionais em ambientes abertos.

No ano 1994, companhias privadas introduziram plataformas de distribuição para gerência de operações, usando um modelo distribuído de orientação a objeto. Os desenvolvimentos trouxeram à gerência de sistemas a coerência com os interesses de gerência de redes. Houve o estudo de um conjunto básico de mecanismos conceituais e modelos para

sustentar as tarefas de gerência, e descreveram um ambiente de gerenciamento baseado na execução remota e de interação [GRE 94].

Também em 1994, M.J. Mendes e E.R.M. Madeira publicaram um trabalho sobre as plataformas que estavam sendo expandidas, e que as mesmas eram suporte ao processamento distribuído aberto de informação de multimídia. Houve o início do desenvolvimento da plataforma *Multiware* na UNICAMP, adotando o modelo básico RM-ODP como modelo funcional e incorporava os conceitos e produtos do mercado: ANSAWare, DCE/DME-OSF e CORBA-OMG. Neste trabalho foi feita a implementação do *software* correspondente à camada *Middleware*, com a metodologia iterativa orientada a objetos [MEN 94].

Já no ano de 1995, Luiz B. Lento e E.R.M. Madeira fizeram a abordagem para acessar objetos em ambientes distribuídos. Esta abordagem era baseada na arquitetura CORBA, onde especificou-se um esquema de acesso aos objetos localizados nos repositórios de interfaces e implementações desta arquitetura. Com isso, conseguiu-se acessar as informações contidas nos repositórios de forma mais simples e eficiente [LEN 95].

Uma das primeiras especificações a serem adotadas pela OMG foi a especificação CORBA, onde são detalhadas as interfaces e características do componente ORB. A última importante atualização da especificação CORBA foi feita na metade do ano de 1995, quando o OMG lançou a CORBA 2.0 [COR 95].

Outros trabalhos foram publicados neste ano. Um deles aborda a descrição de uma arquitetura de gerenciamento de alto nível, onde a monitoração é executada por agentes distribuídos com funcionalidade genérica para filtragem e criação de eventos. Neste trabalho, foi apresentada uma abordagem da freqüência de seleção ajustadora, e aspectos de implementação da arquitetura de gerenciamento em um ambiente CORBA. A freqüência permite aos agentes monitoradores adaptarem sua freqüência de seleção, automaticamente, para diferentes parâmetros de comportamento pertencentes aos componentes de gerenciamento [DIN 95].

Em [CHA 95], Ritu Chadha e Sze-Ying Wuu desenvolveram métodos e ferramentas custo-efetivo para *software* de gerenciamento, no qual os elementos de rede podem ser gerenciados. E para prover flexibilidade e a interoperabilidade, desenvolveram uma infraestrutura correspondente aos padrões TMN, que provê uma camada *Middleware*, a qual protege os programadores de *software* de complexas implementações de gerenciamento OSI/CMISE. Esta infraestrutura pode ser usada para simplificar o processo de implementação de interfaces TMN, descrevendo um sistema fim-a-fim que é gerenciado por um conjunto de

componentes CORBA. Há a descrição de passos requeridos para construir um agente para recursos gerenciados, e para fazer aplicações gerenciáveis.

No trabalho [EDW 95], Nigel Edwards verifica o padrão industrial para o processamento distribuído aberto, desenvolvido pelo *Object Management Group* (OMG). O objetivo é criar um padrão para a interoperabilidade entre aplicações independentemente desenvolvidas entre redes de computadores. Ele trabalha adotando especificações de interface e protocolo dentro do contexto de uma Arquitetura de Gerenciamento do Objeto (OMA). A OMG tem seu foco nos objetos distribuídos como um veículo para a integração de sistemas. O benefício é o encapsulamento e a cópia com heterogeneidade, o que torna as aplicações e as integrações de sistemas mais fácil.

Em 1996, James Hong e Jong-Tae Park viram que um dos requerimentos essenciais em TMN é a distribuição de herança dos sistemas de gerenciamento e dos gerenciados, e devem suportar o processamento distribuído. Então, tiveram o interesse em examinar os requerimentos gerais para o suporte do processamento distribuído. Seu propósito era o de gerenciamento distribuído aberto em TMN, e propõe uma solução. Esta solução é baseada em usos integrados de duas tecnologias de processamento distribuído, o serviço de diretório X.500 e o ODP *Trader* [HON 96].

Em [ZUQ 96] são discutidos os problemas e as respectivas soluções da interoperabilidade entre dois ORBs desenvolvidos por diferentes tecnologias. É apresentado um conjunto de operações para suportar o mecanismo de interceptação, estendendo as transparências de acesso e localização para além do escopo de um ORB, consideradas fundamentais num ambiente distribuído. Também, é apresentada uma forma que provê a transparência de relocação, estendida para suportar a interoperabilidade, e uma implementação dos mecanismos utilizando um protótipo de ORBs.

A análise da Arquitetura de Gerenciamento de Sistemas OSI (SMA) em termos de conceitos e arquitetura RM-ODP é feita em [GEN 96], por Guy Genilloud. A ISO e ITU consideram novas técnicas de modelagem para a implementação de gerenciamento de sistemas distribuídos. As técnicas foram inspiradas em GDMO. É examinado o uso de ferramentas de tradução automática (tradutores GDMO para CORBA-IDL) para integrar os agentes de gerenciamento existentes com a futura Arquitetura de Gerenciamento Distribuída (ODMA).

Kong e Chen discutem as diferentes estratégias de interfaceamento CORBA e OSI baseadas no ambiente TMN. O gerenciamento de rede baseado em OSI não é particularmente

adequado para o gerenciamento de serviço. A tecnologia CORBA provê um ambiente computacional de objeto distribuído, que é formado por uma estrutura básica de gerenciamento distribuído e é capaz de suportar diferentes redes globais. Por isso, achou-se importante que estas duas tecnologias possam trabalhar juntas para prover um serviço de telecomunicação integrada e um ambiente de gerenciamento de rede [KON 96].

No trabalho de [FER 96] é abordado a tradução de especificações, onde estas são traduzidas de uma especificação (GDMO) para outra (CORBA-IDL). São abordados várias comparações entre a gerência OSI e a OMG. Uma grande vantagem encontrada, é que a força do CORBA pode ser combinada com a força do CMIP, para obter o melhor de ambos. O implementador terá um ambiente efetivo, em que a implementação gerente ou agente, será capaz de facilmente integrar componentes de múltiplos fornecedores.

Em um dos trabalhos desenvolvidos por Luís Alberto Scandelari Bussmann e Manoel Camilo Penna, vemos que CORBA é uma tentativa de trazer para ambientes distribuídos os conceitos de encapsulamento e reutilização de código, e os mecanismos da tecnologia de orientação a objetos. Grande número de grandes organizações vê nesta especificação a única tecnologia viável para tratar a interoperabilidade de um modo não proprietário. CORBA está amadurecendo rapidamente e com ela, há várias ferramentas que facilitam a construção de aplicações distribuídas. Juntamente com o DCE é uma tendência, mas ainda não oferece um grau de maturidade tão grande quanto este [BUS 96].

No ano de 1997, novamente R. Chadha e S. Wuu descrevem componentes de uma infraestrutura que provê uma camada *middleware*. Apresentam técnicas que podem ser usadas para simplificar o processo de implementação das interfaces TMN. Esta camada facilitará consideravelmente o processo da incorporação de interfaces de gerenciamento, baseado em padrões, sobre componentes de *software* com o propósito de gerenciamento [CHA 97].

Em outro trabalho apresenta-se um ambiente inovativo (*Application Construction Environment*-ACE) para a especificação, desenvolvimento e geração de serviços de telecomunicações, de acordo com os padrões emergentes tal como TINA e CORBA. Neste ambiente são descritas as especificações de serviço, o desenvolvimento dos requerimentos de ciclo de vida, e a distinção das funcionalidades ACE, para as aplicações em objetos distribuídos no domínio telecom [BOS 97].

No trabalho realizado em [HOW 97], a CORBA é utilizada para a implementação de uma arquitetura em um sistema de gerenciamento de política dirigida. Este sistema pode ser adaptado

dinamicamente para a mudança desta política. A CORBA é vista como um ambiente de computação de objetos distribuídos, adotado pelo topo das recentes facilidades de gerenciamento de sistemas.

Em geral, a monitoração é responsável por prover informações de um sistema, úteis para o processo de tomada de decisões de gerenciamento, com o propósito de controlar o comportamento de seus elementos. Em sistemas distribuídos, os problemas de tais atividades são mais difíceis pelos aspectos da distribuição. O trabalho realizado por João Augusto G. de Queiroz e E.R.M. Madeira, descreve um Sistema de Monitoração de aplicações CORBA para as áreas funcionais de desempenho e contabilização, no contexto da Arquitetura de Gerenciamento Integrado de Sistemas Distribuídos da plataforma *multiware*. Os objetos CORBA apresentam as características de distribuição e encapsulação, e o sistema desenvolvido apresenta uma maneira simples e modular de instrumentar e monitorar tais objetos [QUE 97].

Uma visualização do gerenciamento de Inter-Domínios e dos estados das atividades do XoJIDM, pode ser encontrada em [MAZ 97]. É realizada a especificação/tradução de documentos ASN-1/GDMO para CORBA-IDL (tipos ASN-1 para tipos CORBA-IDL, mapeamento de *templates* GDMO para interfaces e operações CORBA-IDL, mapeamento de SNMPv2 MIB para tipos e interfaces CORBA-IDL). Neste trabalho foi realizado o projeto e a implementação de um *gateway* CORBA/SNMP. O que mais motivou para a realização deste trabalho foi o conjunto de bibliotecas das classes de objetos gerenciados GDMO, onde os objetos gerenciados são padronizados para as redes de telecomunicações, e a linguagem de programação independente (IDL) é utilizada para a implementação de novas interfaces.

O presente trabalho apresenta, no capítulo 2, alguns conceitos de gerenciamento e conceitos relacionados a objetos e supervisão de alarmes. No capítulo 3 é apresentado um estudo detalhado de TMN, com suas arquiteturas e funcionalidades. No capítulo 4, são abordados os requisitos para a TMN suportar o sistema distribuído, bem como algumas características da TMN com CORBA. No capítulo 5, é discutida a idéia de como a arquitetura CORBA pode ser utilizada no gerenciamento distribuído TMN, tendo como base a pesquisa dos TSIG e OMG. No capítulo 6, é apresentado o conceito de Supervisão de Alarmes em TMN obtido da norma Q.821 [Q821], e apresenta um exemplo de como desenvolvê-la em CORBA. Finalmente, no capítulo 7, é apresentada a especificação e implementação das classes que formam a Supervisão de Alarmes

em TMN, são comparados os padrões CORBA e OSI, e são apresentados os resultados do trabalho. No capítulo 8, conclusões importantes são consideradas.

O trabalho de dissertação aqui apresentado foi aceito para publicação no SCS/IEEE Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'98) - Nevada [LOR 98], e teve suporte financeiro do Conselho Nacional de Pesquisa (CNPq).

2. Conceitos Gerais

Neste capítulo são abordados os conceitos utilizados no decorrer da pesquisa sobre gerência distribuída em redes de gerenciamento de telecomunicações.

2.1 Conceitos relacionados ao Sistema de Gerenciamento

Serviço

Um serviço é como um tipo abstrato de dados. Ele define operações que podem ser realizadas sobre um objeto, mas não especifica como essas operações são implementadas[TAN94].

Recurso

É uma visão local/restrita de um equipamento da rede. O gerenciamento de serviço pode utilizar as funções fornecidas pelo recurso.

Função de Gerenciamento

É a menor parte de um serviço observada pelo usuário deste serviço; normalmente ela consiste de uma seqüência de ações sobre um objeto gerenciado [M3400].

Gerente e Agente

No modelo de gerenciamento **Gerente-Agente**, um processo pode assumir qualquer um dos papéis. Os **gerentes** gerenciam recursos mapeados em objetos tornados visíveis pelos **agentes**, chamados **objetos gerenciados**. Cada recurso gerenciado é representado por pelo menos um objeto. Neste modelo, um gerente envia operações de gerenciamento a um agente e recebe notificações dele. Os agentes podem executar operações de gerência sobre um ou mais objetos gerenciados. Um gerente pode administrar vários agentes e estes podem ser gerenciados por vários gerentes.

Operações de Gerenciamento

As operações de gerenciamento são primitivas executadas na fronteira entre um recurso e o objeto gerenciado que o representa [BRI 93]. Podem ser operações orientadas a atributos e operações sobre objetos gerenciados como um todo.

A ISO especifica as seguintes operações orientadas a atributos [LAV 96]: GET ATTRIBUTE VALUE (obtenção do valor do atributo), REPLACE ATTRIBUTE VALUE (substituição do valor do atributo), SET WITH DEFAULT VALUE (substituição do valor do atributo pelo valor *default*), ADD MEMBER (inclusão de valores) e REMOVE MEMBER (remoção de valores).

As operações sobre objetos gerenciados como um todo são as CREATE, DELETE e ACTION.

Evento

Mudança no estado de um objeto. Um comportamento autônomo de um objeto. O evento pode ser sinalizado pela emissão de uma notificação.

Notificações

Um sinal ou mensagem indicando que ocorreu um evento.

Objetos Gerenciados

São os objetos definidos para o desenvolvimento do sistema gerenciado. Um objeto gerenciado é a representação de um recurso real que pode ser manipulado pelas aplicações de gerenciamento. Desta forma, cada recurso que se deseja manipular e controlar deve ser descrito na forma de um objeto gerenciado.

Os objetos gerenciados são definidos em termos de [BRI 93]:

- seus atributos ou propriedades;
- as operações a que podem ser submetidos;
- as notificações que podem emitir para informar sobre a ocorrência de eventos de gerenciamento;
- suas relações com outros objetos gerenciados.

Objetos de Suporte

São os objetos gerenciados que servem de suporte para o desenvolvimento de outros objetos gerenciados.

Classes de Objetos Gerenciados

É um conjunto de objetos que são do mesmo tipo. Os objetos dentro da classe podem ter um ou mais moldes (*templates*) usados para a geração e gerenciamento de objetos.

Base de Informações de Gerenciamento (MIB)

No modelo de gerenciamento gerente-agente, o conjunto de objetos gerenciados dentro de um sistema constitui, juntamente com os atributos, a Base de Informação de Gerenciamento - MIB.

Atributo

São componentes identificadores de objetos, com tipo de valor. É um conjunto de valores de dados (tipo de dados) e qualificadores sobre estes valores.

Comportamento

É o conjunto de operações definidas para o objeto que especificam a maneira pela qual os dados de um objeto são manipulados. Podem corresponder a uma mudança de estado do objeto.

2.2 Conceitos relacionados a Objetos

Objetos

É qualquer coisa, real ou abstrata, a respeito da qual armazenamos dados e ou operações que as manipulam.

Objetos Distribuídos

Objetos podem ser encontrados e acessados em qualquer parte da rede.

Modelo de Objetos OMA (*Object Management Architecture*)

Modelo genérico definido pelo OMG, para facilitar a definição de objetos e os modos pelos quais estes objetos podem interagir. Este modelo proporciona uma forma de mapeamento entre modelos de objetos de diversos fabricantes.

Interoperabilidade

É a propriedade de sistemas heterogêneos se comunicarem. Para compor a interoperabilidade, deve-se entender os aspectos da sintaxe e da semântica da troca de informação. E para termos a interoperabilidade, devemos concordar com ambos estes aspectos.

Domínios

É uma coleção de objetos gerenciados que foram explicitamente agrupados para o propósito de gerência. O próprio domínio pode ser considerado objeto gerenciado e assim, podem estar contidos em outros domínios. Os domínios não isolam seus membros: objetos externos podem comunicar diretamente com um objeto do domínio. Um domínio é um objeto que pode pertencer a outros domínios [QUE 97].

Federações

É uma reunião de grupos que respondem a diferentes autoridades (e deste modo representam domínios diferentes), de modo a cooperar para o alcance de algum objetivo comum.

Interface

É a especificação dos atributos e operações de objetos.

Especificação de Atributo, Operação e Evento

É o processo que descreve os atributos, operações ou eventos de objetos gerenciados, utilizando-se de linguagem natural, técnicas formais, dentre outras.

Referência a Objeto

É um endereço simbólico usado para referenciar, sem ambigüidade, um objeto, durante seu tempo de vida. As referências a objeto são associadas no momento da criação do objeto, e nunca são subseqüentemente substituídas.

Nome

Um símbolo (*token*) usado para referenciar um objeto. Objetos diferentes podem ser associados a um mesmo nome, e um mesmo objeto pode ser associado a nomes diferentes. Os nomes podem ser substituídos durante o tempo de vida de objeto.

Nomes Distintos (DN)

São nomes particulares distribuídos em toda a extensão da hierarquia. Os nomes distintos são construídos hierarquicamente pelo uso recursivo dos Nomes Relativamente Distintos (RDNs).

Nomes Relativamente Distintos (RDN)

São nomes particulares no contexto do seu objeto superior, na hierarquia. Os objetos inclusos são nomeados para os objetos pertencentes ao objeto superior.

Operação

Operações são processos que podem ser solicitados como unidades de processamento. A operação é, normalmente, inicializada pela recepção de uma invocação.

Operações Ciclo de Vida de um objeto

São as operações que permitem criar, copiar, mover e destruir objetos.

Transparência de Acesso

Quando a referência a objeto não revela como o objeto é acessado.

Transparência de Localização

Quando a referência a objeto não revela a localização do objeto.

2.3 Conceitos relacionados a Supervisão de Alarmes TMN

Supervisão de Alarmes

É definida como um conjunto de funções que habilitam a monitoração e/ou interrogação de elementos da rede de telecomunicações quanto a eventos ou condições de alarme [Q821].

Sumarização

É o processo de agregar e, opcionalmente, aplicar algoritmos sobre dados ou sobre objetos observados, para produzir informação sumária.

Falha

Corresponde à causa física ou algorítmica do mau funcionamento de um sistema, manifestada através de erros.

Severidade do Alarme

É o grau de mau funcionamento do sistema gerenciado em decorrência de falhas ocorridas. O nível de severidade do alarme é avaliado por meio do nível de degradação da qualidade de serviço oferecido ao usuário do sistema ou por meio do estado da capacidade de um determinado objeto gerenciado [BRI 93].

Indicação de Tendências

Informação usada para indicar se existem alarmes pendentes, que ainda não foram tratados, e quais as tendências demonstradas pelo alarme corrente em relação àqueles pendentes [BRI 93].

Informação de Valor-Limite

Informação utilizada quando o alarme é resultado da ultrapassagem de algum valor-limite (*threshold*) [BRI 93].

Sugestões de Ações de Reparo

Dependendo do tipo de objeto gerenciado, podem ser definidas sugestões de ações de reparo no caso de ocorrência de algum alarme emitido por este objeto gerenciado [BRI 93].

3. Rede de Gerenciamento de Telecomunicações (TMN)

Para atender às necessidades de gerenciamento de redes de telecomunicações, a ITU-T definiu uma estrutura organizada de rede que faz a interligação dos vários tipos de sistemas de suporte à operação e equipamentos de telecomunicações, usando uma arquitetura genérica com protocolos e interfaces padrão, denominada TMN [M3010], a qual é baseada no modelo de referência OSI de gerência de redes. A TMN fornece meios para transportar, armazenar e processar informações, com relação à gerência e serviços de Telecomunicações.

Este capítulo relata uma breve visão de TMN e suas arquiteturas funcional, física e de informação.

3.1 Visão Global da TMN

O conceito básico de TMN é de proporcionar uma arquitetura organizada para interconexão de vários tipos de sistemas de operação (OSs) e/ou equipamentos de telecomunicações para troca de informações de gerenciamento, utilizando protocolos e interfaces padronizadas.

O objetivo de uma TMN é dar apoio ao gerenciamento de redes e serviços de telecomunicações, oferecendo funções de gerência e comunicação entre ela e a rede de telecomunicações.

A conexão geral entre a TMN e a rede de telecomunicações a ser gerenciada, pode ser vista na Figura 3.1.

Conceitualmente, TMN é uma rede separada, que se comunica com a rede de telecomunicações em vários pontos diferentes para receber informações e controlar suas operações.

TMN tem sido encarada como responsável pela estrutura para a operação, administração, manutenção e provisão (OAM&P) de redes de telecomunicações e serviços. Ela baseia-se em

sistema próprio de operações e gerência de recursos de um sistema distribuído. Ambos, o sistema de gerenciamento (gerente/agente) e o sistema gerenciado (elementos de rede, esquemas de transmissão) são fisicamente e logicamente distribuídos. Em tal ambiente, ambas, a informação relacionada ao recurso e as informações relacionadas ao serviço, sobre o sistema de gerenciamento e sistema gerenciado serão espalhadas por toda a rede. Assim, o suporte para **processamento distribuído** é um requisito essencial.

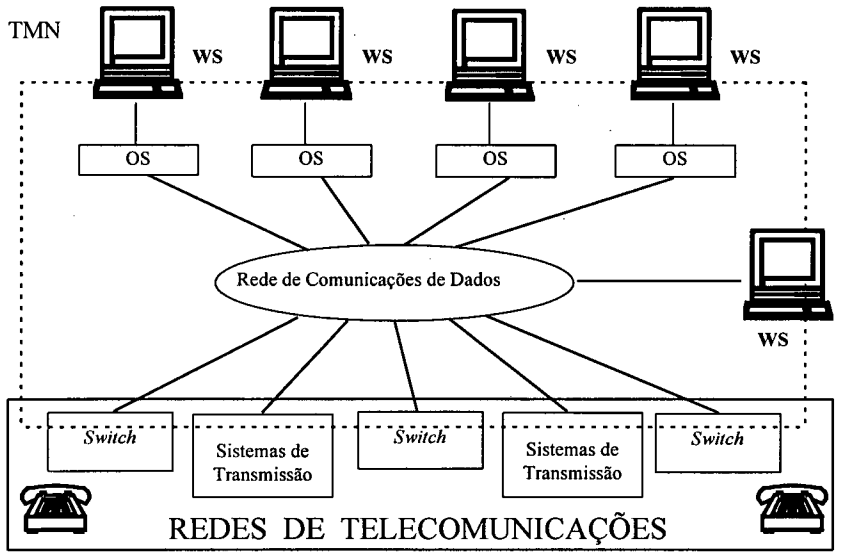

Figura 3-1 - Relações TMN para a Rede Física (M.3010).

Um dos aspectos chaves da TMN é o uso da estrutura de gerenciamento OSI como base para a estrutura TMN. O modelo OSI possui uma rica e poderosa metodologia de modelagem, protocolos de comunicação e o paradigma genérico de **gerenciamento gerente/agente** é também utilizável para modelagem e gerenciamento de recursos de telecomunicações.

Atualmente existem diversos estudos em andamento buscando definir e viabilizar modelos ou padrões alternativos, de forma a permitir uma possível integração entre os diversos produtos, para os vários níveis da gerência de redes de telecomunicações.

3.2 Funcionalidades Associadas a uma TMN

De acordo com as necessidades das empresas e para prover o suporte completo para gerência de redes de telecomunicações, uma TMN apresenta várias funções que por suas

similaridades podem ser agrupadas em áreas funcionais. Cada área reúne funções que têm objetivos semelhantes, assim cada área cuida de um aspecto importante da gerência.

A recomendação X.700 da ITU-T define cinco áreas funcionais. Dentro de cada área, aplicações apropriadas cuidam de determinados suportes de gerenciamento. São elas :

- **Gerência de Desempenho**: Monitora a eficiência e a qualidade dos serviços prestados pela rede e pelos equipamentos de telecomunicações através da coleta e análise de dados e estatísticas;
- **Gerência de Falhas**: Cuida da detecção, isolamento e correção de falhas dos recursos e serviços que podem afetar a qualidade dos serviços prestados. É dentro de Gerência de Falhas que encontramos a Supervisão de Alarmes, da qual foram selecionados alguns Objetos Gerenciados para a especificação em IDL (Anexo III), no presente trabalho;
- **Gerência de Configurações**: Permite ao usuário estabelecer e mudar os recursos físicos e lógicos da rede de telecomunicações;
- **Gerência de Contabilizações ou Tarifações**: Calcula e registra o custo associado ao uso da rede de telecomunicações;
- **Gerência de Segurança**: São funções cuja incumbência são os vários aspectos de segurança do sistema, principalmente os acessos aos recursos e aos serviços.

As funcionalidades de uma TMN consistem na capacidade de trocar informações de gerenciamento com os equipamentos de telecomunicações, converter informações de um formato para outro, transferir informações de gerenciamento entre as diferentes localizações no ambiente TMN, analisar uma informação de gerenciamento e reagir de forma apropriada, converter informações de gerenciamento para uma forma útil e significativa para o usuário, e assegurar acesso seguro das informações de gerenciamento.

3.3 Arquiteturas da TMN

A arquitetura TMN é definida a partir de três perspectivas:

- **Funcional** - descreve as funções de gerência através de blocos funcionais;
- **Física** - descreve as interfaces que constituem a TMN;
- **Informação** - possibilita o mapeamento dos princípios de gerência OSI em princípios TMN através de uma abordagem orientada à objetos;

Maiores detalhes destas três arquiteturas , seus componentes e interfaces são obtidos na recomendação M.3010.

3.3.1 Arquitetura Funcional TMN

A arquitetura funcional TMN é mostrada na figura 3.2.

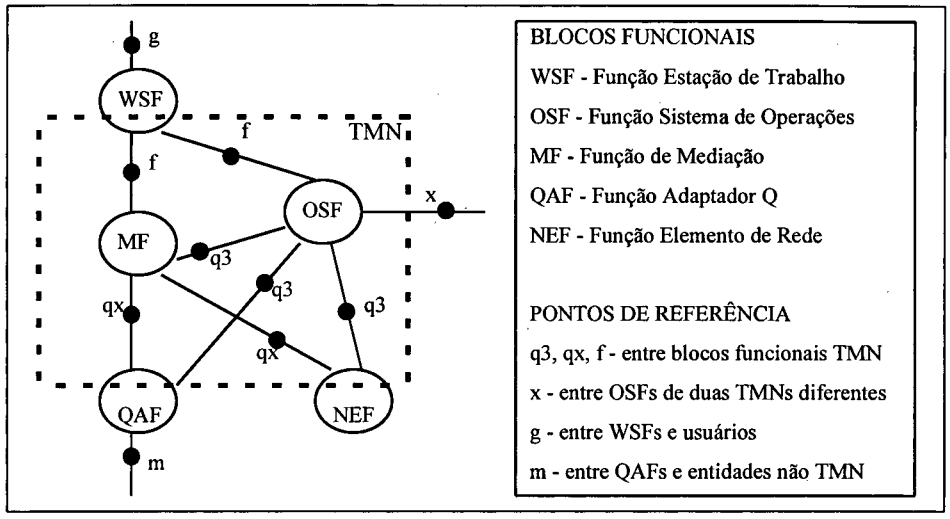

Figura 3-2 - Blocos Funcionais e Pontos de Referência

Basicamente, esta arquitetura descreve a distribuição das funcionalidades suportadas pela TMN através da definição de alguns blocos funcionais que provêem funções genéricas para uma TMN. Também descreve a distribuição apropriada destes blocos, através dos quais uma TMN de qualquer complexidade pode ser implementada. Os blocos funcionais são constituídos por componentes funcionais e comunicam-se entre si através de pontos de referência (elos ou ligações entre os blocos funcionais). Os blocos funcionais são compostos por combinações de componentes funcionais, e estes podem ser encontrados em mais de um bloco funcional.

Blocos Funcionais

Os blocos funcionais representam o agrupamento de funções genéricas da TMN.

São definidos como blocos funcionais:

- Função Sistemas de Operações (**Operations Systems Function - OSF**): este bloco funcional representa o processamento de informações que monitoram, coordenam e controlam as funções de telecomunicações e as próprias funções de gerência da TMN;

- Função Elemento de Rede (**Network Element Function**): é o bloco funcional que está em contato direto com o ambiente de telecomunicações e seu papel é o de ser monitorado e/ou controlado pelos OSFs (através das MFs). Os NEFs representam para a TMN os recursos de telecomunicações;
- Função Estação de Trabalho (**WorkStation Function**): interpreta as informações da TMN para o usuário, ou seja, inclui o suporte para a interface homem-máquina;
- Função de Mediação (**Mediation Function**): assegura que as informações que trafegam entre os OSFs e NEFs (ou QAFs) estão no formato esperado pelos blocos. É um mediador/tradutor entre os blocos interconectados. Pode armazenar, adaptar, filtrar e condensar informações;
- Função Adaptador Q (**Q Adaptor Function**): atua como um tradutor entre uma interface TMN e uma interface não-TMN. Realiza a conversão entre um ponto de referência **q3** ou **qx** da TMN e um ponto de referência não-TMN.

Componentes Funcionais

Os blocos funcionais são compostos por componentes funcionais que pela combinação dos serviços prestados por elas, permitem que os blocos funcionais possam cumprir determinadas funções para a TMN.

Os componentes funcionais representam agrupamentos de funções mais específicas para a TMN. Na arquitetura Funcional TMN são definidos os seguintes componentes funcionais:

- Função de Aplicação de Gerência (MAF): implementa funcionalidades das aplicações de gerência ou parte dela, e pode assumir o papel de gerente ou de agente;
- Base de Informação de Gerenciamento (MIB): repositório conceitual das informações de gerência. Representa o conjunto de objetos gerenciados dentro de um sistema gerenciado;
- Função de Conversão de Informação (ICF): usada em sistemas intermediários para traduzir mensagens de um modelo de informação de uma interface para o de outra interface;
- Função de Apresentação (PF): traduz informações mantidas em um modelo de informação TMN para um formato capaz de ser exibido em uma interface homem-máquina e vice-versa;
- Adaptação Homem - Máquina (HMA): converte as informações usadas na MAF do modelo de informação à Função de Apresentação, e vice-versa;
- Função de Comunicação de Mensagens (MCF): associada a todos os blocos funcionais que têm uma interface física, e sua utilização limita-se à troca de mensagens contendo informações de gerência entre os blocos.

Componentes funcionais característicos de cada bloco funcional:

- OSF : MIB, OSF-MAF, ICF, HMA;
- WSF : PF;
- NEF : MIB, NEF-MAF;
- MF : MIB, QAF-MAF, ICF;

Pontos de Referência da TMN

Os Pontos de Referência definem as fronteiras de serviços entre dois blocos funcionais de gerência, permitindo identificar as informações trocadas entre estes blocos. Estes pontos de referência são pontos conceituais de troca de informações entre os blocos funcionais, que quando implementados correspondem às interfaces físicas.

Há três classes de pontos de referência TMN:

- **classe q**: entre NEF e OSF, NEF e MF, MF e OSF, e OSF e QAs. Estas associações podem acontecer diretamente ou através da função de comunicação de dados. Dentro da classe de pontos de referência **q** há duas subclasses:

  pontos de referência qx : entre NEF e MF, e QAF e MF.

  pontos de referência q3 : entre NEF e OSF, QAF e OSF, e MF e OSF.
- **classe f** : entre WSF e OSF, e WSF e MF.
- **classe x** : entre duas TMNs (OSF de uma TMN e o OSF de outra TMN) ou entre um OSF de uma TMN e um bloco funcional com funcionalidades equivalentes de outra rede.

Há mais duas classes de pontos de referência não-TMN:

- **classe g**: entre WSF e usuários;
- **classe m**: entre QAF e entidades não-TMN.

A figura 3.3 representará a arquitetura física onde poderemos observar claramente os pontos de referência.

3.3.2 Arquitetura física TMN

As funções TMN podem ser implementadas em uma variedade de configurações físicas. A figura 3.3 mostra um modelo simplificado da arquitetura fisica TMN com seus blocos e pontos de referência citados anteriormente.

- **OS** - Sistemas de Suporte às Operações: a arquitetura física dos OSs deve possibilitar a centralização ou a distribuição das funções de dados. Estas funções podem ser: programas de aplicação de suporte, suporte aos terminais de usuários, ou relatórios;

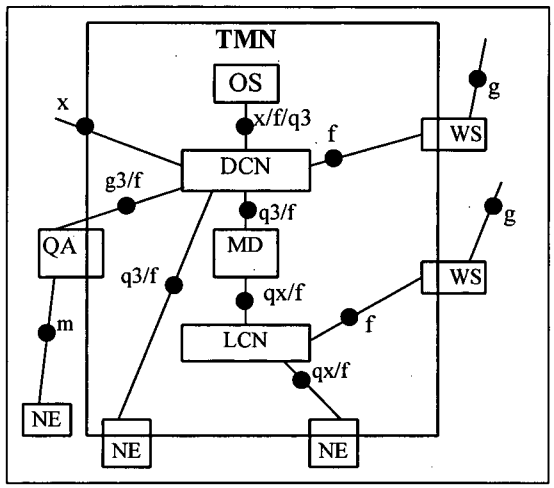

Figura 3-3 - Arquitetura Física TMN

- **DCN** - Rede de Comunicação de Dados: responsável pela implementação da função de comunicação de dados e as respectivas Funções de Comunicação de Mensagens;
- **MD** - Dispositivos de Mediação: responsável pela implementação de funções de mediação (MFs). Esse dispositivo atua sobre a troca de informações entre NEFs, QAFs e os OSFs, e provê funcionalidades de gerenciamento local para os NEs;
- **NE** - Elementos de Rede: corresponde às entidades de telecomunicações (equipamentos ou facilidades) que são monitorados e/ou controlados. Os NEs podem executar as funções de um ou mais OSFs, MFs ou QAFs;
- **QA** - Adaptadores Q: o bloco funcional QAF é usado para interconectar equipamentos e OSs, as quais não provêem interfaces padronizadas Qx ou Q3 à TMN;
- **WS** - Estações de Trabalho: atua como um terminal ligado, via DCN, a um OS ou a um dispositivo com funções de mediação (ou também, qualquer componente da TMN).

3.3.3 Arquitetura de informação TMN

A arquitetura de informação TMN é baseada no modelo de informação orientado a objetos, isto fornece fundamentos para a utilização dos princípios e conceitos do gerenciamento de sistemas OSI.

Assim, a arquitetura de Informação, além de utilizar 6os conceitos de Gerente/Agente, Domínios e Conhecimento de Gerenciamento Compartilhado (SMK - *Shared Management Knowledge*) do gerenciamento de sistemas OSI, introduz o conceito de Arquitetura em Camada Lógica (LLA - *Logical Layered Architeture*).

A informação trocada pelos sistemas de gerenciamento é modelada em termos de objetos gerenciados. Um objeto gerenciado é uma abstração do recurso gerenciado e representa suas propriedades, podendo também representar um relacionamento entre recursos ou uma combinação de recursos.

A definição de um objeto gerenciado é dada pelos seus atributos visíveis, pelas operações de gerenciamento que lhe podem ser aplicadas, pelo comportamento apresentado em resposta a estímulos internos ou externos, e pelas notificações emitidas por ele.

■ **Conhecimento Compartilhado de Gerenciamento (SMK)**

Para garantir o seu interfuncionamento, sistemas de gerenciamento em comunicação devem compartilhar uma visão comum das informações referentes aos protocolos de comunicação, às funções de gerenciamento, às classes de objetos suportadas, às instâncias disponíveis de objetos gerenciados, às capacitações autorizadas e aos relacionamentos de *containment* entre os objetos gerenciados. O conjunto destas informações é definido como Conhecimento Compartilhado de Gerenciamento.

A figura abaixo mostra que a informação compartilhada é relacionada com a comunicação entre entidades pares. Nesta figura, o SMK entre a função do sistema A e a função do sistema B não é o mesmo SMK entre a função do sistema B e a função do sistema C.

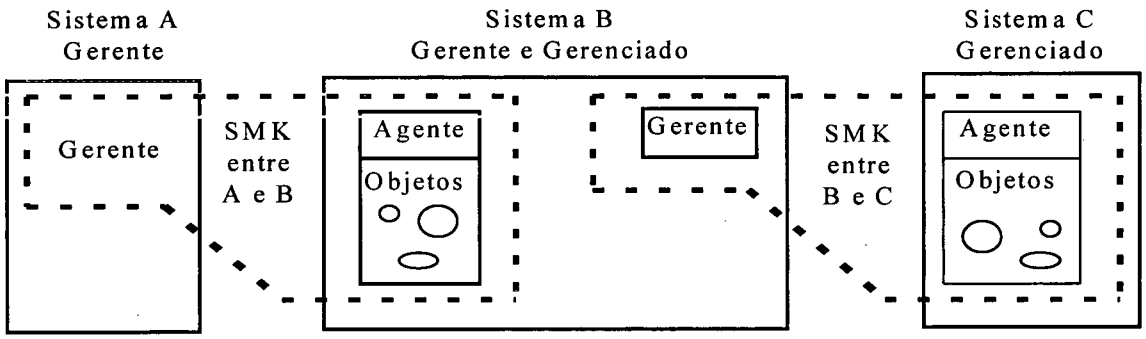

Figura 3-4 - Relações da informação compartilhada com as entidades

■ **Domínios**

Dois dos requisitos organizacionais para gerenciamento de uma coleção de objetos gerenciados é o particionamento do ambiente de gerenciamento em várias áreas funcionais, tais como segurança, contabilização e gerenciamento de falhas; e, a atribuição ou modificação temporária dos papéis de Gerente e Agente para cada um dos propósitos de gerenciamento dentro de cada coleção de objetos gerenciados.

Quando os objetos gerenciados são organizados em conjuntos para atender os requisitos organizacionais, eles são chamados de Domínio Gerencial.

■ **Arquitetura Lógica em Camadas**

O conceito de Arquitetura Lógica em camadas (LAA - *Logical Layered Architeture*) foi desenvolvido baseado no princípio hierárquico (arquitetura baseada em uma série de camadas). Esta arquitetura implica em agrupar as funcionalidades de gerenciamento em camadas. Apenas quando uma instância específica da LLA é definida, as funções ou grupos de funções podem tornar-se processos e os pontos de referência entre processos podem tornar-se interfaces.

■ **Nomeação e Endereçamento na TMN**

Para o sucesso de uma TMN em uma administração, é importante definir um esquema integrado e lógico de endereçamento e nomeação para identificar e localizar os vários objetos de comunicação dentro de uma TMN.

Assim, é necessário que os nomes sejam únicos e não ambíguos, o que requer a existência de mecanismos de coordenação das atividades de nomeação entre as administrações. Isto se torna viável através da divisão sistemática do conjunto de todos os nomes possíveis em subconjuntos.

