UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES

AVALIAÇÃO DE DESEMPENHO, PORTABILIDADE E INTEROPERABILIDADE DE SISTEMAS DE GERÊNCIA DE REDES DE TELECOMUNICAÇÕES BASEADOS EM PLATAFORMAS TMN COMERCIAIS

por: Flavia Azevedo Schneider

Orientador: Prof. Dr. Lee Luan Ling

Banca:

Dr. Rege Romeu Scarabucci

Prof. Dr. Maurício Ferreira Magalhães

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP como requisito parcial para a obtenção do Título de **Mestre** em Engenharia Elétrica.

1999

Este exemplar corresponde a redação final da tese defendida por Flavia Azevedo Schneider e aprovada pela Comissão Julgada em 15 / 07 / 99
Orientador

UNICAMP
BIBLIOTECA CENTRAL

BAE 20 000497 87

UNIDADE BC
N.º CHAMADA :
T/UNICAMP
Sch57a
V. Ex.
TOMBO BC/ 39926
PROC. 278/00
C [ ] D [X]
PREÇO R$ 11,00
DATA 12/04/00
N.º CPD

CM-00137779-3

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Schneider, Flavia Azevedo
Sch57a Avaliação de desempenho, portabilidade e interoperabilidade de sistemas de gerência de redes de telecomunicações baseados em plataformas TMN comerciais. / Flavia Azevedo Schneider.--Campinas, SP: [s.n.], 1999.

Orientador: Lee Luan Ling
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas de telecomunicação. 2. Redes de computação - Gerência. I. Lee, Luan Ling. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

À minha mãe

Maria Santana

À minha avó

Honorina

E ao Edmilson

AGRADECIMENTOS

Gostaria de fazer um agradecimento geral aos colegas do CPqD e da Unicamp que colaboraram de forma direta ou indireta para a realização desse trabalho. Entretanto, não poderia deixar de ressaltar, de modo especial o meu agradecimento:

Na Unicamp

Lee Luan Ling pelo trabalho de orientação.

No CPqD Telebrás:

- Luis Benedito Fayan não só pela coordenação do projeto, mas também pelo incentivo, apoio, companherismo e conhecimentos transmitidos.
- Milton, Lorena, Ana Rita, Armando e Renata pelo companherismo, discussões e trabalho em grupo.

Em especial a Edmilson pelo carinho, paciência e apoio, e a professora e amiga Eliana pelo apoio e incentivo.

RESUMO

A investigação dos aspectos de desempenho de sistemas de gerência de redes de telecomunicações baseados em plataformas TMN torna-se fundamental para as empresas operadoras de telecomunicações e para os fornecedores de sistemas de gerência. Uma vez que o número de sistemas especificados/desenvolvidos baseados na arquitetura TMN cresce de forma muito acentuada e que não existem parâmetros quantitativos para o estabelecimento de arquiteturas físicas otimizadas para tais sistemas.

Esta dissertação objetiva a investigação de fatores que influenciam no desempenho, portabilidade e interoperabilidade de sistemas de gerência TMN. Apresenta-se primeiramente a estrutura básica de uma plataforma TMN genérica, com o objetivo de identificar os componentes que mais podem afetar o desempenho de um sistema de gerência. São descritos então os testes que foram especificados/desenvolvidos para quantificar alguns parâmetros de desempenho de tais plataformas e apresenta-se os resultados obtidos deste trabalho. Estes resultados, muito embora não caracterizem todos os fatores a serem analisados, já podem ser muito úteis para a especificação de arquiteturas de sistemas de gerência, auxiliando na definição da distribuição física de dispositivos mediadores, sistemas de adaptação e etc.

Este trabalho tem apoio do Departamento de Sistemas de Operação do CPqD da Telebrás.

ABSTRACT

This work presents an analysis of the performance, portability and interoperability aspects in the development of telecommunications management applications running in commercial TMN platforms. This work was done at CPqD TELEBRÁS, which has a TMN laboratory that studies and analysis the implementation aspects of TMN architecture applied to telecommunications enterprises.

Firstly, it is presented the basic architecture of a generic TMN platform and it is identified its components which could be responsible for the main performance, portability and interoperability aspects. After that, it is presented the tests that were specified/developed in order to obtain figures regarding performance. This figures address mainly the number of transactions per second that a system can perform depending of some factors as the number of agents running in the system, etc. Finally, the results are presented, and nevertheless it is necessary to get more data about the subject, it is already possible to use these results to aid in the specification of telecommunication management systems.

Índice

Índice de Figuras

Índice de Tabelas

Índice dos Gráficos

Abreviações

ACSE	Association Control Service Element
ANSI	American National Standards Institute
API	Application Program Interface
ASN.1	Abstract Syntax Notaion.1
CLNP	ConnectionLess Network Protocol
CLNS	ConnectionLess Network Service
CCITT	Consultative Commitee for International Telegraph Telephone
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
CMOT	CMIP over TCP/IP
CONS	Connection Oriented Network Service
DCE	Distributed Computing Environment
DCF	Data Communication Function
DCN	Data Communication Netowrk
DN	Distinguished Name
FTAM	File Transfer, Access Management
GDMO	Guidelines for the Definition of Managed Objects
GIRS	Gerência Integrada de Redes e Serviços
HMA	Human Machine Adaptation
ICF	Information Conversion Function
IP	Internet Protocol

ISO	International Organization for Standardization
ITU	InternationalTelecommunications Union
ITU-TS	InternationalTelecommunications Union Telecommunications Standardization Sector (antigo CCITT)
MD	Mediation Device
MF	Mediation Function
MIB	Management Information Base
NE	Network Element
NEF	Network Element Function
OAM&P	Operação, Administração, Manutenção e Provisionamento
OS	Operation System
OSF	Operations System Function
OSI	Open Systems Interconnection
PF	Presentation Function
PDU	Protocol Data Unit
QA	Q Adaptor
QAF	Q Adaptor Function
RFC	Requests for Comments
ROSE	Remote Operations Service Element
SDH	Synchronous Digital Hierarchy
SMK	Shared Management Knowledge
TCP/IP	Trasmission Control Protocol/Internet Protocol
TMN	Telecomunications Management Network
TP	Transport Protocol

TSAP	Transport Service Acess Point
UDP	User Datagram Protocol
WAN	Wide Area Network
WS	Workstation
WSF	Workstation Function

CAPÍTULO 1

1.1 INTRODUÇÃO

A contínua evolução dos sistemas de telecomunicações deu origem a um ambiente de rede bastante heterogêneo, seja pelas diferenças tecnológicas, seja pela variedade de fornecedores de equipamentos e produtos. Entretanto, a competição, a evolução das redes, o crescimento da demanda por novos e melhores serviços, bem como a evolução tecnológica, obriga as empresas de telecomunicações a reduzirem seus custos, melhorarem a qualidade dos serviços e agilizarem o atendimento às necessidades dos clientes. Assim, a competição, o crescimento das redes, em número e tecnologia, a busca pela melhoria da qualidade dos serviços estão levando as empresas de telecomunicações a fortes investimentos na Gerência Integrada de Redes e Serviços (GIRS) .

Todos os sistemas de voz, dados e meios de transmissão utilizam sistemas comercialmente vendidos no mercado, para agregar novos e melhores serviços aos clientes. Assim, provocando quantidade e diversidade de fornecedores, equipamentos e produtos. Esses equipamentos e produtos são, normalmente, gerenciados por sistemas proprietários[1] e incompatíveis entre si, resultando em:

- **Multiplicidade de sistemas-** para cada novo tipo de equipamento/fabricante é necessário um novo sistema de supervisão específico;
- **Multiplicidade de terminais e formas de operação** - cada sistema tem seus próprios terminais e própria linguagem de comunicação homem-máquina;
- **Multiplicidade de base de dados-** cada sistema emprega sua própria base de dados local.

---

[1] Sistema Proprietário é um sistema não padronizado por normas de Organizações, sendo desenvolvido por fabricantes possuindo geralmente uma arquitetura fechada que impede a interoperabilidade com outros sistemas semelhantes.

Dessa forma os custos operacionais da empresas de Telecomunicações são elevados e a agilidade da gerência de redes e serviços de telecomunicações é comprometida uma vez que as soluções de gerência adotadas não têm capacidade de atuar de forma integrada e eficiente sobre tal ambiente. A sofisticação e a qualidade atualmente exigidas dos serviços de telecomunicações requerem uma gerência integrada e eficiente não sendo compatíveis com este ambiente heterogêneo. Assim, um grande esforço de padronização vem sendo feito nos últimos anos por várias organizações internacionais na área de gerência de redes de telecomunicações com o intuito de estabelecer e padronizar arquiteturas e tecnologias para esse ambiente. A base destes esforços foi o estabelecimento da Arquitetura TMN por parte do ITU-T (na época ainda chamado CCITT) em 1985, em que foi iniciada a definição de um sistema básico de gerência para a indústria de telecomunicações. Um conjunto de informações, especificado como OAM&P (*Operations, Administration, Maintenance and* Provisioning) foi definido, e o termo TMN (*Telecommunications Management Network*) foi atribuído para designar a arquitetura de gerência para a indústria de telecomunicações. O resultado foi uma série de recomendações (série M^2) do ITU-T contendo regras gerais a serem seguidas com o objetivo de se obter um ambiente multifornecedor entre sistemas de gerência e equipamentos de telecomunicações. A base destes princípios foi a definição de interfaces abertas, orientadas a objeto, com protocolos e modelos de informação padronizados, que devem ser implementadas tanto por fabricantes de equipamentos de telecomunicações como por fabricantes de sistemas de gerência.

Apesar destas definições TMN já terem sido feitas há algum tempo, somente agora é que o volume de trabalho na área de padronização tem tomado força de mercado, o que está permitindo que um número cada vez maior de sistemas de gerência de redes de telecomunicações sejam especificados com base nas regras/recomendações estabelecidas por essas organizações internacionais.

---

[2] Recomendações série M3000, série X.700.

Este crescimento do mercado de sistemas TMN levou ao surgimento (ou a aceleração das especificações) de uma série de novas características tecnológicas na área de desenvolvimento de software, tais como APIs (*Application Program Interface*) padronizadas para protocolos de gerência de rede, aplicações especialistas voltadas para armazenamento, correlação e filtragem de eventos de rede, mecanismos de distribuição, armazenamento de objetos entre outras funções.

Assim, a competição, o crescimento das redes, em número e tecnologia, a busca pela melhoria da qualidade dos serviços, a redução dos custos levaram ao crescimento do mercado de sistemas TMN. E esse contexto ocasionou o desenvolvimento de uma série de soluções tecnológicas, que aglutinadas em produtos comerciais são chamadas genericamente de **Plataformas TMN** [Plat93]. O objetivo desses produtos é oferecer uma série de serviços básicos e ferramentas para o desenvolvimento de aplicações de gerência TMN, tornando mais rápido e mais barato a obtenção de software para sistemas de gerência de redes de telecomunicações. Com isso, atualmente quase que a totalidade de sistemas TMN são desenvolvidos em Plataformas comerciais TMN, isto é, as aplicações de gerência se utilizam de uma série de serviços básicos providos por um *middleware*[3] especializado para telecomunicações.

1.1.2 Contribuição

Este trabalho contribui principalmente com a seleção de parâmetros que são importantes na avaliação de desempenho de sistemas TMN, em conjunto com a elaboração de uma metodologia de testes para verificar como estes parâmetros afetam o desempenho das plataformas TMN. Também faz parte deste trabalho avaliar a portabilidade de aplicações de gerência de redes de Telecomunicações em diferentes plataformas TMN comerciais, assim como verificar a interoperabilidade de aplicações de gerência em diferentes plataformas TMN comerciais.

---

[3] *Middleware* - é uma camada de software entre as aplicações e os dispositivos físicos, sistema operacional. Sua função é padronizar e simplificar o acesso aos dispositivos físicos.

1.1.3 Objetivo

O objetivo deste trabalho é especificar, descrever e comparar características de desempenho, portabilidade e interoperabilidade de sistemas TMN que são relevantes na especificação de sistemas TMN. Para explicar a necessidade e os conceitos técnicos que levaram ao desenvolvimento deste trabalho é necessário discutir os fundamentos sistêmicos de TMN, Gerência Integrada de Redes e Serviços (GIRS), protocolos utilizados, linguagens do modelo de informação. Portanto, também é o objetivo deste trabalho a discussão dos itens acima citados e a caracterização da plataforma TMN através da identificação dos componentes mínimos para formar uma plataforma TMN.

1.1.4 O Contexto do Desenvolvimento do Trabalho

Os sistemas de gerência que atuam diretamente sobre a rede de telecomunicações são sistemas de missão crítica, isto é, necessitam atuar em condições rigorosas de disponibilidade, robustez e desempenho, assim torna-se fundamental a investigação de fatores que são determinantes no desempenho de tais sistemas. Deve-se também verificar alguns parâmetros de desempenho quantitativamente, de forma a auxiliar na especificação de futuros sistemas TMN.

Foi neste cenário que o Laboratório TMN (LabTMN) do Departamento de Sistemas de Operações do CPqD da Telebrás iniciou os trabalhos de caracterizar, avaliar e quantificar os parâmetros de desempenho relacionados a uma Plataforma TMN genérica. O LABTMN está especialmente voltado para o estudo e análise dos aspectos de implementação da arquitetura TMN nas empresas operadoras de telecomunicações do Sistema Telebrás.

Deve-se ressaltar que o objetivo deste trabalho não é comparar características de desempenho de diferentes fornecedores de plataformas, mas estabelecer alguns critérios e parâmetros que possam ser utilizados pelas empresas operadoras de telecomunicações e pelos fornecedores de sistemas de gerência como patamares no dimensionamento de futuros sistemas TMN.

1.2 Gerência Integrada de Redes e Serviços (GIRS)

Nesta seção será discutido as mudanças que ocorreram no ambiente das empresas operadoras de Telecomunicações e no ambiente dos clientes das empresas operadoras nas ultimas três décadas para entendermos o surgimento da Gerência Integrada de Redes e Serviços - GIRS. Será então discutido quais os objetivos das GIRS e quais são os requisitos necessários para atingir os objetivos das GIRS.

Nas últimas três décadas ocorreram grandes mudanças no ambiente das empresas operadoras, tanto do ponto de vista tecnológico quanto do surgimento de novos serviços. A Tabela 1-1 mostra a evolução do ambiente de Telecomunicações nas últimas 3 décadas, tanto para as empresas operadoras quanto para os clientes das empresas operadoras.

Década de 70	
Empresas Operadoras de Telecomunicações	**Clientes**
⇒ Utilização de equipamentos predominantemente analógicos.	⇒ Voltado essencialmente para serviços de voz.
⇒ A supervisão da rede era realizada exclusivamente por operadores.	⇒ Aspiração ao serviço.
⇒ Caráter fortemente expansionista.	⇒ Qualidade em segundo plano.
⇒ Processos manuais de operação e controle de rede.	⇒ Sem estrutura própria de telecomunicações.
Década de 80	
⇒ Surgimento de equipamentos digitais (inicialmente em transmição e posteriormente em comutação).	⇒ Além do serviço de voz, surgimento de serviços de dados.
⇒ Surgimento de forma incipiente, dos primeiros projetos de sistemas de gerência automáticos.	⇒ Após obtenção do serviço, os clientes começam a exigir a qualidade do serviço.
⇒ Ocorrência de um forte controle de investimentos, diminuindo o ritmo de expansão.	⇒ Os grandes clientes começam a formar estrutura própria para gerenciar seus serviços e sistemas de telecomunicações.
⇒ Primeiros esforços de automação de tarefas operacionais e administrativas.	
⇒ Ampliações da planta executadas com recursos próprios (projeto, compra, venda e	

instalação).	
Década de 90	
⇒ Fim da contratação de equipamentos analógicos.	⇒ Diversificação de serviços: voz, dados, imagem, móvel celular, rede inteligente.
⇒ Proliferação de sistemas proprietários de gerência de sistemas.	⇒ Grande importância à agilidade e qualidade na prestação de serviços: Qualidade/Produtividade.
⇒ Planta instalada multifornecedor, implicando em equipamentos de fabricantes diferentes e com funcionalidades, recursos e interfaces diferentes.	⇒ Criação de serviços e redes próprias de telecomunicações.
⇒ Ampliação da planta executada em parceria com iniciativa privada.	⇒ Expectativa de desregulamentação, privatização e liberação no setor de telecomunicações; abre-se um novo nicho para atuação da iniciativa privada.
⇒ Terceirização de algumas atividades.	
⇒ Surgimento da Gerência Integrada de Redes e Serviços (GIRS).	

Tabela 1-1- Evolução do ambiente de Telecomunicações

Conforme mostrado na Tabela 1-, na década de 80 começaram a surgir os primeiros sistemas de supervisão e gerência de equipamentos e sistemas de suporte à administração dos serviços. Os primeiros sistemas, usualmente utilizados em PCs ou *workstations* apresentam arquitetura proprietária e interfaces não padronizadas com os equipamentos que supervisionam e gerenciam. Assim, para cada equipamento/fabricante é necessário um sistema de supervisão específico, cada sistema tendo seus próprios terminais e linguagens de comunicação, e consequentemente necessitando de uma base de dados local. Para obtermos uma gerência "integrada" precisaríamos colocar numa mesma sala todos os equipamentos com seus sistemas de gerência e um técnico especializado para cada um desses equipamentos/fabricantes para fazer a operação do sistema de gerência de cada equipamento/fabricante.

No caso de sistemas de suporte à administração dos serviços, normalmente utilizados em *mainframes*, não existe comunicação com os equipamentos utilizados na prestação de serviços e, às vezes, nem com outros sistemas envolvidos no suporte ao mesmo processo.

Para exemplificar, pode ser citado o caso dos centros de serviços de telecomunicações onde não era possível a integração de sistemas envolvidos no processo de registro de reclamações com sistemas de testes de circuitos, uma vez que os sistemas que atendiam ao centro de serviço não se conectavam entre si com sistemas de supervisão e gerência de redes ou com outros equipamentos. Pôde ser observado que com esta falta de integração não tinha-se uma visão global do estado da rede e dos serviços, o que facilitaria a identificação da causa da falha e difusão das informações sobre o estado dos circuitos e serviços. Esta deficiência acarretava muitas vezes na identificação e correção de alguns problemas/falhas só quando ocorria reclamação por parte do cliente.

Ao falarmos de gerência integrada estamos nos referindo a vários tipos de integração: integração entre aplicações, com respeito a apresentação das informações para os operadores, integração da base de dados, etc.

A integração entre aplicações pode ser analisada sob dois aspectos:

1. Integração entre sistemas de gerência das várias camadas do modelo OSI de referência, melhor explicado no Capítulo 3. É esta integração que possibilita, por exemplo, que o sistema que suporta o atendimento de reclamações obtenha dados de um circuito reclamado com informações dos alarmes nos equipamentos por onde passa esse circuito.

2. Integração entre entidades de uma mesma camada. É com esta integração que alarmes e dados, oriundos de vários equipamentos e sub-redes, são consolidados e correlacionados em uma visão de rede, de circuito fim-a-fim.

Assim, surgiu a idéia das GIRS, que foi definida como: "Conjunto de ações realizadas visando obter a máxima produtividade da planta e dos recursos disponíveis, integrando de forma organizada as funções de Operação, Administração, Manutenção e Provisão (OAM&P) para todos os elementos, redes e serviços de telecomunicações."

Com as GIRS objetiva-se: aumento da qualidade do serviço prestado pela mais rápida provisão e recuperação, redução dos custos operacionais pela racionalização das atividades de manutenção e redução dos custos dos sistemas de gerência. Mas para que os objetivos acima sejam atingidos são necessários os seguintes requisitos: processos operacionais com fluxo contínuo, facilidades de reconfiguração em tempo real agilizando a

manutenção; detecção da causa raiz das falhas; terminal de operação universal com apresentação padrão; eliminação da multiplicidade de sistemas de supervisão; dados de configuração confiáveis.

Para obter os requisitos acima é necessário que:

- Os terminais de operação estejam ligados em uma rede de comunicação de dados e não a um sistema de operações.
- Os sistemas/aplicações possam trocar informações e ativar processos entre si de forma padronizada.
- Os sistemas possam utilizar base de dados comum, evitando duplicação desnecessária de dados.
- Os sistemas possam utilizar dados comuns recebidos dos Elementos de Redes (*Network Element* - NE) atuando nos níveis de rede, serviços e negócios.
- Os sistemas realizem a correlação de dados para descobrir a causa raiz das falhas e degradação de desempenho.

É importante ressaltar que Gerência **Integrada** não é Gerência **Centralizada**. A flexibilidade obtida com a tecnologia da informação proporciona a possibilidade de disponibilidade das informações de gerência junto aos pontos de atuação, aumentando a velocidade com que os problemas são resolvidos, independente da rede ou da localização geográfica.

A adoção de padrões TMN, descrito mais detalhadamente no próximo capítulo, em conjunto com os demais padrões industriais, formam hoje uma base sólida para implantação das GIRS.

1.3. Organização do Trabalho

Este trabalho foi organizado da seguinte forma:

- Capítulo 1: apresenta a contribuição, o objetivo, o contexto do desenvolvimento deste trabalho e introduz Gerência Integrada de Redes e Serviços (GIRS).
- Capítulo2: aborda os conceitos teóricos do Modelo TMN: sua arquitetura funcional, arquitetura física e arquitetura de informação.
- Capítulo 3: trata dos protocolos utilizados na TMN.
- Capítulo 4: trata do modelo de informação de gerência e das linguagens utilizadas no modelo de informação (GDMO e ASN.1)
- Capítulo 5: é feita a caracterização da Plataforma TMN.
- Capítulo 6: trata do desempenho, portabilidade e interoperabilidade entre plataformas TMN.
- Capítulo 7: apresenta a descrição dos testes com os resultados e conclusões deste trabalho.

CAPÍTULO 2

Arquitetura TMN

2.1. INTRODUÇÃO

Este capítulo apresenta o modelo TMN. Sendo assim, serão abordados as 5 áreas funcionais de gerência e as três arquiteturas básicas que compõem a TMN. Dentro de cada arquitetura são descritas as funções, os blocos e componentes que pertencem a cada uma.

Como foi visto no capítulo anterior, a implementação de GIRS não é trivial devido a vários fatores, tais como heterogeneidade e complexidade da Rede de Telecomunicações. Entre os requisitos, citados no capítulo anterior, para implementar a GIRS destacaram-se:

- *Deve permitir a gerência de redes, serviços e equipamentos heterogêneos.*
- *Deve possibilitar mudanças tecnológicas e funcionais.*
- *Deve permitir o interfuncionamento entre redes gerenciadas separadamente.*
- *Deve prover integridade e segurança no suporte à gerência.*
- *Deve permitir aos clientes, provedores de serviço e outras administrações o acesso às informações de gerência de maneira segura e controlada.*

O ITU-T atento ao problema publicou as recomendações série M.3000 que conceitua TMN da seguinte forma:

> *Telecommunication Management Network (TMN) - Conjunto de padronizações abrangendo arquitetura, protocolos e interfaces, que tem como objetivo garantir a interoperabilidade de todos os elementos de redes, dando suporte às ações de gerência descrita acima.*

Estas recomendações indicam que TMN e Gerência de Sistemas OSI são aplicáveis como solução para suportar GIRS. Sendo assim, a próxima seção é dedicada a apresentação do modelo TMN.

2.2. MODELO TMN

A TMN oferece a possibilidade de alcançar uma série de objetivos gerenciais:

- Minimizar o tempo de reação a eventos da rede.
- Minimizar a carga causada pelo tráfego de informações de gerência.
- Prover mecanismos de isolação para minimizar riscos de segurança e para localizar e conter falhas de redes.
- Melhorar o serviço de assistência e interação com os clientes.

O conceito básico de TMN é prover uma estrutura organizada para interconectar vários tipos de sistemas operacionais (OS - *Operations System*) a equipamentos de telecomunicações para a troca de informação de gerência utilizando interfaces padronizadas que incluem a definição de protocolos e mensagens. Assim a TMN pode gerenciar:

- Redes Públicas e Privadas, incluindo redes de telefonia móvel, redes virtuais, de longas distâncias, metropolitanas e locais.
- A própria TMN.
- Terminais de transmissão como multiplexadores, roteadores e SDH (*Synchronous Digital Hierarchy*).
- Sistemas de transmissão digital e analógica baseados em cabo coaxial, par trançado, fibra óptica, rádio e satélite.
- *Mainframes*, processadores *front-end*, controladores de cluster e servidores de arquivo.
- Serviços de suporte e teleserviços.
- PABX, acessos de PABX e terminais de usuários.

- *Softwares* providos por ou associados a serviços de telecomunicações.
- Sistemas de suporte e infra-estrutura, tais como módulos de teste, sistemas de energia, unidades de ar-condicionado e sistemas de alarme de edifício.
- Entidades distribuídas e serviços oferecidos pelo agrupamento de itens anteriores.

Conceitualmente a TMN é uma rede sobreposta que realiza a interface com uma rede de telecomunicações em vários pontos, com a finalidade de enviar e receber informações para controle de sua operação. Podemos dizer que a TMN é uma rede sobreposta porque ela se sobrepõe à rede de telecomunicações com o objetivo de gerenciá-la, com a Figura 2-1 é possível visualizar bem o relacionamento entre a rede de Telecomunicações e a TMN. A rede de telecomunicações é composta por vários tipos de equipamentos de telecomunicações analógicos e digitais, tais como sistemas de transmissão, centrais, terminais de sinalização, *mainframes*, etc.

A Figura 2-1 é um exemplo genérico de uma Rede de Telecomunicações gerenciada. Os sistemas de Telecomunicações (Sistemas de Comutação e Transmissão) são gerenciados através da DCN (*Data Communication Network*). A DCN é uma rede de dados utilizada para transportar as informações de gerência através dos protocolos de gerência. A DCN utiliza parte da própria rede de Telecomunicações para transportar as informações de gerência, como por exemplo a rede de gerência do SDH que utiliza os *bytes* D1 à D12 do cabeçalho SDH para transportar informações de gerência. A DCN também possibilita a conexão entre várias TMN. Os OSs (*Operation System*) representam as aplicações de gerência.