4. Gerência Distribuída em TMN

Este capítulo identifica os mais importantes requisitos para o suporte do processamento distribuído em TMN, e apresenta a arquitetura física TMN com OSI e com CORBA. Uma TMN provê funções de gerenciamento para redes e serviços de telecomunicações, que podem ser geograficamente distribuídas pelo mundo. Ela mesma é uma rede composta de um conjunto de sistemas em cooperação. Assim, um sistema baseado em TMN opera e gerencia recursos de um ambiente inerentemente distribuído. Isto é, ambos os sistemas de gerenciamento (gerentes e agentes) e os sistemas gerenciados (elementos de rede, multiplexadores, terminais, etc) são fisicamente e logicamente distribuídos.

4.1 Provendo o Processamento Distribuído

No ambiente TMN, ambas, a **informação relacionada ao recurso** e a **informação relacionada ao serviço**, nos sistemas gerentes e sistemas gerenciados, serão espalhadas por toda a rede. Portanto, a TMN deve suportar o processamento distribuído, e para isso deve haver mecanismos apropriados para armazenar e prover cada informação em qualquer lugar da rede, onde ela seja necessária. Neste ambiente, é também essencial que a capacidade de processamento distribuído seja provida a uma única TMN, bem como entre TMNs, possibilitando assim, o gerenciamento distribuído em TMNs interconectadas [HON 96].

O primeiro requisito para o suporte do processamento distribuído em TMN é haver um **mecanismo repositório de informação**, para suportar o armazenamento e a distribuição da informação relacionada ao recurso. E cada informação deve ser facilmente acessada e amplamente disponível para vários sistemas de gerenciamento, sempre quando necessária.

O segundo requisito para o suporte do processamento distribuído em TMN é haver um **mecanismo corretor apropriado para encontrar a informação** sobre vários serviços disponíveis na rede e para provê-los aos sistemas de gerenciamento que desejam utilizá-los. Os sistemas de gerenciamento devem ter o conhecimento de quais e onde os serviços são disponíveis, e sua provisão deve ser feita transparentemente pelo mecanismo.

O terceiro requisito para suportar o processamento distribuído em TMN é suportar as **interações muitos-para-muitos** entre gerentes e agentes. Há situações onde um gerente deseja gerenciar múltiplos agentes, simultaneamente. Também, há situações onde múltiplos gerentes desejam interagir com um simples agente. E ainda, gerentes necessitam se comunicar com outros gerentes para trocarem informações de gerenciamento. E finalmente, agentes necessitam se comunicar com outros agentes para trocarem informações de gerenciamento. Cada interação de muitos-para-muitos pode ser suportada pelo desenvolvimento e utilização de protocolos de comunicação, diretamente às partes envolvidas ou através de uma entidade de mediação intermediária.

Assim, tendo-se o conhecimento dos requisitos exigidos, pode-se apresentar as soluções oferecidas pelas tecnologias de processamento distribuído, nos contextos OSI e CORBA.

4.2 Tecnologias de Processamento Distribuído

A seguir serão examinados mecanismos que podem satisfazer os requisitos de processamento distribuído, discutidos anteriormente, para o suporte do gerenciamento distribuído aberto em TMN. Esta seção mostra como se pode obter o suporte para gerenciamento distribuído através do modelo OSI e da arquitetura CORBA, ressaltando as soluções correspondentes nestes dois contextos.

4.2.1 Serviço de Diretório X.500 (OSI)

Os padrões X.500 especificam um serviço de diretório que provê e gerencia a informação sobre entidades. Este serviço de diretório e sua informação de diretório é fisicamente distribuída, provê uma informação unificada e uma simples visão funcional lógica [HON 96].

A informação que o diretório possui é chamada de **Informação Base do Diretório** (DIB). Esta informação são entradas (ou objetos) que contém a informação sobre as entidades. Cada entrada consiste de um conjunto de atributos, cada um com um tipo e um ou mais valores. Os tipos de atributos que estão presentes em cada entrada são dependentes na classe de entidades que a entrada descreve. As classes podem ser definidas pelo usuário, e os vários tipos de informação podem ser armazenados no diretório. As entidades na DIB são representadas pelas

entradas no espaço de nomes hierárquico e global, chamado **Árvore de Informação do Diretório** (DIT). As entradas são localizadas na DIT de acordo com as relações organizacionais entre as entidades que elas representam.

A definição do serviço X.500 define portas e operações resumidas, as quais provê ao usuário, a funcionalidade de recuperar, pesquisar e modificar a informação do diretório. Estas operações formam um protocolo de interface simples, chamado de **Protocolo de Acesso ao Diretório** (DAP). Os atuais padrões do X.500 suportam leitura, comparação, lista, pesquisa e o abandono de funções de interrogação, e basicamente adiciona, remove e modifica as funções de manipulação de entrada.

O serviço de diretório, incluindo a DIT, é distribuída através de entidades fisicamente separadas, chamadas de **Agentes do Serviço de Diretório** (DSAs). Esta distribuição é transparente ao usuário através do uso de operações do **Protocolo do Serviço de Diretório** (DSP), entre DSAs. Cada usuário ou usuário-processo é representado pelo **Agente Usuário de Diretório** (DUA), responsável pela interação com o diretório. Assim, os usuários podem facilmente e eficientemente acessar a informação do diretório através de DSAs locais ou próximos.

Assume-se que alguns serviços serão providos sem levar em consideração a divisão da rede. Isto é realizado pela natureza distribuída de DSAs e réplicas da informação de diretório. O X.500 também provê um mecanismo de segurança, prevenindo o acesso à informação do diretório pelos usuários não autorizados.

Portanto, o Serviço de Diretório X.500 satisfaz o primeiro requisito para a armazenagem e distribuição da informação relatada para recursos gerenciados, bem como sistemas de gerenciamento em TMN. Uma descrição do Serviço de Diretório X.500 pode ser encontrada em [TEI 96].

4.2.2 Serviço de Nomes no CORBA

CORBA oferece como mecanismo repositório de informação o Repositório de Interfaces e o Repositório de Implementações, os quais utilizam os serviços de nome e de *trading* para encontrar a informação.

O serviço de nomes no padrão CORBA permite que clientes encontrem objetos baseando-se em nomes. Este serviço fornece um grafo de contexto de nomes que pode possuir diversas raízes. Um contexto contém um conjunto de ligações de nome para objeto. Em um sistema de arquivo, os contextos são equivalentes aos diretórios. Através do grafo de contexto de nomes, outros contextos podem ser ligados a um contexto específico.

Analisar um nome é determinar o objeto associado com o nome em um determinado contexto. *Ligar* um nome é criar uma ligação de nome em um determinado contexto. Um nome é sempre analisado relativo ao contexto (não há nomes absolutos).

Quando um contexto é semelhante a qualquer outro objeto, ele pode também ser ligado a um nome em um contexto de nomeação. Os contextos de ligação em outros contextos criam um gráfico de nomeação - um grafo direcionado com nós e margens rotuladas, onde os nós são os contextos. Um gráfico de nomeação permite mais nomes complexos para referenciar um objeto. Dado um contexto em um gráfico de nomeação, uma seqüência de nomes podem referenciar um objeto. Esta seqüência de nomes define um caminho no gráfico de nomes para navegar o resultado do processo, como pode ser vista na figura 4.1.

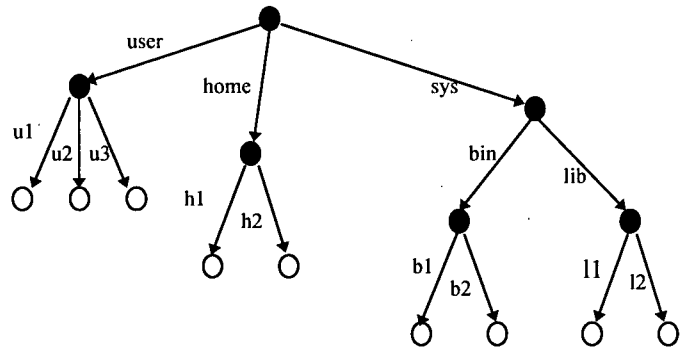

Figura 4-1 - Gráfico de Nomeação

Muitas das operações definidas no contexto de nomes têm os nomes como parâmetros. Estes nomes têm estruturas e um nome é uma seqüência ordenada de componentes. Um nome com um componente simples é chamado de nome simples; um nome com componentes múltiplos é chamado de uma seqüência de nomes (*compound*). Cada componente, exceto o último, é usado para nomear um contexto; o último componente denota o objeto ligado.

Um componente-nome consiste de dois atributos: o identificador e a classe. Ambos são representados por caracteres IDL. O atributo classe adiciona uma descrição para nomes em um modo independente da sintaxe. Alguns exemplos de valor do atributo classe é *executable* e

postscript. O sistema de nomeação não interpreta, determina, ou gerencia estes valores, mas altos níveis de *software* podem policiar seu uso e gerenciamento. Esta característica distingue a necessidade de aplicações que utilizam as convenções de nomeação sintáticas dos objetos relatados. Por exemplo, *Unix* utiliza os sufixos **.C** e **.O** .

Para evitar emissões de diferentes sintaxes de nomes, o serviço de nomeação sempre negocia com nomes em sua forma estrutural. É assumido que vários programas e serviços do sistema manterão nomes da representação dentro da forma estrutural, de uma maneira que é conveniente para eles.

O módulo de nomeação é uma coleção de interfaces que definem o serviço de nomes. Este módulo contém as interfaces *NamingContext* e *BindingIterator*.

A interface *NamingContext* apresenta as seguintes operações:

- objetos de ligação: as operações de ligação nomeiam um objeto em um contexto de nomeação; quando um objeto é ligado, ele pode ser encontrado com a operação *resolve*;
- resolução de nome: a operação *resolve* é o processo de recuperação de um objeto ligado por um nome em um dado contexto;
- desligamento: a operação de desligamento remove um nome ligado de um contexto;
- criação de contextos de nomes: para a criação de novos contextos utiliza-se as operações *new_context* (retorna um contexto de nome implementado pelo serviço de nome, como o contexto no qual a operação foi invocada) e *bind_new_context* (cria um novo contexto e o liga ao nome fornecido como um argumento);
- exclusão de contextos: a operação *destroy* exclui um contexto de nome;
- lista de contexto de nomeação: permite a um cliente interagir através de um conjunto de ligações em um contexto de nomeação.

A interface *BindingIterator* permite um cliente interagir através de ligações, utilizando as operações *next_one* (retorna a próxima ligação) ou *next_n* (retorna o número mais requisitado de ligações).

O serviço de *Trading* provê um serviço para encontrar objetos. Ele provê o registro de serviços disponíveis através da invocação de uma operação no *trader*, passando como

parâmetros, a informação sobre o serviço oferecido. A exportação da operação carrega uma referência a objeto que pode ser usada pelo cliente, para invocar operações em serviços já anunciados, uma descrição do tipo de serviços oferecidos, e a informação em atributos distintos do serviço.

O gerenciamento por *traders* pode ser dividido para facilitar a administração e navegação. Esta informação é armazenada pelo *trader*. Toda vez que um cliente desejar obter uma referência para um serviço que executa um trabalho particular, ele invoca uma importação da operação, passando como parâmetros uma descrição de serviços requeridos. Devido a este pedido, o *trader* verifica as propostas apropriadas para a aceitabilidade. Para ser aceito, uma proposta deve ter um tipo que combine (*conforms*) as propriedades consistentes e requisitadas com as informações especificadas pela importação.

O serviço de *trading* em um domínio simples, pode ser distribuído através de um número de objetos *trader*. Os *traders* em diferentes domínios podem ser federados. A federação possibilita aos sistemas de domínios diferentes negociarem o compartilhamento de serviços fora do controle de seus serviços e políticas próprias. Um domínio pode assim, compartilhar a informação com outros domínios, com os quais têm sido federados, e ele pode agora ser pesquisado pelo serviço apropriado.

4.2.3 ODP *Trader*

O ODP é um conjunto de padrões que têm como objetivo uma variedade de arquiteturas, redes e sistemas operacionais para prover um ambiente de processamento distribuído aberto. O ODP *Trader* é um componente do ambiente ODP, cujo propósito é conciliar os objetos ODP.

A vantagem do ODP *Trader* está em grandes ambientes distribuídos, onde os objetos precisam ser feitos com a informação dos serviços disponíveis. O *trader* permite que objetos ODP sejam configurados em um ambiente ODP, sem o conhecimento dos serviços ou fornecedores de serviços dentro deste ambiente. O *trader* permite a seleção do serviço dinâmico e a ligação de clientes e servidores.

O documento ODP *Trader* [ODP 94] discute um grande número de componentes que incluirão o *trader*: determinados componentes com a política de *trading*, requerimentos de

segurança, requerimentos de contabilização, requerimentos de transferência, qualidade de serviço e federação.

No núcleo do sistema ODP *Trader* há as interações entre quatro diferentes tipos de objetos: comerciantes, importadores, exportadores e serviços. Um exportador é termo ODP para um provedor de serviço. É um objeto com um serviço que ele deseja disponibilizar para outros objetos. Ao prover um serviço, é completado o serviço de exportação do *trader*. Um exportador é também capaz de mais tarde, indisponibilizar o serviço.

Na terminologia ODP, um solicitador de serviços é conhecido como um importador. No ambiente ODP, estes importadores podem operar sem qualquer conhecimento de onde os serviços requeridos estão ou quais os objetos eles provêm. Para encontrar estes serviços, o importador deve requisitar um serviço ao *trader*. O *trader* então retorna para o importador, os detalhes dos serviços, igualando ao pedido do serviço se existir. Um serviço é uma função fornecida pelo exportador para outros objetos ODP o utilizarem. Um serviço pode ser um dos seguintes tipos: uma operação atômica (por exemplo, *write*), uma seqüência de operações (*open, write, close*), ou um conjunto de operações (*read, write, open, close*).

Um serviço é exportado na forma de um serviço *proposto*, o qual descreve o serviço que está sendo disponível. Um importador descobre os serviços através do envio de pedidos importados ao *trader*. O principal componente de um pedido importado é o serviço *request*, que é um conjunto de afirmações as quais descrevem o serviço desejado. A importação do pedido também provê a informação, descrevendo o método e o escopo da pesquisa para ser utilizada pelo *trader*. Esta é uma proposta do *trader* para igualar os serviços *requests* dos importadores com os serviços *propostos* dos exportadores. Isto é feito para a combinação das afirmações no serviço *request* com as afirmações que compõe os serviços *proprietários* dos serviços oferecidos. O *trader* envia para um importador os detalhes dos serviços (incluindo locação) que iguala seus requerimentos de serviços.

O trabalho no inter-funcionamento (federação) de *traders* têm sido um modo inferior na ordem para suportar o comércio de serviços, transversalmente a domínios múltiplos. Através do inter-funcionamento de *traders*, não somente a informação nos serviços em ambientes distribuídos podem ser alterados, mas também os próprios serviços podem ser invocados transversalmente em domínios diferentes.

O ODP *Trader* satisfaz o segundo requisito de processamento distribuído para TMN, pois provê transparência na localização e transparência na migração de operações, bem como pode prover serviços de gerenciamento em TMN.

4.2.4 ORB do CORBA

Na arquitetura CORBA, um cliente para fazer uma requisição, deve conhecer a referência do objeto e a operação que deseja executar. Após, o cliente pode chamar a rotina *stub* correspondente, ou construir a requisição dinamicamente através da DII (Interface de Invocação Dinâmica). O receptor da mensagem não conhece qual dos métodos foi utilizado na requisição, apenas o ORB trata internamente todas as diferenças entre eles, e sabe qual método foi utilizado.

Ao ser realizada a requisição, o ORB localiza o código da implementação, transmite os parâmetros e transfere o controle para a implementação do objeto através de um esqueleto IDL, específico à interface e ao adaptador do objeto.

A interface do ORB é idêntica para todas as implementações, independentemente do adaptador de objetos utilizado. Através dela, podem ser invocadas as operações que são executadas pelo ORB, as quais são úteis tanto para os clientes quanto para as implementações de objeto.

O núcleo da arquitetura CORBA é o ORB. Ele provê os mecanismos de representação de objeto e de comunicação para que seja possível processar e executar as requisições. A arquitetura CORBA foi estruturada na forma de sobrepor seus componentes ao núcleo ORB, permitindo assim, que as diferenças entre núcleos de ORBs sejam mascaradas.

Devido ao fato de o tipo que representa uma referência de objeto poder variar de uma implementação para outra, convém armazenar referências de outra maneira quando estas tiverem que passar por diferentes ORBs. As referências podem ser convertidas para *strings*, as quais podem ser manipuladas pelas operações usuais associadas a este tipo de dado.

Duas operações são definidas com tal objetivo na interface do ORB. A operação object_to_string converte um elemento do tipo object em um tipo string, enquanto a operação string_to_object faz o caminho inverso. Esta representação para a referência de objeto deve seguir um padrão compreendido por todos os ORBs.

Existe um conjunto de operações que, independentemente de fatores relacionados à heterogeneidade do sistema de objetos e do adaptador utilizado, pode ser executado sobre qualquer objeto. As operações desse conjunto, não são realizadas por um objeto, mas diretamente pelo ORB e podem ser descritas na forma de interface chamada Object.

```
Interface Object {
        ImplementationDef  get_implementation ( );
        InterfaceDef       get_interface ( );
        boolean            is_nil ( );
        Object             duplicate ( );
        void               release ( );
        Status             create_request ( );
};
```

As operações get_implementation() e get_interface() retornam respectivamente as definições de implementação e de interface de um objeto, contidas nos repositórios de implementações e de interfaces.

A operação is_nil() determina se uma referência não corresponde a um objeto qualquer, ou seja, testa se seu valor é Object_Nil.

As operações duplicate() e release() são operações para cópia e desalocação de um objeto. Estas operações são necessárias devido às dependências de implementação na representação de uma referência de objeto. As referidas operações alocam e liberam uma área de memória correspondente ao tipo utilizado pela implementação para referenciar o objeto.

A operação create_request() faz parte da interface do ORB, apesar de estar mais diretamente relacionada à interface de invocação dinâmica. Esta operação cria uma requisição para o objeto, que será montada e invocada com a ajuda das demais rotinas de interface da DII.

O ORB do CORBA provê a representação básica de objetos e a comunicação de pedidos, como também um conjunto de serviços para clientes e implementações de objetos, os quais podem ter diferentes propriedades e qualidades. Devido a isso, o ORB é tido como o mecanismo corretor para manter o "caminho" da informação sobre vários serviços disponíveis na rede, e para provê-los aos sistemas de gerenciamento que desejam utilizá-los.

4.2.5 Interações de Muitos-para-Muitos

Em uma TMN, as atividades de gerenciamento são realizadas por sistemas de gerenciamento através de gerentes TMN e agentes TMN. Os gerentes TMN são as partes do

sistema de gerenciamento que são embutidas nos sistemas de operação TMN, iniciando as atividades de gerenciamento. Os agentes TMN são as partes do sistema de gerenciamento que estão embutidas nos elementos de rede TMN, realizando as atividades de gerenciamento de interesse dos gerentes. Um ou mais gerentes serão envolvidos no envio dos pedidos de gerenciamento para um ou mais agentes que estão disponíveis por toda TMN, e no processamento das respostas recebidas ou notificações de eventos dos agentes, para realizar várias funções de gerenciamento. Os agentes realizam várias funções de gerenciamento ou de interesse dos gerentes, através da interação com as partes dos elementos de rede que são gerenciadas por eles.

Em um ambiente TMN, vários serviços de gerenciamento são disponíveis e utilizados por vários gerentes e por outros serviços de gerenciamento. Estes serviços precisam ser localizados antes deles serem utilizados. O ODP *Trader* pode prover o serviço de localização dos serviços requeridos, quando necessário. Isto é realizado pelos provedores de serviços (na forma de agentes), exportando seus serviços com os atributos de cada serviço para o *trader*. Ao requerer um serviço, podem obter a informação do mesmo, solicitando ao *trader* para localizar o serviço disponível que lhes interessa. Uma vez que o serviço é localizado, o requisitor importa o serviço e o utiliza.

Para o suporte de interações muitos-para-muitos entre gerentes e agentes, poderia-se desenvolver protocolos de comunicação separados, mas isto seria muito dispendioso. Estas interações são permitidas através do uso de um mecanismo intermediário oferecido pelo X500. A informação relatada do gerenciamento pode ser armazenada dentro do Diretório X.500, por uma ou mais partes (gerentes e agentes), e estas partes podem acessar a informação simultaneamente, possibilitando assim, o suporte para interações muitos-para-muitos.

4.2.6 GIOP, Armazenamento CORBA e Grupos de Objetos

Para haver o suporte do processamento distribuído em TMN é necessário que haja um suporte à interações de muitos-para-muitos entre gerentes e agentes no sistema de gerenciamento. Há situações em que um gerente deseja utilizar agentes de gerenciamento múltiplo, ou múltiplos gerentes interagirem com um simples agente, ou os gerentes trocarem informações de gerenciamento entre eles, ou ainda, os agentes trocarem informações de gerenciamento entre eles.

As interações muitos-para-muitos podem ser suportadas pelo desenvolvimento e utilização de protocolos de comunicação diretamente para as partes envolvidas ou através de uma entidade de mediação intermediária. É possível utilizar os protocolos de comunicação para o suporte do processamento distribuído considerando o protocolo GIOP do CORBA (Anexo V). A informação relatada do gerenciamento pode ser armazenada em algum lugar não determinado (mas ligado à *factory*) de uma ou mais partes (gerentes e agentes), e estas partes podem acessar a informação simultaneamente, podendo com isso, suportar as interações muitos-para-muitos.

Na arquitetura CORBA há o processamento em grupo, essenciais para aplicações distribuídas, o qual determina uma associação de membros que apresentam uma relação abstrata comum ou uma política comum. No sistema de gerenciamento, podem existir situações em que grupos de gerentes desejam enviar funções de gerenciamento para um grupo de agentes ou a um simples agente, e para isto é utilizado o protocolo de comunicação GIOP e o armazenamento CORBA (ligada à *factory*).

O suporte para grupo tem como um dos pontos essenciais o desempenho da comunicação. Um ORB pode ser visto como canalizando pedidos e respostas de serviços, mediando interações através de protocolos apropriados (GIOP). O pedido de um cliente deve ser mapeado transparentemente, como pedidos de operação em objetos membros do grupo.

4.2.7 Sumário dos Mecanismos OSI e CORBA

A tabela 4.1 mostra, resumidamente, as soluções oferecidas dos mecanismos OSI e CORBA, vistas anteriormente, para satisfazerem os requisitos para o suporte do processamento distribuído em TMN.

	TMN em OSI	**TMN em CORBA**
Primeiro Requisito	Serviço de Diretório X.500	Serviço de Nomes e de *Trading*
Segundo Requisito	*Trader*	ORB
Terceiro Requisito	Diretório X.500	GIOP e Armazenamento CORBA

Tabela 4-1 - Solução dos requisitos em X500 e CORBA

4.3 Uma Visão de Arquitetura Física TMN

4.3.1 Modelos de Gerenciamento OSI e CORBA

Atualmente, existe o Modelo de Gerenciamento da Informação OSI (MIM) [GHE97] onde diversas aplicações de gerenciamento estão desenvolvidas. O surgimento de uma nova tecnologia para implementação de sistemas distribuídos, a arquitetura CORBA, nos leva a pensar que possa ser utilizado como um modelo de gerenciamento. Temos a seguir as características e comparações gerais entre estes principais modelos de gerenciamento [GHE 97].

O paradigma orientado a objeto é baseado nos conhecidos conceitos de orientação a objetos, e estes conceitos podem ser aplicados em uma estrutura de gerenciamento. Algumas das características encontradas nestes modelos são identificador do objeto, que é um identificador único e independente de valores; as agregações e objetos complexos, os quais são objetos com muitos atributos e muitas instâncias de atributos, bem como atributos de objeto que representam outros objetos; métodos e encapsulamento, que incluem atributos de objetos representando o estado, o acesso sobre o envio de mensagens (que invocam métodos correspondentes), os métodos especificados pelo nome, as classes de argumentos, e classes de resultados.

Outro conjunto de características são as classes, com objeto compartilhando os mesmos atributos; as instâncias que representam os objetos computacionais; a herança, que é uma classe de especialização da superclasse, como também classes de objetos organizadas em uma estrutura de herança simples ou múltipla; *binding*, que é uma seleção dinâmica do método a ser executado no objeto; e polimorfismo, que é a aceitação da operação de outras superclasses, onde é iniciada a classe de objeto gerenciado.

O modelo de gerenciamento da informação **OSI** é um bom exemplo de um modelo orientado a objeto genuíno, onde cada entidade é modelada como um objeto gerenciado. Cada objeto gerenciado têm uma lista de atributos e exibe comportamentos como o resultado de operações de gerenciamento realizados nos objetos. Cada atributo têm uma área de valores que representam as instanciações de objeto. Todas as propriedades da modelagem orientada a objeto são suportadas no modelo MIM OSI, tais como encapsulação, herança, nomeação, especialização e alomorfismo. Os atributos são agrupados em pacotes que podem ser obrigatórios ou condicionais. O modelo de gerenciamento OSI utiliza três estruturas de árvores distintas: a árvore de registro ISO / ASN.1, a árvore de herança e a árvore de nomeação. O modelo também

utiliza *templates* padronizados para documentar a ligação de nome (*binding*) e pacotes que caracterizam cada objeto gerenciado [GHE 97].

Em contraste com este modelo, a arquitetura **CORBA** provê uma plataforma de objeto distribuído, utilizando como padrão as definições da interface do objeto, e esconde o verdadeiro modelo de informação usado em implementações de objeto. O interesse dos objetos é diferente, por que eles são vistos como objetos computacionais e o CORBA é mais voltada para os aspectos de execução, do que aspectos da especificação. CORBA define as referências a objetos para os seus próprios serviços que são as verdadeiras representações de objetos dentro de seus ORBs. Estas referências de objeto podem variar de ORB para ORB, mas eles têm claras estruturas de relacionamentos, tal como herança. De acordo com a linguagem de definição de interfaces utilizada (IDL) o mapeamento com várias aplicações cliente podem ser idênticas para todos os ORBs. Os serviços de objeto e facilidades comuns, que estão fora dos principais serviços da CORBA, podem usar modelos orientados a objeto, por exemplo o modelo GDMO OSI.

A tabela a seguir nos mostra, resumidamente, as características e comparações de cada modelo de gerenciamento.

	OSI	**CORBA IDL**
Características dos Objetos	biblioteca do objeto, classes, instâncias, atributos, comportamento, operações e notificações	modelo de referência a objeto, atributos e operações
Notação	ASN.1	CORBA IDL
Codificação	BER ou outros	não autorizado
Relações e estruturas de Objetos	herança, nomeação, pacotes condicionais, alomorfismo, estrutura árvore de nomeação	herança, adaptadores de objeto
Guia de definição do objeto	GDMO ITU-T X.722	OMG IDL CORBA 1.x, CORBA 2.x
Serviços de comunicação subjacente	CMIP / CMIS ou OSI em cima de TCP/IP	GIOP/IIOP/TCP, ESIOP/DCE RPC
Naming	DNs, *naming* local e global baseado em nomeação, suporte serviço de diretório X.500 , árvore identificador do objeto ASN.1	suporte serviço de diretório compatível com X.500

Tradução do modelo de informação e algoritmos	GDMO para MIB (NM Forum 030) e GDMO-IDL (X Open JIDL)	CORBA-SMI/ MIB, CORBA-GDMO (X/Open JIDM)
Padrões	X.720 MIM, X.721 DMI, X.722 GDMO, X.724 GMI	IDL-CORBA 1.x, IDL-CORBA 2.0
Operações de Objeto	seis operações diferentes, confirmada / não confirmada	pedidos / respostas
Notificações de Objeto	notificação de evento: relatório de alarme, relatório de evento	notificações externas através dos serviços CORBA

Tabela 4-2 - Comparações entre os Modelos de Gerenciamento OSI e CORBA

4.3.2 TMN com OSI

Redes TMN contém um grande número de objetos relativamente simples. A tecnologia de gerenciamento da rede TMN, baseada em OSI, é conveniente para o gerenciamento de elementos de rede. Objetos modelam os elementos de rede reais. Eles são relativamente simples no sentido em que os objetos normalmente não têm operações e relações complicadas. GDMO [X.722] é uma linguagem adequada para especificar estes objetos e suas operações, comportamento e notificações. CMIS/CMIP [CMI 94] é usado para definir serviços e protocolos de comunicação entre diferentes entidades, responsáveis pela implementação e invocações de objetos. As demais operações realizadas nestes objetos são simples e normalmente efetuadas em um grupo de objetos determinados pelas condições de filtro e escopo.

Noutro modo, os serviços TMN podem ser modelados por um pequeno número de objetos complexos. Estes objetos representam um alto nível de abstração da rede, relações dos componentes da rede e serviços computadorizados, os quais são suportados pela rede. Por exemplo, o serviço de teleconferência pode ter um número pequeno de instâncias, representando os serviços distribuídos, oferecendo e gerenciando os componentes, isto contudo é suportado por uma grande porção de elementos subjacentes, redes e vários sistemas de informação de gerenciamento. O acesso para um serviço de teleconferência pode invocar funções complicadas incluindo contabilização, segurança, performance, interoperabilidade, qualidade de serviços, entre outros.

Dado estes parâmetros e características de gerenciamento, os serviços TMN são difíceis de serem modelados com as facilidades GDMO e implementados usando a arquitetura de gerenciamento de rede OSI.

A figura 4.2 nos mostra uma possível arquitetura física TMN com OSI. Por exemplo, para que um OS monitore um NE, pode ser necessário a utilização de dispositivos de mediação, e para a comunicação entre eles são utilizados as interfaces q, x e f (capítulo 3).

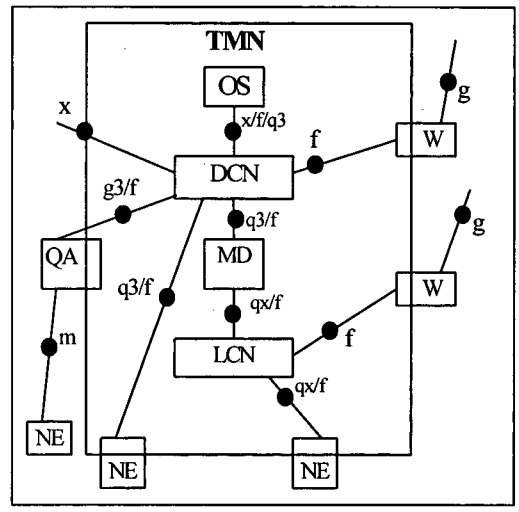

Figura 4-2 - Arquitetura Física TMN com OSI

A figura 4.3 nos mostra relações da arquitetura física TMN com OSI. As relações podem ser conexões entre TMN e a Rede de Telecomunicações a ser gerenciada. Esta pode ser vista como uma rede separada, que se comunica com a rede de telecomunicações em vários pontos, para receber informações e controlar suas operações (capítulo 3).

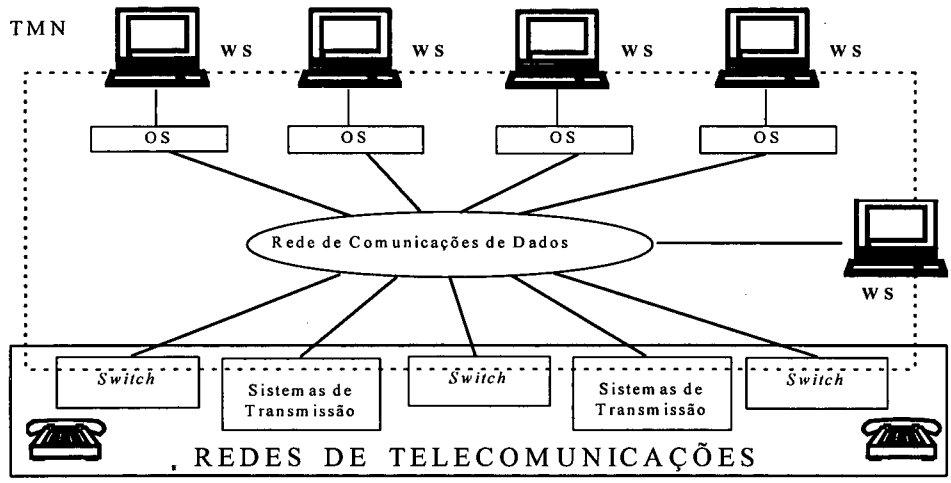

Figura 4-3 - Relações TMN para a Rede Física com OSI

4.3.3 TMN com CORBA

A tecnologia CORBA oferece uma outra alternativa para modelar complicados objetos de serviço para o gerenciamento de serviço TMN. CORBA suporta um ambiente para a definição, transporte, implementação e invocação de objetos e pedidos para o acesso destas funções. Ela também suporta a separação e a distribuição da implementação de objetos de suas definições, igual aos objetos de serviço que podem ser prontamente designados, implementados, estendidos e gerenciados pelo ambiente. A linguagem de especificação IDL oferece facilidades para especificar operações complicadas para objetos, e interfaces genéricas para invocar estas operações. Devido à estas características, a tecnologia CORBA se tornou um forte candidato, como um ambiente para implementar o gerenciamento de serviços TMN [KON 96].

Embora CORBA seja um candidato desejável para implementar os serviços TMN e gerenciamento de serviços, essa arquitetura é vista como não adequada para gerenciar redes e elementos TMN, devido ao seguinte raciocínio [KON 96]:

- no gerenciamento da rede TMN, a topologia de rede e a distribuição de recursos são relativamente estáveis, a característica da transparência do manuseio dos objetos providos pelo CORBA, em muitos casos, não beneficia mas causa *overheads*;
- o modelo de comunicação gerente-agente é bem processado para o gerenciamento de uma grande quantidade de redes e elementos, os quais não são altamente distribuídos e dinamicamente reconfiguráveis. A comunicação entre os gerentes e agentes usando conexões estabelecidas são mais efetivas que usar um "corretor" de pedidos;
- dado a hierarquia de objetos de rede, ela é ineficiente para a CORBA modelar as operações de escopo e filtro, existentes no modelo de gerenciamento OSI.

Entretanto, este trabalho têm o objetivo de mostrar o que se pode conseguir em termos de gerenciamento de recursos, usando-se CORBA em TMN.

Formas de Integração CORBA e OSI

São mostradas a seguir quatro idéias sobre a integração de CORBA e OSI.

1) Idéia de Integração

A proposta do trabalho KON 96 seria a de combinar CORBA e o gerenciamento de rede TMN para prover uma rede integrada e a solução do gerenciamento de serviço.

O maior problema desta integração é a modelagem do objeto. Estas diferentes tecnologias utilizam diferentes acessos para modelarem os objetos gerenciados:

- no gerenciamento de rede OSI, um conjunto de gerenciamento de objetos é definido utilizando GDMO e o tipo de dados usado nos objetos são definidos em ASN.1. Há também um conjunto de funções de gerenciamento de sistemas definidas nos padrões OSI;
- CORBA têm sua própria técnica de modelar objetos. Os objetos são definidos em CORBA IDL. Um conjunto de serviços de objeto são também definidos como COSS (*Common Object Services Specification*) (Anexo II).

2) Idéia de utilizar *Gateway*

Uma maneira de promover o interfaceamento entre CORBA e TMN baseado em OSI, é construir um *gateway* entre o ambiente CORBA e o ambiente de gerenciamento de rede OSI. Neste interfaceamento, os agentes são desenvolvidos no ambiente OSI (objetos GDMO são definidos e é usado o protocolo de comunicação CMIP) e os gerentes são desenvolvidos no ambiente CORBA (as definições do objeto são baseados em IDL e o ORB é usado como o meio de comunicação). Há também um mapeamento um-a-um entre os objetos GDMO e IDL e entre as operações GDMO e os métodos de invocação ao objeto IDL [KON 96].

Os problemas encontrados nesta interface, é que o acesso inclui operações específicas, tais como a operação *get* com filtro, escopo e sincronização, e o manuseio das notificações e respostas. O padrão CORBA não suporta totalmente os conceitos de filtro e escopo, necessitando da introdução dos objetos representantes (*proxy*) para notificações [KON 96] e [MAZ 97], e o conceito de filtro pode ser gerenciado através da extensão do serviço de eventos CORBA [HAU 97].

3) Idéias de Complemento e de Reconciliação

As diferenças entre o modelo OSI e a arquitetura CORBA podem ser complementares, isto é, é possível explorar o fato das interfaces serem orientadas a comunicação no OSI e orientadas à programação na CORBA, usar esta interface de programação sobre os protocolos padronizados OSI. Mas isto não é uma tarefa fácil, pois ambos têm algumas características diferentes e complexas [FER 96].

Outra idéia é a reconciliação entre estes modelos através de mudanças em ambos, a fim de simplificar os mapeamentos entre eles. O grupo que vêm estudando a muito tempo esta possibilidade, o JIDM (*Joint Inter-Domain Management*), diz que este método não é viável, pois

foram empregados grandes investimentos para produzir cada um dos modelos já existentes, e encontraram dificuldades ao mapear as especificações em cada modelo, pois os mesmos são restringidos pelas próprias definições existentes [FER 96].

4) Idéia Alternativa

Devido às conclusões obtidas no decorrer deste trabalho, achamos que uma melhor alternativa é utilizar o gerenciamento de rede TMN, diretamente com as características apresentadas pela arquitetura CORBA, onde podemos enviar as operações de gerenciamento e notificações através de mensagens GIOP (Anexo V). Hoje já é possível encontrarmos ambientes de programação que oferecem soluções para as características OSI não fornecidas pela arquitetura CORBA, por exemplo o DCOM (*Distributed Component Object Model*) suporta objetos com múltiplas interfaces e fornece um método padrão para navegar entre elas, como também introduz a noção de um objeto e múltiplas interfaces representantes/*stubs* remotamente [CHU 97].

Em TMN, as funções de gerenciamento, tais como o gerenciamento de configuração e de falha, são realizados por blocos de funções específicas, os quais têm capacidades de processamento distribuído. Cada bloco de função pode ser considerado como uma unidade para a provisão da interface de gerenciamento TMN [PAR 96]. As capacidades de gerenciamento distribuído do bloco de função pode ser realizado utilizando-se a arquitetura de processamento distribuído da CORBA, onde o ORB pode ser designado para cada bloco de função, isto é , cada elemento constituinte do bloco de função é visto como um objeto ORB, que se comunica através do protocolo GIOP (figura 4.4).

O ORB provê a capacidade de processamento distribuído dentro de um bloco de função, bem como entre os blocos. O bloco de função OS utiliza a função de comunicação do objeto ORB, o qual provê invocação a operações de um objeto, a fim de realizar o gerenciamento sobre o elemento de rede.

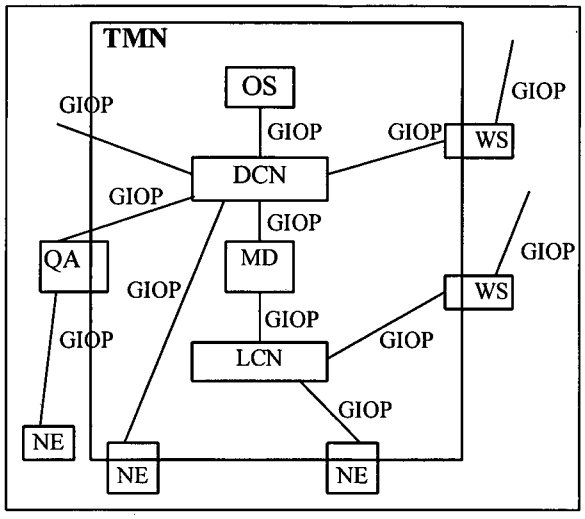

Figura 4-4 - Arquitetura Física TMN com CORBA

A figura 4.5 mostra as relações da arquitetura física TMN com CORBA, onde são utilizados ORBs - GIOPs para a realização do gerenciamento da rede de telecomunicações pela TMN.

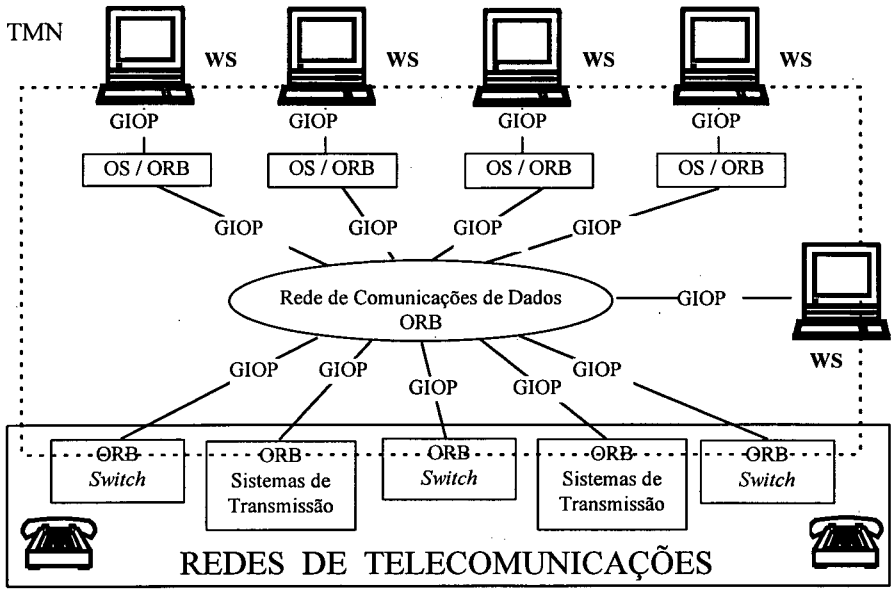

Figura 4-5 - Relações TMN para a Rede Física com CORBA

5. TMN baseado em CORBA

O propósito deste capítulo é visualizar uma arquitetura para um sistema TMN baseado em CORBA. A arquitetura proposta representa uma alternativa de implementação às interfaces abertas OSI e conceitos de gerenciamento de sistemas OSI, usando o paradigma CORBA.

5.1 Arquitetura Lógica

O objetivo desta seção é apresentar alguns princípios da arquitetura lógica de um sistema de gerenciamento na rede de telecomunicação baseada na arquitetura CORBA. Muitos destes princípios são idênticos aos princípios dos padrões TMN [TEL 96].

5.1.1 Classificação de Papéis

A classificação geral de papéis em sistemas dentro da TMN, como um sistema gerenciado e um sistema gerente, podem continuar dentro das atividades de gerenciamento pertencentes ao *Telecom Special Interest Group* (TSIG) - OMG. Um sistema pode assumir ambos os papéis simultaneamente, sendo que em muitas especificações de serviço, componentes dentro de sistemas gerenciados e de sistemas gerentes podem assumir diferentes papéis, dependendo da extensão do serviço específico. Um exemplo, é o serviço de eventos CORBA, onde um componente do sistema gerenciado pode assumir o papel de fornecedor de eventos e um componente do sistema gerente, o papel de consumidor de eventos.

A classificação de um sistema como gerente ou gerenciado não impede a distribuição do sistema. O modelo de referência apresentado na seção 5.2, em determinada circunstância, pode ser considerado como um *framework*, que é utilizado no desenvolvimento de sistemas gerentes, de sistemas gerenciados, ou ambos.

5.1.2 Conceitos de Objetos Gerenciados

O paradigma orientado a objeto é preservado na TMN, incluindo os conceitos de Classe do Objeto Gerenciado (MOC) e de Objetos Gerenciados (MO) (figura 5.1 e figura 5.2) [TEL 96]:

- em um sistema, objeto gerenciado é uma abstração de um ou diversos recursos de redes (entidades);
- os recursos podem ser físicos ou lógicos;
- um recurso de *software* é um exemplo de um recurso lógico que não têm representação física;
- uma unidade de *hardware* pode ser representada por um objeto gerenciado;
- cada objeto gerenciado têm uma única identidade do objeto;
- cada recurso da rede (entidade) têm uma única identidade do objeto;

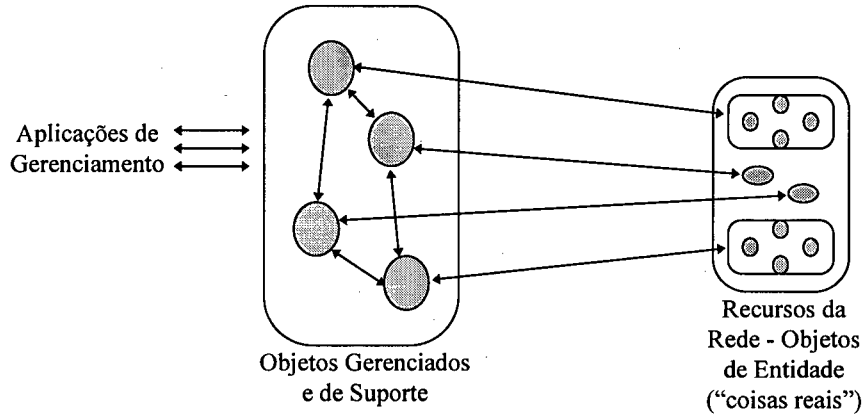

Figura 5-1 - Objetos Gerenciados e de Suporte

- um processo pode ser representado como um objeto gerenciado; quando um objeto gerenciado é criado, também o processo é criado e os dados presentes no banco de dados são copiados para dentro do processo;
- coleções de objetos gerenciados em um sistema são chamadas de MIBs (Base de Informação de Gerenciamento); o gerenciamento do sistema é feito pela manipulação de objetos na MIB, e toda atividade de gerenciamento é feita deste modo, com poucas exceções; um sistema de operação é usado para o gerenciamento, e este gerenciamento trabalha de um modo que o sistema de operação escreve dados dentro de objetos da MIB (com os resultados associados aos recursos da rede), e os dados são lidos pelas aplicações de tráfico, ou vice-versa;
- um objeto gerenciado pode gerar notificações, as quais são enviadas aos sistemas de operação; uma notificação é identificada pela identidade do objeto gerenciado, e a informação que é específica para o tipo de notificação, inclui a identidade do objeto que pertence ao recurso da rede (ou entidades);
- a MIB é considerada como parte do sistema gerenciado;
- objetos gerenciados contém atributos, relações e comportamentos;

- a MIB pode residir internamente ao elemento de rede ou implementada externamente, através de adaptadores Q; para o respectivo sistema de gerenciamento, não existe diferença em a MIB residir em um elemento de rede ou em adaptadores Q.

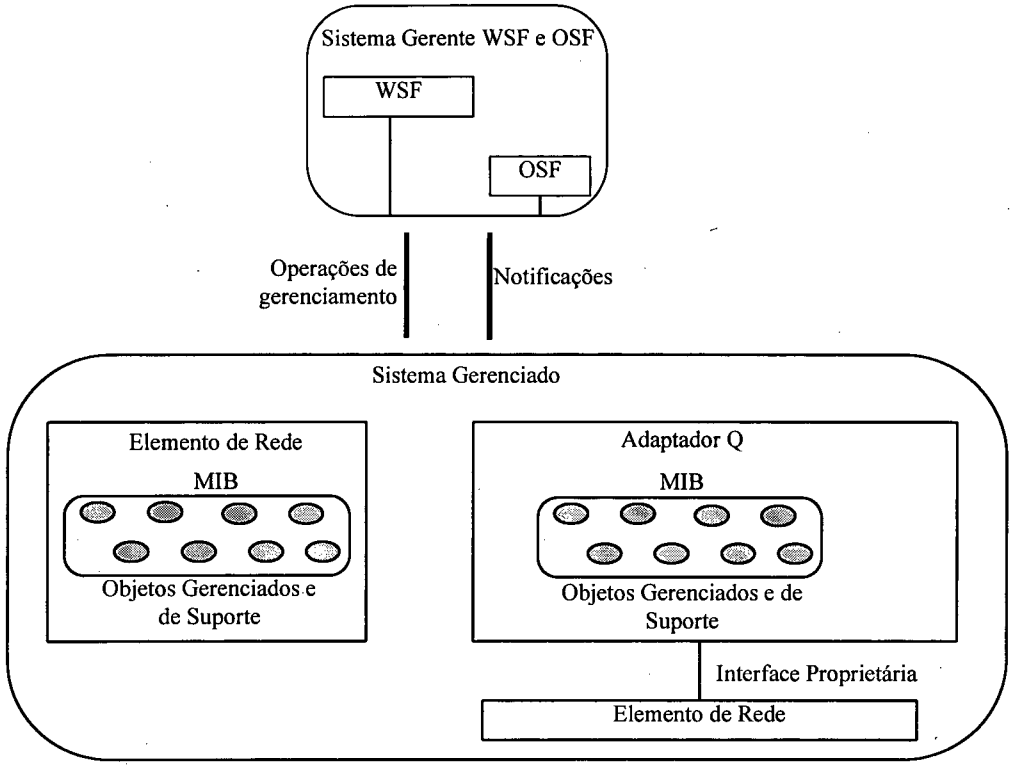

Figura 5-2 - Sistema Gerente e Gerenciado

5.1.3 Modelos de Objetos GDMO e CORBA

Nesta seção é abordada parte do trabalho do grupo *Joint X/Open* e *Network Management Forum* - JIDM (especificações de gerenciamento inter-domínio). O resultado deste trabalho são comparações entre GDMO e modelos de objeto CORBA.

A comparação foi baseada nos seguintes itens: objetivos dos modelos, interfaces, características dos objetos, especificação e instanciação do objeto, taxinomia do objeto, referência a objeto, resolução de endereço e seleção do objeto.

A análise do grupo concluiu que as noções básicas de objetos, taxinomia do objeto, atributos, operações, estado, comportamento, e encapsulamento nos dois modelos são virtualmente idênticas. O trabalho do grupo se estendeu com o mapeamento e *gateways* entre estes dois modelos. As maiores diferenças identificadas entre estes modelos são:

- **Eventos**: no GDMO, notificações são especificadas como parte da definição do objeto gerenciado; no padrão CORBA, os eventos tipados são especificados como parte da definição da interface do receptor. O que foi proposto recentemente, é manusear notificações GDMO via serviço de eventos CORBA, e vice-versa.
- **Réplicas Múltiplas**: explicitamente suportadas pelo CMIP, mas não pelo CORBA. O grupo JIDM recomenda um esquema por meio de invocações, inclui uma exceção, como uso de um objeto de réplicas múltiplas. Estas são retornadas, utilizando um canal de eventos OMG que é indicado no retorno da exceção.

5.2 A Arquitetura Aplicada

Esta seção descreve, resumidamente, um Modelo de Referência para o gerenciamento TMN baseado na arquitetura CORBA, como proposto pelo *Telecom Special Interest Group* (TSIG) - OMG [TEL 96].

5.2.1 Geral

A TMN tem adotado uma arquitetura de informação orientada a objeto baseada nos princípios de gerenciamento do sistema OSI. Os sistemas de gerenciamento alteram a informação em termos dos objetos gerenciados, e estes podem representar propriedades de um recurso como se estivessem verificando as funções de gerenciamento [M.3010].

A TMN não indica como cada sistema de gerenciamento poderia ser desenvolvido. O principal objetivo da TMN é categorizar os componentes do sistema de gerenciamento com uma perspectiva funcional, e definir as interfaces de comunicação entre os sistemas.

A tecnologia baseada no OMG-CORBA é dividida em três áreas:

⇒ no desenvolvimento de sistemas de computadores, os quais podem fazer internamente o uso substancial de implementações, tendo definidos os serviços e facilidades comuns CORBA;

⇒ facilitando a interoperabilidade entre sistemas de computadores: CORBA, em sua forma purista, é um componente central padrão que suporta a interface independente da linguagem de programação, o padrão de mapeamentos de linguagens, e tem suporte multifornecedor. As interfaces para funções de aplicação suportadas pelos sistemas operacionais em Linguagem de

Definição da Interface OMGs (IDL) aumenta o potencial de intertrabalho entre cada sistema operacional;

⇒ CORBA OMG tem a capacidade de prover uma implementação alternativa para o paradigma de comunicações OSI, para TMN. A migração das funções de gerenciamento de sistemas TMN para a arquitetura de gerenciamento de objeto OMGs, facilitará o desenvolvimento da arquitetura CORBA, puramente baseada em sistemas de gerenciamento.

A maioria dos trabalhos desenvolvidos pelo grupo TSIG - OMG, são apropriados para o reuso das pesquisas e especificações geradas pelas atividades com a ITU-T. As vantagens oferecidas pela computação de objetos distribuídos em geral, e especificadamente as de CORBA e da OMA (Arquitetura de Gerenciamento do Objeto), fazem com que haja um melhor desempenho das características de sistemas desenvolvidos neste contexto.

5.2.2 Uma *Framework* para Gerenciamento de Telecomunicações

O objetivo das atividades do grupo TSIG é desenvolver os serviços gerais oferecidos pela CORBA e as facilidades CORBA com a OMA OMGs. O conjunto completo de especificações dos serviços é fundamental para prover uma estrutura ao desenvolvimento das aplicações de gerenciamento de telecomunicações distribuídas.

Distribuição das Aplicações de Gerenciamento

Devido à experiência com padrões TMN, o TSIG assume que o ponto inicial de sua pesquisa é a padronização de Modelos de Informação para tecnologias e áreas de gerenciamento com **recursos** de rede.

Neste contexto, temos as seguintes considerações:

- MIBs podem ser vistas como parte do sistema gerenciado, indiferente da localização de sua implementação (no elemento de rede ou adaptador Q);
- a implementação da MIB para elemento de rede e recursos de rede inclui o comportamento (gerenciamento) dos objetos gerenciados e dos objetos de suporte.

O que está sendo distribuído, neste caso, são as aplicações de gerenciamento entre o sistema-gerente e sistema-gerenciado. O sistema-gerente provê, principalmente, as funções de gerenciamento para o nível da rede dos objetos gerenciados e de suporte, e também a capacidade para invocar os serviços/funções de gerenciamento para o nível de elemento de rede dos objetos

gerenciados e de suporte. Feito isso, o sistema-gerente conta com o próprio uso da interface de invocação dinâmica e estática do CORBA, para acessar os serviços do sistema gerenciado. Isso possibilita a liberação de um novo sistema-gerente a cada momento que uma modificação ocorreu na rede [TEL 96].

Esta arquitetura não impede que companhias de terceiros forneçam as funções de gerenciamento associadas aos elementos de rede de um fornecedor de equipamento particular. Por exemplo, esta arquitetura possibilita a um operador telecom comprar interruptores de um fornecedor de equipamentos, o sistema gerente de um segundo fornecedor, e os adaptadores Q requeridos para gerenciar os interruptores de um terceiro fornecedor.

Para ser possível implementar os princípios TMN, é necessária uma clara separação entre as funções de estação de trabalho (WSF) e as funções de suporte às operações (OSF). No sistema TMN baseado em CORBA, o sistema de gerenciamento é somente composto de WSF e OSF [TEL 96].

Uma importante vantagem do sistema baseado em CORBA é a simplicidade do ambiente de desenvolvimento, onde é apenas necessário conhecer a linguagem IDL (similar a C++). CORBA isola o sistema sendo desenvolvido, de sistemas operacionais e é independente de linguagens. Assim, por exemplo, teoricamente, a WSF pode ser realizada em um ambiente *Smalltalk/Windows 95*, a OSF em um ambiente *C/Windows NT*, e a QAF em um ambiente *C++/UNIX*. Também, CORBA 2.0 permite que diferentes implementações de CORBA sejam usadas para o sistema-gerente e sistema-gerenciado [TEL 96].

Outra importante vantagem, é a distribuição e transparência de localização do CORBA, onde os objetos gerenciados possam ser localizados em diferentes endereços. E os conceitos TMN (arquiteturas de informação, lógica e física) são mantidos em um sistema baseado em CORBA.

CORBA para Sistemas de Gerenciamento

A figura 5.3 apresenta "dois lados" da Arquitetura CORBA, a Interface do Usuário (Aplicações de Gerenciamento) e os Objetos Gerenciados. O que nos propomos estudar neste trabalho são os Objetos Gerenciados (Agente), podendo o lado da Interface do Usuário (Gerente) ser estudado em trabalhos futuros.

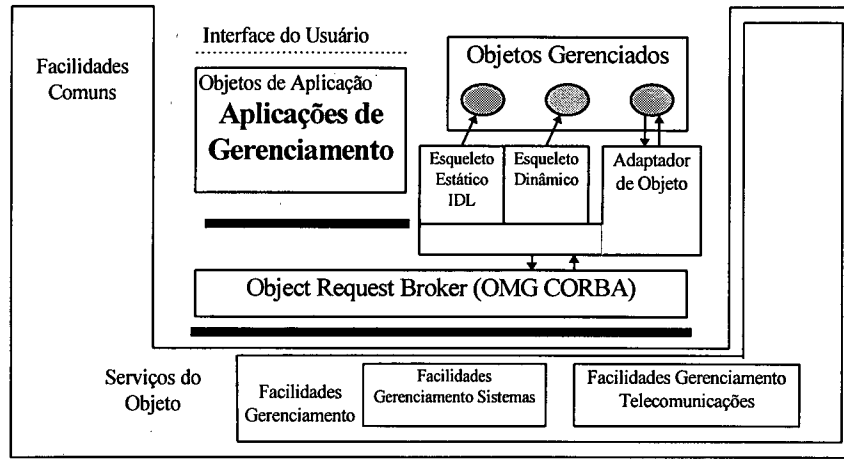

Figura 5-3 - Um Modelo de Referência para Gerenciamento.

O modelo de referência é designado para suportar discussões do TSIG - OMG, quando considerada a introdução de novas especificações tecnológicas. O modelo de referência é considerado flexível e pode ser estendido, como as atividades do grupo. O modelo é uma especialização do Modelo de Referência *X-Open*.

CORBA para Sistemas Gerenciados

Esta seção verifica o uso de CORBA como um candidato para a distribuição e definição de interface, suportando um mecanismo para a integração de sistemas de operação e componentes funcionais dentro de elementos de rede. Somente é considerado o gerenciamento no aspecto do sistema gerenciado.

O gerenciamento de sistemas OMG inicial assume que os objetos CORBA representarão recursos gerenciados em sistemas de computação. CORBA aproxima-se do gerenciamento de sistemas, usando IDL, como a ITU-T se aproxima, usando GDMO para definir os objetos gerenciados, representando os recursos em redes de telecomunicações.

A pesquisa realizada pelo Grupo de Gerenciamento do Conjunto de Inter-Domínios (JIDM), incluindo membros do NM Forum e X/Open, é baseada na equivalência do modelo OSI e os modelos de objeto CORBA/OMG [MAZ 97]. Esta equivalência faz com que CORBA seja um forte candidato para a implementação de uma nova geração de interfaces de gerenciamento de telecomunicações.

Interfaces para Sistemas Gerenciados

A figura 5.4 mostra as várias formas de interfaces para sistemas gerenciados, abordando agentes proprietários, agentes OSI e a possível forma de como utilizar a arquitetura CORBA para prover a interoperabilidade entre objetos de suporte e objetos gerenciados. Neste trabalho, examinamos o que pode ser o ponto de interrogação.

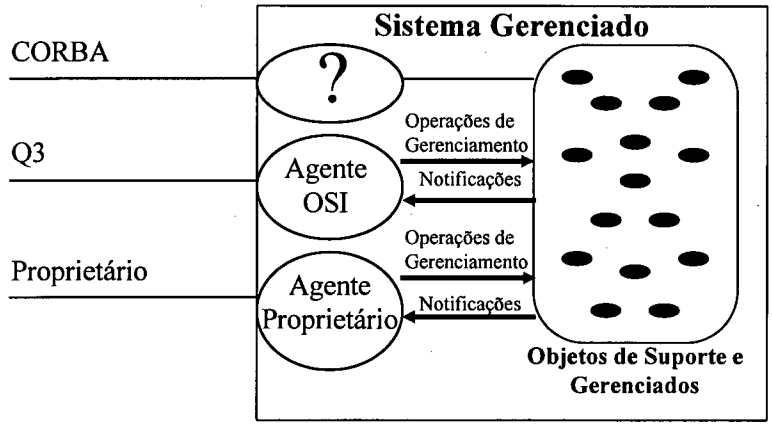

Figura 5-4 - Agentes alternativos do Sistema Gerenciado

A seguir são apresentadas, resumidamente, as interfaces para sistemas gerenciados: a proprietária, a ITU-T e a CORBA.

■ Interface Proprietária

Em sistemas de gerenciamento de telecomunicações, há um problema comum para projetistas, no que se refere ao manuseio de interfaces para dispositivos de rede. O problema está no ajuste da interface de comunicação com o modelo de informação comercializado. O objetivo desta especificação da interface é promover o desenvolvimento de aplicações de gerenciamento capazes de trabalharem com dispositivos de rede, originários de diferentes vendedores.

O *framework* oferecido pela OMG faz com que equipamentos de telecomunicações e de sistemas de gerenciamento suportem tarefas e facilidades comuns associadas ao monitoramento e controle dos dispositivos de rede. Com o desenvolvimento de interfaces básicas associadas com as novas facilidades, pode-se utilizar o encapsulamento oferecido pelas interfaces proprietárias, ocultando características dependentes de equipamentos comercializados e exportando serviços comuns.

■ Interface ITU-T/ISO

Houve um elevado progresso na ITU-T sobre a definição de modelos de informação para o gerenciamento de determinados tipos de redes, por exemplo ATM. É considerado que as interfaces Q3 TMN continuarão sendo utilizadas pelos equipamentos de telecomunicações comercializados, e procuradas pelos operadores em determinadas áreas de aplicação.

O grupo JIDM têm estudado *gateways* CMIP - CORBA, onde a distribuição da parte de gerenciamento (dentro do objeto gerenciado) é manejado pelo mecanismo CORBA.

■ Interface CORBA

Os objetos gerenciados são aceitos dentro da comunidade TMN como componentes centrais de sistemas de gerenciamento. Há uma necessidade em definir uma base comum da classe objeto gerenciado para toda CORBA baseada em sistemas de gerenciamentos. O uso de uma interface de objeto gerenciado com base comum proporcionará um elo entre vários grupos, definindo os serviços na área de gerenciamento baseado em CORBA.

Em um sistema TMN baseado em CORBA, cada classe de objetos gerenciados é definida sobre uma interface IDL, a qual descreve os atributos e operações para essa classe. Uma base gerenciada pela classe de objeto neste contexto, inclui todos os aspectos comuns básicos associados com qualquer objeto gerenciado. Isto pode incluir informações sobre a classe de objeto gerenciado, o identificador para uma instância do objeto gerenciado, as propriedades da instância do objeto gerenciado, etc.

A interface da classe dos objetos gerenciados básica pode então ser especificada pela classe dos objetos gerenciados no sistema. O sistema de gerenciamento deve fazer um uso eficiente de ambas as interfaces dinâmica e estática do CORBA, em uma ordem adequada para invocar serviços de objetos de suporte e objetos gerenciados.

Uma consideração importante é assegurar a compatibilidade da interface IDL estática quando mudanças são requeridas para uma classe de objeto gerenciado. É preciso então prevenir, modificando a aplicação de gerenciamento a cada momento em que o sistema gerenciado é modificado. O sistema de gerenciamento deve usar a Interface de Invocação Dinâmica (DII) para investigar as interfaces e invocar novas, ou modificar serviços oferecidos sobre os objetos gerenciados. Os serviços não modificados devem ser acessíveis diretamente, tanto para interfaces dinâmicas quanto para estáticas.

Uma vez que uma classe de objetos gerenciados base é consentida, então ela pode ser especializada para representar qualquer recurso gerenciado específico no sistema. Quando uma

classe de objetos gerenciados é definida, um conjunto limitado de operações de gerenciamento podem ser invocadas sobre o sistema de gerenciamento, fora do conhecimento de características específicas da classe de objetos gerenciados e especializados. Esta classe de objetos é recomendada no RFP(*Request For Proposals*), que é uma área adotada por todos os grupos, definindo os serviços para CORBA, baseada em sistemas de gerenciamento.

Os padrões de interfaces abertas OSI e ITU-T TMN não provêm uma especificação para comunicação entre Objetos Gerenciados e Objetos de Suporte. Somente especificam interfaces para Agentes. E com a arquitetura CORBA, é possível obter a interoperabilidade entre os Objetos de Suporte e os Gerenciados, através do adaptador de objeto CORBA (figura 5.5).

Considerando que em um sistema TMN baseado em CORBA, este seja o mecanismo provido pela interoperabilidade garantida entre objetos gerenciados e objetos de suporte, um fornecedor não é preparado para fornecer OSI ou Agentes Proprietários com seus sistemas gerenciados. Mas, por exemplo em organizações, este cenário pode facilitar a migração de gerenciadores OSI e OSs proprietários para CORBA [TEL 96].

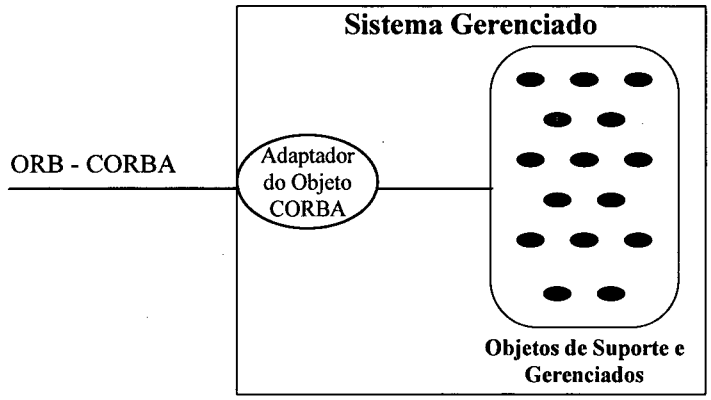

Figura 5-5 - Agente CORBA

O adaptador de objeto CORBA serve como ligação entre as implementações de Objetos Gerenciados CORBA e o próprio ORB do CORBA. O adaptador de objeto é um objeto que adapta a interface de outro objeto a uma interface esperada pela parte chamadora.

6. Uma Experiência de Supervisão de Alarmes TMN com CORBA

Este capítulo tem por objetivo apresentar Supervisão de Alarmes, a partir de conceitos do modelo OSI obtidos da norma Q.821, e apresentar as cinco funções relativas a Supervisão de Alarmes: relatório de alarmes, relatório de sumarização de alarmes; critério de evento de alarme; gerenciamento de indicação de alarmes; e, controle de log.

6.1 *Supervisão de Alarmes* (Alarm Surveillance) em TMN

A **Supervisão de Alarmes** é definida como um conjunto de funções que habilitam a monitoração e/ou interrogação de elementos da rede de telecomunicações quanto a eventos ou condições de alarme [Q821]. Uma informação de alarme é gerada por um NE (*Network Element*) sob a detecção de uma condição anormal ou de falha. Tais alarmes podem ser relatados no instante de sua ocorrência, armazenados para acesso futuro, ou ambos.

A recomendação ITU-T Q821 descreve a estrutura básica de informação e mensagens para a supervisão de alarmes genéricos, fornecendo um modelo de informação para objetos de suporte e um conjunto de mensagens para aplicações em TMN ou entre TMNs relacionadas.

As classes de objetos gerenciados e objetos de suporte relativos à Supervisão de Alarmes da TMN são mostradas na figura que segue [LAV 96].

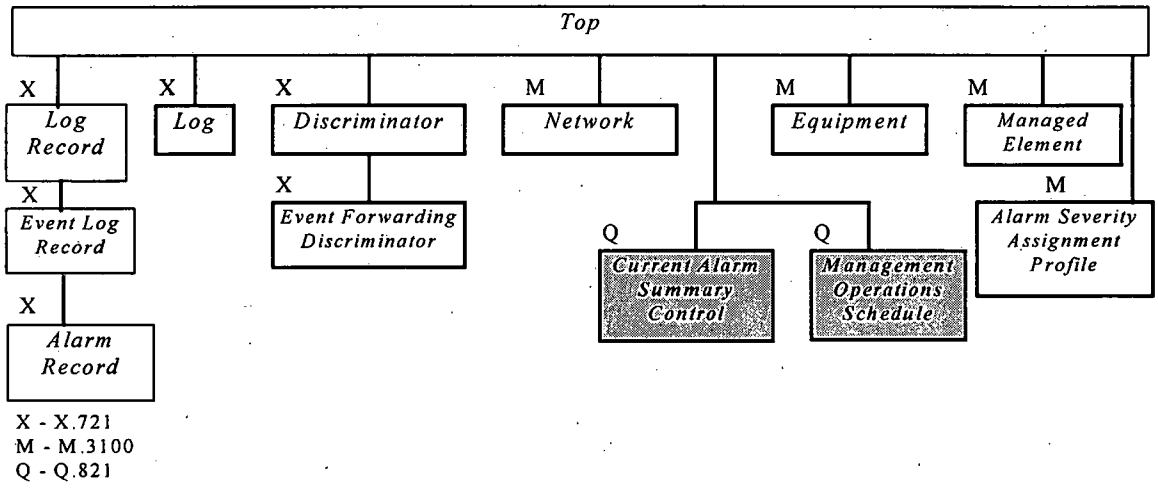

Figura 6-1 - Hierarquia de Herança das Classes de Objetos [M3100]

As classes de objetos gerenciados relacionados à Supervisão de Alarmes para TMN são descritas na recomendação ITU-T Q.821 [Q821], são elas: *Current Alarm Summary Control* e *Management Operations Schedule*, que nos fornecem a sumarização de alarmes. Para que a supervisão de alarmes seja completa, é preciso especificar as classes *Log Record, Alarm Record, Event Log Record, Log, Discriminator* e *Event Forwarding Discriminator*, obtidas da recomendação X.721 [X721]. Além destas classes, é necessário especificar as classes *Network, Managed Element* e *Equipment*, obtidas da recomendação M.3100 [M3100], que representam os elementos de rede de telecomunicações.

Na figura 6.2 é mostrada a **hierarquia de nomeação** (relacionamentos) dos objetos gerenciados, apresentada pela norma M3100. É a partir destes relacionamentos que é formada a árvore de herança (objetos gerenciados e objetos de suporte, como mostrado na figura 6.1).

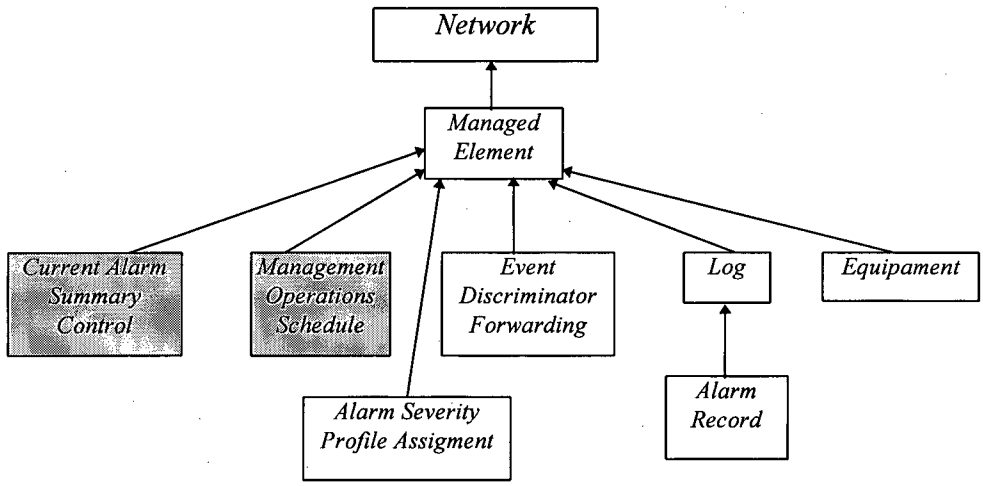

Figura 6-2 - Hierarquia de Nomeação das Classes de Objetos [M3100]

Classe *Log*

A classe *Log* é usada para definir o critério para controlar o armazenamento da informação em um sistema aberto. Usado para armazenar os relatórios de eventos que chegam e notificações de sistema local.

Classe Log Record

A classe *Log Record* é usada para definir os registros (a informação armazenada) contidos em um objeto gerenciado *log*.

Classe Event Log Record

A classe *Event Log Record* é usada para definir a informação armazenada em um *log* como resultado do recebimento de notificações ou relatórios de eventos. Esta é uma super classe da qual são derivados os relatórios para especificar os tipos de eventos.

Classe Alarm Record

A classe *Alarm Record* é usada para definir a informação armazenada em um *log* como resultado do recebimento de notificações ou relatórios de alarme.

Classe Discriminator

A classe *Discriminator* é usada para definir o critério para controlar os serviços de gerenciamento.

Classe Event Forwarding Discriminator

A classe *Event Forwarding Discriminator* é usada para definir as condições que devem ser satisfeitas pelos relatórios de eventos potenciais antes que o relatório de evento seja enviado para um destino particular.

Classe *Network*

A classe *Network* é usada para definir uma coleção de objetos de elementos de telecomunicações inter-conectados capazes de trocar informações.

Classe Equipment

A classe *Equipment* é uma classe de objetos gerenciados que representa componentes físicos de um elemento gerenciado (*Managed Element)* incluindo os RPUs (*Replaceable Units*).

Classe Managed Element

A classe *Managed Element* representa equipamentos de telecomunicações ou entidades TMN da rede de telecomunicações que executam funções de elementos gerenciados, isto é, provêem suporte e/ou serviços ao assinante.

Classe Alarm Severity Assigment Profile

A classe *Alarm Severity Assigment Profile* especifica a indicação de severidade do alarme para objetos gerenciados.

Posteriormente, as classes *Current Alarm Summary Control* e *Management Operations Schedule* serão apresentadas na seção 6.1.3, em Funções de Relatório de Sumarização de Alarme.

A seguir, são apresentados os serviços e funções da monitoração e/ou interrogação de elementos da rede de telecomunicações, que é a essência da Supervisão de Alarmes [LAV 96 e Q821].

6.1.1 Serviços e Funções

Um Serviço de Gerenciamento TMN é uma área da atividade de gerenciamento que fornece suporte para um dos aspectos de Operação, Administração e Manutenção (OAM) da rede que está sendo gerenciada; além disso, ele é parte da especificação da interface TMN [M3200].

Uma Função de Gerenciamento é a menor parte de um serviço observada pelo usuário deste serviço; normalmente ela consiste de uma seqüência de ações sobre um objeto gerenciado [M3400].

Para que a TMN possa executar a Supervisão de Alarmes, os NEs devem permitir:

- ⇨ monitoração de condições de alarmes em tempo quase real ou escalonado;
- ⇨ consultas das condições de alarmes do NE;
- ⇨ relatório de condições de alarmes;
- ⇨ armazenamento e recuperação de informação histórica de alarmes.

As funções da Supervisão de Alarmes necessárias para satisfazer as exigências acima são:

- ⇒ relatório de alarmes;
- ⇒ relatório de sumarização de alarmes;
- ⇒ critério de evento de alarme;
- ⇒ gerenciamento de indicação de alarmes;
- ⇒ controle de log.

Cada função será explicada nas seções posteriores.

Os serviços definidos para suportar as funções da Supervisão de Alarme foram agrupados em unidades funcionais para permitir a negociação do seu uso durante o estabelecimento de uma associação [Q821]. A tabela abaixo fornece os serviços, funções, e objetos gerenciados associados à Supervisão de Alarmes.

Unidade Funcional	Serviço(s)	Classe(s) de Objeto(s)	Função(s)
Kernel	*Alarm Reporting*	*Event Forwarding Discriminator*	*Report Alarm*
Basic Alarm Report Control	*Suspend/Resume Alarm Reporting*	*Event Forwarding Discriminator*	*Inhibit/Allow Alarm Reporting*
Enhanced Alarm Report Control	*Initiate/Terminate Alarm Reporting* *Set/Get Event Forwarding Discriminator*	*Event Forwarding Discriminator*	*Condition Alarm Reporting* *Route Alarm Report*
Alarm Report Retrieval	*Alarm Report Retrieving*	*Log Alarm Record*	*Request Alarm Report History*
Alarm Report Deletion	*Alarm Report Deleting*	*Log Alarm Record*	*Delete Alarm Report*
Current Alarm Summary Reporting	*Current Alarm Summary Reporting*	*Management Operations Schedule* *Currente Alarm Summary Control*	*Report Current Alarm Summary*
Basic Management Operations Scheduling	*Suspend/Resume Management Operations Schedule*	*Management Operations Schedule*	*Inhibit/Allow Current Alarm Summary*
Enhanced Management Operations Scheduling	*Initiate/Terminate/Set/Get Management Operations Schedule*	*Management Operations Schedule*	*Schedule Current Alarm Summary* *Route Current Alarm Summary* *Request Current Alarm Summary Schedule* *Request Current Alarm Summary Route*
Current Alarm Summary Reporting Control	*Initiate/Terminate/Set/Get Current Alarm Summary Control*	*Current Alarm Summary Control*	*Schedule Current Alarm Summary* *Request Current Alarm Summart Schedule*
Current Alarm Summary Retrievel	*Retrieve Current Alarm Summary*	*Current Alarm Summary Control*	*Request Current Alarm Summary*
Alarm Event Criteria Management	*Initiate/Terminate/Set/Get Alarm Severity Assigment Profile*	*Alarm Severity Assignment Profile*	*Condition Alarm Event Criteria* *Request Alarm Event Criteria*
Alarm Indication Management	*Inhibit/Allow Audible and Visual Local Alarms, Reset Audible Alarm*	*Managed Element or its subclass*	*Inhibit/Allow Audible and Visual Local Alarms,* *Reset Audible Alarm*
Basic Log Control	*Suspend/Resume Logging*	*Log Alarm Record*	*Inhibit/Allow Logging*
Enhanced Log Control	*Initiate/Terminate Log* *Set/Get Log*	*Log Alarm Record*	*Condition Logging* *Request Log Condition*

Tabela 6-1 - Unidades Funcionais, Serviços, Objetos e Funções da Supervisão de Alarme [Q821]

6.1.2 Funções de Relatório de Alarmes

As funções de relatório de alarmes são:

- ⇨ emitir relatório de alarmes e informação relatada;
- ⇨ controle dos relatórios de alarmes e informação relatada.