A TMN pode variar em complexidade de uma simples conexão entre um OS e uma parte de um equipamento de telecomunicações a toda rede de telecomunicações com vários Sistemas de Suporte a Operação (OS). Em geral, uma rede de telecomunicações consiste de diversos tipos de equipamentos analógicos e digitais de telecomunicações, tais como, centrais telefônicas, equipamentos de transmissão, *mainframes*, etc. Para a TMN estes equipamentos ou um conjunto deles são chamados de elementos de rede (NE – *Network Element*).

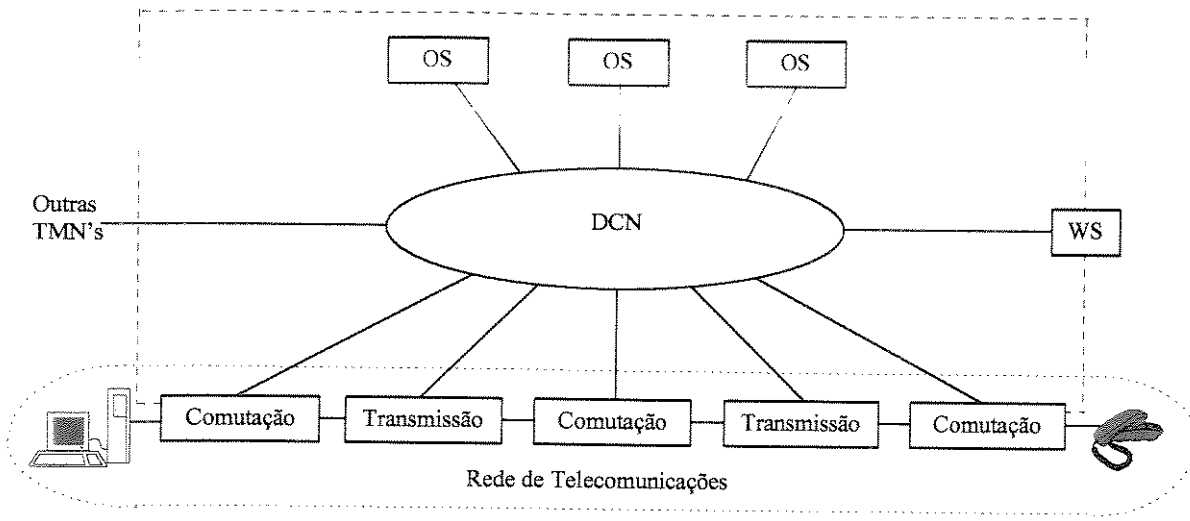

Figura 2-1 - Relações Gerais de uma TMN numa rede de Telecomunicações

Para controlar os diversos e heterogêneos elementos de rede, a TMN é dividida em várias áreas funcionais de gerência. As áreas funcionais de gerência devem atuar em todas as fases: no planejamento, instalação, operação, administração e provisionamento da rede de Telecomunicações de uma empresa operadora. As áreas funcionais de gerência necessárias para todas as fases são:

- Gerência de Desempenho
- Gerência de Falhas
- Gerência de Configuração
- Gerência de Contabilização
- Gerência de Segurança

Estas áreas funcionais constituem processos de aplicação de gerência que utilizam serviços oferecidos pela camada de aplicação do Modelo OSI. A Figura 2-2 abaixo mostra o relacionamento das 5 áreas funcionais no contexto do modelo OSI. O CMISE (*Common Management Information Service Element*) é onde ocorre a troca de informações de gerência, dentro do sistema de gerência OSI, entre agentes e gerentes por meio de um protocolo. O CMISE é especificado em 2 partes: a interface com o usuário e o protocolo. No próximo capítulo, seção 3.1.3, o CMISE será melhor explicado.

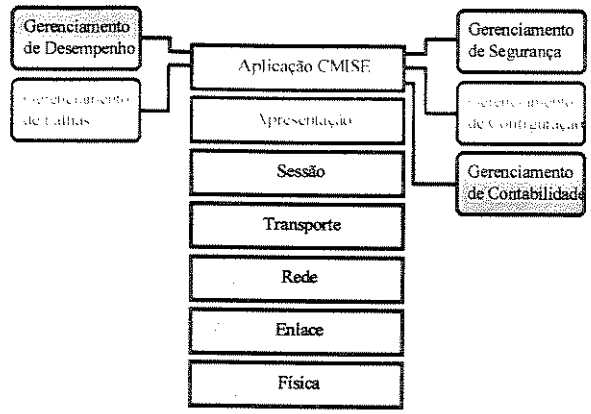

Figura 2-2 - Áreas Funcionais de Gerência

A seguir vamos descrever os 5 tipos de áreas funcionais existentes.

A **Gerência de Falhas** é o conjunto de funções que abrange o tratamento de condições anormais de funcionamento do sistema. Sendo assim, a gerência de falhas atua na:

- Detecção da ocorrência de falha.
- Isolamento da causa da falha.
- E na correção da falha, se possível.

A **Gerência de Configuração** é o conjunto de funções que exerce controle sobre a expansão ou redução de um sistema, o estado das partes que o constituem e a identificação de sua localização. Sendo assim, a gerência de configuração consiste em:

- Obter informações sobre a configuração corrente do sistema.
- Usar estes dados para modificar a configuração do sistema dentro dos dispositivos do sistema.
- Guardar dados e manter atualizado o inventário.

A **Gerência de Desempenho** oferece funções que através da coleta de dados estatísticos permitem controlar, monitorar, relatar e corrigir o comportamento e eficácia da rede. É de vital importância para as atividades de planejamento e controle de qualidade dos serviços de rede. Sendo assim, a gerência de desempenho consiste em:

- Coletar dados.

- Analisar estes dados, para visualizar tendências
- Situar limites de utilização.

A gerência **de Contabilização** é importante na medição dos custos e do volume de recursos utilizados pelos usuários. Sendo assim a gerência de contabilização consiste em:

- Obter dados sobre a utilização dos recursos e serviços do sistema.
- Associar o uso de recursos com escalas de tarifação, combinando custos.
- Tarifar os usuários pelo uso.

A **Gerência de Segurança** trata do uso da gerência de redes para controlar os mecanismos de segurança, abrangendo desde controle de acesso aos sistemas até o controle de informações sigilosas que trafegam nos circuitos de dados. Sendo assim, a gerência de segurança consiste em:

- Identificar as informações "delicadas" que devem ser protegidas.
- Dar segurança aos pontos de acesso.
- Manter esta segurança.

A arquitetura TMN visa a orquestração dos sistemas individuais de gerência de telecomunicações a fim de ter um efeito coordenador sobre a rede com os seguintes requisitos:

- Possibilitar várias estratégias de implementação e graus de distribuição das funções de gerência.
- Considerar a gerência de redes, equipamentos e serviços heterogêneos.
- Levar em conta futuras mudanças tecnológicas e funcionais.
- Incluir capacidade de migração para agilizar a implementação e permitir refinamentos futuros.
- permitir aos clientes, aos provedores de serviços de valor adicional e as outras administrações o acesso a informações e funções de gerência.

- Endereçar tanto um pequeno quanto um grande número de recursos gerenciáveis.
- Possibilitar o interfuncionamento entre redes gerenciadas separadamente, de modo que serviços inter-redes possam ser providos entre operadoras.
- Prover o gerência de redes híbridas baseadas em tipos de equipamentos diversos.
- Proporcionar flexibilidade na escolha do grau de confiabilidade/custo desejado para todos os componentes de gerência de rede e para a rede como um todo.

Assim, a TMN forma a arquitetura básica que assegura a interconexão de vários sistemas de operação (OSs) e os elementos de rede (NEs) por meio de interfaces padronizadas.

A arquitetura geral da TMN está estruturada em 3 arquiteturas básicas que podem ser consideradas separadamente no planejamento e projeto de uma TMN: arquitetura funcional, arquitetura de informação e arquitetura física. Nas próximas seções iremos falar destas arquiteturas.

2.2.1 Arquitetura Funcional

A arquitetura funcional descreve as funções de gerência agrupadas em blocos funcionais através dos quais uma TMN pode ser implementada independentemente da sua complexidade. A definição destes blocos funcionais e dos pontos de referência entre os blocos leva à especificação de interfaces padrão TMN. A arquitetura funcional é dividida em 5 blocos funcionais, e por sua vez os blocos funcionais são constituídos por componentes funcionais. Os blocos funcionais provêem as funções gerais que capacitam uma TMN a executar as funções de a gerência. A Figura 2-3 mostra os blocos e as componentes funcionais. A TMN se compõe de 5 blocos funcionais descritos nos itens abaixo.

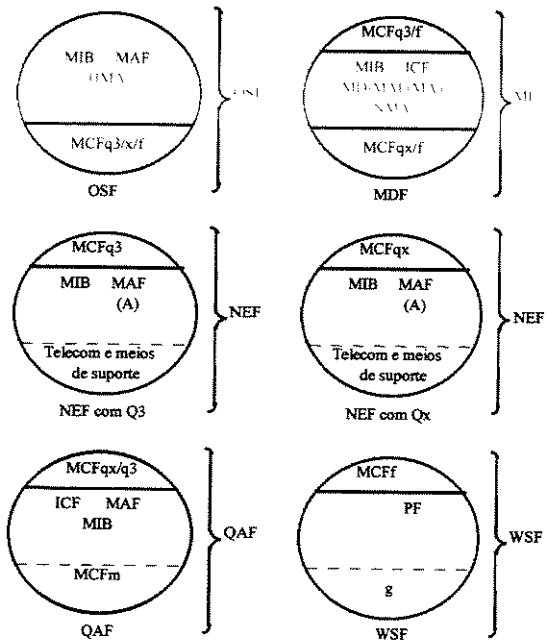

Figura 2-3 - Blocos Funcionais/Componentes TMN

2.2.1.1 Bloco Funcional Sistema de Operação - OSF (*Operation System Function*)

O OSF processa informações relativas à gerência de telecomunicações, a fim de monitorar, coordenar e/ou controlar funções de telecomunicações (incluindo também funções de gerência , isto é, a própria TMN).

2.2.1.2 Bloco Funcional Elemento de Rede – NEF (*Network Element Function*)

O NEF comunica-se com a TMN a fim de ser monitorado e/ou controlado. Além disso o NEF provê funções de telecomunicações e de suporte requeridas pela rede de telecomunicações a ser gerenciada. Apesar destas funções não fazerem parte da TMN elas são representadas pelo NEF para a TMN.

Assim a parte da NEF que provê a representação das funções de telecomunicações para a TMN não faz parte da TMN, apenas as funções de suporte à TMN pertecem à TMN.

2.2.1.3 Bloco Funcional Estação de Trabalho - WSF (*Workstation Function*)

O WSF provê meios para interpretar informações da TMN para o usuário das informações de gerência. A WSF inclui suporte para interface homem/máquina não pertencente à TMN.

2.2.1.4 Bloco Funcional Adaptador Q – QAF (*Q Adaptor Function*)

O QAF é utilizado para conectar entidades não-TMN à TMN, traduzindo pontos de referência proprietários para pontos de referência TMN.

2.2.1.5 Bloco Funcional Mediação - MF (*Mediation Function*)

O MF atua na passagem de informação entre OSF e NEF (ou QAF) para assegurar que as informações sejam adequadas às expectativas dos blocos a serem interconectados pelo MF. Um MF pode armazenar, adaptar, filtrar, nivelar e condensar informações. De maneira simplificada o MF funciona como um tradutor de protocolos e de informação.

2.2.1.6 Componentes Funcionais

Os blocos funcionais, descritos nos itens anteriores, são constituídos pelas componentes funcionais, e uma componente funcional pode estar presente em diferentes blocos funcionais. A seguir descreveremos as 5 componentes funcionais existentes na TMN e em que blocos funcionais elas podem ser encontradas.

2.2.1.6.1 Função de Aplicação de Gerência - MAF (*Management Application Function*)

A MAF é quem realmente implementa os serviços de gerência, podendo assumir o papel de gerente ou agente. Esta componente funcional não está sujeita à padronização dentro da TMN. Dependendo de como são utilizadas podem atuar no papel de gerente ou agente. Quando atua no papel de agente a MAF inclui representações lógicas necessárias para suportar a troca de informação. Estas funções se encontram nos elementos de rede (NE). A MAF pode ser encontrada em todos os blocos funcionais.

2.2.1.6.2 Base de Informações de Gerência - MIB (*Model Information Base*)

A MIB corresponde ao repositório conceitual das informações de gerência. Ela representa o conjunto de objetos (recursos) gerenciados dentro de um sistema gerenciado. Sua estrutura e implementação não estão sujeitos à padronização dentro da TMN. A MIB está em qualquer bloco funcional que tiver dados, assim a MIB pode ser encontrada em todos os blocos funcionais. Mais detalhes sobre a MIB podem ser visto no Capítulo 4.

2.2.1.6.3 Função de Conversão de Informação – ICF (*Information Conversion Function*)

A ICF é utilizada nos sistemas intermediários para traduzir o modelo de informação de uma interface para o modelo de informação de outra interface, por exemplo, convertendo representações de objetos (nível sintático/semântico). Sua implementação não está sujeita a padronização dentro da TMN. Esta função se encontra nos blocos funcionais QAFs e MFs.

2.2.1.6.4 Função de Apresentação – PF (*Presentation Function*)

A PF executa operações gerais para traduzir informações mantidas no modelo de informação da TMN para um formato capaz de ser exibido em uma interface homem-máquina e vice-versa. É utilizada pelo bloco funcional WS.

2.2.1.6.5 Função de Adaptação Homem-Máquina – HMA (*Human Machine Adaptation*)

A HMA executa a conversão do modelo de informação da Função de Aplicação de Gerência (MAF) para o modelo de informação apresentado pela TMN à Função de Apresentação. Pode ser encontrada nos blocos funcionais OSF e MF.

2.2.1.7 Pontos de Referências da TMN

Os pontos de referência da TMN definem fronteiras de serviços entre dois blocos funcionais de a gerência, permitindo identificar as informações trocadas entre estes

blocos. Cada ponto de referência requer características de ponto de referência diferentes para a troca de informações. Existem três classes de pontos de referência:

- classe *q* – entre blocos funcionais OSF, QAF, MF e NEF;
- classe *f* – para ligação de estações de trabalho;
- classe *x* – entre OSFs de duas TMNs ou entre uma OSF de uma TMN e um bloco funcional com funcionalidades equivalentes de outra rede.

Ponto de referência *q*

Este ponto de referência serve para delinear a parte lógica da informação trocada entre os blocos funcionais definidos pelo modelo de informação suportado por estas funções. Os blocos funcionais que se comunicam através deste ponto de referência não suportam todo escopo do modelo de informação, assim a mediação deve ser usada quando houver discrepância.

O ponto de referência *q* é sub-dividido em 2 subclasses:

- ponto de referência *qx*, localizada entre NEF e MF, QAF e MF e entre MF e MF;
- ponto de referência *q3*, localizado entre NEF e OSF, QAF e OSF, MF e OSF, e entre OSFs.

Ponto de referência *f*

O ponto de referência *f* está localizado entre blocos WSF e OSF e/ou WSF e MF.

Ponto de referência *x*

O ponto de referência *x* está localizado entre blocos OSFs em diferentes TMNs.

Ponto de Referência *g*

O ponto de referência *g* está fora da TMN, entre WSF e usuários, não sendo considerado parte da TMN mesmo conduzindo informações da TMN.

Ponto de referência *m*

Está localizado fora da TMN, entre QAF e sistemas de gerência não-TMN.

A Tabela 2-2, abaixo mostra como os pontos de referência (*qx*, *q3*, *f*, *x*, *g* e *m*) podem estar distribuídos entre os blocos funcionais. Por exemplo, entre um bloco funcional NEF e OSF existe um ponto de referência *q3*; entre OSFs pode existir um ponto referência *q3* se os OSFs estiverem na mesma TMN, ou um ponto de referência *x*, se os OSFs pertencerem à TMN diferentes. Os espaços em branco na Tabela 2-2 significam que os blocos funcionais não se conectam.

	NEF	OSF	MF	QAF_{q3}	QAF_{qx}	WSF	Não-TMN
NEF		q3	qx				
OSF	q3	q3,x[1]	q3	q3		f	
MF	qx	q3	qx		qx	f	
QAF_{q3}		q3					m[2]
QAF_{qx}			qx				m
WSF		f	f				g[2,3]
Não-TMN				m	m	g	

Tabela 2-1 - Relações entre os Blocos Funcionais e Pontos de Referência

2.2.2 Arquitetura de Informação

Baseada numa abordagem orientada a objetos, a arquitetura de informação fornece fundamentos para o mapeamento dos princípios de gerência de sistemas OSI (*Open System Management*) em princípios TMN. Neste sentido, os princípios dos sistemas OSI precisam ser expandidos para atender as necessidades específicas do ambiente TMN.

---

[1] O ponto de referência só se aplica quando os blocos OSFs estão em TMNs diferentes.

[2] Os pontos de referência m e g não são pontos de referência TMN.

[3] O ponto de referência g está entre WSF e usuários.

A informação trocada pelos sistemas de gerência é modelada em termos de objetos gerenciados que são abstrações dos recursos e representam suas propriedades, podendo também representar um relacionamento entre recursos ou uma combinação de recursos. Maiores informações sobre o modelo de informação podem ser encontradas no capítulo 4 desta dissertação.

A Arquitetura da Informação utiliza os conceitos de agentes/gerentes (Capítulo 5), domínios e conhecimento de gerência compartilhada (SMK- *Shared Management Knowledge*) da gerência de sistemas OSI.

Pelo SMK, os sistemas de gerência que se comunicam, para garantir seu interfuncionamento, devem compartilhar uma visão comum das informações referentes a protocolos de comunicação, funções de gerência, classes de objetos gerenciados suportados, capacidades autorizadas e relacionamento de *containment*[4] entre os objetos gerenciados.

2.2.3 Arquitetura Física

A arquitetura física descreve interfaces e exemplos de componentes físicos que constituem a TMN. As interfaces padrão TMN correspondem aos pontos de referência definidos na arquitetura funcional. Os blocos físicos correspondem aos blocos funcionais também definidos na arquitetura funcional. A TMN pode ser implementada em várias configurações físicas, a Figura 2-4 exemplifica um modelo físico simplificado desta arquitetura. A Figura 2-4 mostra as interconexões dos vários blocos por um conjunto de interfaces interoperáveis padrão, estas interfaces padrão TMN são definidas em correspondência aos pontos de referência (subsecção 2.2.1.7), sendo aplicadas aos pontos de referência quando conexões físicas externas a eles são necessárias. Os blocos e as interfaces que compõem a arquitetura física TMN são definidos nas subseções abaixo.

---

[4] *Containment* entre objetos – um objeto gerenciado pode estar contido em outro objeto gerenciado.

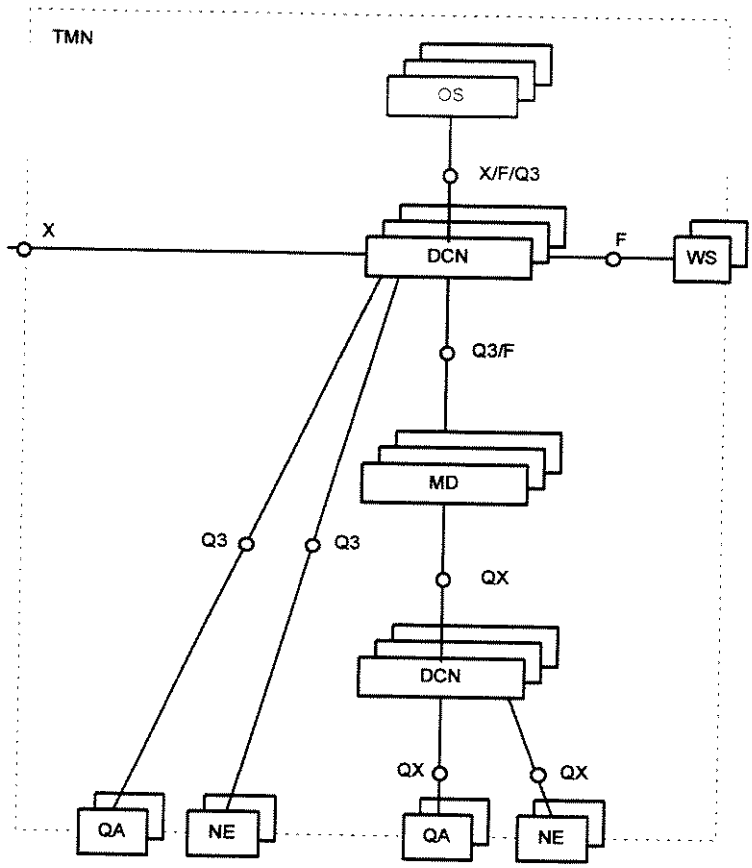

Figura 2-4- Arquitetura Física simplificada para TMN

2.2.3.1 Bloco Funcional Sistema de Operação – OSF (*Operation System Function*)

A arquitetura funcional dos OSs pode ser implementada em diferentes números de OSs, dependendo do tamanho da rede gerenciada, da funcionalidade requerida e da confiabilidade. A escolha do hardware do suporte aos OSs depende se os serviços que eles provêem são em tempo real ou não. As funções de OSs são normalmente implementadas utilizando interface Q_3[5] (Seção 2.2.3.7.1) mas também podem ser

[5] A interface Q_3 é uma subclasse da interface Q que suporta um conjunto mais complexo de funções que a

implementadas num sistema com funções de NE (*Network Element*) ou de MD (*Mediation Device*).

Os OSs englobam todas as funções que permitem realizar o processamento das informações relacionadas com a operação, a administração e a manutenção de redes independentes ou cooperantes. Estas funções incluem:

- programas de aplicações de suporte;
- funções de bancos de dados;
- suporte de terminais de usuários;
- programas de análise;
- formatação de dados e relatórios.

O OS desenvolve as OSFs e pode opcionalmente desenvolver MFs, QAFs e WSFs.

2.2.3.2 Rede de Comunicação de Dados – DCN (*Data Communication Network*)

A Rede de Comunicação de Dados pode ser formada por linhas dedicadas, redes de pacotes, rede digital de serviços integrados, rede telefônica pública comutada, redes locais, entre outros. Os enlaces constituídos a partir desses vários elementos podem ser de uso exclusivo da DCN ou compartilhados com outras redes e serviços.

Os sistemas de Telecomunicações devem fornecer tanto a comunicação espontânea de mensagens quanto diálogos bidirecionais. Um dos OSs deve ser responsável por prover funções que garantam a integridade dos canais de dados utilizados pela DCN.

A DCN é uma rede de comunicação dentro da TMN que suporta a DCF (*Data Communication Function*). E representa uma implementação das camadas OSI 1 a 3, incluindo quaisquer padrões relevantes a ITU-T ou ISO. A DCN pode ser implementada por X.25, WAN, LAN, SSC#7, RDSI ou SDH.

---

outra subclasse Q_x da interface Q. Mais detalhes sobre a interface Q pode ser visto na seção 2.2.3.7.1.

Como visto na Figura 2-4, um DCN conecta NEs, QAs e MDs utilizando a interface Q_3. Adicionalmente o DCN pode ser usado para conectar MDs a NEs e QAs utilizando a interface Q_x , Seção 2.2.3.7.1. O uso da interface padrão Q (Q_x ou Q_3) permite maior flexibilidade no planejamento das comunicações necessárias.

2.2.3.3 Dispositivos de Mediação – MD (*Mediation Device*)

O Dispositivo de Mediação é responsável pela implementação de funções de mediação (MFs), e pode opcionalmente desenvolver as funções OSFs, QAFs e WSFs. A mediação é um processo dentro da TMN que atua na passagem de informações entre NEFs (Network Element Functions) e possibilitam a gerência local para NEs. A mediação utiliza interfaces padrão e podem ser realizadas em dispositivos de mediação separado ou entre NEs.

A mediação pode ser implementada como uma hierarquia de dispositivos em cascata utilizando interfaces padrão. O cascateamento de dispositivos de mediação e várias estruturas interconectadas entre MDs ou entre MDs e NEs oferecem grande flexibilidade a TMN.

Os MDs podem envolver, entre outros, os processos de:

- Conversão de informação entre diferentes modelos de informação.
- Interfuncionamento entre protocolos de alto nível.
- Tratamento de dados (concentração, coleção, formatação, tradução).
- Tomada de decisões.
- Armazenamento de dados.

Assim, os MDs atuam na troca de informações entre componentes da rede, provendo funcionalidades de gerência local para NEs (Elementos de Rede).

2.2.3.4 Elementos de Redes – NE (*Network Element*)

Os NEs implementam as NEFs e podem adicionalmente desenvolver uma ou mais OSFs, MFs ou QAFs. O NE consiste em equipamentos de telecomunicações ou grupos/partes de

suporte ou outros itens pertencentes ao ambiente de telecomunicações que possuam uma ou mais interfaces padrão do tipo Q (opcionalmente F ou X). Equipamentos que não possuem uma interface padrão TMN terão acesso a TMN via a Função de adaptação (QAF), referência [M3010].

Existem 2 classes distintas de funções que podem ser contidas numa NEF:

- Funções de Telecomunicações diretamente envolvidas no processo de telecomunicações, como comutação e transmissão.
- Funções que não estão diretamente envolvidas, como localização de falhas, bilhetagem, comutação, proteção e condicionamento de ar.

É importante observar que as várias partes da NE não estão numa única localização física. Elas podem estar distribuídas ao longo do sistema de transmissão.

2.2.3.5 Adaptadores Q – QA (*Q Adaptor*)

Um QA é tipicamente um conversor de interfaces/protocolos. É um dispositivo que conecta NEs ou OSs com interfaces compatíveis não-TMN (ponto de referência m) a interfaces Q_x ou Q_3 , através das QAFs (Q *Adaptor Function*). O adaptador Q pode conter uma ou mais QAFs.