Estas funções são realizadas segundo um modelo padrão. Este modelo determina que se existe uma condição de alarme, uma notificação de alarme é emitida pelo elemento de rede e o *status* de alarme *Active-Reportable* é atribuído ao objeto gerenciado afetado. Da mesma forma, quando a condição de alarme deixa de existir, uma notificação de alarme é emitida e o *status* de alarme *Cleared* é atribuído ao objeto gerenciado. Um terceiro *status* de alarme *Active-Pending* é atribuído ao objeto gerenciado quando alguma condição de alarme foi reconhecida, mas não teve duração suficiente para ser tratada como não-transitória.

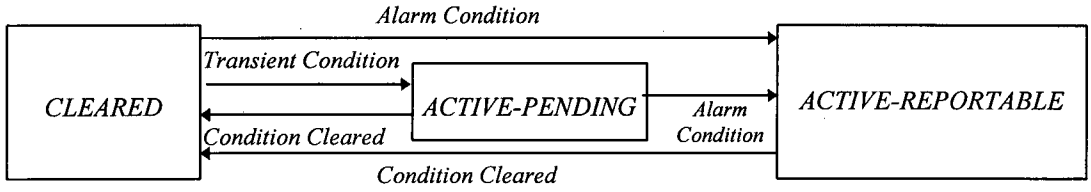

Figura 6-3- *Status* relatados para o Alarme *Status* dos Objetos Gerenciados [M3200]

Um NE deve fornecer um mecanismo para o controle de notificações, ou seja, uma condição de alarme deve resultar em um relatório de alarme. Para isto, foram definidas as seguintes funções de gerenciamento:

- Relatório de Alarme (*Report Alarm*);
- Rota do Relatório de Alarme (*Route Alarm Report*);
- Requisição da Rota do Relatório de Alarme (*Request Alarm Report Route*);
- Relatório da Condição de Alarme (*Condition Alarm Reporting*);
- Requisição da Condição de Controle de Relatório de Alarme (*Request Alarm Report Control Condition*);
- Permitir/Inibir Relatório de Alarme (*Allow/Inhibit Alarm Reporting*);
- Requisição do Histórico de Alarme (*Request Alarm History*).

6.1.3 Funções de Relatório de Sumarização de Alarmes

As funções de Relatório de Sumarização de Alarmes são:

- ➔ emitir o relatório de sumarização das condições de alarme corrente dos objetos gerenciados especificados;
- ➔ controle do relatório de sumarização das condições de alarme corrente dos objetos gerenciados especificados.

Os alarmes são recebidos de objetos específicos e satisfazem condições definidas. O Escalonador de Operações de Gerenciamento requisita a recuperação de um relatório de sumarização de alarme corrente. O Controle de Sumarização de Alarme Corrente responde fornecendo o relatório de sumarização de alarme.

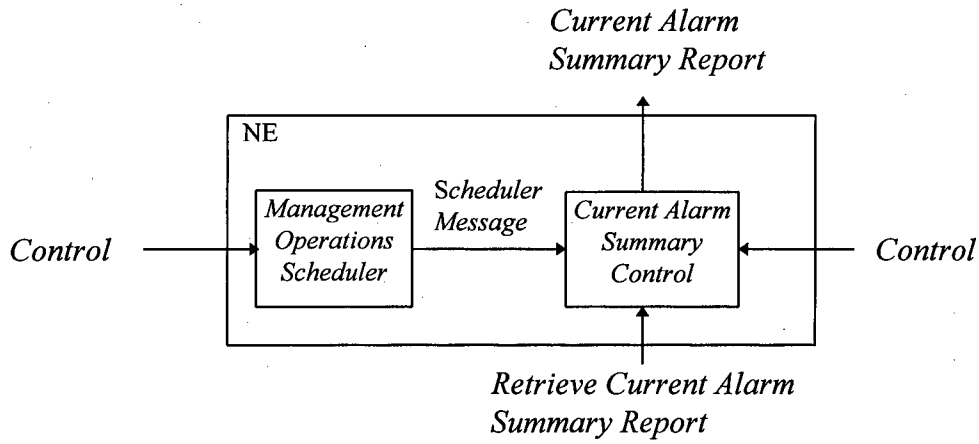

Figura 6-4 - Relatório de Sumarização de Alarmes Corrente [M.3200]

As funções de gerenciamento necessárias para permitir a realização da Função de Relatório de Sumarização de Alarmes Corrente são:

- ➔ o Relatório de Sumarização de Alarme Corrente (*Report Current Alarm Summary*);
- ➔ a Rota de Sumarização de Alarme Corrente (*Route Current Alarm Summary*);
- ➔ a Requisição da Rota de Sumarização de Alarme Corrente (*Request Currente Alarm Summary Route*);
- ➔ o Escalonamento de Sumarização de Alarme Corrente (*Schedule Current Alarm Summary*);
- ➔ a Requisição do Escalonador de Sumarização de Alarme Corrente (*Request Current Alarm Summary Scheduler*);

➔ Permite/Inibe a Sumarização de Alarme Corrente (*Allow/Inhibit Current Alarm Summary*);

➔ a Requisição de Sumarização de Alarme Corrente (*Request Current Alarm Summary*).

Das classes de objetos gerenciados e de suporte (figuras 6.1 e 6.2), escolhemos as classes abaixo, no sentido de se poder comparar com o trabalho realizado em GDMO e ASN.1 [LAV 96].

Classe de Objeto *Current Alarm Summary Control*

A classe de objeto *Current Alarm Summary Control* é uma classe de objetos de suporte que fornece o critério para a geração do relatório de sumarização de alarme corrente [Q821]. Um objeto é incluído em um relatório de sumarização de alarmes correntes se:

- o objeto está incluído na lista de objetos (*ObjectList*);
- o objeto tem um *status* de alarme que consta na lista de *status* de alarme (*AlarmStatusList*);
- o objeto tem um alarme com um percentual de severidade que consta na lista de severidade de alarme (*PerceiveSeverityList*) e uma causa provável que consta na lista de causa provável (*ProvableCauseList*).

Classe de Objeto *Management Operations Schedule*

A Classe de objeto *Management Operations Schedule* é uma classe de objetos de suporte que fornece a facilidade de escalonar um serviço para ocorrer periodicamente [Q821]. O período é definido como um intervalo de tempo, onde o tempo inicial indica a primeira ocorrência do serviço e o tempo final indica o término do intervalo.

A relação destas duas classes ocorre quando a classe *Management Operations Schedule* requisita o relatório de sumarização de alarmes para a classe *Current Alarm Summary Control.* Com o envio da mensagem, uma ação *Retrieve Current Alarm Summary Control* é executada, e em resposta a esta ação a classe *Current Alarm Summary Control* envia o relatório (*current alarm summary report*) para a classe M*anagement Operations Schedule*.

6.1.4 Funções de Critério de Evento de Alarmes

A função de Critério de Evento de Alarmes realiza o gerenciamento do critério usado para determinar quando uma condição deve ser considerada um alarme.

As funções de gerenciamento que permitem a realização da Função de Critério de Evento de Alarmes são Critério de Condição de Evento de Alarme (*Condition Alarm Event Criteria*) e Requisição do Critério de Evento de Alarme (*Request Alarm Event Criteria*).

6.1.5 Função de Gerenciamento de Indicação de Alarme

A Função de Gerenciamento de Indicação de Alarme realiza o controle de indicações de alarmes audíveis.

As funções de gerenciamento que permitem a realização da Função de Gerenciamento de Indicação de Alarme são Inibir/Permitir Indicações de Alarmes Audíveis/Visuais (*Inhibit/Allow Audible/Visual Alarm Indications*) e Habilitar Alarmes Audíveis (*Reset Audible Alarms*).

6.1.6 Funções de Controle de Log

As Funções de Controle de *Log* são o armazenamento de alarmes de um NE e a recuperação do histórico de alarmes de um NE.

Para que a Supervisão de Alarmes realize suas funções, é necessário preservar informação sobre relatórios de alarmes que tenham ocorrido a partir dos objetos gerenciados. Segundo o modelo do Controle de *Log*, os registros de alarme do *Log*, contém a informação de seus correspondentes relatórios de alarmes.

As funções de gerenciamento necessárias para realizar as Funções de Controle de *Log* são Permitir/Inibir Armazenamento (*Allow/Inhibit Logging*), Condição de Armazenamento (*Condition Logging*) e Requisição de Condição de Log (*Request Log Condition*).

6.2 Resolvendo TMN com CORBA

Para mostrar como se pode realizar Supervisão de Alarmes em TMN com CORBA, tomamos o seguinte exemplo:

Podemos executar supervisão de alarmes com CORBA através de um PC (o qual contém o gerente OS TMN CORBA), e uma estação SUN com o sistema operacional SOLARIS (que contém o agente adaptador de objeto CORBA, e o elemento de rede com objetos gerenciados). As operações de gerenciamento são enviadas do gerente OS para o agente em CORBA, e notificações de gerenciamento são enviadas do agente em CORBA para o OS da TMN, através do protocolo de comunicação GIOP, que está presente no ORB do PC e no ORB da SUN.

Para exemplificarmos a utilização de Supervisão de Alarmes TMN com CORBA, especificamos um comutador ATM, como elemento de rede (NE) (seção 7.3).

A TMN pode ser vista como uma rede de telecomunicações que funciona sobre uma rede ATM. Neste caso, estamos supondo que uma rede ATM é a rede gerenciada por uma rede TMN (capítulo 7).

A figura 6.5 esquematiza a experiência TMN com CORBA pretendida neste trabalho.

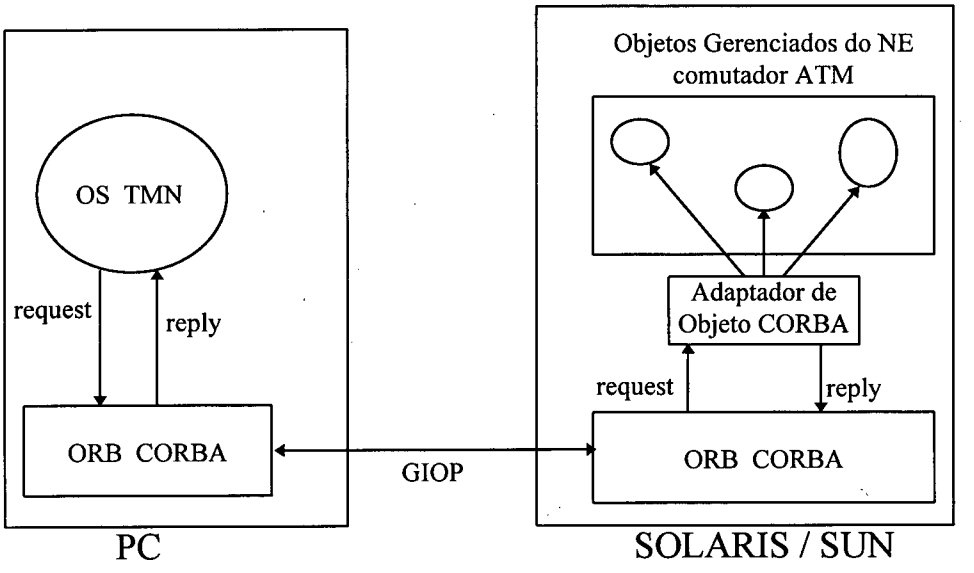

Figura 6-5 - Relação TMN com CORBA e elemento de rede ATM

7. Implementando Supervisão de Alarmes em TMN com CORBA

Este capítulo investiga a utilização da arquitetura CORBA como plataforma de desenvolvimento do serviço de supervisão de alarmes TMN. São enfocados a especificação e a implementação de objetos gerenciados e objetos de suporte da supervisão de alarmes (vistos no capítulo 6), e como exemplo são especificados os objetos gerenciados de um elemento de rede ATM.

7.1 Ferramenta Java

A linguagem de programação Java se parece com o C++, tanto que é possível realizar os mesmos tipos de tarefas e solucionar os mesmos tipos de problemas como é feito nas linguagens de programação C e C++, e ainda oferece mais as seguintes facilidades [VAL 96 e LEM 97]:

- Feita principalmente para Internet (ambiente *World Wide Web*), com o objetivo de a transformar em um megaprocessador através de *applets* que podem ser executados de qualquer máquina, mesmo remotamente. Este *applets* são colocados nas páginas HTML, e mesmo assim, Java é independente da Internet. Os *applets* são pequenos aplicativos específicos, gerados a partir da linguagem Java. Assim, uma vez que um computador possua um *software* Java, ele pode executar qualquer aplicação que venha através da rede.
- É uma ferramenta orientada a objetos que agrupa programas e dados em funções que retratam o mundo real. A orientação a objetos é importante pois facilita a definição clara de interfaces, criação de programas modulares e flexíveis, e torna possível a produção de código reutilizável.
- A interpretação de programas faz com que estes sejam executados mais vagarosamente do que no esquema tradicional de execução de código binário. No entanto, oferece vantagens tais como, o tamanho compacto dos programas, a impossibilidade da proliferação de vírus e falsificações, e a possibilidade de computadores até então incompatíveis, executarem o mesmo *software*. Oferece segurança, pois foi feito para ser utilizado em ambientes

interconectados, e para isso, utiliza a técnica de autenticação baseada em criptografia de chave-múltipla.

- Java possui uma extensa biblioteca de rotinas para copiar facilmente protocolos TCP/IP (*Transport Control Protocol / Internet Protocol*), tais como HTTP (*Hypertext Transfer Protocol*) e FTP (*File Transfer Protocol*). Para abrir e acessar objetos através da rede, as aplicações Java utilizam URLs (*Universal Resource Locator*). E para a auto adequação de novos protocolos, é necessário que estes estejam escritos em Java.
- Java têm um coletor de lixo automático que simplifica a tarefa de programação e também reduz consideravelmente os erros.
- Pode-se produzir *softwares* que rodem em todas as plataformas através do compilador Java, o qual gera instruções de código *byte* (*bytecode*) - independente da arquitetura de computador. As instruções são projetadas para serem tão fáceis de interpretar em qualquer máquina quanto para serem traduzidas no código da máquina instantaneamente. A desvantagem da utilização do código *byte* está na velocidade de execução, pois são processados pelo interpretador, o que se diferencia de programas específicos ao sistema que são executados no *hardware* para o qual são compilados.
- É uma linguagem de programação para aplicações distribuídas. Onde é permitido adicionar novos tipos de conteúdos às páginas (Internet), com adição de dados e códigos, sem a necessidade de se esperar uma nova liberação (*release*) para o tratamento das informações a respeito do conteúdo.
- Grande parte das complexidades de C e C++ foram tornadas transparentes ao usuário ou excluídas do Java, tornando-a mais simples. Não há ponteiros no Java, nem ponteiro aritmético e as seqüências de caracteres e matrizes são objetos reais no Java.

A ferramenta Java utilizada neste trabalho é o *VisiBroker For Java*^{TM} *ORB* da *VisiGenic Software*. Esta ferramenta roda em diversos ambientes, dentre eles no *Windows NT* e *SOLARIS*; facilita o desenvolvimento e extensão de aplicações baseadas em objetos distribuídos, as quais são flexíveis e de fácil manutenção; implementa os padrões IIOP e CORBA 2.0 desenvolvidos pela OMG, e em um ambiente de computação distribuída, pode interoperar com outros ORBs complacentes ao CORBA 2.0.

7.2 Especificação IDL dos Objetos Gerenciados

Um ambiente de gerenciamento é composto por gerentes, agentes e objetos gerenciados. Um gerente transmite operações de gerenciamento aos agentes a fim de obter informações atualizadas sobre os objetos gerenciados e controlá-los. Um agente recebe as operações de gerenciamento emitidas pelo gerente e executa as ações necessárias sobre os objetos gerenciados.

No trabalho em questão chamamos de **objetos de suporte** os objetos gerenciados que necessitam de suporte de redes e **serviços** de gerenciamento de telecomunicações, por exemplo, registro de evento *Log* [X721], alarme de registro [X721], *current alarm summary control* [Q821] e *management operations schedule* [Q821] (classes de supervisão de alarmes). Os **objetos gerenciados** são abstrações de **recursos** de telecomunicações (exemplos: redes, serviços e sistemas), com o objetivo de gerenciamento, e os recursos são independentes de suas necessidades de gerenciamento; um exemplo de recursos de telecomunicações modelados como classes de objetos gerenciados é o Ponto de Terminação [M3100], utilizado pelo comutador ATM.

O objetivo deste trabalho foi obter uma primeira experiência em suportar gerência distribuída em uma rede TMN através de CORBA. Como exemplo, consideramos a parte de TMN que trata de Supervisão de Alarmes, que é baseada na recomendação Q.821 da ITU-T [Q.821]. Desta norma foram extraídas algumas classes de objetos que foram especificadas e implementadas, e para isso, utilizamos o documento oficial para CORBA 2.0 [COR 95].

7.2.1 Utilização de um *framework* e as Classes de Supervisão de Alarmes com CORBA

Para ser possível a especificação e implementação das classes de sumarização de alarmes com CORBA, é preciso ter como base, algumas funções da Classe TOP e de algumas classes que formam a supervisão de alarmes (Capítulo 6), as quais devem ser especificadas em IDL e implementadas em Java. Devido a esta necessidade, utilizamos um *framework* [CHAV 97] que oferece interfaces e operações para a realização de nossa experiência de Supervisão de Alarmes com CORBA.

Especificação IDL do *framework*

Neste *framework*, o papel do gerente é de apenas manipular as notificações de eventos, que no nosso trabalho, é o relatório de sumarização de alarmes. O papel do agente é de associar nomes a objetos, de criar ou destruir objetos, e o de efetuar ou desfazer o registro de classes de objetos. A classe *ClassImplementor* é chamada de fábrica (*factory*), utilizada para criar instâncias da classe, como também novas classes e objetos. Os objetos gerenciados são organizados através da representação de agregação (*containment*) e organização de nomes através de RDN (*Relative Distinguished Name*); e seus atributos e operações são específicas para cada classe ou objeto.

Uma interface *Iterator* permite iterar os elementos de uma coleção. Normalmente, usa-se um objeto *Iterator* em métodos que retornam um conjunto de elementos. O *iterador* permite a navegação nessa coleção de elementos, e permite a iteração de coleções com os mais variados tipos de dados (int, struct, objetos, enum).

A interface *Container* é a interface ancestral de *List* e *PropertyList*.

A interface *List* é uma lista para qualquer tipo de dados (any), útil para declarar atributos com valores compostos.

A interface *PropertyList* é uma lista de propriedades, onde uma propriedade é um par nome-valor, e o valor pode ser de qualquer tipo (any). A sua importância está em permitir que vários parâmetros sejam passados a um objeto, sem obedecer um número fixo (como é exigido nos métodos em IDL).

A declaração prévia da interface *ManagedObject* e da interface *EventForwarder*, é necessária para que o compilador saiba que o nome *ManagedObject* (e o *EventForwarder*) será definido posteriormente, e assim, por exemplo, o campo *source* da struct event, não provocará erro.

A estrutura *enum Operator* é uma estrutura que representa um critério a ser aplicado a um objeto CORBA. Como exemplo, o critério representa para um objeto x, o seguinte: x.className = "Keyboard". Desta forma, pode-se determinar critérios para repasse de eventos. Assim, um gerente interessado apenas nos eventos gerados por teclados usaria um critério semelhante ao exemplificado.

A interface *ManagedObject* é a classe básica de todos os objetos gerenciados, os quais são organizados através da representação de agregação (*containment*) e organização de nomes

através de RDN (*Relative Distinguished Name*); e seus atributos e operações são específicas para cada classe ou objeto.

A interface *ClassImplementor* é utilizada para todas as implementações de classes. É esta a classe que funciona como fábrica para criar instâncias de classes, como também novas classes e objetos.

A interface *Agent* é utilizada como uma interface para acesso a um agente. É esta interface que pede para a *ClassImplementor* criar um objeto/classe, manipula a hierarquia distribuída, associa nomes a objetos (de criar ou destruir objetos), e o de efetuar ou desfazer o registro de classes de objetos;

A especificação em IDL do *framework* de gerenciamento é apresentada no Anexo I.

Especificação IDL dos Objetos de Suporte para a Supervisão de Alarmes

Das classes apresentadas na figura 6.1 do capítulo 6, algumas foram especificadas em IDL e implementadas em Java, sob as características e funcionalidade da arquitetura CORBA. No sentido de se poder comparar com o trabalho realizado em GDMO e ASN.1 [LAV 96], pretende-se implementar as classes relacionadas à Função de Relatório de Sumarização de Alarmes (capítulo 6, seção 6.1.3). Estas classes são representadas pelas interfaces IDL e explicadas a seguir.

Classe *Log*

A classe *Log* é usada para definir o critério para controlar o armazenamento da informação em um sistema aberto. Usado para armazenar os relatórios de eventos que chegam e notificações de sistema local. Têm como resposta à classe *currentAlamSummaryControl* os alarmes armazenados (*Alarms*).

A estrutura *logRecord* é usada para definir os registros contidos em um objeto gerenciado *log*. Ela representa a informação armazenada nos *log*s.

Classe *Event Forwarder*

Quando o objeto *Manager* cria um evento, ele solicita o tratamento do evento à classe *Event Forwarder*, que avalia o critério para o evento (para cada discriminador - a classe *event discriminator forwarding*).

Classe *Event Discriminator Forwarding*

A classe *Event Discriminator Forwarding* é usada para definir as condições que devem ser satisfeitas pelos relatórios de eventos antes que o relatório de evento seja enviado para um destino particular, ou para o *Manager*.

Classe *Network*

A classe *Network* é usada para definir uma coleção de objetos de elementos de telecomunicações inter-conectados capazes de trocar informações. Esta classe representa a própria rede, por exemplo, a rede inf e a rede Irg.

Especificação das Classes para Sumarização de Alarmes

Os alarmes são recebidos de objetos gerenciados específicos, por exemplo dos objetos gerenciados do comutador ATM. A classe *Management Operations Schedule* requisita o relatório de sumarização de alarme, enviando uma mensagem à classe *Current Alarm Summary Control*, a qual fornece como resposta o relatório (figura 7.1).

Classe CURRENT ALARM SUMMARY CONTROL

É uma classe de objetos de suporte que fornece o **critério** para a geração do relatório de sumarização de alarme.

Um objeto é incluído neste relatório se:

- o objeto estiver incluído na lista de objetos, a *objectList*;
- o objeto tiver um status de alarme que conste na lista de status de alarme, a *alarmStatusList*;
- o objeto tiver um alarme com um percentual de severidade que conste na lista de severidade de alarme, a *perceivedSeverityList*;
- o objeto tiver uma causa provável que conste na lista de causa provável, a *probableCauseList*.

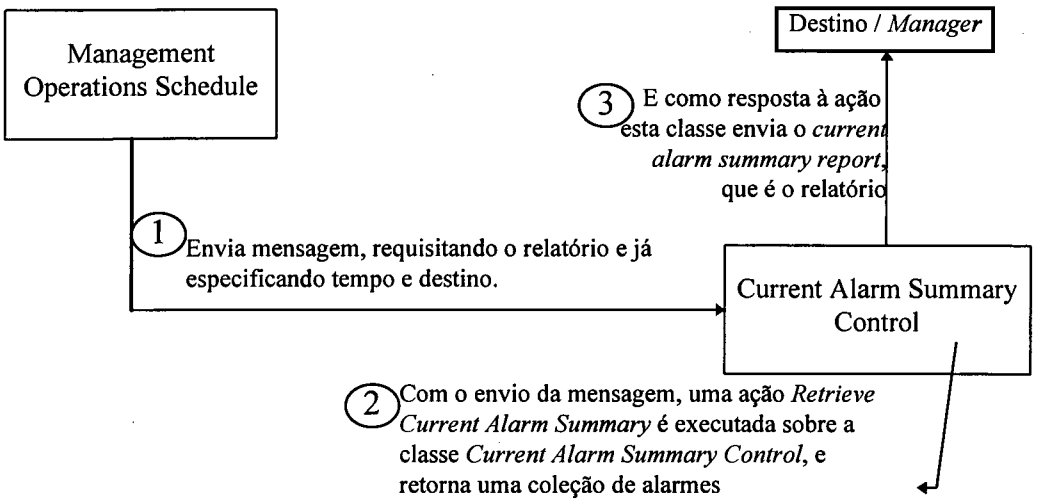

Figura 7-1 - Escalonamento de Operações de Gerenciamento

Esta classe têm a ação *retrieveCurrentAlarmSummary*, usada para requisitar que um relatório de sumarização de alarme seja enviado do sistema gerenciado para o gerente. Esta ação retorna uma coleção de alarmes, que consta dos seguintes parâmetros: o identificador do objeto, o percentual de severidade, o *status* de alarme e a causa provável do alarme.

A notificação *currentAlarmSummaryReport* é emitida pelo objeto, o qual fornece o coleção de alarmes. Esta notificação é emitida de acordo com o intervalo de tempo definido pela classe m*anagementOperationsSchedule*, ou quando o gerente requisita o relatório.

Classe MANAGEMENT OPERATIONS SCHEDULE

É a classe de objetos de suporte que fornece o escalonamento de um serviço de gerenciamento (relatório de sumarização de alarmes), para que ocorra periodicamente. O período é definido como um intervalo de tempo, onde o tempo inicial indica a primeira ocorrência do serviço, especificado como *beginTime*. O tempo final indica até quando o serviço pode ocorrer, e é especificado como o *endTime*.

O objeto que fornecerá o relatório é definido pela *affectedObjectClass* e *affectedObjectInstances* , isto é, o próprio objeto c*urrentAlarmSummaryControl*, quando a ação *retrieveCurrentAlarmSummary* retornará a coleção de alarmes e aciona a operação *currentAlarmSummaryRepor*t para emitir o relatório.

O endereço do destino do serviço é o mesmo de quem o invocou.

O *interval* indica o intervalo de tempo entre as requisições de um relatório pela classe *managementOperationsSchedule* (por exemplo, 5 em 5 segundos). Este intervalo pode ser especificado em segundos, minutos, horas ou dias.

Para que a classe *currentAlarmSummaryControl* possa retornar a coleção de alarmes, ela requisita os alarmes para a classe *log*, conforme as condições pré estabelecidas e o intervalo de tempo especificado. A classe *log* envia os alarmes através da operação *getRecords()*.

A especificação em IDL dos objetos de suporte para supervisão de alarmes com CORBA, é apresentada a seguir.

Arquivo **"as.idl"**:

```
#include "cmf.idl"
module AS {
  enum AlarmStatus {asCleared,asIndeterminated,asWarning,asMinor,asMajor};
  enum PerceivedSeverity {psWarning,psMinor,psMajor};
  enum ProbableCause
{pcQueueSizeExceeded,pcReceiverFault,pcTransmiterFault,pcIntruderDetection,pc
VentilationFault, pcConnectionFault, pcCellFault, pcTransmiterCellFault,
pcVoiceChannelFilled};
  struct Alarm {
    CMF::ManagedObject source;
    PerceivedSeverity perceivedSeverity;
    AlarmStatus alarmStatus;
    ProbableCause probableCause;
  };
  enum LogFullAction {lfaWrap,lfaHalt};
  struct LogRecord {
    any data;
    long long date_time;
  };
  interface Log : CMF::ManagedObject {
    void log (in any data);
    attribute LogFullAction logFullAction;
    readonly attribute short maxSize;
    readonly attribute short currentSize;
    CMF::Iterator getRecords ();
  };
  interface CurrentAlarmSummaryControl : CMF::ManagedObject,CMF::EventHandler
{
    attribute CMF::List alarmStatusList;
    attribute CMF::List objectList;
    attribute CMF::List perceivedSeverityList;
    attribute CMF::List probableCauseList;
    CMF::Iterator retrieveCurrentAlarmSummary ();
  };
  interface ManagementOperationsSchedule : CMF::ManagedObject {
    attribute CMF::List affectedObjectClasses;
    attribute CMF::List affectedObjectInstances;
    attribute long long beginTime;
    attribute long long endTime;
    attribute long long interval;
    attribute CMF::EventHandler destination;
  };
};
```

A figura 7.2 apresenta as relações entre o gerente, o agente e a interface *ManagedObject* pertencente ao *framework* para gerenciamento CORBA [CHAV 97].

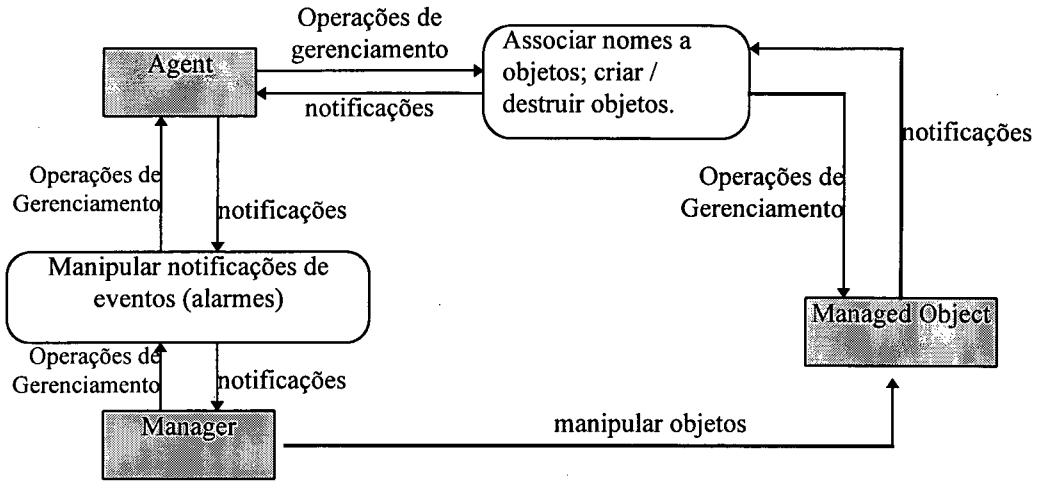

Figura 7-2 - Diagrama de fluxo de dados para modelo de gerenciamento

A figura 7.3 apresenta a hierarquia de herança para as interfaces do Sistema de Supervisão de Alarmes TMN com CORBA, onde *object* representa a interface CORBA, as interfaces nos quadros brancos representam as interfaces do *framework* de gerenciamento [CHAV 97], e as interfaces nos quadros em cor "grafite" representam as interfaces pertencentes ao sistema de supervisão de alarme com CORBA.

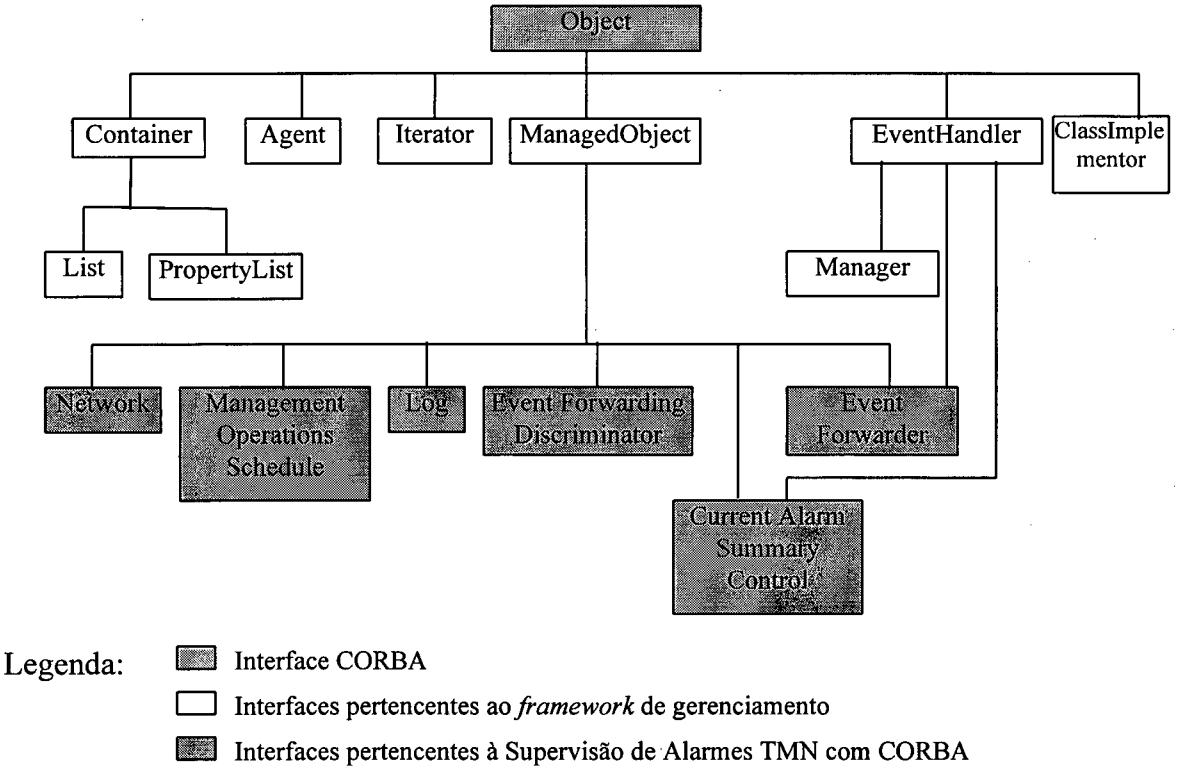

Figura 7-3 - Interfaces para a Supervisão de Alarmes com CORBA

A figura 7.4 apresenta as interfaces do sistema de supervisão de alarme TMN com CORBA, com seus respectivos atributos e operações (quando existirem).

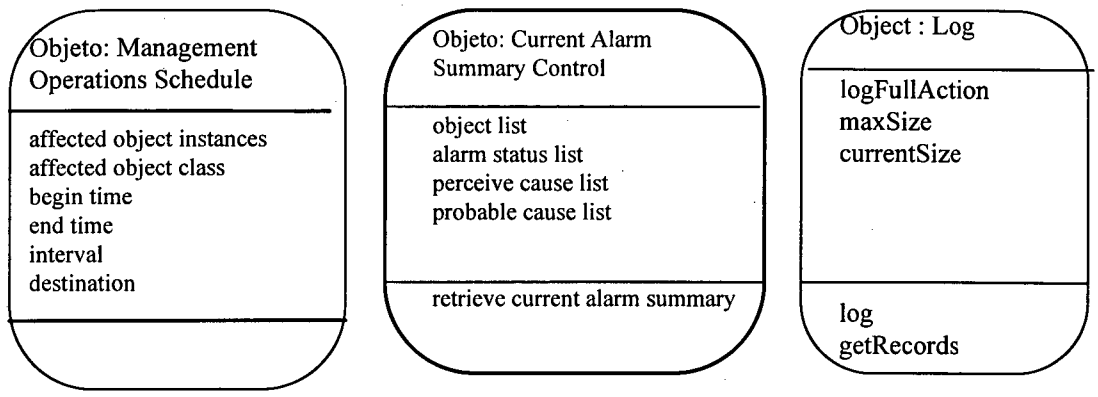

Figura 7-4 - Objetos *management operations schedule*, *current alarm summary control* e *log* com seus atributos e operações

As figuras 7.5 e 7.6 apresentam o diagrama de fluxo de dados para a emissão do relatório de sumarização de alarmes, quando o objeto *managementOperationsSchedule* define o tempo para emissão do relatório, e quando o *manager* requisita o relatório através da ação *retrieveAlarmSummary*, respectivamente.