2.2.3.6 Estações de Trabalho - WS

O WS implementa as WSFs, as WSFs convertem o modelo de informação (Capítulo 4) usado na TMN, disponíveis no ponto de referência *f*, para um formato apresentável ao usuário - ponto de referência *g*. Sua função é atuar como terminal ligado, através da DCN, ao OS ou a um MD, assim deve ter capacidade de processamento e armazenamento de dados para fazer a conversão. As WSs não implementam OSF ou MF, mas se uma OSF e WSF são combinados, esta implementação é considerada OS (ou MD).

Basicamente apresentam as seguintes funções:

- segurança de acesso e *login*;

- reconhecimento e validação de entradas;
- formatação e validação de saídas;
- suporte para menus, telas, janelas, *scrolling* e paginação;
- acesso à TMN;
- ferramentas de desenvolvimento de tela para permitir o desenvolvimento e a modificação do lay-out das telas, a definição de textos fixos e de *helps*, a manutenção da base de dados das telas e facilidades de edição.

2.2.3.7 Interfaces Padrão TMN

Para que dois ou mais blocos da TMN troquem informações de gerência é preciso que os elementos estejam conectados ao mesmo meio de comunicação, e suportem a mesma interface pela via de comunicação. Assim o objetivo da especificação de interfaces interoperáveis é assegurar a compatibilidade dos dispositivos interconectados para acompanhar uma dada função TMN independente do tipo de dispositivo ou fabricante, e evitar a existência de vários padrões de comunicação.

As interfaces interoperáveis definem o conjunto de protocolos e mensagens transportadas pelo protocolo. Elas são orientadas a transação e estão baseadas na visão orientada a objetos de comunicação, formalmente definidos pelo conjunto de protocolos, procedimentos, formato e semântica das mensagens utilizadas para comunicação de gerência.

O que distingue uma interface de outra é o escopo da atividade de gerência que a comunicação na interface deve suportar. Este entendimento comum é definido como conhecimento compartilhado de a gerência (SMK- *Shared Management Knowledge*).

2.2.3.7.1 Interface Q

A interface *Q* é aplicada aos pontos de referência *q*, assim também é subdividida em 2 subclasses:

- Q_x é aplicada ao ponto de referência q_x, suporta um conjunto mais restrito de funções, usando uma pilha de protocolos simples. É apropriada para NEs que requerem poucas funções de OAM.
- Q_3 é aplicada ao ponto de referência q_3, suporta um conjunto complexo de funções, requerendo muitos serviços de protocolos para suportar este conjunto.

Características Principais	
Q_3	Q_x
• Modelo de Informação complexo. • Exige 7 camadas de protocolos. • Exige grande processamento.	• Modelo de Informação simples. • Pode utilizar *short-stack*. • Exige pouco processamento.

Tabela 2-2 – Comparação entre Interface Q_3 e Q_x

2.2.3.7.2 Interface F

Esta interface é aplicada ao ponto de referência *f*. Suporta um conjunto de funções conectando as WSs aos componentes físicos que contêm as OSF e MF, através da DCN. Assim a interface F é um elo que permite ao usuário modificar informações de gerência dentro de uma MAF.

Os protocolos que suportam as interfaces Q e X são diferentes dos que suportam esta interface.

2.2.3.7.3 Interface X

A interface X é aplicada ao ponto de referência *x*. É utilizada para interconectar 2 TMNs, ou uma TMN com outros sistemas de gerência que suportem uma interface TMN-*like*. Assim, por esta interface conectam-se redes de Telecomunicações distintas, deve requerer maior segurança do que para uma interface Q. As componentes funcionais de segurança são: autenticidade, controle de acesso, confiabilidade e integridade dos dados. Em geral, os dois primeiros serviços são mandatórios, os dois últimos vão depender da aplicação.

2.2.3.8 Famílias de Protocolos para TMN

Existe um conjunto de famílias de protocolos para cada interface TMN (Q_3, Q_x, X e F). A escolha do protocolo depende da implementação utilizada na configuração física.

A camada de aplicação é comum para cada família, e é a base para assegurar a interoperabilidade. Nem todas as funcionalidades da camada 7 são necessárias, em algumas interfaces ou todas camadas têm suas funcionalidades reduzidas.

Para equipamentos de redes que não têm interfaces interoperáveis, precisa-se converter protocolos e mensagens para o formato das interfaces interoperáveis. Esta conversão é feita pelas funções MCF e QAF, que podem ser encontrados em Adaptadores Q, Elementos de Rede, Dispositivos de Mediação ou Sistemas de Operação.

No caso da família Q_3, uma única seleção de protocolos (para camadas 4 a 7 do modelo OSI) deve suportar um conjunto de aplicações TMN com necessidade de protocolos similares. Os protocolos para estas camadas podem ser com serviços transacionais ou serviços de transferência de arquivos. Para as camadas 1 a 3, podem ser necessárias opções que permitam o transporte mais eficiente, os protocolos para estas camadas podem ser orientados à conexão ou não orientados à conexão.

No caso da família Q_x ,os atributos funcionais necessários são dependentes das funções de mediação e das diferentes partições das MFs entre diferentes MDs operando em cascata.

CAPÍTULO 3

Protocolos

3.1. INTRODUÇÃO

O objetivo principal deste capítulo é a abordagem dos protocolos OSI necessários para executar a gerência de rede, incluindo os protocolos ACSE, ROSE, CMIS/CMIP. Além disso, ainda são analisados os protocolos RFC1006 e CMOT que são adaptações do protocolo CMIP para redes TCP/IP.

A função fundamental de gerência dos sistemas OSI é a troca de informações de gerência entre agentes e gerentes por meio de um protocolo. Os seguintes serviços-protocolos ISO são necessários para realizar a gerência de rede usando a estrutura OSI: ACSE, ROSE e CMIS/CMIP. Ainda são utilizados os protocolos RFC1006 e CMOT para gerência OSI, eles são uma adaptação do protocolo CMIP para redes TCP/IP. Estes protocolos são definidos utilizando a notação ASN.1 (*Abstract Syntax Notation 1*), que é abordado na seção 4.1.1.1. As regras de codificação para ASN.1 torna a representação dos dados independente da máquina/rede.

3.2 ACSE – Association Control Service Element

O ACSE - *Association Control Service Element* - é utilizado para estabelecer e liberar associações entre entidades de aplicação. Antes que qualquer operação de gerência possa ser efetuada utilizando CMIP é preciso que as duas aplicações envolvidas formem uma associação. O estabelecimento da associação pode ser iniciado tanto pelo agente como pelo gerente. Uma associação ACSE permite ao gerente e agente a troca de A*pplication Entity Titles* que são utilizadas para a identificação e a troca de nomes de contexto de aplicação para estabelecer um contexto de aplicação. Um contexto de aplicação define que elementos de serviços (por exemplo, ROSE e CMISE) devem ser usados sobre a associação.

Depois do estabelecimento da associação, o ACSE só será utilizado novamente na liberação da associação pelo agente ou gerente.

3.3 ROSE – Remote Operation Service Element

O ROSE - *Remote Operation Service Element* - é utilizado para chamada de procedimento remoto no caso de se estar utilizando protocolos OSI. Ele permite que uma operação possa ser executada por um sistema remoto. O protocolo de operação remota contém um identificador de solicitação para correlacionar pedidos e respostas, um código de operação e um campo de argumento para parâmetros específicos à operação. O ROSE só pode ser utilizado depois que uma associação tenha sido estabelecida.

O CMIP utiliza os serviços de transação orientados para todas seus pedidos e respostas assim como as facilidades de respostas de erro providos pelo ROSE.

3.4 CMIS – Common Management Information Service

A troca de informações de gerência, dentro do sistema de gerência OSI, entre agentes e gerentes por meio de um protocolo é referida como CMISE (*Common Management Information Service Element*). CMISE é especificado em duas partes:

- CMIS (*Common Management Information Service*), a interface com o usuário, especificando os serviços fornecidos.
- CMIP (*Common Management Information Protocol*), o protocolo, especificando o formato e *procedures* associadas às PDUs.

O CMIS oferece sete serviços para desenvolver operações de gerência, na forma de primitivas. Além disso, os usuários CMISE precisam ser capazes de estabelecer associações em ordem para desempenhar estas operações de gerência. Estes últimos serviços são providos pelo ACSE, e são fornecidos pelo CMISE como meio de acesso, o CMIP não é envolvido.

O CMISE utiliza o CMIP para troca de PDUs, e o O CMIP, por sua vez, utiliza os serviços do elemento de serviços de operações remotas (ROSE). Tanto o ACSE como o ROSE utilizam os serviços de Apresentação.

A Figura 3-1 abaixo, mostra os serviços oferecidos e utilizados pelo CMISE, e quais são as primitivas utilizadas pelo ACSE, ROSE e CMISE no contexto CMIS e CMIP.

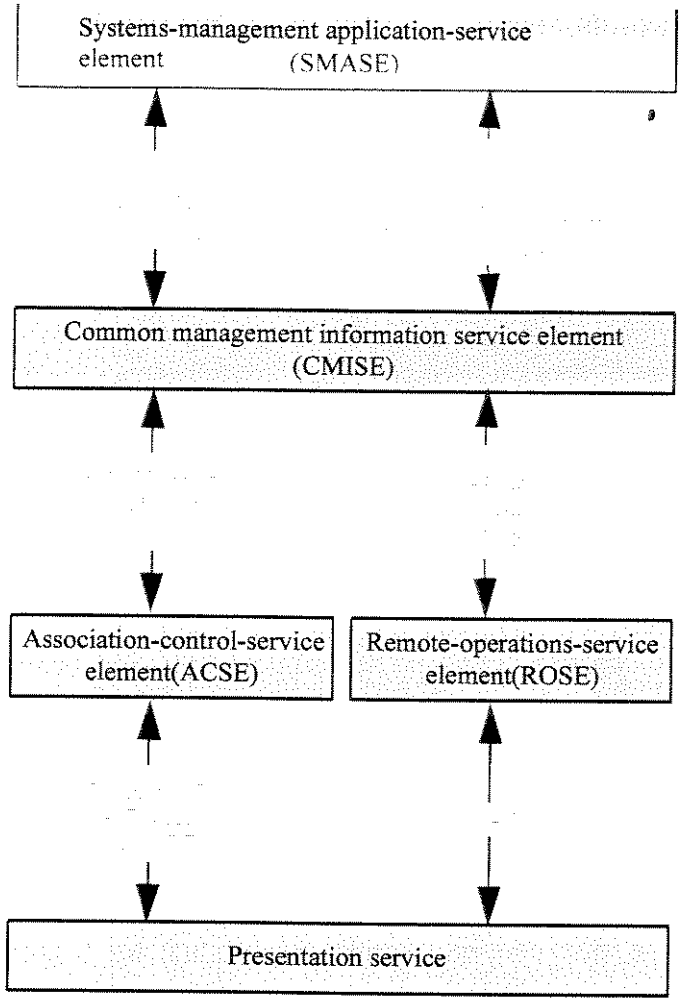

Figura 3-1- Serviços oferecidos e utilizados pelo CMISE

O CMIS define os serviços oferecidos pelos sistemas de gerência OSI. Esses serviços são utilizados por processos de gerência para se comunicarem remotamente. Eles são especificados em termos de primitivas - Tabela 3-1, que podem ser vistas como comandos ou chamadas de *procedures* com parâmetros e são divididos em:

- Serviços confirmados: um processo remoto de gerência envia uma resposta indicando recebimento com sucesso ou falha da operação requisitada.
- Serviços não confirmados: um processo remoto NÃO enviará resposta indicando recebimento com sucesso ou falha da operação requisitada.

Os serviços CMIS podem ser divididos em 2 classes:

1. **Serviços de Associação** - usuários CMIS precisam estabelecer uma associação para se comunicarem. Eles utilizam o ACSE para controle das associações das aplicações. São

os seguintes os serviços invocados: M-INITIALIZE, M-TERMINATE e M-ABORT.

2. **Serviços de Transferência de Informação**. Nesta classe exisem 2 tipos de serviços:

   - **Serviços de notificação de gerência** - Este serviço é utilizado para enviar informações de gerência referentes a uma notificação. A definição de notificações e o conseqüente comportamento das entidades de comunicação são dependentes das especificações do objeto gerenciado que gera as notificações estando fora do escopo do CMIS. O serviço invocado para notificação de gerência é: M-EVENT-REPORT.
   - **Serviços de operações de gerência** - Estes serviços são utilizados para enviar informações de gerência referentes a operações de gerência de sistemas. A definição da operação e o conseqüente comportamento das entidades de comunicação são dependentes da especificação do objeto gerenciado para o qual a operação está direcionada, também está fora do escopo do CMIS. São os seguintes serviços invocados para serviços de operaçõe de gerência: M-GET, M-SET, M-ACTION, M-CREATE, M-DELETE.

O CMIS ainda oferece duas facilidades

- Respostas múltiplas a uma operação confirmada podem ser associadas à operação através do uso do parâmetro *linked-identification* das primitivas.
- Operações podem ser executadas em vários objetos gerenciados, selecionados através de algum critério e sujeitos a uma condição de sincronização.

As Tabela 3-1a e 3-1b descrevem as componentes das duas classes dos serviços CMIS, serviços de Associação de Gerência e Transferência de Informação, respectivamente.

1. Serviços de Associação de Gerência		
Serviço	**Tipo**	**Definição**
M-INITIALIZE	Confirmado	é invocado para estabelecer uma associação com o propósito de troca de informações de gerência.
M-TERMINATE	Confirmado	é invocado para liberar uma associação.
M-ABORT	não-Confirmado	é invocado para liberar uma associação de forma abrupta.

Tabela 3-1a– Serviços CMISE – Serviços de Associação de Gerência

2. Serviços de Transferência de Informação		
Serviços de Notificação de Gerência		
Serviço	Tipo	Definição
M-EVENT-REPORT	Confirmado/não-confirmado	avisa um usuário par de um serviço CMISE a respeito de um evento com um objeto gerenciado.
Serviços de Operações de Gerência		
M-GET	Confirmado	pede a recuperação de informação de gerência de um usuário par de serviço CMISE
M-SET	Confirmado/não-confirmado	pede a modificação de informação de gerência por um usuário par de serviço CMISE
M-ACTION	Confirmado/não-confirmado	pede que um usuário par de um serviço CMISE execute uma ação
M-CREATE	Confirmado	pede que um usuário par de um serviço CMISE crie uma instância de um objeto gerenciado
M-DELETE	Confirmado	pede que um usuário par de um serviço CMISE remova uma instância de um objeto gerenciado
M-CANCEL-GET	Confirmado	pede que um usuário par de um serviço CMISE cancele um pedido pendente do serviço M-GET

Tabela 3-1b - Serviços CMISE – Serviços de Transferência de Informação

3.4.1 Linkage

As primitivas de serviço M-GET, M-SET, M-ACTION e M-DELETE podem especificar operações em vários objetos. Quando isso acontece, uma resposta é devolvida para cada objeto. Alguma técnica especial é necessária para associar as várias respostas com o pedido inicial que as gerou. Esta associação é necessária pois podem existir alguns pedidos ainda pendentes e as respostas que chegam precisam ser associadas aos pedidos anteriormente enviados.

O *Linkage* é oferecido através do parâmetro *linked-identifier*, que aparece em cada primitiva *response* e *confirm* com valor igual ao do parâmetro *invoke identifier* das primitivas de *request* e *indication*. O *invoke identifier* é um identificador único associado a cada operação.

3.4.2 Seleção de Objetos Gerenciados

O CMIS oferece um conjunto de ferramentas que permitem que um usuário selecione um ou mais objetos a serem submetidos a uma operação de gerência. Essas facilidades são oferecidas como parâmetros nas primitivas de serviço M-GET, M-SET, M-ACTION e M-DELETE.

A Seleção de Objetos Gerenciados (*Managed-Object-Selection*) envolve os conceitos de ***scoping, filtering*** **e sincronismo.** Parâmetros baseados nesses conceitos podem estar presentes nas primitivas de serviço citadas anteriormente para especificar um grupo de objetos com determinadas características e aplicar sobre eles os comandos de gerência.

3.4.3 Serviço de Notificação de Gerência

A única primitiva definida para a notificação de gerência é M-EVENT-REPORT, usada para enviar uma notificação ao gerente. Ao contrário das outras operações de gerência, o *event report* é iniciado pelo processo agente. O serviço pode ser confirmado ou não confirmado.

3.4.4 Serviços de Operação de Gerência

3.4.4.1 Serviço *M-GET*

O serviço M-GET permite a recuperação de dados da MIB. Um processo de gerência, desempenhando o papel de gerente, envia um pedido de GET para outro processo de gerência, que desempenha o papel de agente. O pedido pode se relacionar com um ou mais objetos gerenciados, e para cada um deles pode ser pedido o valor de um, vários ou todos os atributos. O serviço M-GET é sempre confirmado.

3.4.4.2 Serviço *M-SET*

O serviço M-SET permite a modificação de dados da MIB. É usado para mudar o valor de um ou mais atributos em um ou mais objetos gerenciados. O serviço M-SET pode ser confirmado ou não confirmado.

3.4.4.3 Serviço M-ACTION

O serviço M-ACTION permite a chamada de um procedimento de ação pré-definido como parte de um objeto gerenciado. O pedido especifica o tipo da ação e os parâmetros de entrada. O serviço M-ACTION pode ser um serviço confirmado ou não confirmado.

3.4.4.4 Serviço *M-CREATE*

O serviço M-CREATE é usado para criar uma nova instância de classe de objeto. Os pacotes condicionais e os valores de atributos que o objeto gerenciado deve ter podem ser especificados como parte do pedido, ou então pode ser utilizada uma instância já existente como modelo. O serviço M-CREATE é sempre confirmado.

3.4.4.5 Serviço *M-DELETE*

O serviço M-DELETE é usado para eliminar um ou mais objetos da MIB, e é sempre confirmado.

3.4.4.6 Serviço *M-CANCEL-GET*

O serviço M-CANCEL-GET é usado para interromper uma operação de GET muito longa. O motivo pelo qual apenas a operação de GET pode ser interrompida é que é difícil garantir a consistência da MIB se uma operação capaz de alterá-la é interrompida depois de ser iniciada. O M-CANCEL-GET é sempre um serviço confirmado.

3.4.5 Serviços de Associação

Para que dois usuários CMIS possam executar operações de gerência, é preciso estabelecer antes uma associação de aplicação. Para isso, o CMIS utiliza os serviços do ACSE (*Asssociation-Control-Service-Element*).

O ACSE oferece duas as funcionalidades aos usuários CMIS. A primeira é estabelecer uma associação de aplicação que possa ser usada para trocar primitivas de serviços CMIS. A segunda ocorre no momento do estabelecimento da associação e é a negociação entre dois usuários pelas características do CMIS que serão usadas nessa associação. Essas características são expressas em termos de unidades funcionais.

todas as primitivas CMIS, exceto M-CANCEL-GET, com a limitação de que respostas múltiplas, *scoping*, *filtering* e sincronização não são suportados. Essas limitações se aplicam aos serviços M-GET, M-SET e M-ACTION. Os dois usuários podem negociar unidades funcionais adicionais a serem acrescentadas ao *kernel* da associação em questão. Uma aplicação é estabelecida usando o serviço A-ASSOCIATE do ACSE. O usuário do CMIS fornece os parâmetros pedidos ao emitir um A-ASSOCIATE.*request*. O parâmetro *user-information* do ACSE é dividido em três partes específicas do CMIS:

1. **Unidades funcionais:** Este parâmetro pode aparecer tanto nas primitivas de *request/indication* como nas de *response/confirm* e é a lista de unidades funcionais adicionais a serem acrescentadas à unidade funcional do *kernel*. Apenas as unidades funcionais que aparecem em ambas listas são acrescentadas ao *kernel*.
2. **Controle de acesso:** informação de formato não especificado a ser usado pelo controle de acesso.
3. **Informação do usuário:** informação específica da aplicação.

Uma associação pode ser liberada por qualquer um dos usuários do CMIS. A liberação em condições normais é excutada através do serviço A-RELEASE, e a liberação abrupta através do serviço A-ABORT.

3.5 CMIP (Common Management Information Protocol)

Enquanto o CMIS define os serviços para as operações de gerência, o CMIP (*Common Management Information Protocol*) define procedimentos para a transmissão da informação de gerência e define a sintaxe para o serviço de gerência do CMIS. O CMIP é definido em termos das PDUs (*Protocol Data Units*) que são trocadas entre pares de elementos de serviço de informação de gerência (CMISEs) para executar o serviço CMIS.

3.5.1 Operação CMIP

Para entender o CMIP é preciso vê-lo em contexto com os serviços CMISE e os serviços que o CMISE oferece. A Figura 3-1 mostra esta relação. Como já mencionado anteriormente, o CMISE oferece 7 serviços para desenvolver operações de gerência, na

forma de primitivas[1]. Os usuários CMISE precisam ser capazes de estabelecer associações em ordem para desenvolverem operações de gerência. Estes serviços são fornecidos pelo ACSE (Association Control Service).

3.5.2 CMOT (Common Management Information Protocol over TCP/IP)

A arquitetura CMOT é baseada na estrutura de gerência OSI e nos modelos, serviços, e protocolos desenvolvidos pelo ISO para gerência de rede. A arquitetura CMOT demonstra como a estrutura e gerência OSI pode ser aplicada a ambientes TCP/IP e usada para gerenciar objetos numa rede TCP/IP.

A arquitetura CMOT especifica todas as componentes essenciais de uma arquitetura de gerência de rede para construir um interoperável sistema de gerência de rede multifornecedor.

3.5.2.1 Modelos de Gerência

As seções seguintes indicam como a arquitetura CMOT se aplica aos modelos de gerência OSI e aponta algumas limitações.

A arquitetura CMOT atual diz respeito somente às operações e características de um único domínio de gerência.

Quanto ao modelo funcional, a arquitetura CMOT provê os fundamentos para atuar nas 5 áreas funcionais: falha, configuração, desempenho, segurança e contabilização. Mas a arquitetura não descreve como estes tipos de gerência devem ser realizados.

3.5.2.2 Arquitetura do Protocolo

O objetivo da arquitetura do protocolo é mapear a arquitetura do protocolo de gerência OSI num ambiente TCP/IP. O que está atualmente padronizado nesta arquitetura é o mínimo requerido para se obter interoperabilidade entre diferentes fornecedores de sistemas de gerência de rede.

Os protocolos utilizados para gerência de rede ISO são: ACSE, ROSE e CMIP. No lugar de implementar estes protocolos nos topos das camadas de apresentação, sessão e

---

[1] Estas primitivas sempre incluem um entre 4 modificadores padrão: *request, indication, response e confirm.*

transporte, as PDUs do ACSE, ROSE e CMIP são mapeadas direto na camada de transporte utilizando os protocolos TCP ou UDP. Isto é possível através da utilização do *Lightweight Presentation Protocol* - LPP **[RFC1085]**, que mapeia ROSE e ACSE sobre TCP/UDP/IP. A Figura 3-2 mostra a representação da arquitetura do protocolo CMOT.

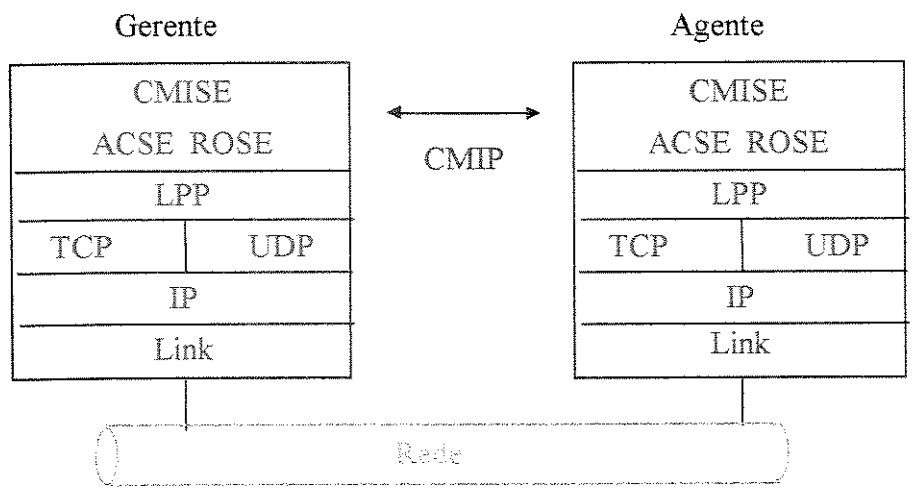

Figura 3-2- Arquitetura do protocolo CMOT

3.5.3 RCF1006

O CCITT e a ISO têm definido várias recomendações de sessão, apresentação e aplicações que têm sido adotadas por comunidades internacionais e numerosos fabricantes. Isto permite uma convergência e estratégia de transição mais fácil de redes TCP/IP para redes baseadas em ISO, a médio e longo prazo.

Existem duas aproximações básicas que podem ser usadas na passagem de aplicações ISO ou CCITT para o ambiente TCP/IP. A primeira aproximação é tornar portável cada aplicação individual separadamente, desenvolvendo protocolos locais sobre TCP/IP. A segunda aproximação é baseada na observação que tanto o protocolo Internet ARPA, quanto o protocolo ISO são sistemas baseados em camadas. Um aspecto chave do princípio de camadas é a independência entre elas.

Existem duas definições para interface de acesso da camada:

- A definição de serviços oferecidos, que descrevem os serviços providos pela camada e as interfaces que a camada provê para acesso a estes serviços.

- A definição de serviços requeridos, que descrevem serviços usados pela camada e as interfaces utilizadas para acesso a estes serviços.

Coletivamente, todas as entidades na rede que cooperam para prover o serviço são conhecidas como provedoras de serviços. Individualmente, cada uma destas entidades é considerada como um ponto de serviço.

Implementada internamente, existe uma definição para camada:

- uma definição de protocolo que descreve as regras que cada ponto de serviço usa quando se comunica com outros pontos de serviços.

Assim, podemos dizer que o provedor de serviço usa o protocolo de serviço da camada inferior para oferecer seu serviço à camada superior.