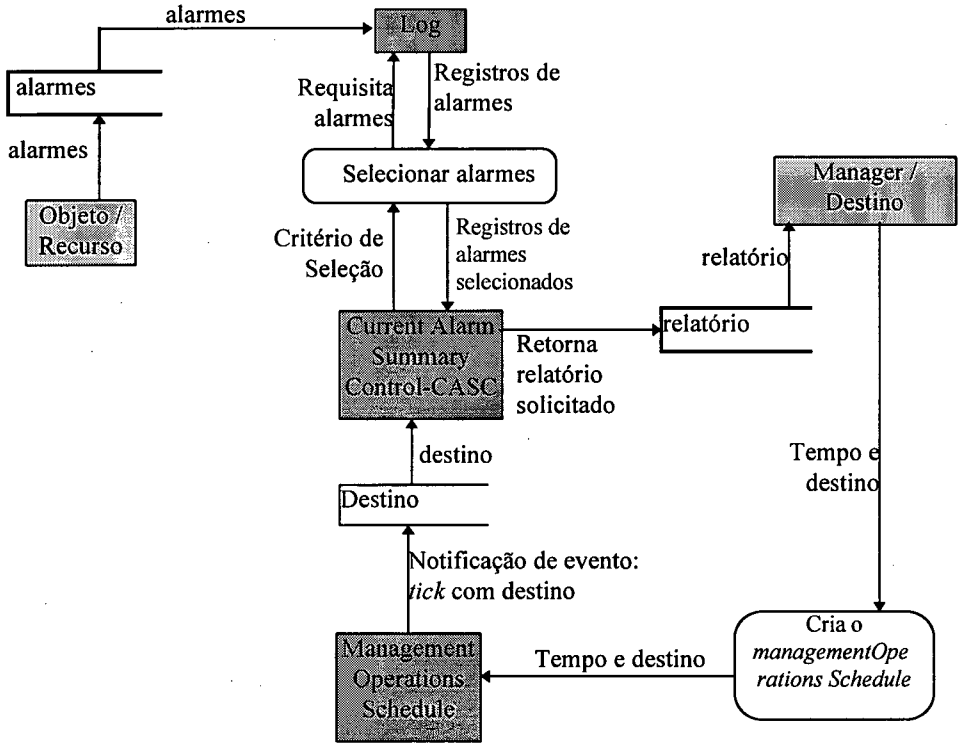

Figura 7-5 - Diagrama do Fluxo de Dados para gerar Relatório de Sumarização de Alarmes no tempo definido pelo objeto *management operations schedule*

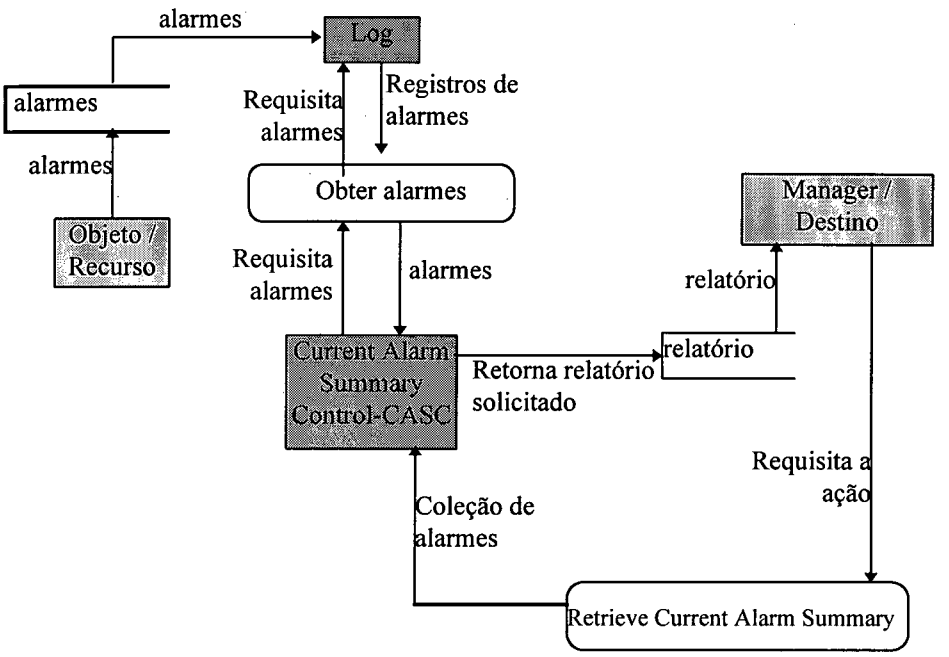

Figura 7-6 - Diagrama Fluxo de Dados para gerar o Relatório de Sumarização de Alarmes requisitado por um *Manager*, através da ação *retrieve current alarm summary*

A figura 7.7 apresenta a troca de eventos durante a criação dos objetos para a sumarização de alarmes, e as figuras 7.8 e 7.9 apresentam a troca de eventos para a definição do intervalo pelo objeto *managementOperationsSchedule*, e quando o *manager* requisitar o relatório de sumarização de alarmes, respectivamente.

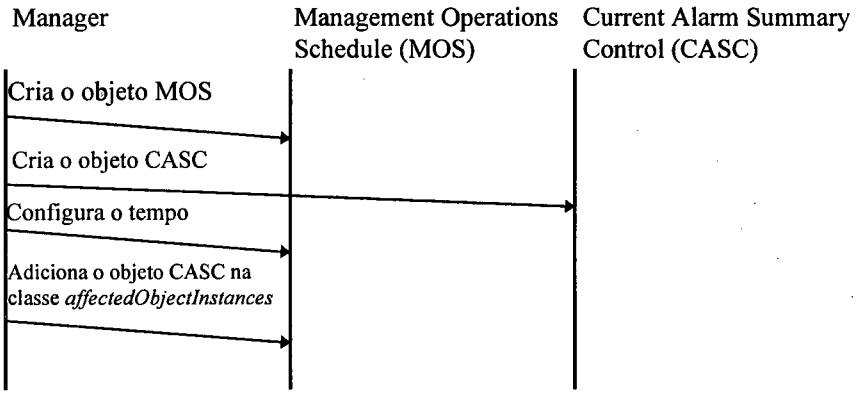

Figura 7-7 - Troca de eventos durante a criação dos objetos *management operations schedule* e *current alarm summary control*

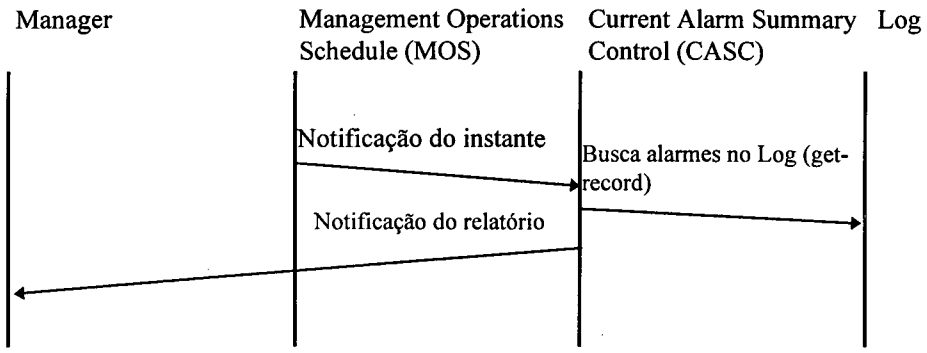

Figura 7-8 - Troca de Eventos para Sumarização de Alarmes através do objeto *management operations shedule*

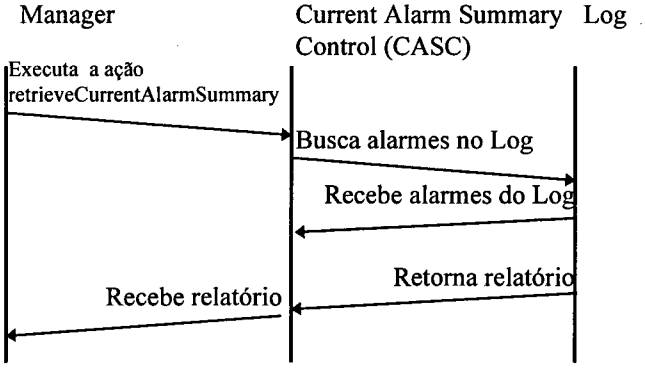

Figura 7-9 - Troca de Eventos quando o *Manager* solicita o Relatório de Sumarização de Alarmes ao objeto *current alarm summary control*

7.3 Especificação IDL do Elemento de Rede ATM

7.3.1 Redes ATM

Asynchronous Transfer Mode (ATM) é uma tecnologia de transmissão digital em banda larga que traz as vantagens de segurança e fidelidade, para prover "*fast packet switching*".

ATM é um modo de transferência orientado a pacote. Ele permite que múltiplas conexões lógicas sejam multiplexadas através de uma simples interface física. O fluxo de informação, em cada conexão lógica, é organizada dentro de pacotes (células) de tamanho fixado. As conexões lógicas em ATM são classificadas como Canais Virtuais (VC). Canal Virtual é um canal entre dois usuários finais da rede, onde são trocados uma taxa variável e o fluxo *full-duplex* de células com tamanho fixado. Estes canais são também usados para a troca

usuários-rede (controle de sinalização) e troca rede-rede (gerenciamento de rede e roteamento). Um Caminho Virtual (VP) é um pacote de canais virtuais que têm os mesmos pontos finais. Assim, todo fluxo de células de canais virtuais, em um simples caminho virtual, "*switched together*".

O gerenciamento ATM visa gerenciar a rede física ATM, e gerenciar os serviços providos através de conexões lógicas da rede ATM.

A padronização ITU-T propõe um conjunto de recomendações contendo novas funções para o gerenciamento das camadas de Canal Virtual e Caminho Virtual. O modelo de informação, especificado nestas recomendações, descreve classes de objetos gerenciados que herdam características das classes de objetos propostas pelas recomendações: M.3100 *Generic Management Information Model* [M3100]; Q.821 *Stage 2, Stage3 Description for the Q3 Interface: Alarm Surveillance* [Q821]; e Q.882 *Stage 1, Stage 2 e Stage 3 Description for the Q3 Interface: Performance Management* [Q822]. A coleção destes objetos gerenciados fornece informações de gerenciamento que podem ser adotadas por elementos de rede ATM, define funções de gerenciamento VP e VC para a camada ATM, e define um modelo de informação para a subcamada de convergência de transmissão [KOR 97].

7.3.2 TMN CORBA para gerenciar um elemento de rede ATM

TMN apresenta uma arquitetura de sistema de operação distribuída com interfaces de operação padronizadas entre os componentes de gerenciamento. Estes componentes podem utilizar o protocolo IIOP (*Internet Inter-ORB Protocol*), que especifica a implementação do GIOP através do transporte baseado no protocolo TCP/IP, para a troca de operações e são modelados por um modelo de informação orientado a objetos.

A arquitetura CORBA OMG e seus serviços provê uma base sólida para o deslocamento da plataforma de gerenciamento de rede do paradigma de comunicação *peer-to-peer*, para um paradigma de objetos distribuídos. Também, provê a tecnologia básica para o desenvolvimento de plataformas e aplicações de gerenciamento com serviços distribuídos.

O ORB é o núcleo da arquitetura CORBA, que provê o canal de comunicação entre clientes e implementações de objetos. Dentre outras características distribuídas, o ORB provê

transparência da implementação de objeto, diferente dos canais habituais (com passagem de mensagem).

As interfaces de objeto CORBA são definidas em IDL, que é uma linguagem de definição abstrata para modelagem de objetos genéricos, os quais provêem "moldes" para definições de classe de objeto. Através do protocolo inter-ORBs genérico (GIOP), são definidos a sintaxe de transferência de dados e o formato de mensagem para invocações a objeto, transversalmente e dentro de domínios ORB (figura 7.10).

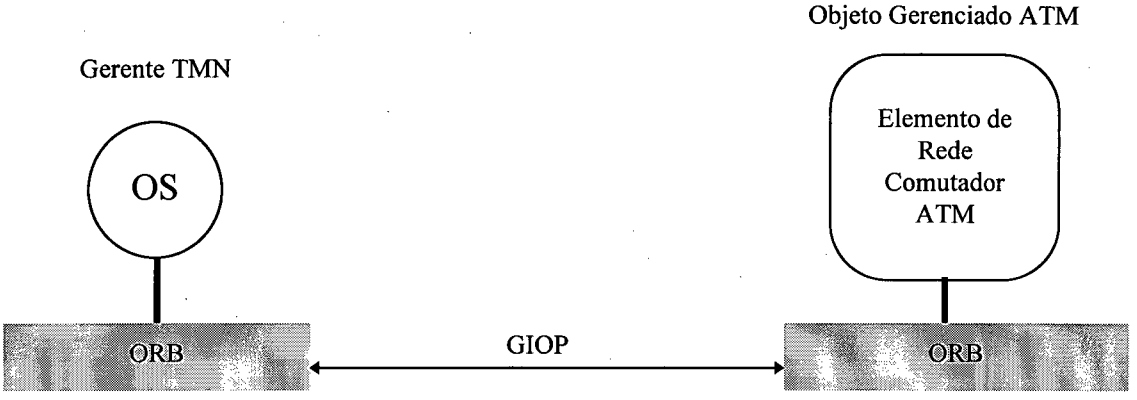

Figura 7-10 - Arquitetura Física do Gerenciamento de um Elemento de Rede

A seguir são citados alguns dos serviços CORBA (Anexo II) que são muito úteis para o desenvolvimento de aplicações e gerenciamento distribuído da rede ATM [CHE 97].

- **Serviço de Transação**: provê um método seguro realizando uma série de operações em objetos, os quais podem ser distribuídos através de muitos elementos de rede ATM. Uma transação têm as propriedades de atomicidade, consistência, isolamento e durabilidade.

- **Serviço de Relação**: armazena a informação sobre as relações entre objetos CORBA. Cada objeto, em uma relação, têm um determinado papel. Por exemplo, em uma relação de nomeação, uma localização ATM contém um nó ATM; o objeto nó ATM realiza o papel "contido" e a localização do objeto ATM realiza o papel "contém".

- **Serviço de Notificação**: permite que objetos emitam notificações quando ocorre um evento. Clientes podem receber estas notificações se anteriormente, eles tenham sido registrados para receberem notificações daquele produtor de notificações. Dados arbitrários podem ser passados adiante com a notificação.

- **Serviço de *Logging***: provê uma facilidade para clientes distribuídos em logar mensagens para um *log* unificado. O *log* parece ser simples, mas ele é armazenado em um modo

transversalmente distribuído em todas as partes da rede. As mensagens podem ser recuperadas mais tarde, com ordenamento mantido corretamente.

7.3.3 Modelo de Informação TMN

O modelo de informação especificado na recomendação M.3100 contém classes de objetos gerenciados que são necessários para a troca de informações, através das interfaces definidas pela arquitetura TMN. Este modelo identifica as classes destes objetos para o gerenciamento de redes de telecomunicações, e pode ser utilizado no gerenciamento de redes independentemente da tecnologia oferecida pelos equipamentos de comunicação. Isso garante que os conceitos TMN sejam viáveis para a modelagem e gerenciamento de componentes específicos em uma rede ATM [KOR 97].

Pretende-se mostrar nesta seção, a especificação IDL das classes que formam um elemento de rede ATM, obtidas da norma G.atmm [G.atmm]. Foram utilizadas as especificações IDL do *framework* de gerenciamento e da supervisão de alarmes, desenvolvidas neste trabalho, para o teste de conexão entre OS e o elemento de rede, como também identificar as falhas correntes que podem ser observadas por um elemento de rede. Em adicional, é mostrado a especificação IDL das classes para a troca de informações na rede ATM, obtidas da norma M.3100. Estas classes são necessárias para o gerenciamento de pontos terminação de *trails* e conexões requisitadas durante o estabelecimento de uma conexão.

Na figura 7.11, é apresentada a hierarquia de herança de alguns objetos presentes no modelo de informação M.3100 e das classes utilizadas pelo elemento de rede ATM, associadas com as conexões VP e VC. As classes são utilizadas para modelar aspectos lógicos que possibilitem a inicialização de um elemento de rede, fornecer mecanismos para a manipulação de eventos e identificar os componentes básicos de um elemento de rede [KOR 97]. Nesta hierarquia, os objetos gerenciados fornecem atributos e operações, úteis para a especificação de um elemento de rede, para o gerenciamento de pontos de terminação de *trails* e conexões requisitados durante o estabelecimento de uma conexão.

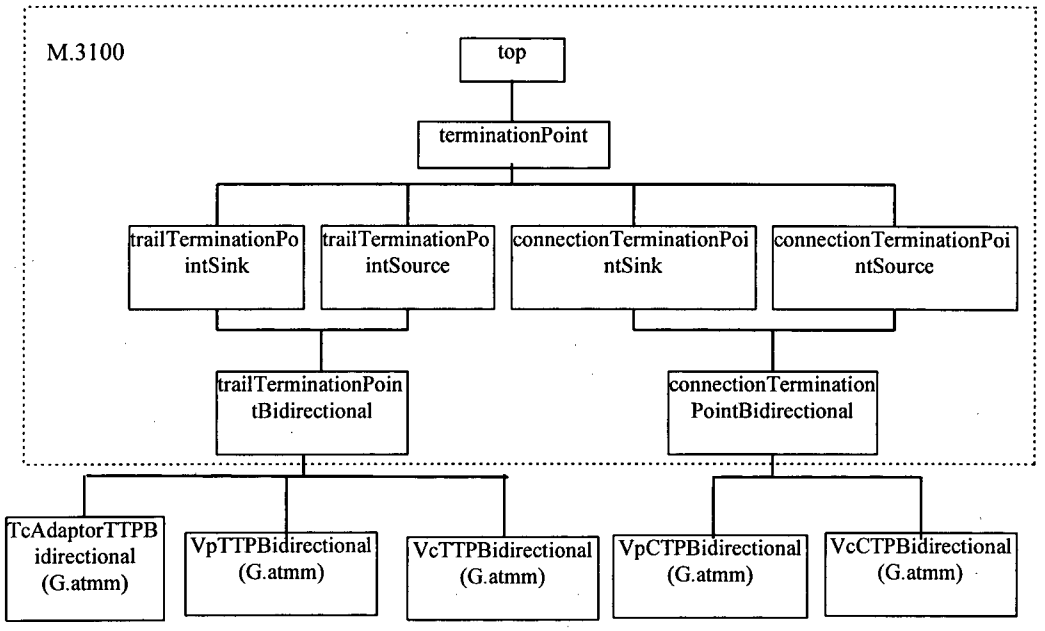

Figura 7-11 - Hierarquia de herança do modelo de informação G.atmm

Estes objetos realizam a construção de um elemento de rede ATM com o suporte de construções especificadas na recomendação M.3100.

Objetos M.3100

No que segue, temos os objetos que formam a hierarquia de herança do modelo de informação presentes na recomendação M.3100.

- **terminationPoint**

É a super classe de todos os objetos gerenciados que representam o ponto de terminação de *trail* ou o ponto de terminação de uma conexão de uma entidade de transporte. Há os parâmetros de controle associados com a criação e remoção de um objeto gerenciado. Esta classe têm como atributos:

- **trailTerminationPointSource** e **trailTerminationPointSink**

Identificação de um ponto de acesso onde um *trail* origina e termina.

- **connectionTerminationPointSource** e **connectionTerminationPointSink**

Fornecem os métodos para iniciar e terminar uma conexão de enlace, e contém estruturas que informam o número do canal (*channelNumber*) e o identificador do ponto de terminação de uma conexão (*CTPDownStreamPointer*).

Para melhor visualizar as camadas ATM no contexto geral:

A camada física é subdivida em [CER 97]:

- subcamada de meio físico: especifica as características mecânicas, elétricas e óticas dos meios de transmissão adotados, como o sincronismo necessário à transmissão e recepção de *bits*;
- subcamada de convergência de transmissão: especifica funções de geração e composição dos conjuntos de *bits*, de verificação dos *bits* de controle de erros, delineamento dos conjuntos de *bits*, dentre outras características.

A camada ATM especifica funções de comutação espacial e temporal dos conjuntos de *bits*, de geração, extração e adaptação dos *bits* correspondentes ao cabeçalho da célula e ao controle de tráfego [CER 97].

A camada de adaptação ao ATM é subdividida em [CER 97]:

- subcamada de segmentação e recomposição: especifica funções de decompor mensagens oriundas das camadas superiores (adaptá-las ao envio) e de recompor as mensagens, a partir dos conjuntos de *bits* recebidos da camada inferior;
- subcamada de convergência: especifica funções de propiciar serviços típicos da camada de transporte às aplicações das camadas superiores.

♦ **trailTerminationPointBidirectional**

Suas características e métodos para a criação e terminação de um ponto terminal de um *trail* são herdados das classes de objeto *trailTerminationPointSink* e *trailTerminationPointSource*.

♦ **connectionTerminationPointBidirectional**

Suas principais características para o início e término da conexão são herdadas das classes de objeto *connectionTerminationPointSource* e *connectionTerminationPointSink*.

Atributos e Operações dos Objetos M.3100

Na interface *tmnCommunicationsAlarmInformation* têm o atributo *alarmStatus* (as.idl) utilizado para representar aspectos de supervisão de alarmes, indicando o grau de severidade do alarme emitido por um objeto. E o atributo *CurrentProblem* que identifica as falhas correntes que podem ser observadas por um elemento de rede, através da lista de falhas. Esta lista contém uma ampla variedade de alarmes, desde a inexistência de uma instância de objeto (*notExistenceInstance*), até a identificação de dois pontos terminais recentemente conectados (*pointsConnectedLately*).

Os alarmes *VPAISupstream* e *VCAISupstream* são gerados pelos nós VPC e VCC para um ponto de conexão alertar os próximos nós VPC e VCC (*downstream*) que uma falha foi detectada anteriormente (*upstream*) ou através de notificação de uma falha física da camada. Quando um ponto final receber um VP AIS (Sinal de Indicação de Alarme) ou VC AIS, ele retornará um RDI (Indicador de Defeito Remoto) - *VPRDIdownstream* ou *VCRDIdownstream*, para alertar aos próximos nós (*upstream*) que uma falha têm sido detectada entre o ponto de conexão e o ponto final (*downstream*).

Os tipos de atributos *CrossConnectionObjectPointer* e *DownStreamConnectivityPointer* são requisitados pelo elemento de rede e identificam as instâncias de objeto que representam o ponto de terminação de uma conexão e um *trail*.

O valor do atributo *characteristicInformation* é usado para verificar a conectabilidade de instâncias das sub classes de ponto de terminação.

O ponteiro *networkLevel* identifica um objeto da rede. Seu valor somente pode ser modificado pelo sistema gerente.

O atributo *alarmSeverityAssigmentPointer* identifica a severidade do alarme de um objeto de transmissão. Se este atributo for NULL, então o agente determina a severidade através da lista *perceivedSeverityList* (as.idl), ou o valor é indeterminado.

O valor do atributo *SupportableClient* é a lista de classes do objeto representando os clientes que o sistema gerenciado é capaz de suportar.

No atributo *trailTerminationPointBidirectional*, seu estado operacional é desabilitado se a parte *sink* ou *source* do ponto de terminação for desabilitado.

O objeto gerenciado *connectionTerminationPointSink* termina uma conexão. O atributo *downStreamConnectivityPointer* aponta para o ponto de terminação do objeto gerenciado, dentro

do mesmo elemento gerenciado, que recebe a informação do tráfego de seu ponto de terminação na mesma camada, ou é nulo. O objeto referenciado deve ser uma instância de uma das seguintes classes: *trailTerminationPointSink, trailTerminationPointBidirectional, connection TerminationPointSource* e *connectionTerminationPointBidirectional.* O *downStream ConnectivityPointer* pode identificar um ou mais objetos, dependendo se o sinal é conectado para um ou mais objetos de ponto de terminação.

O objeto gerenciado *connectionTerminationPointSource* origina uma conexão. O atributo *upStreamConnectivityPointer* aponta para o ponto de terminação do objeto gerenciado, dentro do mesmo elemento gerenciado, que envia a informação do tráfego para o ponto de terminação na mesma camada, ou é nulo. O objeto referenciado deve ser uma instância de uma das seguintes classes: *trailTerminationPointSource, trailTerminationPointBidirectional, connectionTerminationPointSink* e *connectionTerminationPointBidirectional.*

O objeto gerenciado *trailTerminationPointSink* representa um ponto de terminação onde termina um *trail.* Ele representa o ponto de acesso em uma camada de rede que é um ponto para ambas as relações, *trail* e cliente/servidor. O objeto referenciado deve ser uma instância de uma das seguintes classes: *connectionPointSink,* ou *bidirectional,* ou *trail TerminationPointSource* ou *bidirectional.*

O objeto gerenciado *trailTerminationPointSource* representa um ponto de terminação onde um *trail* é originado. Ele representa o ponto de acesso em uma camada de rede que é o ponto para ambas as relações, *trail* e cliente/servidor. O objeto referenciado deve ser uma das classes: *connectionPointSource* ou *bidirectional,* ou *trailTerminationPointerSink* ou *bidirectional.*

A especificação em IDL dos objetos gerenciados do modelo de informação M.3100, associados às conexões VP e VC, é apresentada a seguir.

Arquivo **"atm.idl"**:

```
#include  "cmf.idl"
#include  "as.idl"
module  atm {
       interface  terminationPoint {
              readonly  attribute  managedObject  root;
              readonly  attribute  string  crossConnectionPointer;
              readonly  attribute  string  characteristicInformation;
              readonly  attribute  string  networkLevel;
              readonly  attribute  AS::perceivedSeverityList
                        alarmSeverityAssigmentProfilePointer;
       }
       struct  currentProblemList {
```

```
            string  communicationAlarm,
            boolean  notExistenceInstance,
            string  pointsConnectedLately,
            string  suspectObjectListParameter,
            string  VPAISupstream,
            string  VCAISupstream
      }
      interface  tmnCommunicationsAlarmInformation {
            readonly  attribute  alarmStatusList  alarmStatus;
            readonly  attribute  currentProblemList  currentProblem;
            currentProblemList  Alarms ();
            void  VPRDIdownstream();
            void  VCRDIdownstream();
      }
      struct   supportableClientList {
            theObject
      }
      interface  trailTerminationPointSink : terminationPoint {
            readonly  attribute  string  administrativeState;
            readonly  attribute  supportableClientList  supportableClient;
            readonly  attribute  string  operationState;
            readonly  attribute  string  ttpInstance;
            readonly  attribute  string  upStreamConnectivityPointer;
      }
      interface  trailTerminationPointSource : terminationPoint {
            readonly  attribute  string  administrativeState;
            readonly  attribute  supportableClientList  supportableClient;
            readonly  attribute  string  ttpInstance;
            readonly  attribute  string  operationalState;
            readonly  attribute  string  downStreamConnectivityPointer;
      }
      struct  permittedValues {
            NULL  none,
            short  single,
            short  concatenated,
            short  broadcast,
            short  broadcastConcatenated
      }
      interface  connectionTerminationPointSink : terminationPoint {
            readonly  attribute  string  ctpInstance;
            readonly  attribute  short  channelNumber;
            attribute  permittedValues  downStreamConnectivityPointer;
            readonly  attribute  string  CTPDownStreamPointer;
      }
      interface  connectionTerminationPointSource : terminationPoint {
            readonly  attribute  string  ctpInstance;
            readonly  attribute  short  channelNumber;
            attribute  permittedValues  upStreamConnectivityPointer;
            readonly  attribute  string  CTPDownStreamPointer;
      }
      interface  trailTerminationPointBidirectional : trailTerminationPointSource {
            readonly  attribute  string  trailTerminationPointBidirectional;
      }
      interface  connectionTerminationPointBidirectional :
                  connectionTerminationPointSource;
}
```

Objetos relativos ao Elemento de Rede (G.atmm)

No que segue, temos os objetos específicos para a gerência de configuração de conexões VC e VP (especificada neste trabalho) presentes na recomendação ITU-T G.atmm [G.atmm], as quais formam o elemento de rede ATM.

- **vcTTPBidirectional**

Utilizada para delimitar a conexão do Canal Virtual e fornecer operações para solicitar o ponto de terminação, inserir uma célula OAM no fluxo de dados (*oamCellLoopback*), e relatar se a célula retornou no tempo requerido. Dentre outras características, o elemento de rede deve identificar o ponto de inserção da célula, o local do retorno da célula, e a indicação se a célula está restrita a um segmento, ou se atinge a amplitude fim-a-fim da conexão.

- **vpTTPBidirectional**

Comportamento idêntico ao da classe anterior, onde as mesmas ações são realizadas para células de teste OAM, dentro do escopo do ponto de terminação de *trail* de um Caminho Virtual.

- **tcAdaptorTTPBidirectional**

Representação do ponto onde ocorre a adaptação da camada ATM com a estrutura do meio físico subjacente.

- **vcCTPBidirectional**

Apresenta métodos apropriados para delimitar enlaces de Canal Virtual. Os atributos e operações desta classe descrevem o valor VCI (Identificador de Canal Virtual), a coleção de descritores de tráfego e a classe de qualidade de serviço atribuída à terminação de enlace do Canal Virtual. Uma importante característica desta classe, é a atribuição do valor VCI ao identificador do ponto de terminação de enlace de Canal Virtual. Há também, atributos para a representação de medidas de performance como taxa pico de fluxo de células, taxa sustentável de célula, tolerância a rajadas, tolerância de atraso de células e classe de qualidade de serviço. A respeito de gerenciamento de falhas, esta classe apresenta um teste de conexão de enlace.

- **vpCTPBidirectional**

Seus atributos e operações são idênticos aos da classe anterior, utilizados para realizar funções no âmbito da sub camada VP. Isso possibilita o uso de objetos separados, com processamento similar, para as diferentes sub camadas de Canal Virtual e Caminho Virtual na camada ATM.

Atributos e Operações relativos às interfaces do Elemento de Rede

O objeto gerenciado *tcAdaptorTTPBidirectional* representa um ponto no sistema gerenciado onde ocorre a adaptação da camada ATM com a estrutura do meio físico subjacente. Este objeto é responsável por gerar notificações dos alarmes de comunicação, os quais informam

a falta de capacidade do objeto gerenciado em delinear uma célula ATM no caminho da transmissão digital. O atributo *cellScramblingEnable* identifica se a célula ATM (*scrambling*) está sendo realizada através da interface ATM, e um valor TRUE identifica qual é a célula que está sendo realizada.

Na ação *oamCellLoopback*, é utilizado o parâmetro *loopbackOAMCell* para requisitar um objeto *vpCTPBidirectional, vcCTPBidirectional, vpTTPBidirectional* ou *vcTTPBidirectional*, para inserir uma célula OAM (*loopback*) dentro do fluxo de célula ATM e verificar seu retorno. O valor TRUE-NULL pode ser utilizado para requisitar o último ponto da conexão ATM ou o segmento da conexão para a célula OAM.

O atributo *egressBurstTolerance* representa a tolerância na emissão de rajadas (em células), que têm sido atribuída para a conexão VP ou VC, que está sendo terminada.

O atributo *egressCDVTolerance* representa a tolerância na variação do atraso da célula (CDV) atribuída para a conexão VP ou VC que está sendo terminada. Quando é omitido, seu valor é 0 (zero).

O atributo *egressPeakCellRate* indica a taxa máxima do fluxo de células atribuída ou reservada, no término da conexão VP e VC.

O atributo *egressQOSClass* identifica a classe de qualidade do serviço atribuída à conexão VP ou VC, na emissão da célula. Seus valores são os da lista *qosClass*: *class1*, *class2*, *class3* ou *class4*.

O atributo *egressSustainableCellRate* é o descritor de tráfego que representa a emissão da taxa sustentável de célula, atribuída à conexão que está sendo terminada.

O atributo *ingressBurstTolerance* representa a tolerância à rajadas que têm sido atribuída à conexão VP ou VC.

O atributo *ingressCDVTolerance* representa a tolerância à variação do atraso da célula, atribuída à conexão VP ou VC. Quando omitido, seu valor é 0 (zero).

O atributo *ingressPeakCellRate* indica a taxa máxima do fluxo de células, atribuída ou reservada, no início da conexão VP ou VC.

O atributo *ingressQOSClass* identifica a classe qualidade de serviço atribuída à conexão VP ou VC, na direção inicial da transmissão de célula. Seus valores são os da lista *qosClass*.

O atributo *ingressSustainableCellRate* é o descritor de tráfego que representa a taxa inicial sustentável de célula, atribuída à conexão que está sendo terminada.

A especificação em IDL dos objetos gerenciados do elemento de rede ATM é apresentada a seguir.

Arquivo **"neAtm.idl"**:

```
#include  "cfm.idl"
#include  "as.idl"
#include  "atm.idl"
module  NEAtm {
      interface  tcAdaptorTTPBidirectional : trailTerminationPointBidirectional {
            readonly  attribute  AS::perceivedSeverityList
alarmSeverityAssigmentPointer;
            readonly  attribute  boolean  cellScramblingEnabled;
      }
      interface  vpTTPBidirectional : trailTerminationPointBidirectional {
            void  oamCellLoopback (in  string  loopbackOAMCell);
      }
      interface  vcTTPBidirectional : trailTerminationPointBidirectional {
            void  oamCellLoopback (in  string  loopbackOAMCell);
      }
      struct  CDVTolerance {
            short  cellDelayVariationToleranceCLP0plus1;
            short  cellDelayVariationToleranceCLP0
      }
      struct  peakCellRate {
            short  peakCellRateCLPplus1;
            short  peakCellRateCLP0;
      }
      struct  qosClass {
            short  class1;
            short  class2;
            short  class3;
            short  class4
      }
      interface  vpCTPBidirectional : connectionTerminationPointBidirectional {
            // trafficDescriptor
            attribute  CMF::propertyList  ingressPeakCellRate;
            attribute  CMF::propertyList  egressPeakCellRate;
            attribute  CMF::propertyList  ingressCDVTolerance;
            attribute  CMF::propertyList  egressCDVTolerance;
            attribute  CMF::propertyList  ingressSustainableCellRate;
            attribute  CMF::propertyList  egressSustainableCellRate;
            attribute  CMF::propertyList  ingressBurstTolerance;
            attribute  CMF::propertyList  egressBurstTolerance;
            // oamTrafficDescriptor
            readonly  attribute  peakCellRate  oamIngressPeakCellRate;
            readonly  attribute  peakCellRate  oamEgressPeakCellRate;
            readonly  attribute  CDVTolerance  oamIngressCDVTolerance;
            readonly  attribute  CDVTolerance  oamEgressCDVTolerance;
            // qosClass
            readonly  attribute  qosClass  ingressQOSClass;
            readonly  attribute  qosClass  egressQOSClass;
            // oamCellLoopback
            void  oamCellLoopback (in  string  loopbackOAMCell);
            boolean  segmentEndPoint ();
      }
      interface  vcCTPBidirectional : connectionTerminationPointBidirectional {
            // trafficDescriptor
            attribute  CMF::propertyList  ingressPeakCellRate;
            attribute  CMF::propertyList  egressPeakCellRate;
            attribute  CMF::propertyList  ingressCDVTolerance;
            attribute  CMF::propertyList  egressCDVTolerance;
            attribute  CMF::propertyList  ingressSustainableCellRate;
            attribute  CMF::propertyList  egressSustainableCellRate;
            attribute  CMF::propertyList  ingressBurstTolerance;
            attribute  CMF::propertyList  egressBurstTolerance;
            // oamTrafficDescriptor
            readonly  attribute  peakCellRate  oamIngressPeakCellRate;
            readonly  attribute  peakCellRate  oamEgressPeakCellRate;
```

```
            readonly  attribute  CDVTolerance  oamIngressCDVTolerance;
            readonly  attribute  CDVTolerance  oamEgressCDVTolerance;
            // qosClass
            readonly  attribute  qosClass  ingressQOSClass;
            readonly  attribute  qosClass  egressQOSClass;
            // oamCellLoopback
            void  oamCellLoopback (in  string  loopbackOAMCell);
            boolean  segmentEndPoint ();
        }
}
```

7.4 Implementação Java

Depois de serem especificadas as interfaces em IDL, foi utilizado o compilador do *Visibroker*, o "**idl2java**", que gera de cada interface IDL (que recebeu como entrada) o código referente a determinada classe em Java. Este código representa *stubs* e esqueletos, como também outros arquivos referentes a uma implementação Java, os quais serão explicados adiante.

A interface em Java, derivada da classe **org.omg.CORBA.object**, corresponde à interface **CORBA::Object**, que é descrita na especificação CORBA 2.0 [COR 95].

Cada interface nos módulos que formam o *framework* de gerenciamento e a supervisão de alarmes, corresponde a uma interface Java.

Para melhor exemplificarmos a implementação em Java, vamos utilizar a interface CurrentAlarmSummaryControl do módulo de Supervisão de Alarmes com CORBA (seção 7.2):

```
interface CurrentAlarmSummaryControl :
CMF::ManagedObject,CMF::EventHandler {

    attribute CMF::List alarmStatusList;
    attribute CMF::List objectList;
    attribute CMF::List perceivedSeverityList;
    attribute CMF::List probableCauseList;
    CMF::Iterator retrieveCurrentAlarmSummary ();
};
```

Figura 7-12 - Especificação IDL para a classe *currentAlarmSummaryControl*

Esta interface fornece o critério para a geração do relatório de sumarização de alarme (seção 7.2.1). Este relatório é requisitado pela classe *Management Operations Schedule*, quando esta classe envia uma mensagem à classe *Current Alarm Summary Control*, e esta fornece como resposta o relatório.

O operador "::" pode ser usado para referenciar uma interface base explicitamente; ele permite referência para um nome que tem sido redefinido na interface derivada. Por exemplo,

CMF::ManagedObject, representa que a classe *ManagedObject* herda todos os elementos da interface base CMF (do *framework*) e ainda pode declarar novos elementos.

A figura 7.13 apresenta os arquivos gerados da compilação Java.

```
C:\rp\projeto>idl2java currentAlarmSummaryControl.idl
Creating: CurrentAlarmSummaryControl.java
Creating: CurrentAlarmSummaryControlHolder.java
Creating: CurrentAlarmSummaryControlHelper.java
Creating: _st_CurrentAlarmSummaryControl.java
Creating: _sk_CurrentAlarmSummaryControl.java
Creating: _CurrentAlarmSummaryControlImplBase.java
Creating: CurrentAlarmSummaryControlOperations.java
Creating: _tie_CurrentAlarmSummaryControl.java
Creating: _example_CurrentAlarmSummaryControl.java
C:\rp\projeto>
```

Figura 7-13 - Arquivos gerados na compilação Java correspondentes à interface **currentAlarmSummaryControl.idl**

O arquivo **`CurrentAlarmSummaryControl.java`** é uma interface Java correspondente à sua interface IDL. Java também é formado por interfaces, neste caso, correspondentes a cada interface IDL. Percebe-se que as sintaxes de ambas as interfaces são idênticas, salvo alguns detalhes sobre os atributos e passagem de parâmetros por referência, que serão vistos a seguir.

A interface **`CurrentAlarmSummaryControl.java`** correspondente à interface *CurrentAlarmSummaryControl.idl* demostrada na figura 7.14.

```
public interface CurrentAlarmSummaryControl extends
org.omg.CORBA.Object {
      public CMF.List alarmStatusList();
      public CMF.List objectList();
      public CMF.List perceivedSeverityList();
      public CMF.List probableCauseList();
      public CMF.iterator retrieveCurrentAlarmSummary();
}
```

Figura 7-14 - Interface Java correspondente à interface IDL

Java não têm passagem de parâmetros por referência, que é necessária para parâmetros de saída (*out* e *inout*). Este problema foi contornado com a criação das classes *CurrentAlarmSummaryControlHolder*, que são responsáveis por manter parâmetros de saída. O arquivo **`CurrentAlarmSummaryControlHolder.java`** é utilizado para representar os parâmetro *out* ou *inout*.

Os atributos somente leitura (*readonly*) geram um método Java correspondente para a obtenção do atributo. Os atributos normais geram dois métodos Java correspondentes, um método para obter o valor do atributo e outro método para alterar o valor do atributo.