A característica de independência entre camadas será utilizado para definir *Transport Service Acess Point* (TSAP) que é idêntico aos serviços e interfaces oferecidas pela ISO/CCITT TSAP, mas na realidade foi implementado o protocolo ISO TP0 (transporte classe 0) no topo do TCP/IP, não no topo do protocolo de rede ISO/CCITT. Desde que o protocolo de transporte classe 0 (TP0) é utilizado sobre a conexão TCP/IP, ele realiza funções idênticas às do transporte classe 4. Assim, as camadas altas (sessão, apresentação e aplicação) podem operar normalmente sem o conhecimento do fato que eles estão executando sobre TCP/IP.

A Figura 3-3 mostra a representação da arquitetura do protocolo RFC1006.

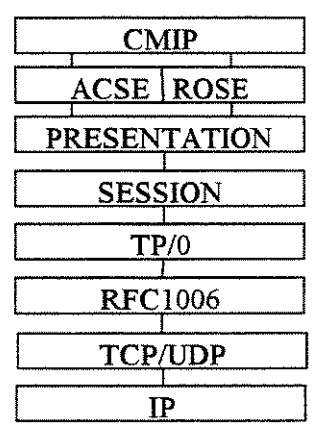

Figura 3-3- Arquitetura do Protocolo RFC1006

CAPÍTULO 4

OSI Management Information Base

4.1. INTRODUÇÃO

Os protocolos de comunicação incluindo TCP/IP e OSI, foram desenvolvidos para fornecer comunicações entre sistemas. Mas para a gerência de redes distribuídas, os protocolos de comunicação sozinhos não são suficientes.

As aplicações distribuídas requerem um padrão de descrição dos recursos a serem gerenciados. Assim o padrão OSI adotou um modelo orientado a objeto para encapsular os recursos e padronizar a interface que eles apresentam à rede. Assim neste capítulo serão vistas as linguagens utilizadas nos modelos de informação: GDMO e ASN.1.

O modelo orientado a objeto promove as seguintes qualidades nas aplicações de gerência de rede:

- modularidade, onde a funcionalidade necessária é decomposta de acordo com o que foi definido e é prontamente implementada em módulos;
- extensibilidade, que permite que aplicações distribuídas sejam construídas sobre funções e serviços de aplicações existentes.

Na gerência de redes orientada a objetos, códigos e dados são unidos em uma entidade chamada objeto gerenciado. Objetos gerenciados são dispositivos, sistemas, ou "qualquer coisa" que requeira alguma forma de monitoramento ou gerência. Um objeto gerenciado é composto de um conjunto de atributos que o caracterizam, ações que ele pode executar e notificações que ele pode emitir espontaneamente. Genericamente, um objeto pode ser definido como uma entidade para a qual operações são requisitadas. Para requisitar alguma ação a um objeto gerenciado envia-se a ele uma mensagem, o objeto pode responder com resultados através do retorno de outra mensagem. Esta troca de mensagem é a única forma de interação possível com os objetos.

Cada recurso que é monitorado e controlado nos sistemas de gerência OSI é representado por objetos gerenciados. As informações sobre os recursos e elementos a serem

gerenciados formam a base de dados do sistema de gerência de rede. Nos sistemas de gerência OSI, esta base de dados recebe o nome de MIB (Management Information Base) [X.720]. Sendo, assim, a MIB é um conjunto estruturado de tais objetos. Abaixo, alguns pontos relevantes sobre os objetos gerenciados devem ser observados:

- Um objeto gerenciado é uma abstração que está disponível para as funções dos sistemas de gerência. Alguns outros mecanismos fora do escopo dos padrões de gerência OSI, mantém a relação entre o objeto gerenciado e o recurso real.
- Um único objeto gerenciado pode representar vários recursos.
- O mesmo recurso pode ser representado por diferentes objetos gerenciados, cada um representando uma característica diferente do recurso.
- Nem todos os recursos precisam ser representados por objetos gerenciados, não significando que tais recursos não existem, apenas eles não estão disponíveis para gerência de sistemas OSI.
- Alguns objetos gerenciados são definidos apenas para suportar as funções de gerência e não representam recursos. Como por exemplo, *logs* de eventos e filtros[1].

A definição de classes de objetos é realizada usando-se *templates* com um formato específico. Para definir classes de objetos utiliza-se o GDMO - *Guidelines for Development of Managed Objects* [X.722] que poderá ser melhor entendido nas próximas seções.

4.2 Linguagens utilizadas no modelo de informação

Paralelamente ao desenvolvimento de protocolos OSI, um conjunto de linguagens não-proprietárias foi desenvolvido e padronizado para que os sistemas abertos pudessem acomodar uma representação de dados (notação de sintaxe abstrata) e uma codificação

---

[1] *Logs* de Eventos são arquivos onde ficam armazenados todos os acontecimentos considerados relevantes pelo sistema de gerência.

Os filtros permitem a seleção de determinados eventos escolhidos pelo ususário.

correspondente para transferência de dados (sintaxe de transferência). Eles provêem uma linguagem universal para programação de aplicações de rede que permite que as aplicações troquem valores de dados sem perder a semântica destes dados - como estão estruturados, que tipos de dados estão presentes em uma estrutura complexa e qual o tamanho da estrutura.

A representação universal para os valores dos dados em sistemas abertos é necessária devido às diferenças entre as representações internas de diferentes sistemas e entre diversas arquiteturas de máquinas. Um inteiro pode ser representado com tamanhos diferentes, pode ter um bit mais significativo ocupando a primeira posição de um byte em um sistema e a última posição em outro sistema. Por causa de problemas deste tipo, não é possível trabalhar com sistemas abertos sem uma linguagem universal, independente da arquitetura da máquina, através da qual é possível representar elementos básicos de informação de forma que:

- a informação possa ser interpretada sem ambigüidades, em qualquer contexto;
- não haja, a princípio, limites para as estruturas de dados;
- a informação possa ser trocada entre sistemas sem perda de semântica.

O OSI captura o significado (semântica) dos dados trocados entre sistemas abertos (sintaxe abstrata) independente da especificação e representação interna desses dados no computador (sintaxe concreta) e dos padrões de bits utilizados para transmitir a estrutura de dados de um computador para outro (sintaxe de transferência). A separação da sintaxe de transferência abstrata da concreta é significativa no sentido de que os dados podem ser representados sem que haja preocupação com espaço em memória ou com a maneira como eles são armazenados em qualquer computador.

4.2.1 ASN.1 - Abstract Syntax Notation 1

ASN.1 **[X.208]** é a linguagem mais aceita para representação de dados para sistemas abertos. Ele foi inicialmente desenvolvido como parte do trabalho em camadas superiores do modelo OSI para servir como representação uniforme de qualquer tipo de dados (tipos de dados pré-definidos, padronizados ou *user-defined*) para que os objetos do ambiente

OSI - cabeçalhos de protocolos e textos de mensagens de correio eletrônico, entradas de diretórios e informações de gerência - possam ser enviados de um sistema para outro de forma que possam ser interpretados sem referência (aderência) a nenhuma máquina ou arquitetura de sistema de informação. O ASN.1 foi adotado desde então como a linguagem de especificação por pessoas de outras áreas além de OSI, como TCP/IP, apesar do fato de ser freqüentemente criticado como dispendioso em termos de ciclos de processador, ao desempenhar a transformação entre representação nativa de um sistema e o ASN.1.

A Recomendação X.209**[X.209]** define as regras básicas de codificação (*basic encoding rules* - BER) para codificar os tipos abstratos do ASN.1 em *streams* de bits que são trocados pelos sistemas abertos.

As descrições do ASN.1 são usadas para especificar PDUs (*protocol data units*), definir objetos que podem ser usados para gerenciar recursos de rede, e para definir objetos e atributos de objetos que podem ser registrados e acrescentados em um diretório ou base de dados global. Utilizando ASN.1 é possível construir estruturas de dados tão complexas quanto se queira, e manter a semântica dessas estruturas através de vários sistemas de computadores diferentes. Pode-se pensar nas PDUs codificadas em ASN.1 como verbos (eles exigem ações como "pegar", "setar", "modificar", "ler", "abrir", "fechar", "buscar" e "inicializar") e seus objetos como substantivos - pode-se "pegar" o valor de um objeto gerenciado, por exemplo, ou "ler" um atributo de uma entrada de diretório. Existe até como associar nomes próprios - *object identifiers*.

4.2.2 GDMO - *Guidelines for Development of Managed Objects*

O GDMO é essencialmente uma linguagem orientada a objetos, usada para definir um conjunto de classes de objetos gerenciados. As normas da ISO especificam como as informações de gerência devem ser definidas, que ferramentas de notação devem ser empregadas em cada definição e que estrutura de documentação deve ser usada na definição das classes de objetos gerenciados.

Uma definição de classe de objeto especifica os atributos, ações, notificações e o comportamento de um objeto. As relações entre objetos são representadas como árvores. Existem 3 relações que fornecem uma estrutura geral para definir e usar objetos: nomeação, registro e herança. Elas constituem uma base de conhecimento comum em redes e comunidades de gerência de sistemas. Nota-se, entretanto, que estas relações não são parte inerente do paradigma de orientação a objetos, mas são importantes para trabalhar em gerência de redes e sistemas.

- **Registro:** a árvore de registro é uma hierarquia de identificadores únicos para classes de objetos, ações, eventos e atributos. A árvore de registro diz respeito a estes objetos e não a instâncias de objetos. Como várias definições de objetos são adicionadas à comunidade de redes, a cada uma é atribuído um único número de identificação. Um identificador de classe de objeto é construído de uma série de inteiros que atravessam um caminho da árvore de registro desde a raiz até o nó a ser identificado. Cada nó desta árvore tem uma autoridade de registro associada que determina como os números na sub-árvore devem ser alocados. A relação registro é usada durante a fase de projeto para construir definições que são unicamente registradas com uma autoridade central. Os números de registro também são usados nas aplicações em tempo de execução, para identificar as classes de objetos gerenciados instanciados, isto é, os que foram criados efetivamente pelo software, tendo memória física alocada.

- **Nomeação:** descreve como cada objeto instanciado está relacionado com outras instâncias dentro de um ambiente particular. Esta relação diz respeito a instâncias e não a classes de objetos. Toda instância de objeto está contida (instância subordianda) dentro de outras instâncias (instância superior).Um objeto existe apenas se sua instância superior continuar existindo. Toda classe de objeto tem um atributo distinto (*distinguished name* - DN) que é usado para identificar cada instância de uma classe. Um *distinguished name* junto com o valor daquele atributo é chamado de RDN - *Relative Distinguished Name.*

- **Herança:** descreve qual classe de objeto deriva de outras. Quando uma classe de objetos é derivada de outra, ela herda os atributos e ações da classe base. Portanto a relação de gerência é utilizada para definir novas classes de objetos baseadas nas classes já existentes. A classe mãe de outra é chamada de superclasse e a filha de subclasse. Uma classe de objeto pode ter mais que uma superclasse, quando isto acontece, esta relação é chamada de múltipla herança.

O acesso a objetos gerenciáveis requer uma definição clara da visão externa do objeto. A modelagem de objetos é uma tarefa de descrição formal das características e operações importantes de uma classe de objetos.

CAPÍTULO 5

PLATAFORMA TMN

5.1. INTRODUÇÃO

Nesta capítulo são identificadas a estrutura e os componentes básicos de uma plataforma TMN. Além disso é introduzido o conceito do paradigma agente/gerente amplamente utilizado pelas aplicações de gerência TMN.

5.2. CARACTERIZAÇÃO DA PLATAFORMA TMN

A Plataforma TMN pode ser entendida como sendo uma camada de *software* que oferece serviços específicos para determinadas classes de aplicações de gerência. Essas classes de aplicações são utilizadas nos sistemas de gerência de rede e gerência de elemento de rede, conforme definido na recomendação M3010 do ITU-T [M3010].

Esta recomendação define que a interligação entre os diferentes sistemas dá-se por meio de interfaces padronizadas e abertas. O conceito de interface na TMN está baseado no paradigma gerente-agente e compreende um modelo de informação associado (composto por um conjunto de objetos gerenciados OSI) e uma pilha de protocolos (onde é obrigatório o uso do protocolo de gerência CMIP). A Figura 5-1 mostra a comunicação entre uma entidade agente e outra gerente através de uma interface interoperável TMN e ilustra como se dá o mapeamento de recursos reais em objetos gerenciados. Estes objetos são vistos pelo sistema de gerência através da interface.

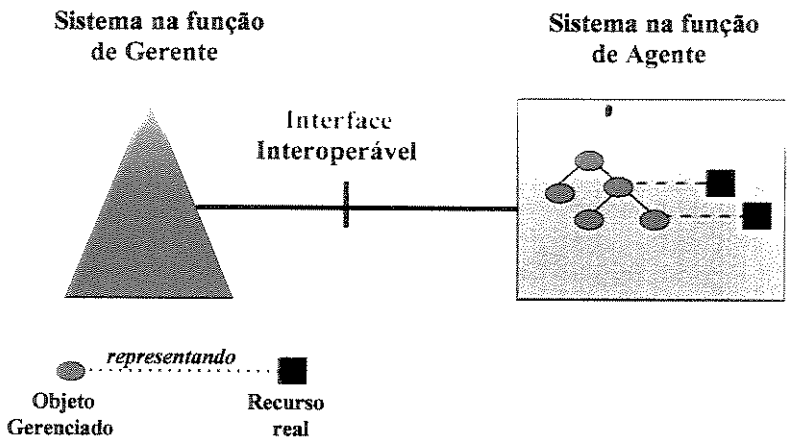

Figura 5-1- Modelo de gerência utilizado na arquitetura TMN

O objetivo de uma plataforma TMN é incorporar soluções tecnológicas que automatizem e acelerem o processo de desenvolvimento de software baseados na arquitetura mostrada acima. Como a padronização dos modelos de gerência de redes de telecomunicações é bastante complexo, um outro objetivo da plataforma TMN é esconder das aplicações desenvolvidas os detalhes envolvidos nesta padronização para simplificar o desenvolvimento das aplicações.

Com a utilização de tais plataformas, pode-se obter benefícios tais como reusabilidade, modularidade e portabilidade das aplicações de gerência, tornando transparente para o usuário os detalhes tecnológicos do ambiente computacional, pois são oferecidos às aplicações um acesso padronizado aos serviços prestados pela plataforma. Além disso, a plataforma oferece suporte a um ambiente distribuído, que propicia independência de localização de objetos e processos.

5.2.1 Estrutura da Plataforma

A seguir são descritos os serviços e funcionalidades básicas providos por uma plataforma TMN. A Figura 5-2 mostra a estrutura geral destes módulos. Dentre os módulos mais importantes de uma plataforma TMN pode-se citar:

- Acesso a qualquer serviço através de interfaces abertas (utilização intensiva de APIs, XOM/XMP, padronizadas)

- Serviço de Comunicação (acesso a protocolos de gerência – CMIP, CMOT, etc.);
- Serviços de Base de Dados (armazenamento de objetos gerenciados – Serviços de Registro);
- Serviços de Distribuição (independência de localização de objetos);

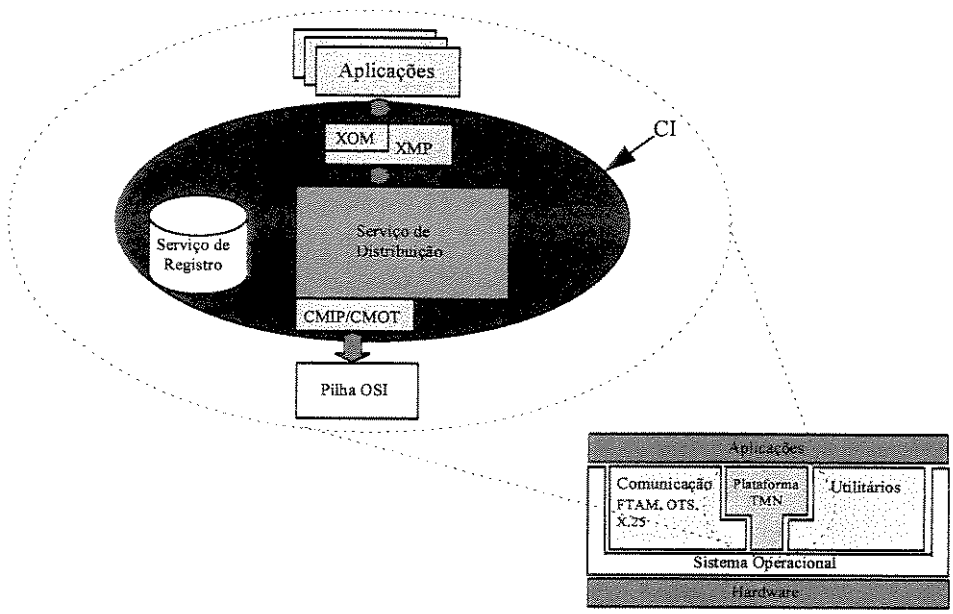

Figura 5-2 - Estrutura básica de uma Plataforma TMN

5.2.1.1 Acesso aos serviços através de interfaces padronizadas

As especificações para a gerência de redes definidas por organismos de padronização, tais como ISO e ITU, dão ênfase à definição de padrões para as interfaces interoperáveis entre dois sistemas de gerência. Estas interfaces são baseadas no serviço CMIS e no protocolo de gerência CMIP, em conjunto com estruturas padronizadas para representação e troca da informação de gerência representada na forma de objetos gerenciados, definidos de acordo com a recomendação X.722 [X.722] do ITU-T, chamada de *Guidelines for the Definition of Managed Objects* (GDMO).

Porém, nenhum padrão recomendado pela ISO ou pelo ITU define *como* uma aplicação de gerência deve ser implementada de forma a se obter uma interface interoperável. A especificação de uma API busca ser uma *ponte* entre as especificações abstratas dos

organismos internacionais e as especificações computacionais, onde são necessárias definições de mais baixo nível.

O objetivo de uma API é prover um conjunto consistente de primitivas e estruturas de dados ao desenvolvedor de aplicações, substituindo as interfaces proprietárias de cada plataforma para acesso aos seus serviços. Assim, a definição de uma API facilita o desenvolvimento de aplicações e permite a portabilidade das mesmas em um ambiente multifornecedor, já que o acesso aos serviços se torna padronizado, escondendo das aplicações parte dos detalhes referentes a comunicação envolvidos na gerência (protocolos e formato de dados).

Foram com estes objetivos que a API XMP (*X/Open Management Protocol*) foi especificada, sendo hoje um padrão *de facto* para as plataformas de aplicações de gerência. A API XMP oferece aos projetistas de sistemas de gerência um mecanismo comum de acesso, tanto para o serviço CMIS, utilizado principalmente em sistemas de gerência de redes de telecomunicações, como para o serviço SNMP, utilizado principalmente em sistemas de gerência de redes de computadores e podendo também fornecer um mecanismo comum de acesso a outros protocolos. Juntamente com a API XMP a X/Open definiu a API XOM (*X/Open Abstract Data Manipulation*) para a manipulação de tipos de dados definidos em ASN.1 [**FOR**], é através da API XOM que é feito o acesso padronizado à base de dados. A Figura 5-3 mostra o contexto de utilização das API XOM/XMP no acesso aos recursos de comunicação e base de dados numa plataforma TMN.

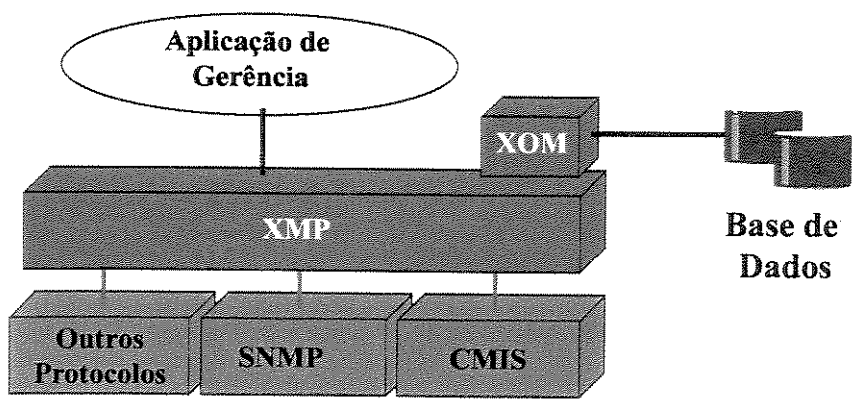

Figura 5-3 - Escopo das API XOM/XMP

5.2.1.2 Serviços de Comunicação

A implementação de interfaces TMN é realizada basicamente através de pilhas de protocolos específicas, que compreendem necessariamente a utilização de protocolos de gerência. Os protocolos de gerência possibilitam a troca de operações e notificações entre o sistema gerenciado e o sistema gerenciador, sendo que o acesso a estes protocolos é feito através de APIs XOM/XMP, como citado no item anterior. Com a finalidade de oferecer as diversas possibilidades de comunicação entre sistemas TMN, foram agregadas às plataformas TMN serviços que oferecem acesso a vários tipos de *stacks* de protocolos. O conjunto desses serviços são aqui chamados de serviços de comunicação. As principais características desses serviços são:

- Disponibilizar os dois protocolos de gerência mais aceitos no mercado, SNMP (*Simple Network Management Protocol*) **[Brisa]** e CMIP (*Common Management Information Protocol*) **[X.711]** para uso das aplicações de gerência. No mercado de telecomunicações, ao contrário da informática, o protocolo CMIP é atualmente o mais difundido.

- Implementar várias opções de protocolos de transporte. Entende-se aqui por protocolos de transporte os protocolos que realizam as funções definidas nas camadas 1 (física) a 4 (transporte) do modelo OSI. As opções de perfis de protocolos utilizados para TMN são definidas na recomendação Q.811 do ITU-T [Q.811]. Os perfis mais comuns são o CONS1 (X.25) e o CLNS1 (ISO 8473 sobre ISO 8802.3). Mais recentemente foi incluída uma proposta de padronização do perfil RFC 1006 (OSI sobre TCP/IP) que pode vir a se tornar um perfil oficial e também ser um protocolo muito utilizado.

- Acesso através de API padronizada (XOM/XMP), conforme descrito no item anterior.

5.2.1.3 Serviços de Base de Dados

O serviço de base de dados é utilizado em conjunto com o serviço de diretórios para que a transparência em relação à localização seja obtida. As consultas e atualizações permitem isolar a aplicação de detalhes específicos de implementação, além de

realizarem conversão de formatos e linguagens de acesso e oferecerem facilidades para acesso a base de dados locais ou remotas.

5.2.1.4 Serviços de Distribuição

As características de sistemas distribuídos são inerentes ao ambiente de telecomunicações. Portanto, para oferecer todo o suporte necessário ao desenvolvimento de aplicações de gerência e facilitar a manipulação dos recursos da rede por parte das aplicações, foram desenvolvidos mecanismos que possibilitam oferecer transparência quanto a localização dos recursos utilizados pelas aplicações. Isto faz com que a aplicação não necessite saber em qual máquina e endereço de um determinado objeto gerenciado a ser acessado.

Como os recursos estão geograficamente dispersos e sujeitos a mudanças de localização, é inteiramente desaconselhável a dependência, por parte da aplicação, de endereços de redes e processos. As funções básicas deste serviço são a de mapear nomes amigáveis a endereços físicos de máquinas e recursos da rede e a de prover um serviço onde o usuário possa obter a localização na rede de algum recurso a ser acessado.

A Figura 5-4 ilustra o mecanismo de atuação da Plataforma TMN que possibilita que as aplicações se abstraiam de endereços de rede e nomes complexos. Para que a entidade A acesse a entidade B, ela simplesmente emite a mensagem a ser transmitida para a Plataforma TMN. A plataforma então possui uma função que consulta uma base de dados que fará a translação do nome da entidade B para um endereço físico na rede mais alguns parâmetros de protocolos. Só depois dessa translação é que a mensagem é encaminhada para o serviço de comunicação (protocolos de gerência) para então ser efetivamente encaminhada à rede. Esse mecanismo, embora contribua para uma simplificação das aplicações, introduz *overheads* no sistema. A análise de alguns desses *overheads* são objetos deste trabalho e serão descritos a seguir.

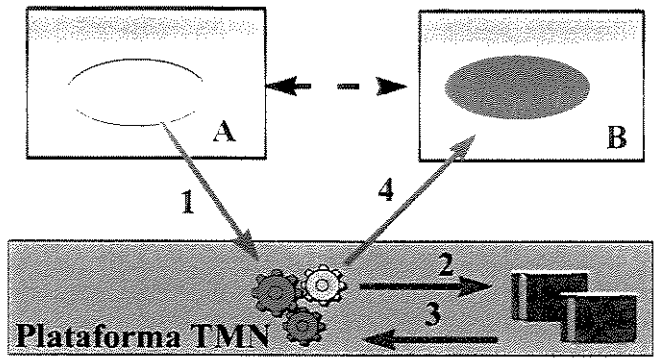

Figura 5-4- Serviço de distribuição na Plataforma TMN

5.3. Paradigma Agente/Gerente

Os conceitos fundamentais básicos de gerência OSI são bastante simples, sendo composto pelo paradigma agente/gerente, onde os processos chamados gerente estão nos sistemas de gerência e os processos chamados agente estão sobre sistemas gerenciados (ou elementos de rede sendo gerenciados). A gerência acontece na troca de informações úteis de controle e monitoração de uma rede e suas componentes. Esta troca é feita através de protocolos padronizados.

O ambiente de gerência OSI inclui os conceitos de gerentes, agentes e objetos gerenciados, descritos a seguir.

O **agente** é um *software* que gerencia um conjunto de objetos, tendo permissão de realizar determinadas ações sobre eles, dependendo de como foram definidos. Ele responde às operações de gerência emitidas pelo gerente fornecendo a este uma visão dos objetos. Também é possível que o agente receba notificações e informações de gerência que venham do recurso gerenciado.

Através dos serviços e protocolos de gerência o **gerente** se comunica com o agente a fim de obter informações a respeito do objetos gerenciados pelo agente , podendo estar na mesma máquina ou em máquina diferente a do agente. O gerente pode receber notificações do agente se o estado dos objetos gerenciados indicarem esta necessidade.

Os termos *agente* e *gerente* são usados para denotar a relação assimétrica entre às aplicações de gerência em que o gerente é o superior e o agente é o subordinado.

Os **objetos** são representações dos recursos que estão sujeitos a algum gerência por parte dos agentes/gerentes , eles são definidos em termos de atributos e comportamentos, das operações a que podem ser submetidos, notificações que podem emitir e das relações com outros objetos gerenciados. O conjunto de todos os objetos gerenciados por um agente compõem a MIB (*Management Information Base*)[**X.721**].

O agente OSI se comunica com o gerente através de protocolos padronizados, mas com objetos gerenciados a comunicação pode ser feita através de protocolos proprietários.

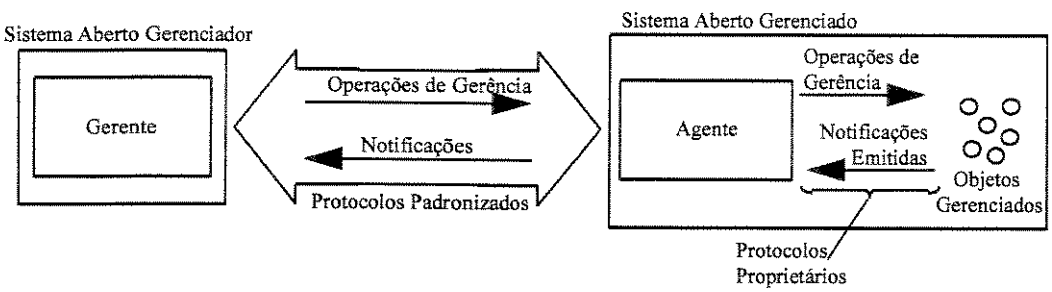

Figura 5-5- Interação entre Gerente/Agente/objetos

A Figura 5-5 mostra um diagrama da interação entre gerente, agente e os objetos. A comunicação entre o agente e gerente é feita sincronamente e confirmada, assim o agente espera até que o gerente faça alguma requisição de gerência (*get*, *set*, *create* e *delete*) para responder. O agente consulta a MIB para preencher a resposta e manda para o gerente. O agente pode ainda enviar espontaneamente notificações para o gerente, como por exemplo, alarmes.

CAPÍTULO 6

DESEMPENHO, PORTABILIDADE & INTEROPERABILIDADE

6.1. INTRODUÇÃO

Neste capítulo serão definidos o fatores que influenciam o desempenho, a portabilidade e a interoperabilidade de sistemas de gerência de redes de Telecomunicações. Assim sendo, no item de "desempenho" a seguir será mostrado que variáveis influenciam o desempenho e qual a configuração utilizada nos testes. Também será feito o estudo para interoperabilidade e portabilidade entre diferentes plataformas comerciais, da mesma forma que foi feito para o desempenho.

6.2. DESEMPENHO

6.2.1 Variáveis de influência

Como já mencionado anteriormente, as Plataformas TMN transformaram-se rapidamente em padrão *de facto* para a implementação de sistemas de gerência da rede de telecomunicações. Porém o uso intensivo por parte das aplicações de gerência desse novo *middleware* acarreta sem dúvida um *overhead* de processamento que deve ser analisado com critério por parte dos usuários dos sistemas de forma que seja verificado o quanto ele vai influenciar o desempenho dos sistemas.

A Figura 6-1 mostra um caso típico de utilização de uma Plataforma TMN. Esse tipo de utilização se dá quando da implementação de um sistema integrador de gerência. Este sistema tem a função de interoperar com os elementos de rede (geralmente de vários fabricantes) através da coleta de dados de gerência e do envio de comandos a esses elementos. Para um dimensionamento apropriado desse sistema de gerência, temos que responder a algumas perguntas básicas tais como:

- Qual é o número de elementos de rede da planta a ser gerenciada ?

- Qual é a estimativa do número de mensagens espontâneas enviadas por cada tipo de elemento de rede?
- Qual é o comprimento médio dessas mensagens?
- Quais são os perfis de protocolos que serão utilizados na arquitetura do sistema?
- Qual é o número de instâncias de objetos a ser manipulado pelo sistema?
- Etc.

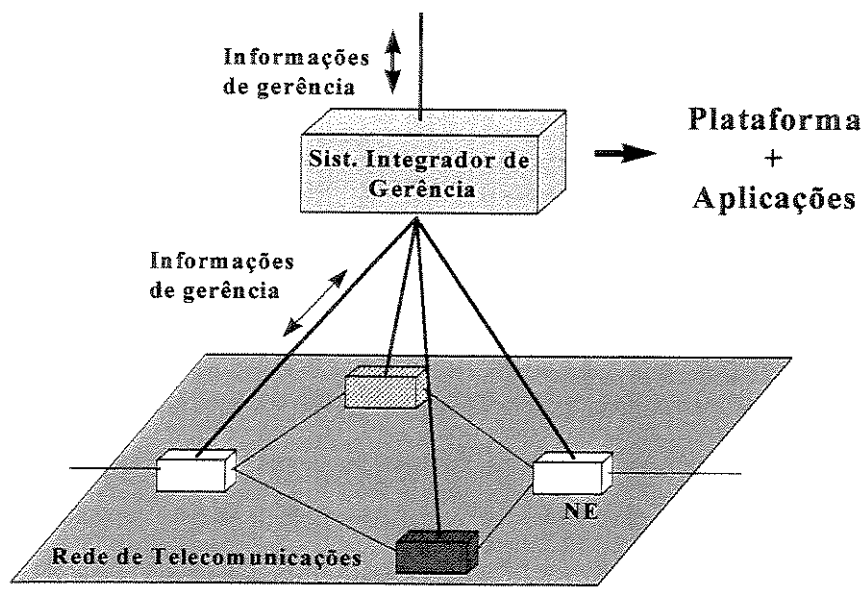

Figura 6-1- Utilização típica de uma Plataforma TMN

Porém, para que as respostas a essas questões sejam realmente úteis no trabalho de especificação da arquitetura de um sistema, torna-se necessário conhecer quais são os valores máximos de desempenho obtidos com sistemas construídos em plataformas TMN, de modo que os requisitos esperados para o sistema especificado sejam realmente factíveis.

Para tanto, tomando-se como base a estrutura de uma plataforma TMN genérica mostrada na seção 5.2.1, foram levantados alguns itens que influenciam no desempenho de um sistema de gerência construído sobre uma plataforma TMN.

Após uma análise preliminar desses fatores, constatou-se duas grandes classes de influência no desempenho desses sistemas. A primeira classe está relacionada a fatores de armazenamento de objetos, onde são fatores importantes o tipo de base de dados utilizada (relacional ou orientada a objetos), o número de instâncias a ser armazenado, a velocidade de tempo de acesso, etc. A segunda classe agrupa fatores relacionados à distribuição do processamento e da utilização de diferentes pilhas de protocolos TMN.

Por uma questão de opção, este trabalho aborda somente a segunda classe de fatores. Ou seja, esta opção se deu principalmente por ser esta classe a menos explorada em termos comparativos (encontram-se dados oferecidos por fabricantes de plataformas relativos a determinadas configurações apenas). Como o Laboratório TMN do CPqD possui vários tipos de plataformas TMN e várias ferramentas de medidas relacionadas a protocolos de gerência, decidiu-se por abordar inicialmente os aspectos relativos aos fatores de comunicação e distribuição de processamento.

Dentre os fatores desta classe que influenciam no desempenho de um sistema, pode-se citar:

- Relação entre o número de agentes e gerentes;
- Tipos e versões de API utilizadas;
- *Stack* de protocolos TMN de nível superior: CMIP ou CMOT;
- *Stack* de protocolos de transporte: X.25, LAN/ISO 8473, TCP/IP, etc.;
- Utilização de funções de serviço diretório;
- Tamanho médio das mensagens de gerência;
- Utilização de mensagens síncronas ou assíncronas;
- Utilização de mensagens confirmadas ou não confirmadas;
- Tipo mais freqüente de mensagem de gerência executada (*create*, *get*, *set*, ou *event report*);

A partir deste elenco de fatores, começou-se a especificar os testes que seriam executados para se saber qual o comportamento de um sistema TMN mediante a variação dos mesmos. Onde, entende-se por comportamento, ao levantamento da taxa de mensagens por segundo que uma aplicação TMN consegue efetivamente manipular e quais desses fatores ocasionam maior impacto no desempenho global do sistema.

6.2.1.1 Configuração do Sistema

Para a realização dos testes aqui descritos e outros em fase de implementação, foram utilizadas 3 estações de trabalho RISC da HP e um Power PC da IBM. Estas estações estão interligadas por uma rede local utilizando padrão CSMA/CD com a taxa de transmissão de 10 Mbit/s.

Foram utilizadas máquinas com capacidades diferentes de processamento, para que se pudesse comparar as diferenças de desempenho entre elas. Mostra-se abaixo as configurações utilizadas para cada estação.

- HP 9000 715/80: configuradas com HP-UX 9.05, 128 MB de memória.
- HP 9000 715/64 configurada com HP-UX 9.05, 64 MB de memória.
- IBM PowerPC RISCsystem/6000/C20 configuradas com AIX 3.2 e 45 MB de memória.

A Figura 6-2 sumariza algumas informações relativas à capacidade de processamento das máquinas utilizadas.

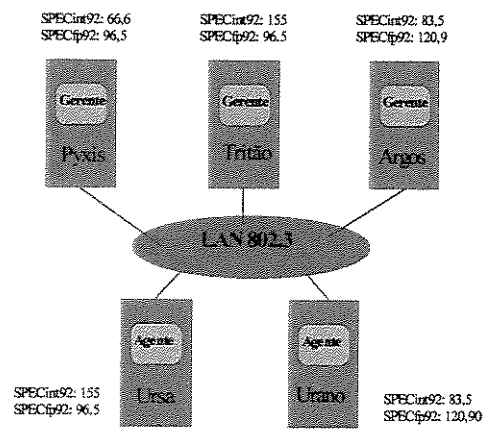

Figura 6-2- Configuração do ambiente de testes de desempenho

6.3. INTEROPERABILIDADE

As especificações para gerência de redes definidas por organismos de padronização, tais como ISO e ITU-T dão ênfase à definição de padrões para as interfaces interoperáveis entre dois sistemas de gerência. Estas interfaces são baseadas no serviço CMIS e no protocolo de gerência CMIP, em conjunto com estruturas padronizadas para a representação e troca de informação de gerência representada na forma de objetos gerenciados, definidos de acordo com a X.722 do ITU-T **[X.722]**, chamada de *Guidelines for Definition of Managed Objects* (GDMO).

Nenhum padrão recomendado pela ISO ou pelo ITU-T define como uma aplicação de gerência deve ser implementada de forma a se ter uma interface interoperável. Para os desenvolvedores de aplicações isso é um fator de dificuldade. Abaixo são mostrados alguns pontos que são ou não objetos de padronização.

- As normas utilizadas na camada de apresentação do modelo OSI definem como um valor, descrito na notação ASN.1 **[X.208]**, é transmitido através de uma interface interoperável, mas não existem normas que definem como este dado deve ser manipulado pela aplicação.
- Os serviços CMIS definem um conjunto de primitivas de gerência, mas não define como uma aplicação acessa a estes serviços.
- As normas definem o conceito de objeto gerenciado como sendo uma abstração do recurso do mundo real que deve ser gerenciado. Definem também as propriedades do objetos gerenciados que são visíveis através de uma interface interoperável (atributos, operações, comportamentos e notificações). Entretanto, as normas não definem como estas propriedades são implementadas ou acessadas por uma aplicação.

Como já mencionado anteriormente (seção 5.2.1.1), o objetivo de uma API é prover um conjunto consistente de primitivas e estrutura de dados ao desenvolvedor de aplicações, substituindo as interfaces proprietárias de cada plataforma para acesso aos seus serviços. Assim sendo, a definição de uma API permite o desenvolvimento de aplicações e facilita

a portabilidade das mesmas em ambiente multifornecedor, já que o acesso aos serviços se torna padronizado, escondendo das aplicações parte dos detalhes referentes a comunicação envolvidos na gerência (protocolos e formato de dados).

6.4. PORTABILIDADE

Neste trabalho foi feito um estudo de portabilidade de agentes/gerentes em diferentes plataformas. A portabilidade depende do compilador C que está sendo utilizado, das APIs XOM/XMP, de arquivos de configuração utilizados pela plataforma TMN, da forma de registro dos agentes, objetos e gerente junto a base de dados e da infra-estrutura de comunicação.

É importante observar qual a versão das APIs XOM/XMP e as funções que estão sendo utilizadas visto que podem existir funções proprietárias de um dado fabricante.

Assim para tornar um código portável é preciso verificar :

- Compilador utilizado.
- Versão/tipo de sistema operacional utilizado.
- Versão XOM/XMP.
- Funções XOM/XMP que estão sendo utilizadas e verificar a presença de agente/gerente na plataforma.
- Como as funções foram implementadas.

Na Tabela 6-1 pode ser visto um exemplo de comparação de códigos em diferentes plataformas.

Normalmente este trabalho de portabilidade não permite que sejam usadas as ferramentas "amigáveis" fornecidas pelos fabricantes para implementação dos agentes/gerentes, pois uma das "facilidades " que elas oferecem é esconder do desenvolvedor a implementação destas APIs (XOM/XMP).

ATabela 6-1 exemplifica a maioria das modificações necessárias num agente/gerente básicos resultantes do estudo de portabilidade entre plataformas TMN de diferentes fabricantes.

Fabricante 1	**Fabricante 2**
• Os diretórios onde se encontram as bibliotecas são diferentes.	
• Existem diferenças na implementação das estruturas XOM/XMP.	
• O nome da estrutura `mp_negotiate` do Fabricante 2 é `mp_version` no Fabricante 1.	

`Typedef struct{`
`  OM_object_identifier feature;`
`  OM_boolean activated;`
`}MP_feature;`
	`typedef struct{`
	`  OM_object_identifier feature;`
	`  OM_boolean activated;`
	`  OM_boolean response;`
	`}MP_feature;`

`Typedef struct{`
`  OM_private_object feature;`
`  OM_boolean activated;`
`}MP_Waiting_Sessions;`
	`typedef struct{`
	`  OM_private_object feature;`
	`  OM_boolean activated;`
	`}MP_waiting_sessions;`

`MP_status APIENTRY mp_receive(`
`  OM_private_object session,`
`  OM_sint *primitive_return,`
`  OM_sint *mode_return,`
`  OM_sint *completion_flag_return,`
`  MP_status`
`  *operation_notification_status_`
`  return,`
`  OM_private_object`
`  *result_of_argument_return,`
	`MP_status APIENTRY mp_receive(`
	`  OM_private_object session,`
	`  OM_private_object context,`
	`  OM_sint *primitive_return,`
	`  OM_sint *completion_flag_return,`
	`  MP_status`
	`  *operation_notification_statu`
	`  s_return,`
	`  OM_private_object`
	`  *result_or_argument_return,`

`OM_sint *invoke_id_return`	`OM_sint32 *invoke_id_return`
O nome das bibliotecas para linkar são diferentes: -lxmp -lntl -lov –lm	O nome das bibliotecas para linkar são diferentes: -lovxomg -lxmpV7 -lntl -lov -lm

Não existem as definições que estão ao lado para este fabricante.	Definições utilizadas neste fabricante:
	`#define MP_ACTIVATE 0`
	`#define MP_DEACTIVATE 1`
	`#define MP_QUERY_STATE 2`
	`#define MP_QUERY_SUPPORTED 3`

A função `at_gmtime_to_gentime()` não foi encontrada.	A função `at_gmtime_to_gentime()` foi encontrada no diretório `/usr/lib/libov.a`

Tabela 6-1- Diferenças encontradas nas APIs XOM/XMP de diferentes fabricantes

Na Tabela 6-1 pode ser observado que os diretórios onde se encontram as bibliotecas são diferentes com isso o projetista tem que tomar cuidado com a localização das bibliotecas informando para os aplicativos o lugar correto das bibliotecas que estão sendo utilizadas. Em relação ao segundo item, diferenças de implementação das estruturas XOM/XMP, têm que ser observados como estas funções estão implementadas, para que elas possam ser usadas corretamente pelos aplicativos. Em alguns casos a mesma função tem nome diferentes para diferentes fabricantes, como por exemplo a função `mp_negotiate()` do fabricante 2 é chamada de `mp_version()` pelo fabricante 1.

No caso dos aplicativos utilizados para o desenvolvimento deste trabalho, foi percebido diferenças nas estruturas `mp_feature()`, `MP_Wating_Sessions` e a função `mp_receive()` tem diferenças de implementação e para que o mesmo aplicativo possa funcionar em diferentes fabricantes é necessário observar como as funçoões estão implementadas para que possam ser usadas corretamente e ter o resultado desejado.

O nome das bibliotecas para *link* são diferentes, assim quando os aplicativos são compilados devem ser passados o correto nome das bibliotecas utilizadas para os aplicativos.

Em alguns casos, observou-se a definição de alguns termos usados nos aplicativos que não estavam definidos em outros fabricantes.

Também foi observado a utilização de algumas funções que estavam definidas apenas em alguns fabricantes.

6.5. XOM/XMP

A API XMP (*X/Open Management Protocol*) é atualmente um padrão *de facto* para as plataformas de aplicações de gerência. Ela oferece ao desenvolvedores de sistemas de gerência um mecanismo comum de acesso, tanto para os serviços CMIS, utilizados principalmente em sistemas de gerência de redes de telecomunicações, como para o serviço SNMP, utilizado principalmente em sistemas de rede computadores. Trabalhando em conjunto com a API XMP existe a API XOM (*X/Open Abstract Data Manipulation*) para a manipulação de tipos de dados definidos em ASN.1. Uma descrição melhor destas duas APIs pode ser vista nas próximas seções.

6.5.1 XOM

O principal objetivo da API XOM é esconder a complexidade da notação ASN.1, provendo um mecanismo geral de representação e manipulação de dados abstratos, sendo especialmente idealizada para uso com outras API OSI. Apesar de definir uma interface orientada a objeto, a API XOM não incorpora todas as características desta interface, como por exemplo a propriedade de encapsulamento.

As variáveis e parâmetros utilizados nos argumentos das funções XMP são definidos através da API XOM. A API XMP utiliza as funções XOM para criar, examinar, modificar e destruir os argumentos das funções XMP.

A API XOM provê um mecanismo geral de manipulação de dados. É a API XOM que vai manipular os dados definidos em ASN.1 (Abstract Syntax Notation One)**[X.208]**. Sendo

um de seus maiores propósitos esconder tanto quanto possível a complexidade do ASN.1, incluindo o BER (*Basic Encoding Rules*) **[X.209]**. O BER são as regras básicas de codificação utilizadas para codificar os tipos abstratos do ASN.1.

Assim, percebe-se que a API XOM não pode existir sozinha, pois ela faz parte da implementação de outra API (como por exemplo a API XMP). Mas é importante salientar que para que uma API utilize a API XOM duas componentes devem ser especificadas:

1. Um conjunto de funções específicos para a API. No caso do XMP, este conjunto inclui os serviços e funções CMIS e SNMP e as várias funções relacionadas. Os parâmetros e respostas destas funções podem ser um pouco complexos, o que provoca a necessidade de um novo componente.
2. Um conjunto de informações de objeto estruturado, que constituem os parâmetros e respostas das funções da API. O principal propósito de XOM é simplificar a gerência destes objetos de informação.

A API XOM provê uma representação de dados, definindo uma estrutura chamada de objetos OM. Um objeto OM é uma estrutura definida em linguagem C, usada para representar tipos de dados abstratos que refletem uma representação dos dados em ASN.1, que são utilizadas pela infra-estrutura de comunicação.

6.5.1.1 Arquitetura da Informação

Aqui serão descritas as estruturas básicas de dados que a aplicação cliente troca com o provedor de serviço. Entende-se por provedor de serviço o software que implementa a API. Para evitar alguma futura confusão entre as funções que são utilizadas pela API XOM e XMP, pois algumas delas têm nomes bem parecidos, as funções XOM são precedidas por "om_" e as funções XMP são precedidas por "mp_", esta regra também se aplica aos nomes das estruturas definidas na API XOM.

Esta regra é importante para evitar confusão entre termos, tais como objeto gerenciado (abstração de um recurso) e objeto OM (estrutura de dados). Assim como, evitar confusão entre uma função `om_create()` e uma função `mp_create_req()` que têm efeitos e utilizações bem diferentes.

Abaixo mostramos algumas definições importantes para API XOM:

- **OM Attribute**: é o componente básico de um objeto OM. Ele é representado por uma estrutura em C chamada `OM_descriptor`, sendo composta de 3 campos: tipo, sintax e valor do atributo:
  - o **tipo** especifica o nome do atributo OM.
  - A **sintax** especifica como o campo *value* deve ser interpretado de acordo com a sintax ASN.1, suportada pela especificação XOM.
  - E o campo **value** contém o valor do atributo.
- **OM Object:** é constituído de uma lista de atributos OM e é representado na linguagem C, por um vetor de estruturas de dados do tipo `OM_descriptor`. O número de atributos, o tipo e a sintaxe de cada um dos atributos que compõem um objeto é especificado pela classe a qual este objeto pertence.
- **OM Class:** Uma classe OM define os seguintes elementos: o nome da classe, a identificação de sua superclasse, a definição dos atributos específicos de cada classe e uma indicação se a classe é abstrata (não instanceável) ou concreta (instanceável).

Muitos dos parâmetros e resultados de funções XOM e XMP são objetos, que por definição são membros de alguma classe OM. A Tabela 6-2 mostra a representação de um objeto OM genérico.

	Tipo	**Sintax**	**Valor**
OM ATRIBUTO$_0$	OM_CLASS	OM_S_OBJECT_IDENTIFIER_STRING	int32 pointer
OM ATRIBUTO$_1$	Uma constante definida em xmp.h, xmp_cmis.h, xmp_snmp.h ou, mais raramente, xom.h	Constante definida nas definições de sintaxe em xom.h	int32 pointer
.	.	.	.
.	.	.	.
.	.	.	.

OM ATRIBUTO$_N$	Uma cte definida em xmp.h, xmp_cmis.h, xmp_snmp.h ou, mais raramente, xom.h	Constante definida nas definições de sintaxe em xom.h	int32 pointer
OM ATRIBUTO$_{N+1}$	OM_NO_MORE_TYPES	OM_S_NO_MORE_SYNTAX	0 (NULL)

Tabela 6-2 - Representação de um objeto OM

- **Packages:** um pacote XOM é um conjunto de classes OM que possuem funcionalidades correlatas. Quando se utiliza a API XOM em conjunto com a API XMP são disponíveis 4 pacotes básicos: *OM package*, *Common package*, *CMIS package*, *SNMP package*. Podem ser definidos outros pacotes XOM para novas classes de objetos XOM para aplicações específicas.

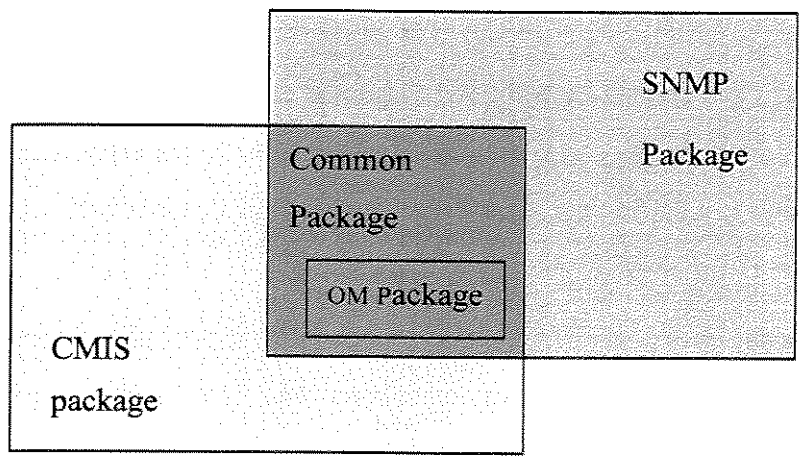

Figura 6-3- Relacionamento entre pacotes CMIS e SNMP

- **Workspace:** é uma área reservada ao controle de recursos por uma particular instância de uma implementação XOM. É necessário criar e destruir um workspace em toda e qualquer aplicação (tanto no agente quanto no gerente). Isto deve ser feito na inicialização da aplicação. Criar um workspace é uma tarefa bastante fácil:

```
workspace = mp_initialize();
```

E para destruir:

```
mp_shutdown(workspace);
```

É importante não confundir os termos definidos acima com os termos de objetos orientados no domínio da gerência de rede. Um `om_object` é uma estrutura de dados e um objeto gerenciado é um recurso a ser gerenciado.

Como a API XOM é orientada a objeto, isto inclui o conceito de herança, isto é, as classes OM estão relacionadas numa hierárquia de superclasse, classe e subclasses. Esta hierárquia forma uma estrutura em árvore sem cruzamentos. Estes conceitos podem ser melhor compreendidos através do exemplo na

Figura 6-4, que mostra o relacionamento entre herança e os tipos de classe em XOM.