O arquivo **`CurrentAlarmSummaryControlHelper.java`** é uma classe utilitária com as seguintes funções:

⇒ *bind*: vincula um *proxy* local, que distribui uma requisição de serviço entre os objetos pertencentes ao cliente usando os serviços de um protocolo de comunicação, à implementação de um objeto remoto [COR 95];

⇒ *narrow*: utilizada para a conversão de instância de uma interface em outra instância [COR 95];

⇒ *read/write*: utilizadas para a serialização do objeto, isto é, transformação de um objeto (seus atributos, suas informações de tempo de execução) em uma forma linearizada (para a transmissão através de uma rede ou armazenamento em um arquivo em disco);

⇒ *extract/insert*: métodos utilizados para a conversão *any->CurrentAlarmSummaryControl* (retirar um objeto *CurrentAlarmSummaryControl* do tipo *any*) e *CurrentAlarmSummary Control->any* (coloca o objeto *CurrentAlarmSummaryControl* no tipo *any*).

Os arquivos **`_st_CurrentAlarmSummaryControl.java`** e **`_sk_Current AlarmSummaryControl.java`** são o *stub* e o *skeleton* para a interface *CurrentAlarmSummary Control*, utilizados apenas pelo ORB (não são diretamente utilizados pelo usuário).

O arquivo **`_CurrentAlarmSummaryControlImplBase.java`** realiza o vínculo da implementação para a interface IDL *CurrentAlarmSummaryControl*, de onde é derivada uma classe chamada **`_exampleCurrentAlarmSummaryControl`**, a responsável pela ligação com o *skeleton.*

Os arquivos **`CurrentAlarmSummaryControlOperations.java`** e **`_tie_CurrentAlarmSummaryControl.java`** são umas das facilidades oferecidas pelo *VisiBroker* (não fazem parte do CORBA) que estão relacionados com a herança de implementação. A linguagem de programação Java permite apenas herança simples. A implementação de CORBA em Java exige que a classe de implementação do objeto derive de sua classe **CurrentAlarmSummaryControlImplBase.java**, impedindo que a implementação de um objeto derive de outra. O mecanismo *tie* utiliza a delegação para permitir a herança de implementação.

Outra facilidade oferecida pelo *VisiBroker* é o arquivo **`_example_Current AlarmSummaryControl.java`** que é um modelo para a construção da implementação de objeto. Java mantém uma classe em apenas um arquivo, e neste arquivo (*_example_Current AlarmSummaryControl.java*) está definida a classe homônima e todos os módulos, atributos e

operações declarados na interface IDL, que se transformam em métodos vazios desta classe. E por estes métodos estarem vazios, é que o programador deve dar significados aos mesmos.

Sendo assim, o único arquivo gerado pela compilação Java, a ser modificado pelo programador, é o **_example_Current AlarmSummaryControl.java**, que no trabalho em questão é alterado para **CurrentAlarmSummaryControlImpl.java**, onde são inseridas as operações para a classe *currentAlarmSummaryControl* receber o aviso do *Manager* ou do objeto *managementOperationsSchedule*, acessar os alarmes armazenados no *Log* e podendo assim, emitir o relatório de sumarização de alarmes (seção 7.2). A seguir, é apresentado o arquivo **CurrentAlarmSummaryControlImpl.java**.

```
import org.omg.CORBA.Any;

public class CurrentAlarmSummaryControlImpl
  extends ManagedObjectImpl
  implements AS.CurrentAlarmSummaryControlOperations {

  protected CMF.List alarmStatusList;
  protected CMF.List perceivedSeverityList;
  protected CMF.List probableCauseList;
  protected CMF.List objectList;

  protected AS.Log alarmLog;

  public CurrentAlarmSummaryControlImpl(CMF.PropertyList args) {
    super(args);
    alarmStatusList = new ListImpl ();
    objectList = new ListImpl();
    perceivedSeverityList = new ListImpl();
    probableCauseList = new ListImpl();

    Any anyValue = args.getPropertyValue ("agent");
    alarmLog = AS.LogHelper.narrow (CMF.AgentHelper.extract
(anyValue).getObject ("alarmLog"));
  }
  public CMF.Iterator retrieveCurrentAlarmSummary() {
    return buildSummary();
  }
  public void handleEvent(CMF.Event ev) {
    System.out.print ("CASC recebendo aviso - ");
    System.out.println ((new java.util.Date() ).toString());
    CMF.EventHandler destination = CMF.EventHandlerHelper.extract
(ev.data.getPropertyValue ("destination"));
    Any anyValue = org.omg.CORBA.ORB.init().create_any();
    CMF.Iterator summary = buildSummary ();

    CMF.IteratorHelper.insert (anyValue,summary);
    CMF.PropertyList evData = new PropertyListImpl ();
    evData.addProperty ("alarmSummaryData",anyValue);
    EventSender.send (new CMF.Event
("periodicAlarmSummary",System.currentTimeMillis(),this,evData),destination);
  }

  protected CMF.Iterator buildSummary () {
/*
Varrer o alarmLog, procurando por registros que satisfacam os criterios aqui
especificados.
```

```
Armazena-los num Vector, e no final retornar um Iterator
*/
    CMF.Iterator records = alarmLog.getRecords();
    alarmLog.reset ();
    return records;
  }
  public void alarmStatusList(
    CMF.List _alarmStatusList
  ) {
    alarmStatusList = _alarmStatusList;
  }
  public CMF.List alarmStatusList() {
    return alarmStatusList;
  }
  public void objectList(
    CMF.List _objectList
  ) {
    objectList = _objectList;
  }
  public CMF.List objectList() {
    return objectList;
  }
  public void perceivedSeverityList(
    CMF.List _perceivedSeverityList
  ) {
    perceivedSeverityList = _perceivedSeverityList;
  }
  public CMF.List perceivedSeverityList() {
    return perceivedSeverityList;
  }
  public void probableCauseList(
    CMF.List _probableCauseList
  ) {
    probableCauseList = _probableCauseList;
  }
  public CMF.List probableCauseList() {
    return probableCauseList;
  }
}
```

Cada objeto tem um tratador de evento. Quando um objeto precisa realizar uma notificação, este objeto invoca o método *handleEvent* para tratar do evento no seu tratador de evento particular. Neste caso, o objeto *CurrentAlarmSummaryControl*.

A seguir é apresentado o arquivo **`ManagementOperationsScheduleImpl.java`**, onde é realizada a operação *notifyAffected*, que passa um evento ao objeto *currentAlarmSummaryControl*, contendo o destino e o intervalo (de 30 em 30 segundos), para este objeto fornecer o relatório de sumarização de alarmes no tempo estabelecido.

```
import org.omg.CORBA.Any;

public class ManagementOperationsScheduleImpl
  extends ManagedObjectImpl
  implements AS.ManagementOperationsScheduleOperations {

  private CMF.List affectedObjectInstances;
  private CMF.List affectedObjectClasses;
  private long beginTime;
```

```
private long endTime;
private long interval;
private CMF.EventHandler destination;

public ManagementOperationsScheduleImpl(CMF.PropertyList args) {
  super(args);
  affectedObjectInstances = new ListImpl ();
  affectedObjectClasses = new ListImpl ();
  if (args.hasProperty ("destination")) {
    Any anyValue = args.getPropertyValue ("destination");
    destination = CMF.EventHandlerHelper.extract (anyValue);
  }
  // por enquanto o intervalo eh fixo (30 segundos)

  interval = 30000;

  Thread t = new MOSThread (this);
  t.start ();
}
public void affectedObjectClasses(
  CMF.List _affectedObjectClasses
) {
  affectedObjectClasses = _affectedObjectClasses;
}
public CMF.List affectedObjectClasses() {
  return affectedObjectClasses;
}
public void affectedObjectInstances(
  CMF.List _affectedObjectInstances
) {
  affectedObjectInstances = _affectedObjectInstances;
}
public CMF.List affectedObjectInstances() {
  return affectedObjectInstances;
}
public void beginTime(
  long _beginTime
) {
  beginTime = _beginTime;
}
public long beginTime() {
  return beginTime;
}
public void endTime(
  long _endTime
) {
  endTime = _endTime;
}
public long endTime() {
  return endTime;
}
public void interval(
  long _interval
) {
  interval = _interval;
}
public long interval() {
  return interval;
}
public void destination(
  CMF.EventHandler _destination
) {
  destination = _destination;
}
public CMF.EventHandler destination() {
```

```
    return destination;
  }
  protected void notifyAffected () {
    // obter um Iterator para a lista de affectedObjectInstances
    // e para cada um invocar handleEvent passando um evento
    // contendo a destinacao e o tipo do evento (timeEvent?).

    CMF.PropertyList evData = new PropertyListImpl();
    Any anyValue = org.omg.CORBA.ORB.init().create_any();
    CMF.EventHandlerHelper.insert (anyValue,destination);
    evData.addProperty ("destination",anyValue);
    CMF.Event ev = new CMF.Event
("timer",System.currentTimeMillis(),this,evData);
    CMF.Iterator iter = affectedObjectInstances.getElements ();
    CMF.EventHandler aoi;
    while (iter.hasMoreElements ()) {
      aoi = CMF.EventHandlerHelper.extract (iter.nextElement ());
      EventSender.send (ev,aoi);
    }
  }
}

class MOSThread extends Thread {
  protected ManagementOperationsScheduleImpl mos;
  MOSThread (ManagementOperationsScheduleImpl _mos) {
    mos = _mos;
  }
  public void run () {
    try {
    do {
      sleep (mos.interval());
      mos.notifyAffected ();
    } while (true);
    } catch (Exception e) {
      e.printStackTrace();
    }
  };
};
```

7.5 *Resultados do Sistema de Supervisão de Alarmes CORBA*

A tecnologia CORBA surge como uma nova alternativa para modelar os complicados objetos de redes de gerenciamento de telecomunicações. CORBA oferece um ambiente para a definição, transporte, implementação e invocação de objetos, além de considerar tudo e qualquer objeto como um **objeto ativo**.

Os resultados do trabalho em questão foram obtidos quando são executados o agente **Agent**, o gerente **Manager** e um **elemento de rede** fictício, chamado de comutador ATM. O gerente - **Manager** cria um objeto *currentAlarmSummaryControl* para si e logo após cria o objeto *managementOperationsSchedule*, configurando o tempo (*beginTime, endTime, interval*),

fazendo com que o objeto *currentAlarmSummaryControl* seja um elemento da lista *affectedObjectInstances*, pertencente ao objeto *managementOperationsSchedule*.

Durante o tempo estabelecido, o gerente **Manager** solicita o relatório através da ação *retrieveCurrentAlarmSummary*, tendo como resposta uma coleção de alarmes. Como também, o objeto *currentAlarmSummaryControl* vai notificando o gerente periodicamente, conforme definido pelo objeto *managementOperationsSchedule*.

Para a realização de testes na implementação, foi criada uma operação que gere alarmes sobre o objeto que representa um elemento de rede fictício, chamado comutador ATM. Nesta operação são feitas consultas periódicas relativas ao seu *status* de alarme, a sua severidade e a sua causa provável, os quais são descritos pelas interfaces *alarmStatusList*, *perceivedSeverityList* e *probableCauseList*, respectivamente (seção 7.2.1). Caso o comutador ATM (contido na lista de objetos - *objectList*) possua um ***status*** de alarme que coincida com um *status* de alarme presente na *alarmStatusList*, ou possua uma **severidade** de alarme que coincida com uma severidade presente na *perceivedSeverityList*, ou ainda, que tenha uma **causa provável** do alarme que coincida com uma causa provável presente na *probableCauseList*, então este objeto, juntamente com seu alarme, seu *status* de alarme, sua severidade, sua causa provável, e a data que o alarme foi gerado, constará no Relatório de Sumarização de Alarmes (figura 7.15).

RELATÓRIO DE SUMARIZAÇÃO DE ALARMES

Object	*AlarmStatus*	*Perceived Severity*	*ProbableCause*	Data-Hora
Comutador ATM	*asMajor*	*psMajor*	*pcConnectionFault*	07/03/98 - 8:35:19 hs
Comutador ATM	*asMinor*	*psMinor*	*pcCellFault*	07/03/98 - 8:35:18 hs
Comutador ATM	*asIndeterminated*	*psWarning*	*pcTransmiterCellFault*	07/03/98 - 8:35:17 hs
Comutador ATM	*asMinor*	*psMinor*	*pcVoiceChannelFilled*	07/03/98 - 8:35:16 hs

Figura 7-15 - Exemplo de um Relatório de Sumarização de Alarmes do Comutador ATM.

No presente trabalho houve a preocupação com a emissão de alarmes correspondentes ao teste de conexão entre OS (*Operations Systems*) e o elemento de rede ATM, como também identificar as falhas correntes que podem ser observadas por um elemento de rede (nas células e a emissão das mesmas). Estes alarmes são caracterizados como Alarmes de Comunicação, os quais estão relacionados a procedimentos e processos usados na transferência de informações através do sistema gerenciado [BRI 93]. Assim, as prováveis causas de um alarme podem ser:

falha na conexão de dois pontos terminais, falha na célula enviada, falha no envio da célula, falha no recebimento da célula, dentre outros.

O nível de severidade pode ser: alerta (*psWarning*), menor (*psMinor*) ou maior (*psMajor*). Uma notificação de alarme é emitida pelo elemento de rede e o nível de severidade é atribuído ao objeto gerenciado afetado. Quando a notificação indica a existência de suspeita sobre a ocorrência de falhas antes que seus efeitos tenham sido percebidos, então o nível de severidade do alarme emitido será de alerta. O nível de severidade menor indica a existência de falhas comprometendo o adequado funcionamento do sistema, sendo necessário tomar ações corretivas para prevenir a ocorrência de falhas mais sérias. E o nível de severidade maior é atribuído ao alarme quando as condições de funcionamento de um sistema são afetadas, sendo necessária uma ação corretiva urgente [BRI 93].

Na figura 7.15 é apresentado o objeto que emitiu alarme, o Comutador ATM. Este objeto apresenta um *status* de alarme (condição de alarme - seção 6.1.2) presente na lista `AlarmStatus`, isto é, `asCleared`, `asIndeterminated`, `asWarning`, `asMinor` ou `asMajor`. Também apresenta a severidade do alarme (possíveis severidades percebidas) presente na lista `PerceivedSeverity (psWarning,psMinor,psMajor)`. E por fim, apresenta a causa provável do alarme, que deve estar presente na lista `ProbableCause`, isto é, `pcReceiverFault`, `pcTransmiterFault`, `pcConnectionFault`, `pcCellFault`, `pcTransmiterCellFault`, `pcVoiceChannelFilled`, dentre outros. O momento em que o alarme é emitido, é apresentado pelo campo **Data-Hora**.

A respeito de informações definidas pela Indicação de Tendências, pela Informação de Valor-Limite e pelas Sugestões de Ações de Reparo [BRI 93], não são abordadas neste trabalho, pois a preocupação foi de relatar todos os alarmes no relatório de sumarização de alarmes, desde que apresente um dos critérios requeridos (seção 7.2).

As funções de gerenciamento de falha em TMN [BRI 93], com CORBA, por exemplo, supervisão de alarmes, são realizados por blocos de funções específicas (capítulo 3), os quais têm capacidades de processamento distribuído. O gerenciamento distribuído do bloco de função é realizado utilizando-se o processamento distribuído da arquitetura CORBA, onde cada elemento constituinte do bloco de função é visto como um objeto ORB, que provê o processamento distribuído dentro de cada bloco de função. Os ORBs se comunicam através do protocolo GIOP.

Uma diferença observada entre o sistema de supervisão de alarmes em OSI e o sistema de supervisão de alarmes desenvolvido com CORBA é a simplicidade na especificação de operações no sistema CORBA, através da IDL. Esta simplicidade inclui uma redução significativa da especificação GDMO para IDL, devido às características da linguagem e da arquitetura CORBA, por exemplo a referência a objeto executa pedidos e respostas (notificações).

O sistema de supervisão de alarmes em CORBA se apresentou como um sistema relativamente simples e economicamente viável ao ser implementado em Java (o sistema em CORBA é simples comparado ao sistema desenvolvido em OSI [LAV 96]). Simples, devido à utilização da linguagem IDL, do protocolo GIOP ou IIOP, e demais características CORBA. Economicamente viável, devido à utilização de uma plataforma de desenvolvimento (*Visibroker*) que explora Java com todas as suas facilidades como linguagem de programação orientada a objeto, reduzindo assim, o tempo de desenvolvimento das aplicações.

Uma vantagem do sistema IDL CORBA é a sua semelhança com Java e C++, o que possibilita maior quantidade de programadores desenvolverem aplicações distribuídas, dentre elas a gerência de redes, sem ser necessária a mudança do paradigma de desenvolvimento.

No sistema de Supervisão de Alarmes TMN com CORBA, aqui desenvolvido, foi considerado apenas o sistema gerente como OS (Sistemas de Operação), e como objeto gerenciado o elemento de rede ATM, onde cada objeto gerenciado ou recurso da rede têm uma única identidade do objeto. O gerenciamento do sistema é feito pela manipulação dos objetos na MIB, que é criada pelo agente, mas não está necessariamente presente neste agente (capítulo 5).

Como uma interface para sistemas gerenciados, CORBA apresenta características importantes, tais como, especificação IDL, invocação estática e dinâmica, e provê a interoperabilidade entre os objetos gerenciados através do ORB que utiliza o protocolo de comunicação GIOP. Também utiliza o adaptador de objetos básico (BOA), o qual realiza a ligação entre a implementação dos objetos e o próprio ORB (seção 5.2.2 - figura 5.5).

7.6 Comparações entre as Arquiteturas de Gerenciamento OSI e CORBA

A rede de gerenciamento de telecomunicações é baseada no modelo de gerenciamento OSI. Este modelo define as interfaces e protocolos, a MIB e os componentes de gerenciamento.

7.6.1 Sobre Gerente e Agente

Os principais componentes OSI de gerenciamento são o gerente, o agente e os objetos gerenciados. Um **gerente** pode monitorar objetos gerenciados, e para isso, envia operações de gerenciamento a um **agente**. Este tem como função, a execução de operações de gerenciamento sobre objetos gerenciados, envia ao gerente o resultado destas operações e as notificações emitidas pelos objetos gerenciados. Os gerentes e agentes trocam informações através de protocolos OSI, e é através da troca destas informações que manipulam a MIB.

O papel de gerente na arquitetura CORBA é similar ao do modelo OSI. Na implementação desenvolvida neste trabalho, a interface Manager(aqui definida) é implementada pelo gerente e possui uma operação para manipular as notificações de eventos enviadas por objetos gerenciados.

O papel de agente também têm um significado similar ao OSI, mas a sua função é reduzida e distribuída entre outros elementos (devido à função dos objetos CORBA). O gerente pode criar e destruir objetos gerenciados através de operações fornecidas pela interface Agent, definida para o trabalho em questão, para obter referências a objetos pelo nome destes objetos gerenciados e permite ainda, a criação e a destruição de classes de objetos em tempo de execução.

7.6.2 Sobre Objeto Gerenciado

Um **objeto gerenciado** é a representação de um recurso real que pode ser manipulado pelas aplicações de gerenciamento. Assim, cada recurso que se deseja manipular e controlar deve ser descrito na forma de um objeto gerenciado. No modelo OSI, ele é definido em termos de seus atributos ou propriedades, as operações a que podem ser submetidos, as notificações que podem emitir para informar sobre a ocorrência de eventos de gerenciamento, e suas relações com outros objetos gerenciados [BRI 93].

Na arquitetura OMG CORBA, um objeto obriga o encapsulamento e é caracterizado pelo conjunto de atributos e operações, definidas em interfaces IDL. Uma das diferenças entre os objetos destas duas arquiteturas é que os objetos CORBA podem ser entidades ativas e os objetos OSI são estruturas de dados passivas onde o agente é o responsável pela execução das operações. É através do conceito de herança que um objeto OMG CORBA pode ter mais que uma interface.

7.6.3 Sobre Operação nos Objetos

No modelo OSI, uma **operação** pode ser inicializada pelo recebimento de uma invocação. Uma operação confirmada gerará uma ou mais respostas, e a operação não confirmada não gerará nenhuma. As exceções são sinalizadas por meio de respostas de exceções, e as invocações e respostas podem ser parametrizadas.

Na arquitetura CORBA, uma operação é iniciada através de um *stub* ou pela interface de invocação dinâmica. A execução normal de uma operação, resulta numa resposta (e retorna o controle para o chamador). A exceção é indicada por meio de um sinal de exceção e a transferência do controle vai para o manipulador de exceções, sendo que todos podem ser parametrizados. É necessário definir uma fábrica OMG (*factory object*) [MAZ 97] de interfaces de objeto, genérica, para criar os objetos CORBA (operação M-CREATE no OSI).

7.6.4 Sobre Eventos

A maior diferença entre os modelos de gerência OSI e OMG é o modo de especificação de **eventos** autônomos. Na gerência OSI, eventos são comportamentos sem conexão com operações. As notificações de eventos são causadas por condições de alarmes no recurso representado por um objeto gerenciado; e uma função de gerência de relatório de evento determina quais notificações de evento são encaminhadas adiante.

Na arquitetura CORBA, os eventos OMG são especificados como sendo suportados pelo receptor dos eventos, e é modelado como uma operação noutro sentido. Nesta arquitetura ainda não se definiu nenhum padrão de filtragem de eventos; deve-se indicar o suporte para cada evento tipado, e quem é responsável por este suporte, é o receptor [FER 96].

7.6.5 Sobre Interoperabilidade

Outras diferenças principais entre estes padrões, são o **ambiente** e interoperabilidade. A arquitetura CORBA oferece um ambiente de desenvolvimento para sistemas distribuídos orientados a objetos, e ainda oferece transparência de acesso e de localização para aplicações de plataformas heterogêneas básicas. O modelo OSI gerencia redes distribuídas e componentes de rede heterogêneos.

No âmbito da **interoperabilidade**/portabilidade, ao nível sintático e semântico, a gerência OSI não oferece nenhuma portabilidade de código e interoperabilidade de comunicações [FER 96]. Já CORBA, oferece a portabilidade de código (através do Java) e a interoperabilidade de comunicação, através do protocolo GIOP (*General Inter-ORBs Protocol*).

7.6.6 Sobre Comportamento e Atributos

O **comportamento** no modelo OSI é suportado correntemente via texto em inglês, e na arquitetura CORBA é suportado pela definição das operações em IDL.

No modelo OSI, um **atributo** é caracterizado por um conjunto de valores de dados e qualificadores que especificam regras de comparação, restrição e escopo, valores iniciais e *default*, e restrições de acesso. Eles também podem ser manipulados através de um nome de grupo de atributos que identifica um ou mais atributos. Já na arquitetura CORBA, um atributo é caracterizado por um identificador, um tipo e um qualificador opcional *readonly* (onde no OSI é a operação GET), e ainda não há nenhum mecanismo para atributos agrupados [FER 96].

7.6.7 Sobre a MIB

O conjunto de objetos gerenciados e seus atributos formam a **MIB**, que tem como função, guardar as informações transferidas ou modificadas pelo uso dos protocolos de gerenciamento OSI. Estas informações correspondem a objetos que representam os recursos reais que estão sendo gerenciados, e incluem os atributos destes objetos, as operações que eles executam e as notificações que eles fornecem.

Sendo a arquitetura CORBA uma plataforma de objetos distribuídos e ativos, um agente é que mantém os objetos, os quais podem não residirem no mesmo local que ele. Na implementação aqui desenvolvida, o agente é uma coleção hierárquica de referências a objetos

gerenciados, sendo a localização destes objetos desconhecida, mas tem-se a certeza de que é associada à *factory* de sua classe de objetos. Isso por causa da transparência de localização natural do ambiente CORBA.

7.6.8 Sobre as Operações de Ciclo-de-Vida

No modelo OSI, as **operações do Ciclo-de-Vida** do Objeto são operações embutidas para a criação e exclusão de instâncias de objetos. Quem é responsável por esta criação, pode especificar o nome ou pode delegar ao agente a definição do nome. Este modelo não prevê nenhum protocolo padrão para mover instâncias de objetos entre sistemas.

A arquitetura CORBA nos fornece objetos de fábrica (*factory*) que provêem operações especializadas para criarem e inicializarem novas instâncias. Esta arquitetura também oferece os outros serviços do ciclo-de-vida COSS, tais como a especificação da exclusão de objetos, bem como operações de cópia e movimento de implementações de objeto [FER 96].

7.6.9 Sobre a Estrutura de Gerenciamento

Na **estrutura de gerenciamento** OSI [LAV 96], são definidas três categorias de gerenciamento: gerenciamento de sistemas, gerenciamento de camadas e operações de camadas. O gerenciamento de sistemas utiliza protocolos da camada de aplicação e se refere à gerência dos vários recursos e seu estado por todas as camadas da arquitetura OSI. Na estrutura deste gerenciamento, o processo gerente utiliza o SMAE (*System Management Application Entity*), que é composto por LMEs (*Layer Management Entity*), entidades que se comunicam com cada camada através de protocolos específicos.

O gerenciamento de camada utiliza protocolos de gerenciamento de propósito especial, que são independentes de protocolos de gerenciamento de outras camadas e não prestam serviços de comunicação às camadas superiores, assim como funções de apoio internas à camada. Este gerenciamento é realizado sobre objetos relacionados com as atividades de comunicação da mesma camada.

A operação de camada utiliza o protocolo normal de cada camada. Deste modo, exige menos funções de apoio para a troca de informações de gerenciamento, e gerencia uma única instância de comunicação em uma camada.

A arquitetura CORBA se abstém de toda a estrutura de camadas apresentada pela OSI através do ORB que possibilita a comunicação direta entre diferentes ORBs, ou entre domínios diferentes utilizando-se de *gateways* [MAZ 97]. CORBA está imediatamente abaixo da camada de aplicação, caracterizando assim, uma subcamada de distribuição (figura 7.17).

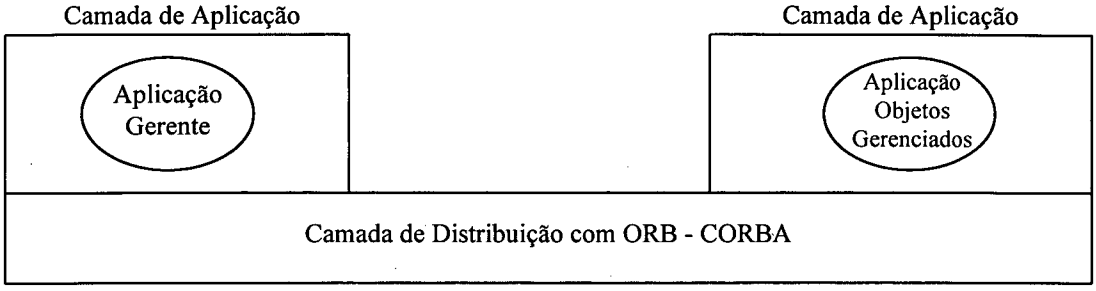

Figura 7-16 - Arquitetura CORBA e a camada de Aplicação

7.6.10 Sobre Referência a Objeto

No modelo OSI, o tipo de dado **Referência a Objeto** é realizado através da instância de objeto, definida por um nome distinguível (DN), o qual é usado extensivamente como parâmetro na interface de comunicação (sintaxe X.500). Na arquitetura CORBA, é realizado através do tipo referência a objeto (*object reference*), onde as operações o utilizam como um de seus parâmetros, e a sua sintaxe é opaca para a operação que a usa [FER 96].

7.6.11 Sobre a Interface

A **interface** no modelo OSI é a interface de comunicação de objetos gerenciados, operações e notificações. Na arquitetura CORBA, uma interface é o conjunto de atributos e operações definidas em IDL. A escolha de interface feita pela gerência OSI é pela comunicação de gerência de rede, e pela OMG é para o desenvolvimento de *software*. Portanto, pode-se pensar que estas escolhas são complementares, isto é, os sistemas de gerência de rede orientado a objeto precisam ser implementados, e os programadores necessitarão de ferramentas de *software* orientados a objeto [FER 96].

7.6.12 Sobre o Modelo de Protocolo

Para operações remotas, no modelo OSI, o **modelo de protocolo** baseia-se na passagem de mensagens não-bloqueante (ROSE), com respostas do tipo normal e a manipulação de

exceções terá que ser adicionada. Na arquitetura CORBA, o uso de *stubs* cliente OMG implica no modelo de chamadas de procedimento bloqueante, e é definida uma interface de invocação dinâmica que permite submeter operações síncronas atrasadas (a operação é chamada assíncronamente e os resultados são coletados posteriormente de modo síncrono) [FER 96].

7.6.13 Sobre a Especificação de Protocolos

No modelo OSI, há três aspectos para a **especificação de um protocolo**: a regra para a troca de informações, a semântica da informação e a sintaxe da informação [FER 96]. Neste modelo, o protocolo de comunicação utilizado nas interações do gerente-agente é o CMIP (Anexo IV). A arquitetura CORBA oferece o protocolo IIOP - GIOP especificado como a interface comum, incluindo o tipo da mensagem e formato, proporcionando assim, a interoperabilidade, além de especificar um mapeamento de linguagem para *Java*, *SmallTalk*, dentre outras [COR 95].

7.6.14 Sobre as Especificações de Atributos, Operações e Eventos

No modelo OSI, as **especificações de atributos, operações e eventos** podem ser ligados dentro de pacotes por meio de uma especificação de pacotes, a qual pode especificar restrições adicionais sobre os atributos (por exemplo: restrições de faixa), e sobre operações e eventos (por exemplo: restrições sobre o tipo dos parâmetros), como também especifica a semântica do pacote como um todo. Neste modelo, as especificações podem ser ligadas dentro de classes de objetos gerenciados por meio de uma especificação da classe de objetos gerenciados, podendo estas ligações serem obrigatórias ou condicionais.

Na arquitetura CORBA, as especificações de atributos e operações são combinadas dentro de interfaces, que podem ser combinadas dentro de módulos. Seus atributos e valores de parâmetros podem conter escolhas nulas, e tem um tipo de exceção pré-definida chamada NOT-IMPLEMENTED. A OMG definiu eventos como operações de uma interface que opera na direção reversa (invocação do objeto), possibilitando assim, que um objeto gerenciado possa definir uma ou mais interfaces contendo todos os eventos que ele possa gerar [FER 96].

7.6.15 Sobre as Ferramentas de Especificação

A gerência OSI, oferece duas **ferramentas de especificação** para especificar a sintaxe de objetos gerenciados e mensagens associadas: GDMO, que é a linguagem de *templates* utilizada para definir as características de objetos gerenciados, e ASN.1, que é uma linguagem de definição de dados utilizada para definir os tipos de dados associados com os atributos e parâmetros. A única ferramenta de especificação utilizada pelo padrão CORBA, é a linguagem para definição de interfaces, chamada IDL, através da qual são definidos os objetos e as mensagens associadas.

7.6.16 Sobre as Classes de Objetos Gerenciados

As **classes de objetos gerenciados**, na gerência OSI, são relacionadas na hierarquia de tipos por meio da herança múltipla e restrita. A raiz principal da árvore de herança é chamada TOP, a qual especifica atributos para a especificação de propriedades estáticas comuns para toda classe. Este modelo possibilita a herança de duas definições de classe que incluem a mesma ação ou atributo.

Na arquitetura CORBA, a classe de objetos gerenciados é definida através de uma interface CORBA-IDL, com o objetivo de criar instâncias de uma nova classe, através de uma *factory* específica. O agente sabe da nova classe de objeto através do registro desta, a informação do seu nome e da *factory* associada a ela. Os tipos de objeto são relacionados numa hierarquia de tipo de interfaces suportadas por meio de herança múltipla, e não é permitido a herança de duas interfaces com o mesmo nome de operação ou atributo.

7.6.17 Sobre Notificação

Notificação é conhecida como um sinal que indica a ocorrência de um evento. A CORBA-IDL não permite a declaração de notificações junto à descrição da interface, mas ela é descrita como tratamento de eventos na implementação Java.

7.6.18 Sobre a Transparência de Acesso

O termo transparência significa "esconder" dos usuários e programadores a separação dos componentes. Na **transparência de acesso,** os objetos locais e os remotos são acessados da

mesma maneira. Esta transparência não é suportada no modelo OSI, necessitando assim, do protocolo CMIP (Anexo IV). Já na arquitetura CORBA ela é suportada, pois os ORBs podem usar qualquer protocolo apropriado, podem empregar camadas baixas do OSI ou Internet, e ainda fornece o protocolo GIOP que possibilita a interoperabilidade [FER 96].

7.6.19 Sobre a Transparência de Localização

Na **transparência de localização** os objetos de informação são acessados sem o conhecimento de sua localização. A arquitetura OSI não suporta esta transparência, sendo necessária uma MIB para armazenar as informações e serem acessadas quando necessário. Já a arquitetura CORBA, suporta transparência de localização, devido às operações oferecidas pelo serviço de ciclo de vida, principalmente a operação *move* [FER 96].

7.7 Interfaces Estáticas e Dinâmicas da arquitetura CORBA

No padrão CORBA, toda a comunicação entre os objetos é realizada pelo ORB. Um cliente pode interagir com o ORB através de *stubs* IDL ou através da interface de invocação dinâmica (DII).

O que mantém as descrições das interfaces disponíveis é o repositório de interfaces, o qual pode ser atualizado e consultado através de chamadas. Quando conhecida a interface de um objeto servidor, um cliente CORBA pode acessar os seus métodos através de dois tipos de chamadas: as estáticas e as dinâmicas.

As chamadas estáticas utilizam *stubs* gerados em tempo de compilação, a partir de descrições IDL dos objetos a serem invocados. As chamadas dinâmicas utilizam a *Dynamic Invocation Interface* (DII), a qual permite a um cliente construir chamadas a servidores em tempo de execução.

Para podermos utilizar as chamadas construídas estaticamente, a aplicação de gerência deve incorporar os *stubs* gerados pelo compilador IDL, isto é, ao ser gerada, a aplicação de gerência deve ter conhecimento dos objetos aos quais ela fornecerá acesso. Em um ambiente inter-redes, esta interface pode ser bastante restrita pois um administrador de sistemas está continuamente tomando conhecimento de novos problemas e de novos serviços. Neste ambiente, o uso da interface de invocação dinâmica seria importante, pois ela permite que uma ferramenta

de gerência possa incorporar acesso a novos objetos sem a necessidade de se recompilar a ferramenta, ou mesmo interromper sua execução [ROD 97].

Atualmente, a grande maioria das ligações (*bindings*) oferecidas para CORBA são baseadas no uso da interface de invocação estática. Isso acontece, principalmente, devido ao fato das linguagens de programação serem tipadas estaticamente (C, C++ e Java), e dentre as linguagens, temos apenas uma exceção, a ligação feita pelo Smalltalk.

Com o uso de chamadas construídas estaticamente, é limitado o benefício oferecido ao modelo de gerência pelo uso do repositório de interfaces. Um exemplo, é acessar o repositório através de uma ferramenta de estilo *browser*, e baseando-se na informação coletada, gerar uma nova versão da aplicação de gerência. No entanto, isso causa um passo de recompilação e reinstalação da aplicação de gerência para cada mudança detectada pelo *browser* [ROD 97].

A interação do cliente com o ORB através de *stubs*, funciona de maneira idêntica ao modelo de chamadas remotas de procedimento. Quando uma especificação escrita em IDL é compilada, é gerado um *stub* cliente e um esqueleto servidor (*stub* servidor). Depois, o programador pode criar o programa cliente e a implementação do servidor. Para o cliente, os serviços desejados são requisitados ao *stub* cliente, que é o encarregado de repassá-los ao servidor, codificar (*marshalling*) os parâmetros de entrada, tratar a resposta e decodificar (*unmarshalling*) os resultados. O esqueleto é encarregado de receber o pedido do *stub* cliente e repassá-lo à implementação do servidor (Figura 7.17).

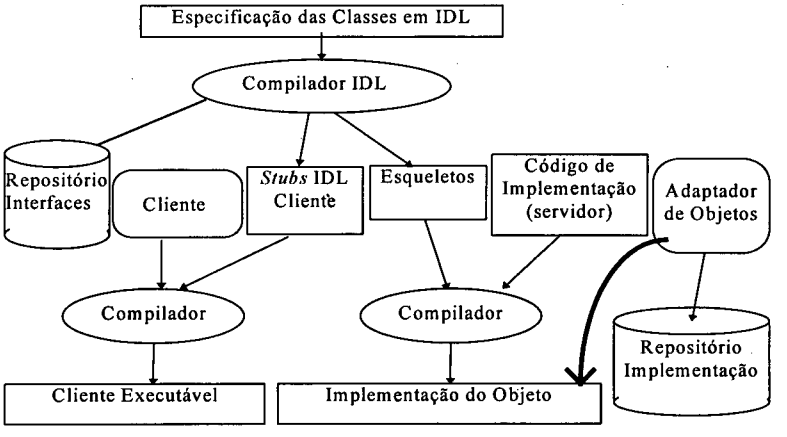

Figura 7-17 - Interface de Invocação Estática

A interface de invocação dinâmica (DII) é uma facilidade genérica utilizada para a invocação de qualquer operação CORBA e para construir, em tempo de execução, a chamada da operação e a lista de parâmetros. A interação através da DII envolve a identificação do objeto

servidor, a obtenção de sua interface, a construção e realização da chamada e a obtenção dos resultados (Figura 7.18).

A identificação do objeto a ser invocado pode ser obtida através de serviços disponíveis no ambiente CORBA, como o serviço de nome ou o de *trader*. É possível também, que o usuário forneça a identificação de um novo objeto explicitamente. Isto é provável depois de obter o conhecimento desse objeto através de algum serviço de divulgação.