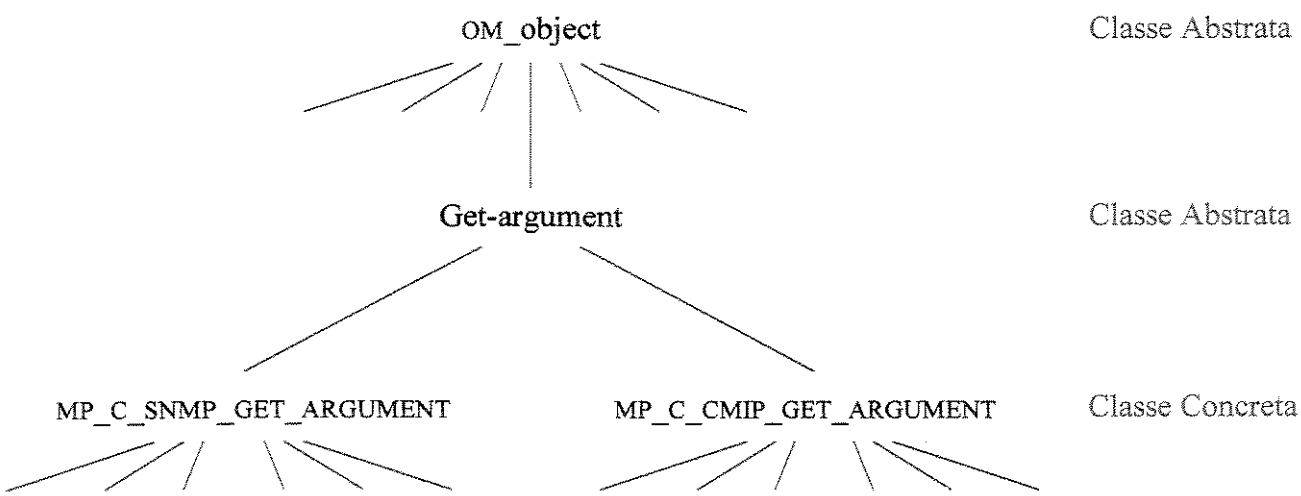

Figura 6-4- Herança e tipos de classe em XOM

Uma ***subclasse*** herda todos os atributos OM da classe OM pai. A classe OM pai é chamada de ***superclasse*** da subclasse. Assim uma instância de uma classe OM é sempre uma instância de cada uma das suas superclasses.

No exemplo da Figura 6-4, a função `mp_get_req()`, pela API XMP, tanto pode ser utilizada para operações no serviço CMIS como no serviço SNMP. Isto é possível por ela ter como parâmetro a classe abstrata OM `get_arg()`. Esta classe é superclasse das classes concretas CMIS-Get-Argument e SNMP-Get-Argument. Assim, a função `mp_get_req()` pode aceitar como parâmetro qualquer objeto das subclasses que

derivam da classe `get_arg()`. É desta forma que XMP pode servir a usuários CMIS e SNMP.

Assim, com o uso das APIS percebe-se uma uniformidade nas chamadas das funções, mas esta uniformização não é total, já que a aplicação precisa saber o protocolo de gerência que está sendo utilizado.

O exemplo ainda mostra que existem classes **abstratas** e **concretas**. As classes OM abstratas existem para prover (através da herança) atributos OM comuns a um grupo de classes concretas. Não é possível criar objetos OM cuja classe OM é abstrata, objetos OM só podem ser criados se vierem de classes concretas. Na

Figura 6-4, as classes concretas têm o prefixo `MP_C_` e são escritas todas em maiúsculas. É desta forma que se identifica classes concretas na linguagem C.

Enfatizando, XOM existe para criar, examinar, modificar e destruir parâmetros complexos e resultados de funções, chamados de objetos OM , para manter isto XOM pode manter representações dos objetos OM no *workspace*. Para criar e destruir estes dados existem dois tipos de objetos OM:

- **Objetos Privados:** um objeto privado é mantido em um *workspace* e sua representação interna não está acessível para manipulação direta pelo desenvolvedor. A manipulação destes objetos deve ser feita através da utilização de funções definidas pela API XOM.

Por exemplo, para mudar o atributo OM de um objeto privado deve-se utilizar a função `om_put()`. Objetos privados OM podem ser transformados em públicos utilizando-se a função `om_get()`.

- **Objetos Públicos:** um objeto público consiste de uma estrutura de dados em C que pode ser manipulada diretamente pela aplicação. Os objetos públicos também podem ser transformados em privados (função `om_put()`).

A saída de funções XOM ou XMP são sempre objetos privados, com uma única exceção, a função `om_get()`. Os objetos públicos podem ser de dois tipos:

1. Gerados por serviços: são objetos considerados apenas para leitura e são criados quando requisita-se uma cópia pública de algum objeto OM privado.
2. Gerado pelo cliente: estes objetos são criados e mantidos pelo desenvolvedor da aplicação, sendo assim, a gerência da memória utilizado para criação destes objetos é responsabilidade do desenvolvedor da aplicação.

6.5.1.2 Funções da API XOM

A Tabela 6-3 abaixo sumariza as funções XOM.

Função	Descrição
`om_copy()`	Cria uma cópia independente de um objeto OM privado.
`om_copy_value()`	Copia uma *string* de um objeto OM privado para outro.
`om_create()`	Cria um novo objeto OM privado.
`om_decode()`	Cria um objeto OM privado que representa um objeto OM privado codificado.
`om_delete()`	Destrói um objeto OM privado ou um objeto OM público que foi gerado por um serviço.
`om_encode()`	Cria um novo objeto OM privado codificado usando *Basic Encoding Rules* (BER) a partir de um objeto OM privado existente.
`om_get()`	Cria uma cópia pública de todo ou de partes de um objeto OM privado.
`om_instance()`	Verifica se um objeto OM é membro de uma determinada classe OM.
`om_put()`	Coloca os valores de atributos de um objeto OM (público ou privado) em um objeto OM privado.

`om_read()`	Lê um segmento de uma *string* de um objeto OM privado.
`om_remove()`	Remove o valor de um atributo ou o atributo de um objeto OM privado.
`om_write()`	Escreve o segmento de uma *string* em um objeto OM privado.

Tabela 6-3 - Funções XOM

Observar que algumas funções estão inversamente relacionadas:

- `om_delete()` e `om_create()` têm efeitos inversos.
- `om_decode ()` e `om_encode ()` têm efeitos inversos.
- `om_get ()` ou `om_remove ()` tem efeito inverso a `om_put()`.
- `om_read ()` e `om_write ()` têm efeitos inversos.

6.5.2 XMP

A API XMP (*X/Open management Protocols*) consiste de uma biblioteca de funções que mapeam serviços dos protocolos de Gerência CMIP e SNMP. Como já pode-se observar, XMP utiliza as funções da API XOM para criar, examinar, modificar e destruir os argumentos de suas funções.

As chamadas aos serviços providos pela API XMP podem ser síncronas ou assíncronas. A interface XMP é simétrica, provendo todas as funcionalidades requeridas para implementar todos os tipos de agentes e gerentes.

As funções XMP suportam os sete serviços CMIS e os quatro serviços oferecidos pelo SNMP, tanto no modo *requestor*, onde a aplicação solicita o serviço, quanto no modo *responder*, onde a aplicação recebe uma solicitação de seviço para ser executada. O mapeamento dos serviços oferecidos pelos protocolos de gerência nos serviços oferecidos pela API XMP são mostrados na Tabela 6-4.

As funções requisitantes têm o sufixo `_req()` no final, as funções do *responder* têm o sufixo `_rsp()` no final e só são usadas para responder o requisitante.

As funções dos serviços GET, SET e de EVENT REPORT aceitam parâmetros específicos para o protocolo que estar sendo utilizado, CMIP ou SNMP.

A Tabela 6-4 mostra o mapeamento dos serviços de Gerência (que podem ser serviços CMIS ou serviços SNMP) nos serviços da API XMP.

Serviço CMIS	**Serviço SNMP**	**Funções XMP**	**Descrição**
Action	-	`mp_action_req()` `mp_action_rsp()`	Requisita ao *responder* que execute uma das ações definidas para um objeto.
Cancel Get	-	`mp_cancel_get_req()` `mp_cancel_get_rsp()`	Requisita ao *responder* que termine um pedido de *get* antes que ele seja completado.
Create	-	`mp_create_req()` `mp_create_rsp()`	Requisita que o *responder*crie uma instância (objeto) de uma classe de objeto especificado.
Delete	-	`mp_delete_req()` `mp_delete_rsp()`	Requisita que o *responder* destrua uma instância (objeto) de uma classe de objeto especificado.
Get	Get	`mp_get_req()` `mp_get_rsp()`	Requisita que o *responder* forneça os valores de zero ou mais atributos de um objeto.
Set	Set	`mp_set_req()` `mp_set_rsp()`	Requisita que o *responder* modifique um ou mais valores de atributos de um objeto.
Event Report	Trap	`mp_event_report_req()`	Emite uma das notificações definidas para o objeto gerenciado.

		mp_event_report_rsp	
-	Get Next	mp_get_next_req() mp_get_next_rsp()	Requisita que o *responder* forneça o tipo e o valor da próxima variável SNMP do objeto.

Tabela 6-4 - Mapeamento dos serviços de Gerência nos serviços da API XMP

Além de prover uma interface para os serviços oferecidos pelos protocolos de gerência, a API XMP define uma série de serviços adicionais que são necessários para gerenciar alguns serviços de suporte, como por exemplo, a gerência de ambiente. Alguns destes serviços podem ser vistos na Tabela 6-5.

Funções XMP	**Descrição**
mp_bind()	Abre uma sessão com a infra-estrutura de cominicação, retornando um objeto OM MP_C_SESSION.
mp_initialize()	Inicializa um *workspace* XOM.
mp_shutdown()	Descarta o *workspace* XOM. Depois da chamada desta função o uso de qualquer função XMP é inválido.
mp_unbind()	Termina uma sessão aberta com a infra-estrutura de comunicação. Esta função não destrói o objeto OM MP_C_SESSION associado, mas ele fica inválido para uso nesta aplicação.
mp_negotiate()	Associa pacotes OM com o *workspace* da aplicação.

Tabela 6-5- Serviços adicionais da API XMP

CAPÍTULO 7

Resultados e Conclusões

7.1. Descrição dos testes

A seguir descreve-se como foram realizados os testes, apresentando-se também os resultados dos mesmos. Com os resultados obtidos, notou-se que alguns testes apresentaram certas anomalias. Estas anomalias (geralmente não-linearidades nas taxas de tratamento de mensagens em relação ao tamanho das mesmas) estão sendo objeto de estudos mais aprofundados para se verificar sua causa.

Deve-se ressaltar que os resultados aqui apresentados não tem o objetivo de comparação de desempenho de produtos comerciais, devendo ser utilizados apenas como parâmetros de referência na especificação e implementação de sistemas de gerência de redes de telecomunicações baseados em Plataformas TMN.

Nos testes descritos abaixo, foram realizadas trocas de mensagens do protocolo CMIP, de comprimento variando de 100 até o máximo de 64K - 512 bytes, por pares gerente/agente. Nos resultados apresentados, não foram mostrados os resultados obtidos para mensagens de comprimento menores que 1 kbyte, visto que os valores encontrados foram praticamente os mesmos que para as mensagens de 1 kbyte.

A comunicação entre agentes e gerentes neste trabalho deu-se apenas no modo síncrono. Todas as medidas foram realizadas com gerente e agente realizando transações CMIP de *m-get* síncronos e confirmados, conforme detalhado na Figura 7-1. Para o cálculo do número de transações executadas, gerente e agente executam um *loop* infinito, sendo interrompidos por um sinal do sistema operacional (temporização) para o fim do procedimento. Assim para obtenção das medidas foi desenvolvidos dois tipos de pares agentes/gerentes. O primeiro tipo de par agente/grente efetivamente fazia a medida, mandando pedidos de *m-get* e medindo o tempo que o agente levava para processar e responder. O segundo tipo de par agente/gerente executava um *loop* infinito sendo interrompidos por um sinal do sistema operacional para o fim do procedimento.

Testes com outros tipos de primitivas, por serem considerados de importância secundária poderão ser realizados posteriormente. Deve-se ressaltar que na configuração mostrada na Figura 7-1, gerentes e agentes podem estar em máquinas diferentes, sendo neste caso as Plataformas TMN interligadas pelos perfis de protocolos já mencionados anteriormente no Capítulo 3.

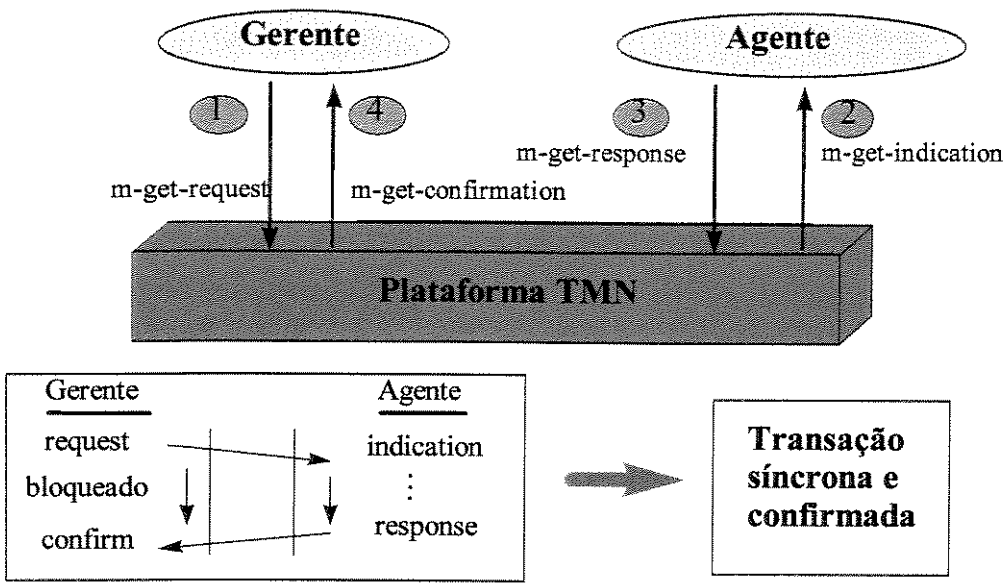

Figura 7-1- Implementação gerente-agente utilizada nos testes de desempenho

7.1.1 Desempenho em função da capacidade de processamento do servidor

Para se obter medidas do número de mensagens CMIP tratadas por um sistema de gerência em função da capacidade de processamento do servidor, foi implementado um teste onde uma aplicação gerente e uma aplicação agente se comunicam através da Plataforma TMN numa mesma máquina. Este teste foi feito em duas máquinas com capacidade de processamento distintas, a saber Urano e Pyxis (ver capacidade de processamento na Figura 6-2). Quando da realização dos testes, não havia outros processos sendo executados nas máquinas que não fossem essenciais ao sistema operacional e à Plataforma TMN. Alguns dos resultados obtidos são mostrados no Gráfico 1, 2 e 3.

O Gráfico 1 apresenta os resultados obtidos com apenas o par agente/gerente sendo executado localmente, isto é, o agente e gerente estavam sendo executados na mesma

máquina. Os Gráfico 2 e o Gráfico 3 representam os resultados obtidos com mais 5 e 10 pares de agentes/gerentes sendo executados em *loop* respectivamente. Pode-se perceber pela comparação entre estes 3 gráficos que eles apresentam o mesmo comportamento. A análise sobre o desempenho em função da variação de agentes a serem gerenciados é outro fator a ser analisado na próxima seção.

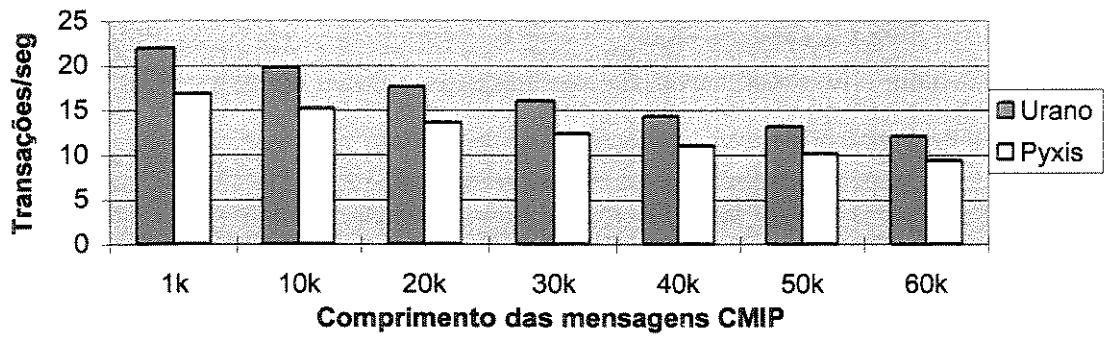

Gráfico 1: Desempenho de Plataformas TMN com gerente e agente em execução local

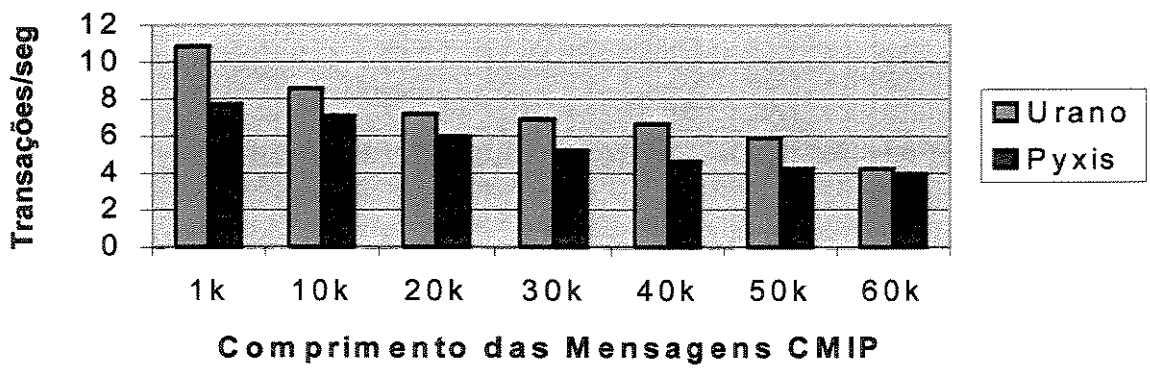

Gráfico 2: Desempenho de Plataformas TMN com gerente e agente em execução local com mais 5 pares de agentes/gerentes sendo executados localmente

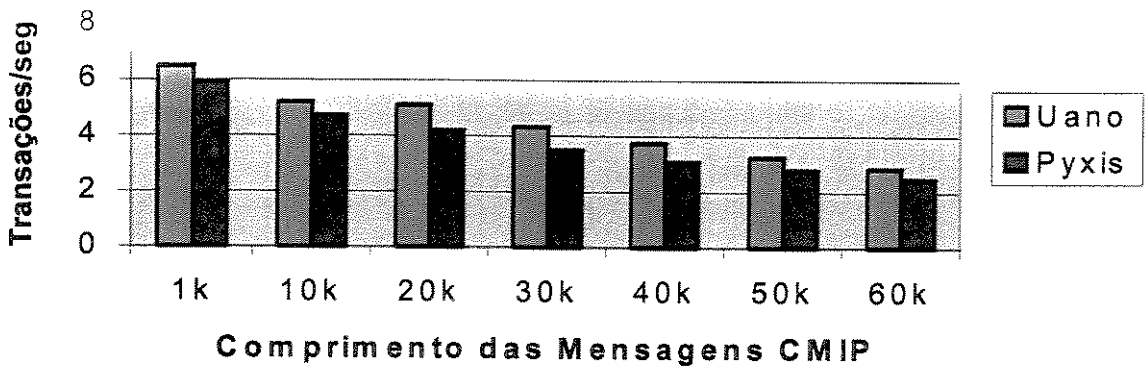

Gráfico 3: Desempenho de Plataformas TMN com gerente e agente em execução local com mais 10 pares de agentes/gerentes sendo executados localmente

Este teste mostrou uma relação linear entre a capacidade de processamento e o número de mensagens tratadas por uma máquina quando da execução local do gerente e do agente. Para a máquina Urano, que possui uma capacidade de processamento aproximadamente 25% superior à Pyxis, obtivemos um desempenho superior variando entre 30 e 35%. Deve-se notar, que mesmo com processos extremamente simples e mensagens pequenas, não foi possível obtermos desempenho superior a 25 transações por segundo, valor que decai para aproximadamente 12 transações por segundo quando da utilização de mensagens longas.

7.1.2 Desempenho em função da variação do número de agentes a serem gerenciados

Um outro fator importante a ser analisado, é o que diz respeito à concorrência de processamento entre vários agentes e gerentes numa mesma Plataforma TMN. Este tipo de configuração é de grande importância na competitividade de um sistema de gerência, pois através dessa aglutinação de agentes/gerentes, pode-se estabelecer uma arquitetura menos onerosa do ponto de vista econômico para a arquitetura física do sistema.

O objetivo deste teste era o de simular esta concorrência, através da inclusão paulatina de pares gerentes/agentes idênticos, executando troca de mensagens CMIP em *loop*, enquanto media-se o número de transações por segundo efetuada por um desses pares. A

Figura 7-2 mostra uma topologia genérica para este teste, com gerentes e agentes podendo estar em máquinas separadas ou permitindo qualquer outro tipo de variação.

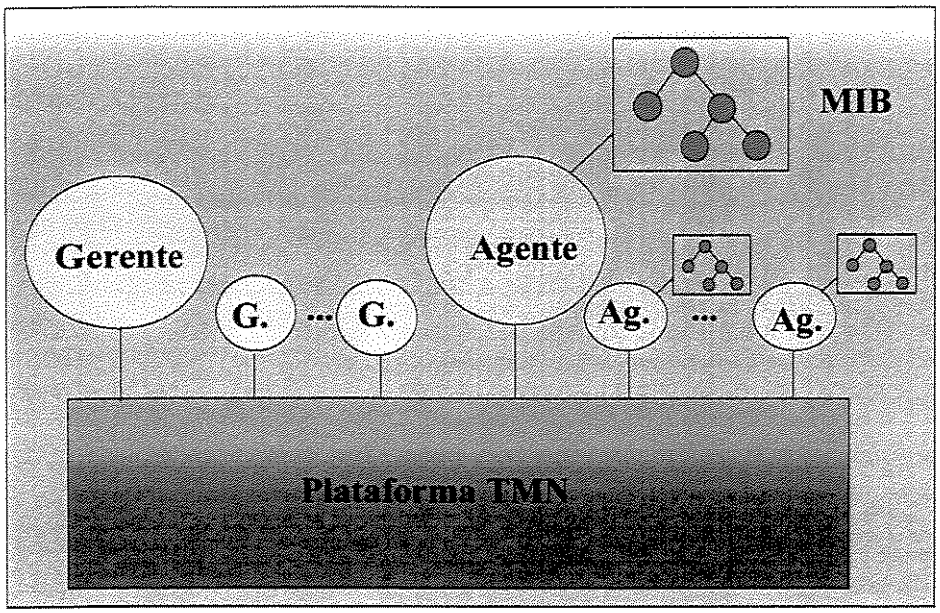

Figura 7-2 - Execução de múltiplos gerentes e agentes na Plataforma TMN

O Gráfico 4 e o Gráfico 5 mostram os resultados da execução do teste acima descrito na *Pyxis* e *Urano* respectivamente , em que gerentes e agentes foram executados localmente. Os resultados deste teste poderão ser usados como parâmetros máximos de desempenho quando do dimensionamento de dispositivos mediadores e de sistemas de adaptação baseados na arquitetura TMN. Nos gráficos seguintes, zero agentes/gerentes significa que não existe nenhum par agente/gerente em *loop*, apenas o par que faz a medição das transações por segundo trocadas. Neste gráfico também é possível observar que existe uma diminuição de 5 a 20 % no número de transações por segundo a medida que aumenta-se o tamanho das mensagens. Por exemplo, o número de transações por segundo para uma mensagem de 10K com zero agentes/gerentes sendo executados em *loop* é 10% menor que o número de transações por segundo para uma mensagem de 1K com zero agentes/gerentes sendo executados em *loop*. Apesar das diferenças de hardware entre as duas máquinas, elas apresentam o mesmo comportamento.

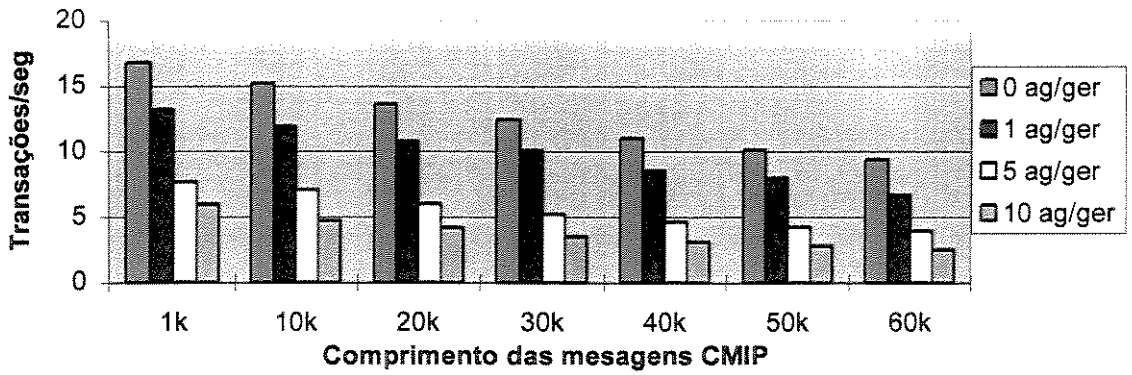

Gráfico 4: Desempenho de uma Plataforma TMN executando vários pares agente/gerente na Pyxis

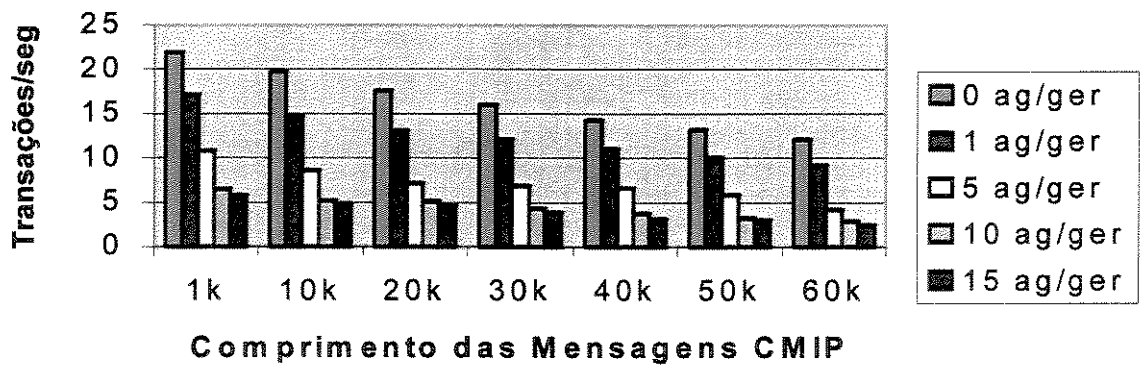

Gráfico 5: Desempenho de uma Plataforma TMN executando vários pares agente/gerente na Urano

7.1.3 Desempenho em função da execução local ou remota de agentes e gerentes

Este teste comparou o número de transações por segundo obtidos quando o agente e o gerente de teste executam numa mesma máquina (execução local) e quando executam em máquinas separadas (execução remota). Os resultados obtidos são mostrados no Gráfico 6. Analisando-se este gráfico, pode-se notar que foram colocados também os valores

obtidos para mensagens de comprimento menor que 1 kbyte. Isto se deve ao fato de haver uma não-linearidade nos resultados apresentados.

Pela análise dos resultados, concluí-se que para mensagens pequenas, a execução remota, gerente e agente executando em máquinas distintas, é mais eficiente do que a execução local. Com o aumento do comprimento da mensagem, a execução local tem melhor desempenho. Uma possível explicação para o fato é a influência da rede de transporte (no caso uma rede local) nos resultados. Para mensagens pequenas, o tempo para transmissão das mensagens é muito pequeno e como o tempo de processamento é dividido, isto ocasiona uma vantagem para o processamento remoto. Para mensagens grandes, o tempo de transmissão torna-se significativo, eliminando a vantagem da distribuição de processamento anteriormente obtida.

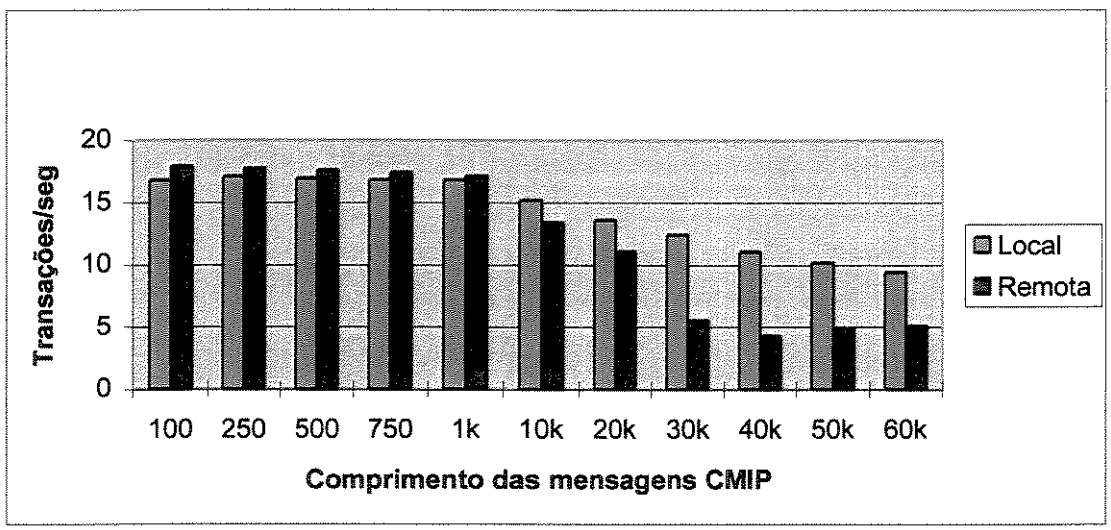

Gráfico 6: Desempenho de uma Plataforma TMN executando pares agente/gerente local e remotamente

O gráfico 7 mostra a comparação entre execução remota quando dez pares agente/gerentes em *loop* estavam sendo executados localmente, primeiro na máquina do gerente (Urano) depois na máquina onde estava sendo executado o agente (Argos). Percebe-se que existe uma diferença em torno de 5% nas Transações por segundo que as máquinas podem processar. O resultado da máquina onde está sendo executado o agente foi inferior ao da máquina onde estava sendo executado o gerente, se outros pares agente/gerentes estiverem sendo executados, porque devido aos outros processos em execução o agente leva mais tempo para processar o *m-get* do gerente e respondê-lo. As máquinas Urano e Argos tem a mesma configuração de *hardware*.

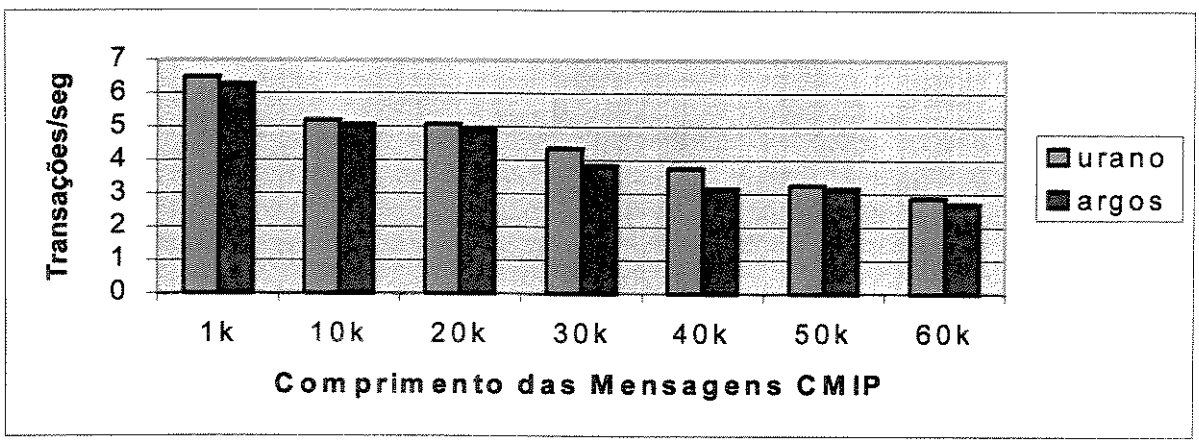

Gráfico 7: Desempenho de uma Plataforma com execução remota quando mais 10 pares agente/gerente estão sendo executados localmente

7.2. Conclusões

Este trabalho apresenta os primeiros resultados obtidos com a atividade de avaliação de desempenho, portabilidade e interoperabilidade de Plataformas TMN. Pretende-se com esse trabalho tornar mais claro às empresas operadoras de telecomunicações quais são os fatores a serem abordados quando da especificação de sistemas de gerência baseados na arquitetura TMN. Espera-se também uma cooperação com os fabricantes de Plataformas TMN para que o restante do trabalho possa ser realizado, visto que seus resultados

podem ser utilizados para aprimorar os requisitos de sistemas de gerência que venham a ser desenvolvido no país nos próximos anos.

Atualmente uma série de outros fatores que afetam o desempenho dos sistemas estão sendo investigados, tais como a utilização de API padronizadas, o uso ou não de procedimentos automáticos para estabelecimento de associação de gerência, e etc. Porém, espera-se que os resultados aqui apresentados já possam ser considerados como uma contribuição para a área de gerência de redes de telecomunicações no Brasil, visto ser esta uma atividade efervescente e que necessita de trabalhos que abordem também aspectos quantitativos da implementação de sistemas utilizando ferramentas atuais.

Com este trabalho pode-se comprovar que os parâmetros abaixo afetam de forma significativa o desempenho das plataformas:

- tamanho de mensagens afeta de maneira linear, isto é, à medida que aumentamos o tamanho das mensagens diminuímos linearmente o número de transações por segundo entre o agente e gerente.
- o processamento local ou remoto dos agentes/gerentes para mensagens menores que 1 KByte tem melhor desempenho se o agente e o gerente estiverem remotos, e para mensagens maiores que 1 KByte o número de transações por segundo que podem ser processadas é maior se o agente e gerente estiverem na mesma máquina. A diferença de processamento apresentou um comportamento não linear.
- capacidade de processamento das *workstations*: foi verificado que não existe uma relação constante entre as capacidades de processamento das máquinas mediante diferentes tamanho de mensagens. Embora esperássemos uma relação constante, o fato de possuírem memórias e caches diferentes mascararam esta medida.

Em relação à portabilidade, observou-se que apesar de existir normas com as definições das APIs, normas para definição das interfaces, ainda existe flexibilidade nas forma de implementação destas APIs pelos fabricantes, o que pode acarretar em diferenças de implementação. Também existe liberdade para implementação de novas funções que tornam-se proprietárias de cada fabricante. E ainda podem existir diferentes versões para

as APIs que estão disponíveis nas plataformas de diferentes fabricantes. Assim para que o mesmo código funcione em diferentes plataformas é necessário checar como cada função está implementada e a existência de cada função nos diferentes fabricantes. Sendo assim, em alguns casos, perde-se a vantagem de implementação de aplicativos (agentes/gerentes) utilizando as ferramentas gráficas disponibilizadas por alguns fabricantes, uma vez que, para utilização das ferramentas gráficas o desenvolvedor dos aplicativos perde o acesso ao código fonte, só tendo acesso ao código executável.

Para o caso da interoperabilidade entre diferentes plataformas, é necessário um conhecimento profundo do funcionamento dos protocolos utilizados e suportados em cada plataforma, das interfaces utilizadas e suportadas em cada plataforma, como deve ser feita a configuração da pilha de protocolos em cada plataforma, e como os agentes e seus objetos devem ser configurados de tal forma que possam ser acessados não só localmente como remotamente pelos gerentes. As pilhas suportadas por cada fabricante podem ser diferentes e a forma como agentes e seus objetos devem ser configurados na plataforma para que possam ser acessados localmente ou remotamente varia para cada fabricante.

Um próximo passo a ser dado após o encerramento deste trabalho é o levantamento do perfil típico de mensagens de gerência de algumas tecnologias de telecomunicações (número típico de alarmes, comprimento de mensagens, etc). Com esses dados, mais os que estão sendo levantados em relação às Plataformas TMN, espera-se poder estabelecer uma arquitetura física otimizada para a implementação de dispositivos mediadores e de sistemas de gerência para as empresas de telecomunicações.

REFERÊNCIAS

[Plat93] Fayan, B. L.; Imai, C. T.; Faber, M. B.; Lorena, P. S. ; Plataforma de Suporte a Aplicações de Gerência - Revista Telebrás Tecnologia, nº 59, vol.17. Dez. 1994.

[OSF] OSF - Open Software Foundation Distrubuted computing Emviroment - DCE - Features and Funcionality.

[Brisa] BRISA (Sociedade Brasileira para interconexão de Sistemas Abertos), Gerenciamento de Redes - Uma abordagem de sistemas abertos, 1993. Makron Books.

[X.208] Recomendação X.208 - ITU-T, Specification of Abstract Syntax Notation (ASN.1).

[X.711] Recomendação X.711(1993) - ITU-T,"Common Management Information Protocol Specification for CCITT Applications".

[X.720] Recomendação X.720(1992) - ISO/IEC 10165-1: 1992, Information Technology - Open System Interconnection - Structure of Management Information - Management Information Model.

[X.721] Recomendação X.721 (1992) - ISO/IEC 10165-2: 1992, Information Technology - Open System Interconnection - Structure of Management Information: Definition of Management Information.

[X.722] Recomendação X.722 (1992) - ISO/IEC 10165-4: 1992, Information Technology - Open System Interconnection - Structure of Management Information: Guidelines for the definition of Managed Object.

[M3010] Recomendação M3010, Principles for a Telecommunications Management Network.

[M3100] Recomendação M3010, Principles for a Telecommunications Management Network - Generic Network Information Model.

[DevGui] HP 9000 Series and Sun Systems, Manual Part Number: J1064-90009, HP Open View Distributed Management Developer's Guide, Setembro 1994.

[AdmRef] HP 9000 Series and Sun Systems, Manual Part Number: J1064-90009, HP Open View Distributed Management Agent Plataform Administrator's Reference, Setembro 1994.

[Q.811] ITU-TSS Recomendation Q.811 - Protocols for lower layers of Q3 interface

[NMF] NETWORK MANAGEMENT FORUM. RGB Management API. Issue 1.0 Draft 1.

[NecSis] Perotoni, I.A; Simons H.; A Necessidade De Sistemas de Supervisão e Gerência de Rede nas empresas Operadoras de Telecomunicações. Revista TELEBRÁS, páginas 56-61, dezembro de 1993.

[Tirib] Tiribelli, P. Information technology and telecommunications business: integrated operations support systems in Italy. Sip headquartes, Roma, 1991.

ANEXOS

ANEXO.1 Código do programa do agente

Este item apresenta o código do programa do agente que foi utilizado nas medidas das transações por segundo efetuadas nas medidas de desempenho. A linguagem de programação utilizada foi o ANSI C em conjunto com as APIs XOM/XMP. Este código está portável para as duas plataformas utilizadas no desenvolvimento deste trabalho.

ANEXO.2 Código do programa do gerente

Este item apresenta o código do programa do gerente que foi utilizado nas medidas das transações por segundo efetuadas nas medidas de desempenho. A linguagem de programação utilizada foi o ANSI C em conjunto com as APIs XOM/XMP. Este código está portável para as duas plataformas utilizadas no desenvolvimento deste trabalho.

ANEXO 1

CÓDIGO DO PROGRAMA DO AGENTE

```
/*****************************************************************
 * @(#) agent.c                                                  *
 *  Flavia A. Schneider                                          *
 *                                                               *
 *****************************************************************/

#include <stdio.h>
#include <xom.h>
#include <xmp.h>
#include <xmp_cmis.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <OV/ove_xmp.h>       /* na manpage do om_create fala para
                              incluir este arquivo */
#include <OV/ov_types.h>
#include <time.h>
#include <OV/at_gentime.h>

#define  ERROR_BUFLEN 128
#define  OID_LEN 128
#define  NUMGET 150

        /*****************************************************************
        *                                                                *
        * Exporta todas classes e pacotes requeridos por este agente.    *
        *                                                                *
        *****************************************************************/

#ifdef _IBM_SOURCE
#ifdef __alpha
OM_IMPORT(MP_C_ATTRIBUTE_ID)
OM_IMPORT(MP_C_ATTRIBUTE)
OM_IMPORT(MP_C_AVA)
OM_IMPORT(MP_C_CONTEXT)
OM_IMPORT(MP_C_DS_DN)
OM_IMPORT(MP_C_DS_RDN)
OM_IMPORT(MP_C_OBJECT_CLASS)
OM_IMPORT(MP_C_OBJECT_INSTANCE)
#else
OM_EXPORT(MP_C_ATTRIBUTE_ID)
OM_EXPORT(MP_C_ATTRIBUTE)
OM_EXPORT(MP_C_AVA)
OM_EXPORT(MP_C_CONTEXT)
OM_EXPORT(MP_C_DS_DN)
OM_EXPORT(MP_C_DS_RDN)
OM_EXPORT(MP_C_OBJECT_CLASS)
OM_EXPORT(MP_C_OBJECT_INSTANCE)
#endif      /*__alpha */

OM_EXPORT(MP_CMIS_PKG)
OM_EXPORT(MP_C_CMIS_GET_ARGUMENT)
OM_EXPORT(MP_C_CMIS_GET_RESULT)
OM_EXPORT(MP_C_CMIS_SERVICE_ERROR)
OM_EXPORT(MP_C_FORMI)
OM_EXPORT(MP_C_SESSION)
OM_EXPORT(OM_C_OBJECT)
OM_EXPORT(MP_C_ENTITY_NAME)
OM_EXPORT(OM_BER)
```

```
#endif /*ibm_source */

#ifdef _HPUX_SOURCE
OM_EXPORT(MP_CMIS_PKG)
OM_EXPORT(MP_C_CONTEXT)
OM_EXPORT(MP_C_SESSION)
OM_EXPORT(MP_C_CMIS_GET_RESULT)
OM_EXPORT(MP_C_ENTITY_NAME)
OM_EXPORT(MP_C_ATTRIBUTE)
OM_EXPORT(MP_C_ATTRIBUTE_ID)
#endif /*_hpux_source */

void preencherMens(int argc,char **argv);    /* funcao para preencher mensagem
                                                com tamanho vari'avel*/

        /*****************************************************************
        *                                                                *
        * A feature list structure contem pacotes e caracteristicas      *
        * que vao ser negociadas no workspace.  Elas podem ser           *
        * inicializadas durante a fase de execucao.                      *
        *                                                                *
        *****************************************************************/
#ifdef _IBM_SOURCE
static MP_feature feature_list[] =
{
    { {0,0},/* 0,*/ OM_FALSE },
    { {0,0},/* 0,*/ OM_FALSE },
    { {0,0},/* 0,*/ OM_FALSE }
};
#endif /*_ibm_source */

#ifdef _HPUX_SOURCE
static MP_feature feature_list[] =
{
    { {0,0}, 0, OM_FALSE },
    { {0,0}, 0, OM_FALSE },
    { {0,0}, 0, OM_FALSE }
};
#endif /*_hpux_source */

static OM_descriptor entity_name[] = {
      OM_OID_DESC(OM_CLASS, MP_C_ENTITY_NAME),
      {MP_ENTITY, OM_S_PRINTABLE_STRING, { 14,"object_mansinc" }}, /*nome do
agente*/
      OM_NULL_DESCRIPTOR };

#ifdef _IBM_SOURCE
static OM_descriptor session[] = {
      OM_OID_DESC(OM_CLASS, MP_C_SESSION),
      {MP_ROLE, OM_S_INTEGER, {{MP_T_PERFORMING|MP_T_PERFORMING}} },
      /*{MP_FILE_DESCRIPTOR, OM_S_INTEGER, }, na hp este parametro 'e
obrigatorio*/
      {MP_REQUESTOR_TITLE, OM_S_OBJECT, { 0, entity_name }},
      OM_NULL_DESCRIPTOR };

static OM_descriptor                attribute_idibm[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0,0} },
    OM_NULL_DESCRIPTOR,
```

```
};
#endif /*ibm_source */

#ifdef _HPUX_SOURCE
static OM_descriptor session[] = {
      OM_OID_DESC(OM_CLASS, MP_C_SESSION),
      {MP_FILE_DESCRIPTOR, OM_S_INTEGER, },
      {MP_REQUESTOR_TITLE, OM_S_OBJECT, { 0, entity_name }},
      OM_NULL_DESCRIPTOR };

/* esta' sendo utilizado para passar a mensagem de tamanho vari'avel */
static OM_descriptor                attribute_idmen[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
};
#endif

static OM_descriptor                attribute_id1[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
};

static OM_descriptor                attribute_id2[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
};

static OM_descriptor                attribute_id3[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
};

static OM_descriptor                attribute_id4[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
};

#ifdef _IBM_SOURCE
static OM_descriptor                attributeIbm[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE),
    { MP_ATTRIBUTE_ID, OM_S_OBJECT, {0, attribute_idibm} },
    /*{ MP_ATTRIBUTE_VALUE, OM_S_INTEGER,  0 },*/
        { MP_ATTRIBUTE_VALUE, OM_S_VISIBLE_STRING, {0,0}/* OM_STRING("Mickey
Mouse")*/ },
    OM_NULL_DESCRIPTOR,
};

#endif /*_IBM_SOURCE */

#ifdef _HPUX_SOURCE
/*estou colocando este atributo para passar uma mensagem de tamanho
  variavel para o gerente     */
static OM_descriptor                attributeMen[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE),
    { MP_ATTRIBUTE_ID, OM_S_OBJECT, {0, attribute_idmen} },
```

```
    { MP_ATTRIBUTE_VALUE, OM_S_VISIBLE_STRING, /*OM_STRING("MENSAGEM")*/ {0, 0}
},
    OM_NULL_DESCRIPTOR,
};
#endif
/*
static OM_descriptor                attribute1[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE),
    { MP_ATTRIBUTE_ID, OM_S_OBJECT, {0, attribute_id1} },
    { MP_ATTRIBUTE_VALUE, OM_S_INTEGER, 25 },
    OM_NULL_DESCRIPTOR,
};

static OM_descriptor                attribute2[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE),
    { MP_ATTRIBUTE_ID, OM_S_OBJECT, {0, attribute_id2} },
    { MP_ATTRIBUTE_VALUE, OM_S_VISIBLE_STRING, OM_STRING("Mickey Mouse") },
    OM_NULL_DESCRIPTOR,
};

static OM_descriptor                attribute3[] = {
    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE),
    { MP_ATTRIBUTE_ID, OM_S_OBJECT, {0, attribute_id3} },
    { MP_ATTRIBUTE_VALUE, OM_S_ENUMERATION, 1 },
    OM_NULL_DESCRIPTOR,
};
*/
#ifdef _IBM_SOURCE
static OM_descriptor                cmis_get_result[] = {
    OM_OID_DESC(OM_CLASS, MP_C_CMIS_GET_RESULT ),
    { MP_MANAGED_OBJECT_CLASS, OM_S_OBJECT, { {0, NULL} } },
    { MP_MANAGED_OBJECT_INSTANCE, OM_S_OBJECT, { {0, NULL} } },
    /*     { MP_CURRENT_TIME, OM_S_GENERALISED_TIME_STRING,{ {0, 0} } }, hp    */
    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, { {0, attributeIbm} } },
    /*{ MP_ATTRIBUTE_LIST, OM_S_OBJECT, { {0, attributeMen} } },*/  /*
MP_C_ATTRIBUTE */
    /*{ MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute1} },  */ /*MP_C_ATTRIBUTE
*/
    /*      { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute2} },  MP_C_ATTRIBUTE
*/
/*    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute3} },   MP_C_ATTRIBUTE */
/*  { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute4} },   * MP_C_ATTRIBUTE */
    OM_NULL_DESCRIPTOR,
};
#endif /*_ibm_source */

#ifdef _HPUX_SOURCE
static OM_descriptor                cmis_get_result[] = {
    OM_OID_DESC(OM_CLASS, MP_C_CMIS_GET_RESULT ),
    { MP_MANAGED_OBJECT_CLASS, OM_S_OBJECT, { {0, NULL} } },
    { MP_MANAGED_OBJECT_INSTANCE, OM_S_OBJECT, { {0, NULL} } },
    { MP_CURRENT_TIME, OM_S_GENERALISED_TIME_STRING,{ {0, 0} } },
    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, { {0, attributeMen} } },  /* MP_C_ATTRIBUTE
*/
/*    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute1} },   MP_C_ATTRIBUTE */
/*    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute2} },  MP_C_ATTRIBUTE */
/*    { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute3} },   MP_C_ATTRIBUTE */
/*  { MP_ATTRIBUTE_LIST, OM_S_OBJECT, {0, attribute4} },   * MP_C_ATTRIBUTE */
    OM_NULL_DESCRIPTOR,
};
```

```
#endif /*_hpux_source */


/*variavel utilizada para preenchimento da mensagem*/
char *vetMens;

int   main(int argc,  char *argv[])
{
OM_return_code          om_ret;
OM_workspace            workspace;
MP_status               mp_status;
/* DECLARE VARIABLES HERE */
char                    oid_string[OID_LEN];
char                    err_buf[ERROR_BUFLEN];
OM_private_object bound_session;
OM_private_object get_result;
OM_private_object       priv_context/*[NUMGET]*/;
OM_private_object       request;
OM_public_object        response, obj;
OM_sint                 completion_flag_return;
OM_sint                     primitive_return;
OM_sint32               invoke_id_return;
OM_string          gen_time;
OM_object_identifier    oid_oid;
OM_value_position       num;
MP_status               operation_notification_status;
int                i,cont=0;

#ifdef _HPUX_SOURCE
MP_waiting_sessions     session_list[2];
#endif /* hpux_source */

/*declaracao de variaveis necessarias para utilizacao das funcoes xom/xmp da
ibm*/
#ifdef _IBM_SOURCE
OM_sint                 mode;       /*utilizada no mp_revceive */
OM_private_object get_result_ibm;  /*objeto criado com copia do get_arg para ser
passado pelo
                                mp_get_rsp */
MP_Waiting_Sessions     session_list[2];
#endif /*ibm_source */

      if(!argc){
        fprintf(stderr,"argc=0\n",*argv);
        exit(-1);
      }
        /*****************************************************************
        *                                                                *
        * A API mp_initialize() 'e utilizada para inicializar o workspace.*
        * se o workspace 'e NULL, deve-se sair do programa com uma       *
        * mensagem de erro, pois nada mais pode ser feito.          *
        *                                                                *
        *****************************************************************/

      workspace = mp_initialize();

      if (workspace == NULL) {
          printf ("mp_initialize() failed \n");
          return(-1);
      }
```

```
        /*****************************************************************
        *                                                                *
        *  Vai ser testado se o pacote CMIS 'e suportado com o           *
        *  mp_negotiate(). Mas primeiro deve-se setar a primeira         *
        *  feauture da lista de features para especicar o pacote cmis    *
        *  antes de fazer a chamada. Depois da chamada testa-se          *
        *  o valor do campo ativado para verificar se o pacote 'e        *
        *  suportado.                                                    *
        *                                                                *
        *****************************************************************/
#ifdef _IBM_SOURCE
      feature_list[0].feature = MP_CMIS_PKG;
        feature_list[0].activated = OM_FALSE ; /*request = MP_ACTIVATE*/

        mp_status = mp_version (workspace, feature_list);
       /*mp_negotiate(feature_list,workspace);*/
#endif /*_ibm_source */

#ifdef _HPUX_SOURCE
      feature_list[0].feature = MP_CMIS_PKG;
        feature_list[0].request = MP_ACTIVATE;

        mp_status = mp_negotiate (feature_list, workspace);
#endif /*_hpux_source */

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_version/negotiate() failed: %s \n", err_buf);
          mp_shutdown(workspace);   /* ibm ?? */
          return(-1);
      }

#ifdef _HPUX_SOURCE
      /* para HP */
      if (!feature_list[0].response) {
          printf("CMIS Package not negotiated successfully\n");
          return(-1);
      }

#endif /*_hpux_source */

#ifdef _IBM_SOURCE
      if(!feature_list[0].activated){
        printf("cmis_pkg not available\n");
        mp_shutdown(workspace);     /* ibm  */
        return(-1);
      }
#endif /*_ibm_source */
```

```
  /*****************************************************************
  *                                                                *
  * Use o comando mp_bind() para ligar ao o  postmaster. Como      *
  * este e' o agente, pode-se definir uma sessao de objeto         *
  * estaticamente com um requestor title. Agentes devem avisar a   *
  * CI seu nome.                                                   *
  *                                                                *
  *****************************************************************/

mp_status = mp_bind(session, workspace, &bound_session);

if (mp_status != MP_SUCCESS) {
    if (mp_status == MP_NO_WORKSPACE)
      strcpy (err_buf, "MP_NO_WORKSPACE");
    else if (mp_status == MP_INVALID_SESSION)
      strcpy (err_buf, "MP_INVALID_SESSION");
    else if (mp_status == MP_INSUFFICIENT_RESOURCES)
      strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
    else {
      mp_error_message (mp_status, ERROR_BUFLEN,
          (unsigned char *) &err_buf);
      (void)om_delete(mp_status);
    }
    printf("mp_bind() failed: %s \n", err_buf);
    mp_shutdown(workspace);   /* ibm ?? */
    return(-1);
}

  /*****************************************************************
  *                                                                *
  * Um ponteiro para o obeto bound session deve ser colocado no    *
  * primeri slot do session_list.  O segundo slot e' inititalizado*
  * com NULL.  O timeout e' configurado para 0 na chamada do       *
  * mp_wait, assim ficara  bloqueado ate chegar alguma coisa       *
  *                                                                *
  *****************************************************************/

session_list[0].bound_session = bound_session;
session_list[0].activated = OM_FALSE;
session_list[1].bound_session = NULL;

/*vai esperar NUMGET pedidos de get's do gerente, mas um de cada vez */
for(cont=0;cont<NUMGET;cont++){
if(!(cont%50))
  printf("\nWaiting .....,  cont= %d\n",cont);

mp_status = mp_wait(session_list, workspace, 0);

if (mp_status != MP_SUCCESS) {
    if (mp_status == MP_NO_WORKSPACE)
      strcpy (err_buf, "MP_NO_WORKSPACE");
    else if (mp_status == MP_INVALID_SESSION)
      strcpy (err_buf, "MP_INVALID_SESSION");
    else if (mp_status == MP_INSUFFICIENT_RESOURCES)
      strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
    else {
      mp_error_message (mp_status, ERROR_BUFLEN,
          (unsigned char *) &err_buf);
      (void)om_delete(mp_status);
    }
```

```
            printf("mp_wait() failed: %s \n", err_buf);
            mp_unbind(session);
            om_delete(session);
            mp_shutdown(workspace);
            return(-1);
        }


          /*****************************************************************
          *                                                                *
          * O mp_receive() requer um objeto de context.  Um objeto de      *
          * context sera criado e initializado .                           *
*
          *                                                                *
          *****************************************************************/
#ifdef _HPUX_SOURCE
        om_ret = om_create( MP_C_CONTEXT,
                            OM_TRUE,
                            workspace,
                            &priv_context/*[cont]*/ );
        if (om_ret != OM_SUCCESS) {
            printf ("om_create() failed: %d /n", om_ret);
            mp_unbind(session);
            om_delete(session);
            mp_shutdown(workspace);
            return(-1);
          }
#endif /*hpux_source */

          /*****************************************************************
          *                                                                *
          * Chegou alguma coisa no bound session, assim o mp_receive deve *
          * ser chamado para esvaziar o socket buffer.                     *
          *                                                                *
          *****************************************************************/
        /*printf("antes do mp_receive\n");*/

#ifdef _IBM_SOURCE
        mp_status = mp_receive( bound_session,
                            /*nao existe este parametro na ibm:
priv_context*//*[cont],*/
                            &primitive_return,
                            &mode,
                                  &completion_flag_return,
                            &operation_notification_status,
                            &request,
                            &invoke_id_return );
#endif /*_ibm_source */

#ifdef _HPUX_SOURCE
          mp_status = mp_receive( bound_session,
                                  priv_context/*[cont]*/,
                                  &primitive_return,
                                  &completion_flag_return,
                                  &operation_notification_status,
                                  &request,
                                  &invoke_id_return );

#endif /*_hpux_source */
```

```
      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_receive() failed: %s \n", err_buf);
          mp_unbind(session);
          om_delete(session);
          mp_shutdown(workspace);
          return(-1);
      }


        /*****************************************************************
        *                                                                *
        * omX_print o request.                                           *
        *                                                                *
        *****************************************************************/
#ifdef _HPUX_SOURCE
/*    omX_print( stdout, "REQUEST", workspace, request );*/
#endif /*hpux_source */


#ifdef NOTDEF
      om_ret = om_create( MP_C_CMIS_GET_RESULT,
                          OM_FALSE,
                          workspace,
                          &get_result );

        if (om_ret != OM_SUCCESS) {
            printf("om_create of result object failed: om_ret = %d\n",
                   om_ret);
            exit(1);
        }

      omX_print(stdout, "RESULT1", workspace, get_result);
#endif /* NOTDEF */

#ifdef _IBM_SOURCE
      om_ret = om_get(request,(OM_exclusions) 0 /*OM_NO_EXCLUSIONS*/, NULL,
OM_FALSE, 0, 0,
                      &response, &num);

        /*printf("num=%d\n");*/

      om_ret =  om_create(MP_C_CMIS_GET_RESULT, OM_TRUE, workspace,
&get_result_ibm);
        if (om_ret != OM_SUCCESS) {
            printf("om_create of get_arg_ibm object failed: om_ret = %d\n",
                   om_ret);
          mp_unbind(session);
          om_delete(session);
          mp_shutdown(workspace);
```

```
          exit(1);
        }
#endif /* ibm_source */

#ifdef _HPUX_SOURCE
        om_get(request, OM_NO_EXCLUSIONS, NULL, OM_FALSE, 0, 0,
               &response, &num);

#endif /*hpux_source */

      for (i=1; i < num; i++) {
            obj = response[i].value.object.object;

            switch(response[i].type) {   /* mudei o acesso de . para -> */
            case MP_BASE_MANAGED_OBJECT_CLASS:
                  cmis_get_result[1].value.object.object = obj;
                  /*printf("case object_class\n");*/
                  break;
            case MP_BASE_MANAGED_OBJECT_INSTANCE:
                  cmis_get_result[2].value.object.object = obj;
                  /*printf("case object_instance\n"); */
                  break;
            }
      }
#ifdef _HPUX_SOURCE
      /* o compilador nao consegue encontrar em nenhuma biblioteca na ibm, o q e'
mais
         interressante e' q existe a biblioca */
      om_ret = at_gmtime_to_gentime(gmtime(NULL), &gen_time);
      cmis_get_result[3].value.string.length = gen_time.length;
      cmis_get_result[3].value.string.elements = gen_time.elements;

      if(!cont)
        preencherMens(argc,argv);

#endif /* _hpuxsource */

#ifdef _IBM_SOURCE
      /*vou preencher o atributo ...*/
        if (!strcpy(oid_string, "1.2.76.1.1.1.1.1.2.1"))
            printf("strcpy failed");
        om_ret  = at_str_to_oid (oid_string, &oid_oid);
        if (om_ret != OM_SUCCESS) {
            printf ("at_str_to_oid() failed: %d for object class.\n",
                om_ret);
            return(-1);
        }


      /*attributeIbm[2].value.integer = 9;*/
      if(!cont){
              attribute_idibm[1].value.string.length = oid_oid.length;
                attribute_idibm[1].value.string.elements = oid_oid.elements;

        preencherMens(argc,argv);
      }
      om_ret = om_put(get_result_ibm, OM_REPLACE_ALL, cmis_get_result, NULL,
(OM_value_position) 0,
                  (OM_value_position) 0);
        if (om_ret != OM_SUCCESS) {
```

```
            printf("om_put of get_result_ibm object failed: om_ret = %d\n",
                    om_ret);
            mp_unbind(session);
            om_delete(session);
            mp_shutdown(workspace);
            exit(1);
        }

#endif


    /*          if (!strcpy(oid_string, "1.3.6.1.4.1.11.1.7.1.3.2.1"))
            printf("strcpy failed");
        om_ret  = at_str_to_oid (oid_string, &oid_oid);
        if (om_ret != OM_SUCCESS) {
            printf ("at_str_to_oid() failed: %d for object class.\n",
                om_ret);
            return(-1);
        }

        attribute_id1[1].value.string.length = oid_oid.length;
        attribute_id1[1].value.string.elements = oid_oid.elements;

        if (!strcpy(oid_string, "1.3.6.1.4.1.11.1.7.1.3.2.1"))
            printf("strcpy failed");
        om_ret  = at_str_to_oid (oid_string, &oid_oid);
        if (om_ret != OM_SUCCESS) {
            printf ("at_str_to_oid() failed: %d for object class.\n",
                om_ret);
            return(-1);
        }

        if (!strcpy(oid_string, "1.3.6.1.4.1.11.1.7.1.3.2.6"))
            printf("strcpy failed");
        om_ret  = at_str_to_oid (oid_string, &oid_oid);
        if (om_ret != OM_SUCCESS) {
            printf ("at_str_to_oid() failed: %d for object class.\n",
                om_ret);
            return(-1);
        }

        attribute_id2[1].value.string.length = oid_oid.length;
        attribute_id2[1].value.string.elements = oid_oid.elements;

        if (!strcpy(oid_string, "1.3.6.1.4.1.11.1.7.1.3.2.3"))
            printf("strcpy failed");
        om_ret  = at_str_to_oid (oid_string, &oid_oid);
        if (om_ret != OM_SUCCESS) {
            printf ("at_str_to_oid() failed: %d for object class.\n",
                om_ret);
            return(-1);
        }

        attribute_id3[1].value.string.length = oid_oid.length;
        attribute_id3[1].value.string.elements = oid_oid.elements;
    */
    /*    if(cont ==9){
        omX_print(stdout, "RESULT1", workspace, cmis_get_result);
    }*/
```

```
        /*****************************************************************
        *                                                                *
        * execute o mp_get_rsp() usando a sessao, context e o            *
        * cmis get result.                                               *
        *                                                                *
        *****************************************************************/
      /*printf("vou mandar a resposta\n");*/

#ifdef _IBM_SOURCE
      mp_status = mp_get_rsp( bound_session,
                        MP_DEFAULT_CONTEXT,
                        cmis_get_result,
                        invoke_id_return);
#endif /* IBM_SOURCE */

#ifdef _HPUX_SOURCE
      mp_status = mp_get_rsp( bound_session,
                        priv_context,
                        cmis_get_result,
                        invoke_id_return);
#endif /* hpux_source */

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_get_rsp() failed: %s \n", err_buf);
          mp_unbind(session);
          om_delete(session);
          mp_shutdown(workspace);   /* ibm  */
          return(-1);
      }

      /*printf("sai do mp_get_rsp \n");*/

#ifdef _HPUX_SOURCE
      om_delete(priv_context);
      om_delete(cmis_get_result);
      om_delete(request);
#endif /*HPUX_SOURCE*/

#ifdef _IBM_SOURCE
      om_delete(get_result_ibm);
      om_delete(cmis_get_result);
      om_delete(request);
      om_delete(obj);
#endif

      }/*fim do for para esperar NUMGET get's */

        /*****************************************************************
        *                                                                *
```

```
        * Unbind do postmaster.                                          *
        *                                                                 *
        ******************************************************************/

      mp_status = mp_unbind(bound_session);

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_unbind() failed: %s \n", err_buf);
          return(-1);
      }


        /*****************************************************************
        *                                                                 *
        * Liberacao do context e objeto da sessao.  Usar                  *
        * om_delete.                                                      *
        *                                                                 *
        ******************************************************************/

      /*for(cont = 0; cont < NUMGET ; cont++){
        om_delete(priv_context[cont]);
      }*/
      free(vetMens);
      om_delete(bound_session);
      /*    om_delete(get_result);
      om_delete(request);
      */
        /*****************************************************************
        *                                                                 *
        * Liberacao do workspace com o comando mp_shutdown                *
        *                                                                 *
        ******************************************************************/
      mp_shutdown(workspace);

      return(0);
}

void preencherMens(int argc, char **argv){

  int tamMens=100;
  int i=0;
  /*  char *vetMens;*/

  tamMens = (argc-1)? atoi(argv[1]):100;
  /*printf("\ntamanho da mensagem 'e %d",tamMens);*/

  vetMens=(char *)malloc(tamMens*sizeof(char));
  for(i=0; i<tamMens; i++)
    vetMens[i]='m';
```

```
  vetMens[i]='\0';

#ifdef _HPUX_SOURCE
  attributeMen[2].value.string.length=tamMens;
  attributeMen[2].value.string.elements=(char *)vetMens;
#endif

#ifdef _IBM_SOURCE
  attributeIbm[2].value.string.length=tamMens;
  attributeIbm[2].value.string.elements=(char *)vetMens;
  printf("\nelements[%d]= %s\n",i, attributeIbm[2].value.string.elements);
#endif


}
```