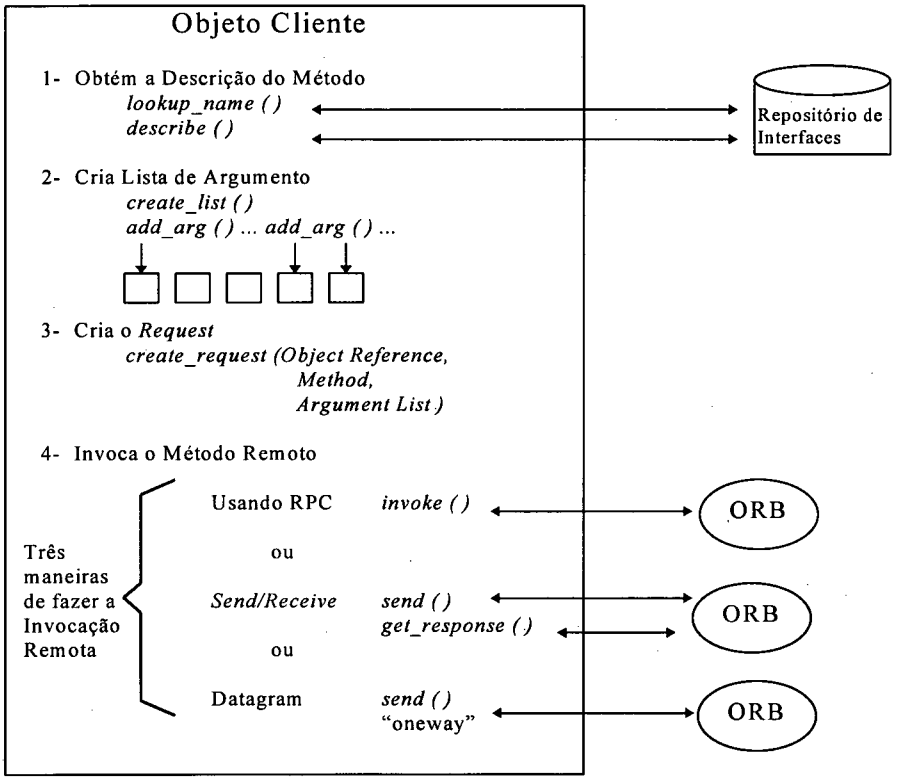

Figura 7-18 - Requisição de serviço usando a Interface de Invocação Dinâmica CORBA

Quando o programa tiver uma referência para o servidor, ele pode requisitar meta-dados que descrevem sua interface. O método get_interface, oferecido por todo objeto CORBA, pode ser usado para se ter acesso à descrição do objeto. Este método retorna um objeto interfaceDef, que funciona como um ponto de entrada no repositório de interfaces.

Verificando o repositório de interfaces, pode-se obter informação detalhada sobre a interface de um objeto. Com esta informação e uma estrutura de dados específica para a descrição de listas de parâmetros, a *NamedValueList*, é possível construir a lista de argumentos a ser usada na chamada de um método.

A chamada create_request é utilizada para criar um novo pseudo-objeto CORBA. Este pseudo-objeto é usado para armazenar o nome do método, a lista de argumentos e, após a

chamada, o valor retornado. A realização da chamada é feita através da invocação de um método do objeto *request* (lembrando-se que as diferentes semânticas de chamada são implementadas por diferentes métodos).

As aplicações CORBA atuais exigem que as chamadas sejam feitas através da DII. Esta interface oferece maior flexibilidade que os *Stubs* Estáticos, mas a criação e a chamada de um pedido DII podem exigir muitas chamadas remotas, tornando o pedido mais caro que uma chamada estática equivalente. As chamadas estáticas não sofrem a sobrecarga de acessos ao repositório de interface, pois consistem da informação já compilada dentro da aplicação [VIN 97].

As invocações estáticas são fáceis de programar, verificam tipos robustamente (se forem objetos diferentes ocorre erro na compilação), a interface com o sistema é mais simples, e é uma interface auto-documentada. Já nas invocações dinâmicas pode-se adicionar novas classes, sem alterar o código cliente, e são muito úteis em ferramentas que precisam de serviços providos em tempo de execução [MAL 96].

As invocações dinâmicas são necessárias para aplicações em que as interfaces não são conhecidas, e portanto, não podem ser especificadas, e assim, devem ser geradas durante a execução. Por exemplo, é preciso desenvolver um *gateway* entre duas tecnologias diferentes, CORBA e SNMP; as interfaces devem ser geradas durante a execução para que CORBA e SNMP possam se comunicar, pois o programador não sabe quais interfaces devem ser especificadas para a adaptação de uma tecnologia à outra.

No trabalho em questão (implementação de Supervisão de Alarmes) foi utilizada extensivamente a invocação estática e em alguns casos a invocação dinâmica. A invocação estática por que têm-se o conhecimento de quais interfaces são necessárias para o desenvolvimento com CORBA (através dos *stubs* gerados automaticamente pelo **`idl2java`** para todas as interfaces). E a invocação dinâmica pois toda a invocação de métodos executada nos *stubs* é síncrona, ou seja, o cliente faz a requisição e fica bloqueado esperando o retorno do servidor (confirmação). No caso da notificação de eventos, o gerador de eventos não precisa de qualquer confirmação do destinatário do evento, portanto é utilizada a invocação dinâmica para realizar uma requisição não confirmada. Dessa forma, o notificador do evento pode continuar a executar e, imediatamente após, enviar a notificação (através da ação **`send_oneway`**)

As formas (métodos de **`org.omg.CORBA.Request`**) de se realizar requisições dinamicamente são [VIN 97]:

- *invoke* - requisição síncrona e confirmada (bloqueante);
- *send_deferred/poll_response* - requisição assíncrona e confirmada (não-bloqueante);
- *send_oneway* - requisição assíncrona e não-confirmada (não-bloqueante).

INTERFACE DE ESQUELETO DINÂMICA (DSI) - é o lado servidor e permite que servidores sejam escritos através de esqueletos para objetos que são invocados e compilados dinamicamente dentro de um programa. A DSI foi introduzida ao CORBA 2.0, pois precisava-se de suporte para a implementação de *gateways* entre ORB's, utilizando diferentes protocolos de comunicação. Ela é uma ferramenta útil para ligações entre ORB's, e para aplicações que servem de "ponte" entre os sistemas CORBA e os serviços e implementações não CORBA.

8. Conclusões e Trabalhos Futuros

Dentre tantas aplicações de gerenciamento TMN OSI, foi escolhido o sistema de supervisão de alarmes TMN (desenvolvido pelo grupo de pesquisa do Laboratório de Redes e Gerência - LRG) para ser o sistema de comparação entre o padrão OSI e o padrão CORBA. A arquitetura CORBA surgiu como uma tecnologia emergente e completa para aplicações distribuídas, por isto, pensou-se em desenvolver o sistema de gerenciamento para supervisão de alarme TMN em CORBA, iniciando o estudo sobre o ambiente de redes de telecomunicações e explorar as interfaces CORBA para a construção de processos de gerenciamento necessários ao sistema TMN.

Para que TMN suporte o processamento distribuído, três requisitos básicos devem ser satisfeitos: haver um **mecanismo de repositório para informação**, que suporta o armazenamento e a distribuição da informação relacionada ao recurso; haver um **mecanismo de correção apropriada para encontrar a informação** sobre vários serviços disponíveis na rede e para provê-los aos sistemas de gerenciamento que desejam utilizá-los; e, suportar as **interações muitos-para-muitos** entre gerentes e agentes. Dentre outros mecanismos, a arquitetura CORBA oferece o serviço de nomes e de *trading* como mecanismo de repositório para informação, possuindo ORB como o mecanismo de correção para encontrar a informação. Para suportar as interações muitos-para-muitos utiliza o protocolo GIOP, a "armazenagem" CORBA e grupo de objetos.

A arquitetura CORBA é vista por um grande número de organizações como a única tecnologia viável para tratar a interoperabilidade de um modo não proprietário [BUS 96]. CORBA oferece mecanismos de orientação a objetos com a tentativa de trazer para ambientes distribuídos os conceitos de encapsulamento e reutilização de código.

As facilidades e serviços CORBA oferecem um conjunto de interfaces que possibilitam a construção de componentes TMN, bem como seu gerenciamento e comunicação entre eles. O protocolo de comunicação utilizado pelo CORBA é o GIOP, que realiza a comunicação entre ORBs de ambientes de desenvolvimento iguais ou diferentes. Para a comunicação entre CORBA e algum outro padrão que não fala CORBA, é necessário a utilização de *gateway* [MAZ 97].

Com o uso da arquitetura CORBA para TMN, a aplicação do lado gerente passa a atuar como um cliente CORBA, e o agente tradicional é substituído por objetos servidores. CORBA oferece mais transparências e uma arquitetura com as facilidades do protocolo GIOP, sendo assim, não é necessário um agente intermediário, como existe nos modelos SNMP/CMIP. O uso de CORBA como plataforma de gerência resulta em uma arquitetura mais simples, onde as entidades de gerente e objetos gerenciados com interfaces IDL continuam a desempenhar o mesmo papel, porém não sendo mais necessário o agente, como parte explícita do modelo de gerência.

Em relação à MIB, os objetos mantidos pelo agente CORBA não necessariamente residem no mesmo local, como no caso de agente OSI, pois a arquitetura CORBA é tratada como uma infra-estrutura de objetos distribuídos. Este agente mantém uma coleção hierárquica de referências a objetos gerenciados, onde a localização destes objetos é desconhecida para o cliente, mas é a mesma da *factory* associada a sua classe de objetos.

A grande maioria de *bindings* oferecidos hoje para CORBA são baseados no uso da interface de invocação estática. Isso é natural pelo fato de linguagens tais como C, C++ e Java serem tipadas estaticamente. Sendo assim, esse trabalho utiliza a interface de invocação estática, por ser mais adequado usar os *stubs* gerados em tempo de compilação, a partir de descrições IDL dos objetos gerenciados a serem invocados. A invocação dinâmica é utilizada no caso da notificação de eventos, para que o notificador do evento possa continuar a executar e logo após enviar a notificação (ação **`send_oneway`**).

O uso dos serviços e facilidades da arquitetura CORBA associados à OMG IDL (definição das interfaces de objetos) para o gerenciamento de redes de telecomunicações, permite a comunicação eficiente entre diferentes plataformas de computação. Por exemplo, a comunicação entre um Agente em um PC (apenas para a criação e exclusão de objetos), e um Gerente em uma estação SOLARIS, e também, um elemento de rede que emita alarmes, o armazenamento dos mesmos e posterior relatório de sumarização de alarmes. Isto nos mostra evidentemente que a arquitetura CORBA pode ser usada para implementar aplicações distribuídas e heterogêneas.

Uma grande vantagem do uso de CORBA na área de gerência TMN é que os desenvolvedores de aplicação se abstém de protocolos de gerência, como CMIP, e passam a atuar no nível mais abstrato de chamadas de métodos.

Devido à linguagem de definição de interfaces (IDL) e sua similaridade a C++, o padrão CORBA possibilita a simplicidade de ambientes de desenvolvimento com relação a GDMO e ASN1. Estes ambientes podem utilizar diferentes linguagens de programação, onde as diferenças de cada linguagem são mascaradas pela CORBA, através do mapeamento IDL. É por isso que CORBA é tida como "isoladora de Sistemas Operacionais e independente de linguagem", devido ao uso da IDL para tratar a heterogeneidade do ambiente (diferentes tipos de máquinas, sistemas operacionais e linguagens de programação), onde para cada objeto CORBA é especificada uma interface.

Uma importante e principal característica deve ser notada na arquitetura CORBA, a interoperabilidade e portabilidade. A portabilidade de código (através do Java) e a interoperabilidade de comunicação, através do protocolo GIOP (*General Inter-ORBs Protocol*).

CORBA oferece transparência de acesso e de localização, onde os objetos gerenciados, locais ou remotos, podem ser acessados da mesma maneira através do ORB, e localizados em diferentes endereços, sem o usuário saber sua localização.

A arquitetura CORBA possibilita a reutilização de conceitos e especificações disponíveis na TMN [FER 96], quando for necessário ou viável. Também, os conceitos TMN de suas arquiteturas lógica, física e de informação são mantidas em um sistema TMN baseado em CORBA [TEL 96].

Para alcançar a integração de três domínios: elementos de redes, sistemas e aplicações, pode-se utilizar uma arquitetura de gerência integrada (aberta e distribuída) [QUE 97] que oferece uma infra-estrutura e um conjunto de serviços úteis para a gerência em um ambiente CORBA. Os componentes desta arquitetura são objetos CORBA, acessíveis por operações ORB como protocolo de gerência.

Esta arquitetura é formada por um conjunto de aplicações cooperativas oferecendo uma interface única ao usuário, abordando as áreas funcionais de desempenho, de contabilização, de falhas e de segurança. Estas áreas funcionais habilitam uma visão logicamente centralizada do domínio gerenciado, independente da distribuição física de seus componentes. Também faz parte desta arquitetura um conjunto de Facilidades para Monitorização, para Controle, para Configuração, para o estabelecimento de Políticas e de Domínios, usadas para o desenvolvimento das aplicações. O nível de suporte da arquitetura de gerência integrada é composto por um ORB com CORBAservices (Anexo II) e CORBAfacilities (Anexo II), e

permite o desenvolvimento de aplicações de gerência, simplificando a implementação e garantindo interoperabilidade [QUE 97].

Com base em tudo o que foi desenvolvido e apresentado por este trabalho, chega-se a conclusão que é possível desenvolver um sistema de gerenciamento distribuído aberto em TMN, através da emergente tecnologia de processamento distribuído orientado a objeto: a arquitetura CORBA.

Durante a realização deste trabalho e através dos resultados obtidos, surgiram idéias que podem dar continuidade a este, tais como:

- desenvolver no ambiente CORBA a Interface do Usuário, parte do modelo de referência de gerenciamento (visto no capítulo 5);
- verificar se todos os serviços oferecidos pelo CORBA, pertencentes ao COSS, possam ser utilizados pela rede de gerenciamento de telecomunicações;
- implementar a especificação IDL do elemento de rede, comutador ATM, bem como o lado aplicação-gerente TMN para o sistema gerenciado ATM, através do ambiente CORBA;
- implementar o sistema de supervisão de alarmes em outros ambientes de desenvolvimento CORBA, bem como com a utilização de outras linguagens de programação; realizar a comunicação e comparação entre eles;
- para aumentar a automatização do sistema de supervisão de alarmes TMN com CORBA, seria necessário desenvolver o tratamento de alarmes e ações de reparo para cada alarme.

Anexo I - Especificação IDL do *Framework* de Gerenciamento

Arquivo **"cmf.idl"**:

```
module CMF {

      interface Iterator {
            any nextElement ();
            boolean hasMoreElements ();
      };
      /* Um Iterator permite iterar os elementos de uma colecao. Normalmente usa-se um objeto Iterator em
      metodos que retornam um conjunto de elementos. O iterator permite que se navegue nessa colecao de
      elementos, e permite a iteração de coleções com os mais variados tipos de dados (int, struct, objetos, enum,
      etc...). */

      interface Container {
            Iterator getElements ();
      };
      // interface ancestral de List e PropertyList

      interface List : Container {
            void add (in any value);
            void remove (in any value);
      };
      /* lista para qualquer tipo de dados (any), útil para declarar atributos com valores compostos. */


      interface PropertyList : Container {
            void addProperty (in string name,in any value);
            void removeProperty (in string name);
            any getPropertyValue (in string name);
            boolean hasProperty (in string name);
      };
      /* lista de propriedades, onde uma propriedade é um par nome-valor, e o valor pode ser de qualquer tipo
      (any); a sua importância está em permitir que vários parâmetros sejam passados a um objeto, sem
      obedecer um número fixo (como é exigido nos métodos em IDL). */


      interface ManagedObject;
      /* declaração prévia, necessária para que o compilador saiba que este nome ManagedObject será definido
      posteriormente, e assim, o campo source da struct event (abaixo), não provocará erro. */

      struct Event {
            string name; // nome do evento, por exemplo: objectCreation
            long long time; // momento da ocorrência do evento
            ManagedObject source; // objeto responsável pela geração do evento
            PropertyList data; // informações adicionais pertinentes ao evento especifico
      };
      // O evento propriamente dito
```

```
enum Operator {equal,notEqual,less,greater,lessOrEqual,greaterOrEqual};
// Estrutura que representa um critério a ser aplicado a um objeto CORBA.

struct Criterion {
      string attributeName; // qual atributo (ex.: "className")
      Operator operator; // qual operador (ex.: equal)
      any value; // qual o valor de comparação (ex.: "Keyboard")
};
/* Como exemplo, o critério representa para um objeto x, o seguinte:
x.className == "Keyboard"
Desta forma, pode-se determinar critérios para repasse de eventos. Assim, um gerente interessado apenas
nos eventos gerados por teclados usaria um critério semelhante ao exemplificado. */

interface EventForwarder;
// declaração prévia


interface ManagedObject {
      readonly attribute EventForwarder eventForwarder;
      readonly attribute string className;
      readonly attribute string DN;
      attribute string RDN;
      attribute ManagedObject parent;
      ManagedObject getSubObject (in string subRDN);
      Iterator selectSubObjects (in string conditions);
      void addSubObject (in ManagedObject subObject);
      void removeSubObject (in string subRDN);
      void debug ();
};
// Classe básica de todos os objetos gerenciados, os quais são organizados através da representação de
agregação (containment) e organização de nomes através de RDN (Relative Distinguished Name); e seus atributos e
operações são específicas para cada classe ou objeto.


interface ClassImplementor {
      readonly attribute string className;
      ManagedObject createObject (in PropertyList args);
      void deleteObject (inout ManagedObject theObject);
};
// interface para todas as implementações de classes; é esta a classe que funciona como fábrica para criar
instâncias de classes, como também novas classes e objetos;


interface Agent {
      readonly attribute ManagedObject root; // referencia ao objeto
raiz
      readonly attribute string name; // nome do agente
      void registerClass (in ClassImplementor ci);
      void unregisterClass (in ClassImplementor ci);
      ManagedObject createObject (in string DN,in string className,in
      PropertyList args);
      void deleteObject (in string DN);
      ManagedObject getObject (in string DN);
};
// interface para acesso a um agente; é a interface que pede para a ClassImplementor criar um objeto/
classe, manipula a hierarquia distribuída, associa nomes a objetos (de criar ou destruir objetos), e o de
efetuar ou desfazer o registro de classes de objetos;
}
```

Anexo II - Serviços e Facilidades da Arquitetura CORBA

Os Serviços de Objeto são interfaces de domínio independente que são usadas por muitos programas de objetos distribuídos. Por exemplo, um serviço fornecido para a descoberta de outros serviços disponíveis é quase sempre necessário no que diz respeito ao domínio da aplicação. Este serviço pode ser o de nomeação (que permite aos clientes acharem objetos baseados em nomes) e o de *trading* (que permite aos clientes acharem objetos baseados em suas características).

Os serviços foram especificados em cinco RFP's (*Request For Proposals*) ao longo dos últimos quatro anos [EDW 95] e [TEL 96].

RFP 1

- ***EVENT Service:***

- Permite aos objetos registrar e anular (o registro) dinamicamente seus interesses em eventos.

- ***LIFE CYCLE Service:***

- Provê operações para criar, copiar, mover e destruir objetos.

- ***NAMING Service:***

- Mapeamento entre nome e objeto.

- ***PERSISTENCE Service:***

- Um objeto persistente precisa manter seu estado entre sessões de execução, exigindo armazenamento não-volátil.

- Acomoda diversos serviços de armazenamento, bancos de dados relacionais e orientados a objetos, documentos compostos, ...

- É a interface única de objeto para armazenamentos diversos.

- É a persistência de objetos independente do tempo de vida da aplicação cliente e implementação que executa os métodos.

RFP 2

- ***CONCURRENCY Service:***

- Permite que os objetos coordenem seus acessos a recursos compartilhados.

- Complementa o Serviço de Transação oferecendo obtenção e liberação de bloqueios nos limites transacionais.

- ***EXTERNALIZATION Service:***

- Um objeto é "exteriorizado" para um *stream* para ser transportado a um diferente processo, máquina ou ORB. O objeto é "interiorizado" a partir do *stream* em seu novo destino para ser usado.

- É a conversão do estado do objeto para transmissão entre sistemas, através de meio diferente do ORB.

- ***RELATIONSHIP Service:***

- Permite criar relacionamentos arbitrários entre objetos que são totalmente despreparados para os relacionamentos.

- É a associação entre dois ou mais objetos.

- ***TRANSACTION Service:***

- Transações na execução de operações de *atomicidade* (quando ocorrem falhas, os resultados parciais são desfeitos), *isolamento* (execução concorrente, mas independente) *e durabilidade* (resultado bem sucedido nunca é perdido).

RFP 3

- ***SECURITY Service*:**

- Controle de acesso aos objetos (*autenticação, autorização, integridade, privacidade, auditoria, criptografia).*

- ***TIME Service*:**

- Mantém uma noção única de tempo no sistema de objetos distribuídos.

RFP 4

- ***LICENSING Service*:**

- Permite definir uma gama de opções de licenciamento dos objetos.

- Controla e gerencia remuneração dos serviços executados.

- ***PROPETY Service*:**

- Permite criar e associar propriedades a objetos dinamicamente.

- ***QUERY Service*:**

- Permite encontrar objetos cujos atributos atendem ao critério de seleção especificado na consulta.

- As operações são feitas sobre conjuntos e coleções de objetos.

- Está sendo desenvolvido um trabalho para criar uma única linguagem de consultas para objetos.

RFP 5

- ***COLLECTION Service:***

- Permite manipular objetos em grupos, provendo uma forma de criar e manipular as coleções mais comuns.

- As operações de coleções são aplicadas em grupos em vez de objetos individuais, e.g., *queus, stacks, lists, arrays, trees, sets e bags.*

- ***CHANGE MANAGEMENT Service:***

- Possibilita gerenciar diferentes versões de objetos (e suas interfaces) conforme evoluam. Também mantém a história desta evolução.

- Identificação e avaliação da consistência de configurações do objeto.

- ***TRADER Service:***

- Mapeamento entre propriedades e objeto; busca de objetos; clientes o usam para encontrar listas de serviços, serviços por tipo... como páginas amareladas.

As facilidades comuns são a mais nova área de padronização em CORBA. Estas facilidades são componentes que devem ser implementados como modelos e representam o mais alto nível de recursos disponíveis para o desenvolvedor de aplicações. Estes componentes são definidos em IDL e disponíveis para os objetos da aplicação.

Como as interfaces de Serviço de Objeto, estas interfaces também são orientadas horizontalmente, mas diferentemente de Serviços de Objeto, elas são aplicações orientadas para o usuário final.

As facilidades comuns podem ser configuradas para requisitos específicos de uma configuração particular [EDW 95]. Existem duas categorias de facilidades, as **horizontais** e as **verticais**:

- **Facilidades Horizontais**: devem concentrar as funções exigidas pela maioria das aplicações. Atualmente, não existem muitos modelos para auxiliar construção de aplicações;
- **Facilidades Verticais**: devem prover interfaces IDL e modelos para o desenvolvimento de aplicações em áreas específicas como automação industrial, processamento de imagens, exploração de petróleo e outras infinidades de áreas que necessitam de recursos específicos. Uma facilidade comum e várias áreas específicas tende a se tornar uma facilidade horizontal.

USER INTERFACE

- Compound Presentation
- Desktop Management
- Rendering Management
- Scripting
- User Support

INFORMATION MANAGEMENT

- Compound Interchange
- Data Enconding / Representation
- Data Interchange
- Information Exchange
- Information Modeling
- Information Storage / Retrieval
- Time Operations

SYSTEM MANAGEMENT

- Collection Management
- Consistency
- Instrumentation
- Policy Management

- Customization
- Data Collection
- Event Management
- Instance Management
- Process Launch
- Quality of Service Mgnt
- Sheduling Management
- Security

TASK MANAGEMENT

- Agent
- Automation
- Rule Management
- Workflow

VERTICAL MARKET

- Accounting
- Computer Integrated Manufacturing
- Distributed Simulation
- Information Superhighways
- Mapping
- Security
- Application Development
- Currency
- Imagery
- Internationalization
- Oil / Gas Exploration / Production
- Telecommunication

Tabela A.II. 1 - Facilidades Comuns da CORBA

Anexo III - Interface Definition Language (IDL)

A linguagem de definição de interface da arquitetura CORBA, a IDL, é utilizada para descrever as interfaces das implementações de objetos, que são acessadas por seus clientes.

Uma interface descrita em IDL especifica o formato da chamada das operações providas pelo objeto e cada um dos parâmetros necessários para efetuar a operação.

De posse de um arquivo de definição de interface de uma implementação de objeto, de extensão .idl , o cliente possui toda a informação que necessita para obter uma determinada operação deste objeto.

As interfaces definidas no arquivo IDL podem ser acessadas através de rotinas *stub*, geradas na compilação, ou através da Interface de Invocação Dinâmica (DII), que utiliza a informação contida no depósito de interfaces para montar uma invocação a um objeto.

Aspectos Léxicos e o Pré-processamento da Linguagem IDL

As regras léxicas da linguagem IDL são as mesmas da linguagem C++, já considerando modificações introduzidas pelo trabalho de padronização realizado pela ANSI, e acrescida de algumas palavras-chave para suportar a arquitetura definida pela OMG.

As regras gramaticais da linguagem IDL são um subconjunto das regras de ANSI C++, acrescidas de construções para o tratamento do mecanismo de invocação de operações.

A IDL é uma linguagem puramente declarativa, sem definição de variáveis ou estruturas algorítmicas. São descritos em IDL apenas os tipos, constantes e operações necessários para especificar uma interface de objeto.

Os *tokens* identificados pelo compilador IDL podem ser de cinco tipos diferentes: identificadores, palavras-chave, literais (1, 2.37, 'a', "string", ...), operadores (| , & , * , = , ...) e separadores (espaços, tabulações, caracteres de nova linha e de comentário). Os separadores são utilizados apenas para delimitar *tokens*, sendo ignorados pelo processo de compilação.

PARTE DA ESPECIFICAÇÃO OMG IDL

➔ < specification > ::= < definition >

➔ < definition > ::= < type-dcl > ";"
| < const-dcl > ";"
| < except-dcl > ";"
| < interface > ";"
| < module > ";"

➔ < module > ::= "module"< system-identifier > "{" < definition > "}" ";"

➔ < interface > ::= < interface-dcl >
| < forward-dcl >

➔ < interface-dcl > ::= < interface-header > { < interface-body > } ";"
< interface-header > ::= "interface" < object-identifier > [< inhentance-spec >]
< interface-body > ::= < type-dcl > ";"
| < const-dcl > ";"
| < except-dcl > ";"
| < attr-dcl > ";"
| < op-dcl > ";"
< inheritance-spec > ::= ":" < scoped-name > { ","< scoped-name > } *

➔ < forward-dcl > ::= "interface" < identifier >

➔ < scoped-name > ::= < identifier >
| "::"< identifier >
| < scoped-name > "::" < identifier >

➔ < op-dcl > := [< op-attribute >] < op-type-spec > < identifier > < parameter-dcls >
[< raises-expr >] [< context-expr >]

< op-attribute > ::= "oneway"

< op-type-spec > ::= < param-type-spec >

| "void"

< param-type-spec > ::= < base-type-spec >

| < string-type >

| < scoped-name >

< parameter-dcls > ::= "(" < param-dcl > { ","< param-dcl > } * ")"

| "(" ")"

< param-dcl > ::= < param-attribute > < param-type-spec > < identifier >

< param-attribute > ::= "in"

| "out"

| "inout"

< raises-expor > ::= "raises" "(" < scoped-name > { ","< scoped-name> } * ")"

< context-expr > ::= "context" "(" < string-literal > { "," < string-literal > } * ")"

➔ < attr-dcl > ::= ["readonly"] "attribute" < param-type-space > <identifier>

{ "," < identifier > } *

➔ < except-dcl > ::= "exception"< identifier > "{"< member> * "}"

< member > ::= < type-spec > < declarators > ";"

< declarators > ::= < declarator > { ","< declarator > } *

< declarator > ::= < simple-declarator >

| < complex-declarator >

< simple-declarator > ::= < identifier >

< complex-declarator > ::= < array-declarator >

< type-dcl > ::= "typedef <type-declarator>

| < struct-type >

| < union-type >

| < enum-type >

< type-declarator > ::= <type-spec > < declarators >

Anexo IV - Common Management Infomation Protocol (CMIP)

Este anexo tem por objetivo apresentar o protocolo CMIP, seus serviços e operações. CMIP é o protocolo de comunicação usado para a troca de mensagens de gerenciamento. Ele faz parte do CMISE, que é o elemento de serviço de gerenciamento de redes do modelo OSI.

1. Os Serviços Oferecidos

Os serviços oferecidos pelo elemento CMISE são agrupados em três categorias: serviços de associação, serviço de notificação de gerenciamento e serviços de operações de gerenciamento [CMI 94]. Os serviços listados na Tabela A.IV.1 podem ser confirmados e não-confirmados. Os serviços confirmados requerem que o respondedor do serviço de gerenciamento sinalize o sucesso ou o erro na execução da requisição. O serviço de notificação de gerenciamento também é definido para operação em modo confirmado, para que haja a garantia de recepção de eventos críticos. Os serviços não-confirmados não emitem respostas às requisições e sinalizações recebidas.

Serviço	Modo
Associação	
A-Associate	confirmado
A-Release	confirmado
A-Abort	não-confirmado
Notificação de Gerenciamento	
M-Event-Report	confirmado/não-confirmado
Operações de Gerenciamento	
M-Get	confirmado
M-Cancel-Get	confirmado
M-Set	confirmado/não-confirmado
M-Action	confirmado/não-confirmado
M-Create	confirmado
M-Delete	confirmado

Tabela A.IV. 1 - Serviços do CMISE.

Enquanto o CMIS define os serviços para operações de gerenciamento, o CMIP define *procedures* para a transmissão de informações de gerenciamento e define a sintaxe para o serviço

de gerenciamento do CMIS. O CMIP é definido em termos de unidades de dados do protocolo CMIP que são trocadas entre iguais CMISEs levando do serviço CMIS.

1.1. SERVIÇOS DE ASSOCIAÇÃO

Utilizados para estabelecer associações de aplicação, baseadas no ACSE.

Os Serviços de Associação são:

- A-ASSOCIATE: solicita o estabelecimento de uma associação;
- A-RELEASE: solicita a liberação normal da associação;
- A-ABORT: solicita e informa a liberação repentina da associação.

Para que dois usuários façam uso dos serviços de gerenciamento oferecidos pelo CMIS, eles devem inicialmente estabelecer uma associação. Para isso, o CMIS faz uso dos serviços oferecidos pelo ACSE, que além de cuidar da associação, ele estabelece os recursos do CMIS que serão usados pelos usuários na associação.

Os recursos que podem ser negociados são agrupados nas chamadas Unidades Funcionais. O serviço padrão oferecido pelo CMIS faz parte da chamada Unidade Funcional *Kernel*, que contém todas as primitivas de serviço CMIS com exceção do M-*Cancel-Get*, e que não permite a realização de seleção múltipla de objetos através dos parâmetros de filtragem, escopo e sincronismo, e conseqüente presença de respostas múltiplas.

1.2. SERVIÇOS DE OPERAÇÕES DE GERENCIAMENTO

Eles são usados pelo gerente para a aplicação de operações sobre os dispositivos controlados pelo agente. Os Serviços de Operações de Gerenciamento são:

- → M-*Get*: solicita a busca de informações de gerenciamento;
- → M-*Cancel-Get*: solicita o cancelamento de um serviço M-Get previamente requisitado e ainda pendente;
- → M-*Set*: solicita a modificação da informação de gerenciamento;
- → M-*Action*: solicita a execução de uma ou mais ações em cima dos objetos gerenciados;
- → M-*Create*: solicita a criação de uma instância de um objeto gerenciado;
- → M-*Delete*: solicita a exclusão de uma ou mais instâncias de objetos gerenciados.

O serviço M-*Get* é o único que tem a opção de cancelamento (M-*Cancel-Get*) definida pelo CMISE. Isto porque seria difícil assegurar a consistência da base de gerenciamento (MIB)

se as outras operações que alteram seu conteúdo também pudessem ser canceladas durante a execução.

O CMISE define a seleção de múltiplos objetos para as operações de gerenciamento M-*Get*, M-*Set*, M-*Action* e M-*Delete* em modo confirmado.

A descrição dos serviços de operação de gerenciamento e do serviço de notificação de gerenciamento apresenta tabelas com os parâmetros de invocação e os parâmetros retornados como resposta. Para as tabelas teremos a seguinte representação:

- O : o parâmetro tem presença obrigatória;
- U : o uso do parâmetro é uma opção do usuário do serviço;
- - : o parâmetro não está presente na primitiva em questão;
- C : a presença do parâmetro é condicionada à resposta do serviço.

A seguir, veremos cada parâmetro com seus códigos de erro pertencentes ao Serviço de Operação de Gerenciamento CMIS:

→ M-GET

Parâmetro	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Linked Identifier	-	C
Base-Object Class	O	-
Base-Object Instance	O	-
Scope	U	-
Filter	U	-
Access Control	U	-
Synchronization	U	-
Attribute-Identifier List	U	-
Managed-Object Class	-	C
Managed-Object Instance	-	C
Current Time	-	U
Attribute List	-	C
Errors	-	C

Tabela A.IV. 2 - Parâmetros do serviço M-*Get*

Erros reportados pelo serviço: acesso negado, conflito de classe/instância, limitação de complexidade, requisição duplicada, erro de obtenção de lista, filtro inválido, escopo inválido, argumento incorreto, classe inválida de objeto, erro no processamento, limitação de recurso, sincronismo não suportado, e operação não reconhecida.

➔ M-CANCEL-GET

Parâmetros	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Get Invoke Identifier	O	-
Errors	-	C

Tabela A.IV. 3 - Parâmetros do serviço M-*Cancel-Get*

Erros reportados pelo serviço: requisição duplicada, operação incorreta, identificador inexistente, erro no processamento, limitação de recurso, e operação não reconhecida.

➔ M-SET

Parâmetro	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Linked Identifier	-	C
Mode	O	-
Base-Object Class	O	-
Base-Object Instance	O	-
Scope	U	-
Filter	U	-
Access Control	U	-
Synchronization	U	-
Managed-Object Class	-	C
Managed-Object Instance	-	C
Modification List	O	-
Attribute List	-	U
Current Time	-	U
Errors	-	C

Tabela A.IV. 4 - Parâmetros do serviço M-*Set*

Erros reportados pelo serviço: acesso negado, conflito de classe/instância, limitação de complexidade, requisição duplicada, filtro inválido, escopo inválido, argumento incorreto, classe de objeto inexistente, instância de objeto inexistente, erro no processamento, limitação de recurso, erro de atribuição de lista, sincronismo não suportado, e operação não reconhecida.

➔ M-ACTION

Parâmetro	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Linked Identifier	-	C
Mode	O	-
Base-Object Class	O	-
Base-Object Instance	O	-
Scope	U	-

Filter	U	-
Managed-Object Class	-	C
Managed-Object Instance	-	C
Access Control	U	-
Synchronization	U	-
Action Type	O	C
Action Information	U	-
Current Time	-	U
Action Reply	-	C
Errors	-	C

Tabela A.IV. 5 - Parâmetros do serviço M-*Action*

Erros reportados pelo serviço: acesso negado, conflito de classe/ instância, limitação de complexidade, requisição duplicada, valor do argumento inválido, filtro inválido, escopo inválido, argumento incorreto, ação inexistente, argumento inexistente, classe de objeto inexistente, instância de objeto inexistente, erro no processamento, limitação de recurso, sincronismo não suportado, e operação não reconhecida.

➔ M-CREATE

Parâmetro	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Managed-Object Classe	O	U
Managed-Object Instance	U	C
Superior-Object Instance	U	-
Access Control	U	-
Reference-Object Instance	U	-
Attribute List	U	C
Current Time	-	U
Errors	-	C

Tabela A.IV. 6 - Parâmetros do serviço M-*Create*

Erros reportados pelo serviço: acesso negado, conflito de classe/instância, requisição duplicada, instância de objeto gerenciado duplicada, valor inválido do atributo, instância inválida do objeto, falta de valor do atributo, argumento incorreto, atributo inexistente, classe de objeto inexistente, instância de objeto inexistente, referência de objeto inexistente, erro no processamento, limitação de recurso, e operação não reconhecida.

➔ M-DELETE

Parâmetro	*req/ind*	*resp/conf*
Invoke Identifier	O	O
Linked Identifier	-	C
Base-Object Class	O	-

Base-Object Instance	O	-
Scope	U	-
Filter	U	-
Access Control	U	-
Synchronization	U	-
Managed-Object Class	-	C
Managed-Object Instance	-	C
Current Time	-	U
Errors	-	C

Tabela A.IV. 7 - Parâmetros do serviço M-*Delete*

Os erros reportados pelo serviço: acesso negado, conflito de classe/instância, limitação de complexidade, requisição duplicada, filtro inválido, escopo inválido, argumento incorreto, classe de objeto inexistente, instância de objeto inexistente, erro no processamento, limitação de recurso, sincronismo não suportado, e operação não reconhecida.

1.3. SERVIÇO DE NOTIFICAÇÃO DE GERENCIAMENTO

Ele é definido pelo CMIS (M-*Event-Report*) e é usado pelo agente para informar ao gerente a ocorrência de eventos relacionados com os dispositivos gerenciados.

M-Event-Report		
Parâmetro	*req/inf*	*resp/conf*
Invoke Identifier	O	O
Mode	O	-
Managed-object class	O	U
Managed-object instance	O	U
Event type	O	C
Event time	U	-
Event information	U	-
Current time	-	U
Event reply	-	C
Errors	-	C

Tabela A.IV. 8 - Parâmetros do serviço M-*Event-Report*

Os erros reportados pelo serviço são tratados como exceções em IDL. São eles: requisição duplicada, valor do argumento inválido, argumento incorreto, argumento inválido, classe inválida de objeto, instância inválida de objeto, erro no processamento, limitação de recurso, e operação não reconhecida.

2. OPERAÇÃO DO CMIP

Para entender a operação do CMIP, precisamos vê-lo no contexto com o uso do serviço CMISE e os serviços em que o CMISE confia. O CMIS prevê serviços para executar operações de gerenciamento, na forma de primitivas de serviço.

Os usuários do CMISE são capazes de estabelecer associações ordenadas para executar operações de gerenciamento. Para os serviços de operação de gerenciamento, o CMISE emprega um CMIP para trocar PDUs.

Em função das características dos serviços CMIS, são onze as PDUs que compõem o CMIP: m-*EventReport*; m-*EventReport-Confirmed*; m-*Get*; m-*Linked-Reply*; m-*Set*; m-*Set-Confirmed*; m-*Action*; m-*Action-Confirmed*; m-*Create*; m-*Detete*; m-*Cancel-Get-Confirmed.*

Em cada PDU existem três tipos de informação: os argumentos (parâmetros necessários para realizar a operação requisitada pela primitiva de serviço CMISE), os resultados e os erros que podem ser retornados após a execução das operações. A Tabela A.IV.9 mostra o mapeamento entre as primitivas CMIS e as PDUs do CMIP.

PRIMITIVA CMIS	MODO	LINKED-ID	PDU CMIP
Notificação de Gerenciamento			
M-Event-Report req/ind	não-confirmado	N/A	*m-EventReport*
M-Event-Report req/ind	confirmado	N/A	*m-EventReport-Confirmed*
M-Event-Report resp/conf	N/A	N/A	*m-EventReport-Confirmed*
Operações de Gerenciamento			
M-Get req/ind	confirmado	N/A	*m-Get*
M-Get resp/conf	N/A	ausente	*m-Get*
M-Get resp/conf	N/A	presente	*m-Linked-Reply*
M-Cancel-Get req/ind	confirmado	N/A	*m-Cancel-Get-Confirmed*
M-Cancel-Get resp/conf	N/A	N/A	*m-Cancel-Get-Confirmed*
M-Set req/ind	não-confirmado	N/A	*m-Set*
M-Set req/ind	confirmado	N/A	*m-Set-Confirmed*
M-Set resp/conf	N/A	ausente	*m-Set-Confirmed*
M-Set resp/conf	N/A	presente	*m-Linked-Reply*
M-Action req/ind	não-confirmado	N/A	*m-Action*
M-Action req/ind	confirmado	N/A	*m-Action-Confirmed*
M-Action resp/conf	N/A	ausente	*m-Action-Confirmed*
M-Action resp/conf	N/A	presente	*m-Linked-Reply*
M-Create req/ind	confirmado	N/A	*m-Create*
M-Create resp/conf	N/A	N/A	*m-Create*
M-Delete req/ind	confirmado	N/A	*m-Delete*
M-Delete resp/conf	N/A	ausente	*m-Delete*

M-Delete resp/conf	N/A	presente	*m-Linked-Reply*

Tabela A.IV. 9 - Correspondência entre as primitivas do CMISE e as PDUs do CMIP

A abordagem OSI de gerenciamento de redes estabelece uma especificação que descreve as mensagens, os objetos e os atributos mencionados nas normas. No presente trabalho, cada elemento de informação é descrito em IDL.

Temos três tipos de informação conduzidas pelas unidades de dados. Os argumentos da entrada definem os argumentos, ou parâmetros, conduzidos na unidade de dados que são derivados da primitiva de serviço do CMISE disparador. Os resultados e erros de entrada contém informações da entidade que está executando o resultado da operação de gerenciamento de sistemas. Estes valores são derivados da primitiva response do CMISE.

A unidade de dados do CMIP inclui um parâmetro *linked*-Id, em ordem, para múltiplas respostas de *link* causadas pela operação invocada.

Anexo V - General Inter-ORB Protocol (GIOP)

Este anexo apresenta a comunicação entre ORBs da arquitetura CORBA, realizada pelo protocolo GIOP [COR 95].

1. O Protocolo GIOP

GIOP (*General Inter-ORB Protocol*) é o Protocolo de Inter ORBs Genérico utilizado para prover a interoperabilidade entre os ORBs. Ele pode ser mapeado em qualquer protocolo de transporte orientado a conexão que ofereça um conjunto mínimo de suposições/hipóteses [COR 95]. O GIOP é especificado como a interface comum, que inclui o tipo da mensagem e formato, proporcionando assim, a interoperabilidade comum.

Um protocolo específico existente atualmente é o IIOP (*Internet Inter-ORB Protocol*), que especifica a implementação do GIOP através do transporte baseado no protocolo TCP/IP. Um protocolo não GIOP é o ESIOPs (*Environment-Specific Inter-ORB Protocol*), o qual permite ORBs se comunicarem mantendo a característica de herança no ambiente de computação distribuída (DCE - *Common Inter-ORB Protocol*).

A especificação do GIOP consiste dos seguintes elementos:

- **A Definição da Representação de Dados Comum (CDR):** define representações para todos os tipos de dados IDL OMG, incluindo tipos básicos e/ou estruturados, tais como, estrutura, seqüência e enumerados. A principal característica oferecida pela CDR é que todos os GIOPs compartilham uma representação de dados comum, facilitando assim, a ligação desta parte da mensagem, para a comunicação entre ORBs e pontes inter ORBs (agentes);
- **Formatos da Mensagem GIOP:** O GIOP define sete mensagens distintas para simplificar as comunicações ORB-ORB, que são trocadas entre agentes para facilitar os pedidos do objeto, localizar implementações do objeto, e gerenciar canais de comunicação. Algumas características da transferência de mensagens são: a conexão é assimétrica (os papéis de cliente e servidor são distintos); os pedidos podem ser multiplexados (clientes múltiplos com

um ORB originário podem compartilhar uma conexão para ORB remoto); e, os pedidos podem sobrepor-se (se forem assíncronos, os pedidos e respostas são suportados através dos identificadores *request/reply*). Das sete mensagens, três podem ser enviadas pelo cliente: *request, cancelRequest e locateRequest*; três podem ser somente enviadas pelo servidor: *reply, locateReplay e closeConnection*; e ambos podem enviar *messageError*.

- **Suposições de Transporte GIOP:** a especificação GIOP descreve as suposições gerais sobre qualquer camada de transporte de rede que pode ser usada para transferir mensagens GIOP. A especificação também descreve como conexões podem ser gerenciadas, e reservadas na ordenação da mensagem GIOP.

1.1 Representação de Dados Comuns (CDR)

A CDR é uma sintaxe de transferência, que possibilita o mapeamento de tipos de dados em IDL OMG e a representação de baixo nível para a transferência entre agentes. Ela apresenta as seguintes características:

- Ordenando *Byte* Variável: as máquinas com um *byte* comum podem permutar mensagens sem a troca de *byte*. Quando as máquinas de comunicação têm *byte* diferente, a mensagem originária determina a mensagem do *byte*, e o recebedor é responsável pela troca de *bytes* em sua ordenação original. Cada mensagem GIOP (e a encapsulação CDR) contém um sinal que indica o *byte* apropriado.
- Tipos Primitivos Alinhados: os tipos de dados primitivos IDL OMG são alinhados em seus limites simples dentro das mensagens GIOP, permitindo que os dados sejam manipulados eficientemente pelas arquiteturas que impõe o alinhamento de dados na memória.
- Mapeamento IDL OMG Completo: a CDR descreve representações para todos os tipos de dados IDL OMG, incluindo os pseudo-objetos transmissíveis, como *TypesCodes*. Necessariamente, a CDR define as representações de tipos de dados, as quais são indefinidas ou dependentes da implementação nas especificações CORBA.

Sintaxe de Transferência CDR

A sintaxe de transferência da Representação de Dados Comum é o formato em que o GIOP representa os tipos de dados IDL OMG em um *stream* de oito *bits*.

Um *stream* de oito *bits* é uma noção abstrata de um armazenamento na memória, que é enviado para outro processo ou máquina, dentro de algum mecanismo de IPC (*Inter-Process Comunication*) ou transporte pela rede. Ele é uma seqüência extensa e arbitrária (mas finita) de valores de oito *bits* (*octet*) com um começo bem definido. Os oito *bits* no *stream* são numerados de 0 a n-1, onde n é o tamanho do *stream*. A posição numérica de um conjunto de oito *bits* no *stream*, é chamada de ***index***. Seus indicadores são usados para calcular os limites do alinhamento.

O protocolo GIOP define dois *streams* de oito *bits* distintos: as mensagens e encapsulações. As mensagens são unidades básicas da troca de informação no GIOP (descritas posteriormente).

As encapsulações são *streams* de oito *bits* nas quais as estruturas de dados IDL OMG podem ser montadas/organizadas independentemente, separadas do contexto de mensagem particular. Uma vez que a estrutura de dados tenha sido encapsulada, o *stream* de oito *bits* pode ser representado através do tipo de dados **sequence<octet>**, o qual pode subseqüentemente ser montado/organizado em uma mensagem ou em outra encapsulação. As encapsulações permitem que constantes complexas sejam pré-montadas/organizadas, e que determinados componentes da mensagem sejam manipulados, sem requererem a desmontagem/desorganização completa.

1.2. Formato das Mensagens GIOP

Na descrição de mensagens GIOP, é necessário definir regras de cliente e servidor. Um cliente é o agente que inicia uma conexão e origina pedidos. Um servidor é um agente que aceita conexão e recebe pedidos.

Todas as mensagens GIOP iniciam com o seguinte cabeçalho, definido em IDL:

```
module GIOP {
    enum MagType {
        Request, Reply, CancelRequest,
        LocateRequest, LocateReply
        CloseConnextion, MessageError };
    struct MessageHeader {
        char        magic [4];
        Version     GIOP-Version;
```

```
        boolean     byte-order;
        octet       message-type;
        unsigned    long    message-size; };
};
```

Que é utilizado para identificar as mensagens GIOP, mas é definido por *byte* de ordenação independente, desde que o mesmo defina o *byte* de ordenação dos elementos da mensagem subseqüente. Os membros do cabeçalho são:

- **magic**: identifica as mensagens GIOP;
- **GIOP-Version**: contém o número da versão do protocolo GIOP sendo usado na mensagem. O número da versão é aplicado para os elementos independentes do transporte da especificação que constitui o GIOP;
- **byte-order**: indica o byte de ordenação usado em elementos subseqüentes da mensagem;
- **message-type**: indica o tipo da mensagem;
- **message-size**: contém o tamanho da mensagem seguindo o cabeçalho da mensagem, em octetos.

MENSAGEM *REQUEST*

Mensagens *request* codifica invocações ao objeto CORBA, incluindo operações acessoras do atributo, e operações **CORBA :: Object, get-interface** e **get-implementation**. Os *requests* circulam do cliente para o servidor.

As mensagens *request* têm três elementos, codificados nesta ordem:

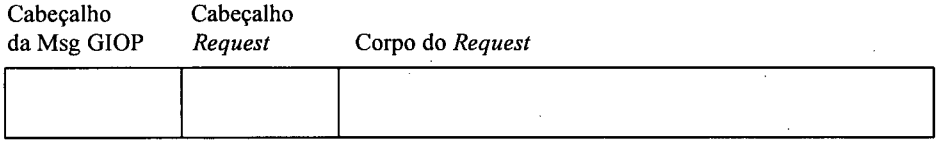

Figura A.V. 1 - Formato da mensagem *request*

CABEÇALHO REQUEST:

O cabeçalho do *request* é especificado como a seguir:

```
module GIOP {
    struct RequestHeader {
        IOP :: ServiceContextList service_context;
```

```
    unsigned    long        request_id;
    boolean                 response_expected;
    sequence <octet>        object_key;
    string                  operation;
    Principal               requesting_principal; };
};
```


Os membros têm as seguintes definições:

- **service_context**: contém os dados do serviço ORB sendo passados do cliente para o servidor;
- **request_Id**: usado para associar as mensagens *reply* com as mensagens *request*;
- **response_expected**: é estabelecido TRUE se o *request* esperado tem um *reply* associado. O valor é FALSE se a operação é definida como *oneway*, ou se a operação é invocada com a DII e a invocação de *flags* inclui o *flag* INV_NO_RESPONSE;
- **object_key**: identifica o objeto que é o alvo da invocação;
- **operation**: contém o nome da operação sendo invocada;
- **requesting_principal**: contém um valor identificando o pedido principal.

CORPO DO REQUEST:

O corpo do *request* inclui todos os elementos **in** e **inout**, na ordem na qual eles são especificados na definição das operações IDL, da esquerda para a direita.

No trabalho em questão, a forma do *request* [COR 95] será incluída no corpo do *request*, parte da mensagem GIOP.

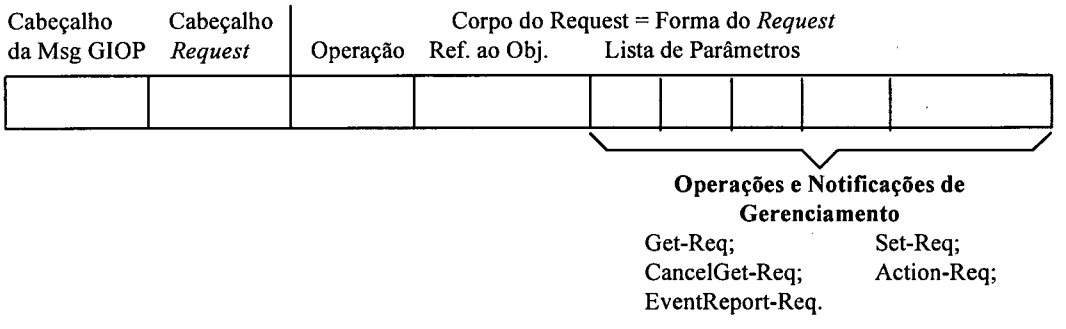

Figura A.V. 2 - Formato da mensagem *request* incluindo a forma do *request*

MENSAGEM *REPLY*

As mensagens *reply* são enviadas em resposta para as mensagens *request*. As respostas incluem parâmetros **out** e **inout**, resultados da operação, e pode incluir valores de exceção. Em adição, mensagens *reply* podem prover a informação da localização do objeto. As respostas circulam do servidor para o cliente.

As mensagens *reply* são codificadas nesta ordem:

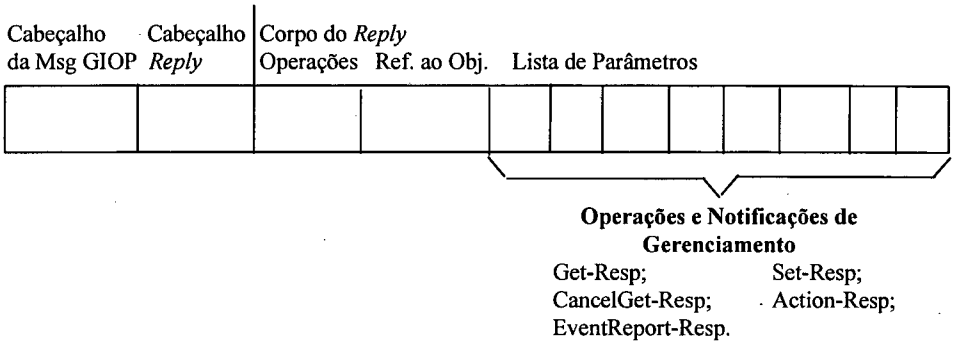

Figura A.V. 3 - Formato da mensagem *reply*

CABEÇALHO DO REPLY:

O cabeçalho do *reply* é definido como segue:

```
module GIOP {
        enum ReplyStatusType {
                NO_EXCEPTION,
                USER_EXCEPTION,
                SYSTEM_EXCEPTION,
                LOCATION_FORWARD };
        struct ReplyHeader {
                IOP :: ServiceContextList       service_context;
                unsigned      long              request_Id;
                ReplyStatusType                 reply_status; };
};
```

Os membros do cabeçalho *reply* têm as seguintes definições:

- **service_context**: contém os dados do serviço ORB sendo passados do servidor para o cliente;

- **request_Id**: usada para associar respostas com *requests*. Ele contém o mesmo valor do *request_Id* como o *request* correspondente;
- **reply_status** indica o estado do *request* associado, e também determina parte do corpo do *reply*.

CORPO DO REPLY:

O formato do corpo do *reply* é controlado pelo valor do **reply_status**. Há três tipos de corpo do *Reply*:

- se o valor do **reply_status** é NO_EXCEPTION, o corpo é codificado como se ele fosse uma estrutura de qualquer operação que retorne valor;
- se o valor do **reply_status** é USER_EXCEPTION ou SYSTEM_EXCEPTION, então o corpo contém a exceção que foi *raised* pela operação;
- se o valor do **reply_status** é LOCATION_FORWARD, então o corpo contém uma referência a objeto. O cliente ORB é responsável por re-enviar o *request* original para o ORB diferente. Este re-enviar é transparente para o programa cliente que esteja fazendo o *request*.

Um exemplo de corpo de *reply* que obtem resposta com sucesso é :

```
struct example_reply {
        double      return_value;
        string      str;
        Principal   p; }
```

Note que o campo **object_key** em qualquer GIOP específico, é servidor relativo e não absoluto. Especificadamente, quando uma nova referência a objeto é recebida em um *Reply* LOCATION_FORWARD ou em uma mensagem *LocateReply*, o campo **object_key** é embutido em novas referências ao objeto do GIOP e podem não ter o mesmo valor, como o **object_key** da referência ao objeto original do GIOP.

MENSAGEM CANCELREQUEST

São mensagens que podem ser enviadas de clientes para servidores. Elas notificam a um servidor que não há mais cliente aguardando um *reply* de um *request* pendente ou mensagem *LocateRequest*.

As mensagens *CancelRequest* têm dois elementos:

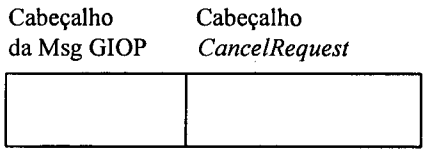

Figura A.V. 4 - Formato das mensagens *cancelRequest*

CABEÇALHO DO CANCELREQUEST:

O cabeçalho do *CancelRequest* é definido como segue:

```
module GIOP {
    struct CancelRequestHeader {
        unsigned    long    request_Id; }
};
```

O membro **request_Id** identifica o *request* ou a mensagem *LocateRequest* para as quais cancela as solicitações. Este valor é o mesmo que o valor do **request_Id** especificado no *request* ou na messagem *LocateRequest* originais.

MENSAGEM LOCATEREQUEST

Pode ser enviada de um cliente para um servidor com o objetivo de determinar: quando a referência a objeto é válida, quando o servidor corrente é capaz de receber pedidos diretamente para a referência a objeto, e se não, como o endereço dos *requests* para a referência a objeto pode ser enviado.

As mensagens *LocateRequest* têm os seguintes elementos:

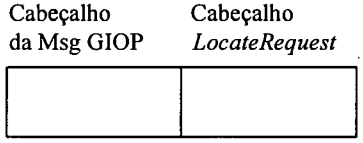

Figura A.V. 5 - Formato da mensagem *locateRequest*

CABEÇALHO DO LOCATEREQUEST:

O cabeçalho do *LocateRequest* é definido como segue:

```
module GIOP {
    struct LocateRequestHeader {
        unsigned    long    request_Id;
        sequence <octet>    object_key; };
```

};

Os membros são definidos como a seguir:

- **request_Id**: usado para associar mensagens *LocateReply* com *LocateRequest*. O cliente (o que requisita) é responsável por gerar valores;
- **object_key**: identifica o objeto sendo localizado.

MENSAGEM LOCATEREPLY

As mensagens são enviadas de servidores para clientes em resposta para as mensagens *LocateRequest*.

As mensagens *LocateReply* têm os seguintes elementos:

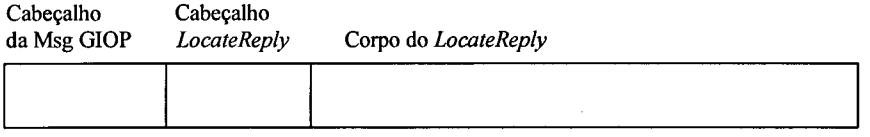

Figura A.V. 6 - Formato da mensagem *locateReply*

CABEÇALHO DO LOCATEREPLY:

O cabeçalho do *LocateReply* é definido como:

```
module GIOP {
    enum LocateStatusType {
        UNKNOWN_OBJECT,
        OBJECT_HERE,
        OBJECT_FORWARD };
    struct LocateReplyHeader {
        unsigned    long    request_Id;
        LocateStatusType    locate_status; };
};
```

Os membros têm as seguintes definições:

- **request_Id**: usado para associar réplicas com *requests*. Este membro contém o mesmo valor *request_Id* que a mensagem correspondente a *LocateRequest*;
- **locate_status**: usado para determinar quando o corpo *LocateReply* existe:

- UNKNOWN_OBJECT: o objeto, especificado na mensagem correspondente *LocateRequest*, é desconhecido para o servidor; nenhum corpo existe;
- OBJECT_HERE: este servidor pode receber pedidos diretamente para o objeto especificado; nenhum corpo existe;
- OBJECT_FORWARD: existe um corpo **LocateReply**.

CORPO LOCATEREPLY:

O corpo é vazio a menos que o valor **LocateStatus** seja OBJECT_FORWARD, neste caso o corpo contém uma referência a objeto que pode ser usada como alvo para os pedidos do objeto especificado na mensagem *LocateRequest*.

MENSAGEM CLOSECONNECTION

É a mensagem enviada somente por servidores. Elas informam aos clientes que o servidor pretende terminar a conexão e não deve ser aguardado para prover respostas futuras.

A mensagem *CloseConnection* consiste somente de cabeçalho da mensagem GIOP, identificando o tipo da mensagem.

MENSAGEM MESSAGEERROR

É enviada em resposta para qualquer mensagem GIOP, cujo número da versão ou o tipo da mensagem é desconhecida para o recebedor, ou qualquer mensagem é recebida cujo o cabeçalho não é formado devidamente.

A mensagem *MessageError* consiste de somente cabeçalho da mensagem GIOP, identificando o tipo da mensagem.

1.3 O Protocolo GIOP e a Camada de Transporte

O GIOP é projetado para ser implementado em uma vasta área de protocolos de transporte. O GIOP tem a seguinte definição do ambiente de transporte:

- Orientado a conexão: o GIOP utiliza conexões para definir o escopo do identificador de *request*;
- Seguro: o transporte garante que os *bytes* são entregues na ordem em que eles foram enviados;

- Visto como um *byte stream*: não há limitações sobre o tamanho da mensagem arbitrária, fragmentações, ou alinhamentos obrigatórios;
- Provê notificação sobre a perda da conexão desordenada: a conexão do proprietário deve receber alguma notificação quando os processos idênticos são abortados, há a colisão de *host* iguais, ou a conectividade da rede é perdida;
- Seu modelo para iniciar conexões pode ser mapeado no modelo de conexão geral do TCP/IP: um agente (descrito como um servidor) divulga um endereço da rede que é usado pelo cliente quando iniciar uma conexão.

O servidor não faz conexões iniciais ativamente, mas é preparado para aceitar *requests* para a conexão. Um outro agente que conhece o endereço pode tentar iniciar conexões, enviando uma requisição de conexão para o endereço. O servidor que está preparado pode aceitar o *request*, formando uma nova e única conexão com o cliente, ou pode rejeitar o *request*.

GERENCIAMENTO DA CONEXÃO

Dois papéis devem ser definidos para cliente e servidor, no que se refere à conexão:
- um cliente inicia a conexão, usando o endereçamento da informação encontrada na referência do objeto, para um objeto que pretende enviar *requests*;
- um servidor aceita conexões, mas não as inicializa.

As conexões não são simétricas, isto é, somente os clientes podem enviar as mensagens *request, locateRequest* e *cancelRequest* na conexão; somente um servidor pode enviar as mensagens *reply, locateReply* e *closeConnection* em uma conexão; e, cliente e servidor podem enviar as mensagens *messageError*. Somente mensagens GIOP são enviadas pelas conexões GIOP.

Os identificadores do *request* devem associar respostas sem ambigüidade com *requests* dentro do escopo e da existência de uma conexão. Eles podem ser reutilizados se não houver a possibilidade do *request* anterior usar o identificador que têm um *reply* pendente. É a responsabilidade do cliente gerar e determinar os identificadores do *request*. Os identificadores do *request* devem ser únicos entre as mensagens *request* e *locateRequest*.

ENCERRAMENTO DA CONEXÃO

As conexões podem ser encerradas de duas maneiras: através do desligamento disciplinado ou pela desconexão abortiva.

O desligamento disciplinado é iniciado pelos servidores resistentes, enviando uma mensagem **closeConnection**, ou pelos clientes (que pode ser iniciado em qualquer momento) apropriados para o encerramento da conexão. Se houverem *requests* pendentes quando um cliente desligar a conexão, o servidor deve considerar que todos os *requests* foram cancelados. E um servidor não pode iniciar um desligamento se houver *requests* em processamento, sendo que não recebeu uma mensagem *cancelRequest*, ou não foi enviado um *reply* correspondente.

Se um cliente detecta o encerramento da conexão recebendo uma mensagem **closeConnection**, ele deve assumir que ocorreu uma desconexão abortiva, e deve informar as exceções COMM_FAILURE para todos os *requests* pendentes na conexão.

MULTIPLEXANDO CONEXÕES

Um cliente pode enviar *requests* para múltiplos objetos alvo na mesma conexão, sendo o servidor secundário da conexão capaz de responder aos *requests* dos objetos. É a responsabilidade do cliente otimizar o recurso usado pelas conexões de reutilização, senão, o cliente pode iniciar uma nova conexão para cada objeto ativo suportado pelo servidor.

ORDENANDO A MENSAGEM

As conexões não são inteiramente simétricas. Somente os clientes podem enviar mensagens *request, locateRequest* e *cancelRequest*; podem ter múltiplos *requests* pendentes e não necessitam aguardar por um *reply* do *request* anterior, antes de enviar outro *request*.

Os servidores podem responder *requests* pendentes em qualquer ordem, pois as mensagens *reply* não são requeridas na mesma ordem que os *requests* correspondentes. Aqui também, somente os servidores emitem mensagens **closeConnection** quando as mensagens *reply* tiverem sido enviadas, em resposta a todas as mensagens *request* que requerem respostas.

1.4 Localização do Objeto

O GIOP é definido para suportar serviços de localização e migração do objeto, sem o preceito da existência de características ou arquiteturas ORB específicas. As características do protocolo são baseadas em um endereço de transporte, não necessariamente que corresponda a qualquer componente da arquitetura de um ORB específico. Somente implica na existência de

alguns agentes com o qual uma conexão pode ser aberta, e para o qual os *requests* podem ser enviados.

O "agente" (proprietário do servidor secundário de uma conexão) pode assumir um dos seguintes papéis, em relação a uma referência a objeto particular:

- Pode ser capaz de aceitar *requests* do objeto, diretamente do objeto e retorna respostas: ele pode ser uma ponte de Inter-ORB que transforma o pedido e o passa para outro processo ou ORB. Em perspectiva do GIOP, é somente importante que os pedidos possam ser enviados diretamente ao agente;
- Pode não ser capaz de aceitar pedidos diretos de qualquer objeto, mas aciona um serviço de localização: qualquer mensagem *request* envia para o agente o resultado das exceções ou respostas com o estado LOCATION_FORWARD, provendo novos endereços para os pedidos que podem ser enviados. Cada agente responde também para as mensagens *locateRequest* com as mensagens apropriadas *locateReply*;
- Pode responder diretamente a alguns pedidos e provê localizações para outros objetos;
- Pode responder diretamente aos pedidos de um objeto particular em um momento, e provê uma localização em um momento posterior.

Os agentes não são requeridos para a implementação de mecanismos de localização. Um agente pode ser implementado com a política de uma conexão que suporta o acesso direto para o objeto, ou retorna exceções. Cada ORB retorna mensagens *locateReply* com o estado OBJECT_HERE ou UNKNOWN_OBJECT, e nunca o estado OBJECT_FORWARD.

Os clientes devem ser capazes de aceitar e processar as mensagens *reply* com o estado LOCATE_FORWARD, desde que o ORB possa escolher a implementação de um serviço de localização. É durante a descrição do cliente que é definido o envio de mensagens *locateRequest*.

Um cliente não deve fazer qualquer suposição sobre a duração do objeto, que fora retornado pelos mecanismos da localização. Quando uma conexão baseada na informação da localização é encerrada, os esforços subseqüentes para enviar pedidos para o mesmo objeto devem iniciar com o endereço original, que é especificado na referência a objeto inicial.

Depois de realizar invocações sucessivamente usando um endereço, um cliente deve ser preparado para ser ativado. O cliente deve somente esperar o endereço do objeto, fornecido pela referência a objeto, para continuar trabalhando com segurança. E, se uma invocação usando este endereço retornar UNKNOWN_OBJECT, o objeto deve ser considerado como não-existente.

Em geral, a implementação dos mecanismos de localização é feita durante a descrição dos ORBs, a qual é disponível para otimização e suporte dos ambientes de localização e migração do objeto flexível.

9. Bibliografia

[BOS 97] BOSCO, P.G., GIUDICE, D.Lo, MARTINI, G. & MOISO, C., **"ACE: An environment for specifying, developing and generating TINA services"**, International Symposium on Integrated Network Management, San Diego, (CA) USA, May 1997, pg 515 a 526.

[BRI 93] BRISA, **"Gerenciamento de Redes - Uma Abordagem de Sistemas Abertos"**, São Paulo, Makron Books, 1993.

[BUS 96] BUSSMANN, Luís A.S., & PENNA, Manoel C., **"Utilização de Sistemas Distribuídos em Telecomunicações"**, Relatório de Pesquisa - parte do Projeto PLAGERE, CEFET - Paraná, 1996.

[CER 97] CEREDA, R.L.D. & CRUZ, M.A.C. & DUTRA, L.S.V. & SEWAYBRICKER, R.R., **"ATM: O futuro das redes"**, São Paulo : Makron Books : Brisa, 1997

[CHA 95] CHADHA, Ritu, & WUU, Sze-Ying, **"Incorporating Manageability into Distributed Software"**, IEEE Second International Workshop on Systems Management, Toronto - Canada, 19-21 June, 1996.

[CHA 97] CHADHA, R., & WUU, S., **"Incorporating Manageability into Distributed Software"**, International Symposium on Integrated Network Management, San Diego, (CA) USA, May 1997, pg 489 a 502.

[CHAV 97] CHAVES, R., **"Um Framework CORBA para gerenciamento de redes"**, Trabalho Parcial de Conclusão do Curso de Bacharelado em Ciência da Computação, UFSC, Florianópolis, Dezembro 1997.

[CHE 97] CHEN, Graham, & RIXON, Jeremy, & KONG, Qinzheng, **"Integrating CORBA and Java for ATM Connection Management"**, DSOM, Sidney, Austrália, outubro 1997.

[CHU 97] CHUNG, P.Emerald, & HUANG, Yennun, & YAJNIK, Shalini, & LIANG, Deron, & SHIH, Joanne C., & WANG, Chung-Yih, & WANG, Yi-Min, **"DCOM and CORBA Side by Side, Step by Step, and Layer by Layer"** - Paper, Bell Laboratories, Lucent Technologies, Institute of Information Science, AT&T Labs Research, October 1997.

[COR 95] OMG, "**The Common Object Request Broker: Architecture and Specification**", v. 2.0, Julho 1995.

[CMI 94] LEHMANN, Erny Otto Júnior. "**Especificação e Verificação do Protocolo CMIP para Gerenciamento de Redes**", Dissertação de Mestrado, COPPE, UFRJ, 1994.

[DIN 95] DINI, Petre, BOCHMANN, Gregor v., KOCH, Thomas & KRÄMER Bernd, "**Agent based Management of Distributed Systems with Variable Polling Frequency Policies**", IEEE Second International Workshop on Systems Management, Toronto - Canada, 19-21 June, 1996.

[EDW 95] EDWARDS N. "**CORBA - An Industrial Approach to Open Distributed Computing**", Draft for inclusion in Open Distributed Systems", July 28, 1995,

[FER 96] FERRASA, M., FOLTRAN, D.C., PENNA, M.C. "**Especificações de Gerenciamento Entre-Domínios: GDMO/CORBA**", Relatório de Pesquisa, Projeto PLAGERE, CEFET-Paraná, 1996.

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

[GEN 96] GENILLOUD, Guy, "**An Analysis of the OSI Systems Management Architecture from an ODP Perspective**", IEEE Second International Workshop on Systems Management, Toronto - Canada, 19-21 June, 1996.

[GHE 97] GHETIE, Iosif G., "**Networks and Systens Management - platforms analysis and evaluation**", Bell Comunications Research, New Jersey, Ed. Kluwer Academic Publishers, 1997.

[GRE 94] GREGOIRE, J.-Ch, "**Conceptual Models and Mechanisms for Distributed Management**", DSOM, october 1994.

[HAU 97] HAUW, L.H. & CANELA, Z. & VOYER, F., "**A CORBA-Based TMN Prototype with Web Access**", Alcatel Telecom RD, DSOM 97, Sidney, Australia, october 1997.

[HON 96] HONG, James W. & PARK, Jong-Tae. " **Supporting Distributed Management in a Telecommunications Management Network (TMN) Environment**", IEEE Second International Workshop on Systems Management, Toronto - Canada, 19-21 June, 1996.

[HOW 97] HOWARD, Stephen, LUTFIYYA, Hanan, KATCHABAW, Michael, & BAUER, Michael, **"Supporting Dynamic Policy Change Using CORBA System Management Facilities"**, International Symposium on Integrated Network Management, San Diego, (CA) USA, May 1997, pg 527 a 538.

[LAV 96] LAVINIKI, S.R. & PALMA, M.R. **"Implementação de Objetos Gerenciados Para Supervisão de Alarmes TMN Aplicados à Central EWSD"**, Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação, UFSC, Florianópolis, Dezembro 1996.

[LAVI 96] LAVINIKI, S.R., SCHÖNBERGER, S., WESTPHALL, C.B. **"Implementação de Objetos Gerenciados TMN na Plataforma OSIMIS"**, Anais em CD-ROM do Congresso INFTEL. PETROBRÁS. Rio de Janeiro (RJ), 2 a 6 de dezembro de 1996.

[LEW 97] LEWAY, Laura & PERKINS, Charles L. **"Aprenda em 21 dias Java"**, tradução de Fremem Assessoria de Comunicação, Editora Campus Ltda, Rio de Janeiro, 1997.

[LEN 95] LENTO, Luiz Otávio Botelho & MADEIRA, Edmundo R.M., **"Um Esquema para Acessar Objetos em Ambientes Distribuídos"**, Anais do 13° Simpósio Brasileiro de Redes de Computadores, Universidade Federal de Minas Gerais - MG, Abril de 1995.

[LOR 98] LORENSET, Vera Lúcia & SOBRAL, João Bosco Mangueira & WESTPHALL, Carlos Becker, **"CORBA based Telecommunication Network Management System: An experience on Alarm Surveillance"**, Artigo aceito a ser publicado no SCS/IEEE *Symposium on Performance Evaluation of Computer and Telecommunication Systems* (SPECTS'98), 19-22 de julho de 1998, Reno, Nevada.

[KON 96] KONG, Qinzheng, & CHEN, Graham, **"Integrating CORBA and TMN Environments"**, 0-7803-2518-4/96, IEEE, 1996.

[KOR 97] KORMANN, Luiz Fernando, **"Novas Funções de Gerência para Redes de Telecomunicações e de Alta Velocidade usando a Plataforma OSIMIS"**, Dissertação de Mestrado, CPGCC, UFSC, 1997.

[MAL 96] MALUCELLI, Vinicius Vendrami. **"Conceitos de Objetos Distribuídos"**, Mini-Curso Objetos Distribuídos'96 com Tecnologia Cliente/Servidor, Internet e Java. Curitiba - Paraná - Brasil, dezembro 1996.

[MAZ 97] MAZUMBAR, Subrata. **"Inter-Domain Management: CORBA, OSI, SNMP"**, Tutorial 5, Bell Laboratories, NJ, USA, Maio 1997.

[MEN 94] MENDES, M.J. & MADEIRA, E.R.M., **"Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware"**, Anais do 12° Simpósio Brasileiro de Redes de Computadores, Curitiba - PR, 16 a 20 Maio 1994.

[M3100] Recommendation M.3100 - **Generic Network Information Model**, ITU-T, 1992.

[M3010] Draft M.3010 - **Concepts of TMN**, version GEN/950331/a. ITU-T, march 1995.

[M3200] Recommendation M.3200 - **Maintenance: Telecommunications Management Network, TMN Management Services: Overview**, ITU-T, 1992.

[M3400] Recommendation M.3400 - **Maintenance: Telecommunications Management Network, TMN Management Functions**, ITU-T, 1992.

[ODP 94] ITU-TS - **Draft ODP Trading Function**, ITU-TS SG7.Q16 Draft Recommendation, July 1994.

[PAR 96] PARK, Jong. & HA, Su-Ho. & HONG, James W. & SONG, Joong-Goo. **"Design and Implementation of a CORBA - Based TMN SMK System"**, Anais NOMS, 1996.

[QUE 97] QUEIROZ, João Augusto G. & MADEIRA, Edmundo R.M. **Gerenciamento de Objetos CORBA para a plataforma Multiware**, Anais do 15° Simpósio Brasileiro de Redes de Computadores, Universidade Federal de São Carlos - SP, 19 a 22 maio 1997, pg 432 a 444.

[Q821] Recommendation Q.821 - **Stage 2 and 3 Description for the Q3 Interface - Alarm Surveillance,** ITU-T, 1993.

[Q822] Recommendation Q.822 - **Stage 1, Stage 2 and Stage 3 Description for the Q3 Interface: Performance Management**, ITU-T, Março 1993.

[ROD 97] RODRIGUEZ, Noemi & MOURA, Ana Lúcia de & IERUSALIMSCHY, Roberto & CERQUEIRA, Renato. **"Aplicações de Gerência com Comportamento Dinâmico"**, Anais do Segundo Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Fortaleza - Ceará - Brasil, novembro 1997.

[TAN 94] TANENBAUM, Andrew S. **"Redes de Computadores"**, tradução de PubliCare Serviços de Informática, Ed. Campus Ltda, Rio de Janeiro, 1994.

[TEI 96] TEIXEIRA, José Helvécio Júnior & SAUVÉ, Jácques Philippe & MOURA, José Antão Beltrão. **"Do Mainframe para a Computação Distribuída - Simplificando a Transição"**, Livraria e Editora Infobook S.A., Rio de Janeiro, 1996.

[TEL 96] Telecom Special Interest Group - Object Management Group, **"CORBA-Based Telecommunication Network Management System"**, OMG Doc., Draft 3.0, Framingham Corporate Center, 1996.

[UFR 96] **"Repositório de Documentos Técnicos do Grupo de Redes - UFRGS"**, Gerência de Redes, CMISE e CMIP, CMIP X SNMP, UFRGS 1996.

[VAL 96] VALLE, André & GUIMARÃES, Claudia. **"Java - Manual de Introdução"**, Axcel Books do Brasil Editora, Rio de Janeiro, 1996.

[VIN 97] VINOSKI, Steve. **"CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments"**, IEEE Communications Magazine, IONA Technologies, vol 35, nº 2, Fevereiro 1997.

[X721] Recommendation X.721 - **Data Communication Networks - Information Technology - Open Systems Interconnection - Structure of Management - Information: Definition of Management Information**, ITU-T, 1992.

[X722] Recommendation X.722 - **Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects (GDMO)**. International Organization for Standardization and International Electrotechnical Committee, June 1993.

[ZUQ 96] ZUQUELLO, Nuccio M. S., & MADEIRA, Edmundo R.M., **"Obtendo Interoperabilidade entre ORBs"**, Anais do 14° Simpósio Brasileiro de Redes de Computadores, Fortaleza - Ce, 20 a 23 Maio 1996.