ANEXO 2

CÓDIGO DO PROGRAMA DO GERENTE

```
/*************************************************************************
 * @(#) Este prog faz get's, mede o tempo gasto e faz a media           *
 *    Flavia A. Schneider                                               *
 *   Modo de comunicacao: sincrono                                      *
 *                                                                      *
 ************************************************************************/

#include <stdio.h>
#include <xom.h>
#include <xmp.h>

#include <xmp_cmis.h>

#ifdef _IBM_SOURCE
#include <lnv.h>  /*esta' em /usr/OV/include */
#include <omp_dmi.h>


#include <sys/stdsyms.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include <time.h>
#include <stdlib.h>
#include <string.h>     /*estao sendo usadas para AbreArq()*/
#include <strings.h>

#include <OV/ove_xmp.h>        /* na manpage do om_create fala para
                               incluir este arquivo */
#include <OV/ov_types.h>
#endif

#ifdef _HPUX_SOURCE

/*#include <xmp_snmp.h>*/
#include <sys/stdsyms.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <time.h>
#include <string.h>     /*estao sendo usadas para AbreArq()*/
#include <strings.h>
#include <signal.h>     /* esta sendo utilizada para finalizar o gerente */

#endif


#define  ERROR_BUFLEN 128
#define  OID_LEN 128
#define  NUMGET 150

long ajuste( struct timeval *temp1, struct timeval *temp2, FILE *fpin);
FILE *AbreArq(char *argv1);
void FechaArq(FILE *fpin);
void PegaData(FILE *fpin);
```

```
FILE *iniciaArq(int argc, char **argv);

        /*****************************************************************
        *                                                                *
        * Abaixo estao todas as classes e pacotes requeridos pelo *
        * gerente.                                                *
        *                                                                *
        *****************************************************************/

/*OM_EXPORT (MP_CMIS_PKG)*/
/*declaracao para ibm */
#ifdef _IBM_SOURCE
OM_EXPORT (OM_BER)
     /*OM_EXPORT (C_DMI_PACKAGES)
OM_EXPORT (C_DMI_SIMPLE_NAME_TYPE)
*/
#ifdef __alpha
     /*OM_IMPORT(DMI_A_ADMINISTRATIVE_STATE)*/
OM_IMPORT(MP_C_ATTRIBUTE_ID)
OM_IMPORT(MP_C_ATTRIBUTE)
OM_IMPORT(MP_C_AVA)
OM_IMPORT(MP_C_CONTEXT)
OM_IMPORT(MP_C_DS_DN)
OM_IMPORT(MP_C_DS_RDN)
OM_IMPORT(MP_C_OBJECT_CLASS)
OM_IMPORT(MP_C_OBJECT_INSTANCE)
#else
     /*OM_EXPORT(DMI_A_ADMINISTRATIVE_STATE)*/
OM_EXPORT(MP_C_ATTRIBUTE_ID)
OM_EXPORT(MP_C_ATTRIBUTE)
OM_EXPORT(MP_C_AVA)
OM_EXPORT(MP_C_CONTEXT)
OM_EXPORT(MP_C_DS_DN)
OM_EXPORT(MP_C_DS_RDN)
OM_EXPORT(MP_C_OBJECT_CLASS)
OM_EXPORT(MP_C_OBJECT_INSTANCE)
#endif

     /*OM_EXPORT (MP_C_OBJECT_CLASS)
OM_EXPORT (MP_C_AVA)
OM_EXPORT (MP_C_DS_RDN)
OM_EXPORT (MP_C_DS_DN)
OM_EXPORT (MP_C_OBJECT_INSTANCE)*/
OM_EXPORT(MP_C_CMIS_GET_ARGUMENT)
OM_EXPORT(MP_C_NETWORK_ADDRESS)
OM_EXPORT(MP_C_ENTITY_NAME)
OM_EXPORT (MP_C_CONTEXT)
OM_EXPORT(MP_C_SESSION)
OM_EXPORT(MP_CMIS_PKG)

OM_EXPORT(MP_C_CMIS_GET_LIST_ERROR)
OM_EXPORT(MP_C_CMIS_GET_RESULT)
OM_EXPORT(MP_C_CMIS_SERVICE_ERROR)
OM_EXPORT(MP_C_FORM1)
OM_EXPORT(MP_C_GET_INFO_STATUS)
OM_EXPORT(OM_C_OBJECT)
OM_EXPORT(MP_C_CMIS_PKG)
```

```
OM_EXPORT(SMI_PART2)
#endif /* ifdef ibm_source */

#ifdef _HPUX_SOURCE
OM_EXPORT (MP_CMIS_PKG)
OM_EXPORT (MP_C_OBJECT_CLASS)
OM_EXPORT (MP_C_AVA)
OM_EXPORT (MP_C_DS_RDN)
OM_EXPORT (MP_C_DS_DN)
OM_EXPORT (MP_C_OBJECT_INSTANCE)
OM_EXPORT(MP_C_CMIS_GET_ARGUMENT)
OM_EXPORT(MP_C_NETWORK_ADDRESS)
OM_EXPORT(MP_C_ENTITY_NAME)
OM_EXPORT (MP_C_CONTEXT)
OM_EXPORT(MP_C_SESSION)
OM_EXPORT(MP_CMIS_PKG)

#endif /* _hpux_source */

        /*****************************************************************
        *                                                                *
        * a estrutura feature list contem pacotes e caracteres que serao*
        * negociadas no worspace. elas podem ser inicializadas aqui ou  *
        * durante a execucao do programa ( como esta sendo feito).       *
        *                                                                *
        *****************************************************************/

#ifdef _IBM_SOURCE
static MP_feature feature_list[] =
{
  { {OMP_LENGTH(MP_CMIS_PKG), OMP_D_MP_CMIS_PKG}, OM_FALSE},
  { {OMP_LENGTH(SMI_PART2), OMP_D_SMI_PART2}, OM_FALSE},
  { {0,0}, OM_FALSE }
  /*  { {0,0}, 0, OM_FALSE },
    0*/
};
#endif /* _ibm_source */

#ifdef _HPUX_SOURCE
static MP_feature feature_list[] =
{
    { {0,0}, 0, OM_FALSE },
    0
};
#endif /* _hpux_source */

        /*****************************************************************
        *                                                                *
        * A definicao estatica do cmis_get_argument inclui as estruturas*
       * class objecte object instance, necesarias para fazer um   *
       * request ao agente. Como estamos interessados em medir o tempo *
       * que o gerente manda um get e o agente responde, para analisar *
       * o que acarreta a utilizacao do ORS, o agente nao tem uma MIB    *
       * assim o pedido de get nao tem relacao com nenhum objeto. Serve*
       * apenas para trocar um pacote CMIP. O agente manda de volta      *
       * uma resposta vazia.                                      *
       *                                                 *
```

```
      * Os campos dos objets identifiers serao preenchidos        *
      * dinamicamente para a classe e instancia  com a funcao          *
      * at_string_to_oid(), mas eles devem ser adicionados abaixo,     *
      * mas setados com zero.                                          *
      *                                                                *
      * O oid da classe 'e 1.2.76.1.1.1.1.1.7.1                        *
      * O distiguishing attr 'e o  nmaNumber e seu oid 'e:         *
      * 1.2.76.1.1.1.1.2.1.8                                          *
      *                                                                *
      ******************************************************************/

static OM_descriptor                object_class[] = {
    OM_OID_DESC(OM_CLASS, MP_C_OBJECT_CLASS),
    { MP_GLOBAL_FORM, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    OM_NULL_DESCRIPTOR,
    };

static OM_descriptor                ava[] = {
    OM_OID_DESC(OM_CLASS, MP_C_AVA),
    { MP_NAMING_ATTRIBUTE_ID, OM_S_OBJECT_IDENTIFIER_STRING, {0, 0} },
    { MP_NAMING_ATTRIBUTE_VALUE, OM_S_PRINTABLE_STRING, {10, "Telepar 25"} },
    OM_NULL_DESCRIPTOR,
    };

static OM_descriptor                ds_rdn[] = {
    OM_OID_DESC(OM_CLASS, MP_C_DS_RDN),
    { MP_AVAS, OM_S_OBJECT, {0, ava} },
    OM_NULL_DESCRIPTOR,
    };

static OM_descriptor                ds_dn[] = {
    OM_OID_DESC(OM_CLASS, MP_C_DS_DN),
    { MP_RDNS, OM_S_OBJECT, {0, ds_rdn} },
    OM_NULL_DESCRIPTOR,
    };

static OM_descriptor                object_instance[] = {
    OM_OID_DESC(OM_CLASS, MP_C_OBJECT_INSTANCE),
    { MP_DISTINGUISHED_NAME, OM_S_OBJECT, {0, ds_dn} },
    OM_NULL_DESCRIPTOR,
    };

      /*****************************************************************
      *                                                                 *
      * Criacao da definicao estatica para o cmis_get_arg here. Os      *
      * campos da classe e dainstancia do objeto sao preenchidos com    *
      * object_class e object_instance, definidos acima.
      * Also create a static definition of the cmis_get_arg here      *
      * insert the object class and instance fields above in it       *
      *                                                                 *
      *****************************************************************/

static OM_descriptor                get_arg[] = {
    OM_OID_DESC(OM_CLASS, MP_C_CMIS_GET_ARGUMENT),
    { MP_BASE_MANAGED_OBJECT_CLASS, OM_S_OBJECT, {0,object_class }},
    { MP_BASE_MANAGED_OBJECT_INSTANCE, OM_S_OBJECT, {0,object_instance} },
    OM_NULL_DESCRIPTOR,
```

```
    };

#ifdef _IBM_SOURCE
OM_private_object cmis_get_arg ;
#endif

int main(int argc, char *argv[])
{
OM_workspace            workspace;
OM_return_code          om_ret;
MP_status               mp_status;
/* declaracao de variaveis */
char                    oid_string[OID_LEN];
char                    ava_string[OID_LEN];
char                    ipaddr[4];
char                    err_buf[ERROR_BUFLEN];
OM_sint32               invoke_id[NUMGET];
OM_object_identifier    oid_oid;
OM_object_identifier    ava_oid;
OM_private_object       bound_session;
OM_private_object       result_obj/*[NUMGET]*/;
OM_private_object       priv_context/*[NUMGET]*/;

/* declaracao das variaveis  usadas p/ medir o tempo do get */
/*FILE *p1,*p2;
char first[10], second[10],*dat[30];
long diferenca[NUMGET];
*/
struct timeval time1[NUMGET],time2[NUMGET];
struct timezone tzp1,tzp2;
long som=0;
int cont=0;
FILE *fpin;

/* variaveis utilizadas na ibm */
#ifdef _IBM_SOURCE

/*descriptor com parametos de sessao*/
static OM_descriptor publicSessionObj[]=
{
  OM_OID_DESC(OM_CLASS, MP_C_SESSION),
  {MP_ROLE, OM_S_INTEGER, MP_T_MANAGING |MP_T_MONITORING},
  OM_NULL_DESCRIPTOR
};
OM_private_object session_ibm;
OM_private_object cmis_get_arg ;
OM_public_object  pub_result_obj;
OM_public_object  attribute_list;
OM_value_position total;
char              *pstr;

#endif      /* _ibm_source */

      fpin=iniciaArq(argc,argv);

      /*fclose(fpin);
      exit(-1);*/
```

```
  /*****************************************************************
  *                                                                *
* Os object identifiers devem ser passados para CI como strings *
* codificadas pelo BER. A funcao at_str_to_oid() fara' esta *
* codificacao. Ela retornara' a string codificada com BER e o   *
* comprimento desta string. Nao pode ser utilizado o strlen()    *
* para encontrar o tamanho desta string porque as strings BER    *
* podem ter NULLs incluidos nela.                                *
* BER -> Basic Encoding Rules                                    *
*                                                               *
*****************************************************************/

/*este registro deve estar igual ao do agent.lrf !!!  */

if (!strcpy(oid_string, "1.2.76.1.1.1.1.1.7.1"))
    printf("strcpy failed");
  om_ret  = at_str_to_oid (oid_string, &oid_oid);
  if (om_ret != OM_SUCCESS) {
    printf ("at_str_to_oid() failed: %d for object class.\n",
          om_ret);
      return(-1);
  }


if (!strcpy(ava_string, "1.2.76.1.1.1.1.1.1.8"))
          printf("strcpy failed");
  om_ret  = at_str_to_oid (ava_string, &ava_oid);
  if (om_ret != OM_SUCCESS) {
    printf ("at_str_to_oid() failed: %d for object instance.\n",
          om_ret);
      return(-1);
  }

  /*****************************************************************
  *                                                               *
* deve-se colocar o oid_oid acima na estrutura object_class *
* Os mesmos passos devem ser feitos para o oid do distinguished *
* attr, na estrutura ava, j'a definida anteriormente.       *
*                                                               *
  *                                                                *
  *****************************************************************/

object_class[1].value.string.length = oid_oid.length;
object_class[1].value.string.elements = oid_oid.elements;

ava[1].value.string.length = ava_oid.length;
ava[1].value.string.elements = ava_oid.elements;

  /*****************************************************************
  *                                                                *
  * A API mp_initialize() 'e utilizada para inicializar o workspace.*
  * se o workspace 'e NULL, deve-se sair do programa com uma       *
  * mensagem de erro, pois nada mais pode ser feito.          *
  *                                                                *
  *****************************************************************/

workspace =(OM_workspace) mp_initialize();
```

```
      if (workspace == NULL) {
            printf ("mp_initialize() failed \n");
            return(-1);
        }

        /*****************************************************************
        *                                                                *
        *  Vai ser testado se o pacote CMIS 'e suportado com o           *
      *  mp_negotiate(). Mas primeiro deve-se setar a primeira          *
        *  feauture da lista de features para especicar o pacote cmis    *
        *  antes de fazer a chamada. Depois da chamada testa-se             *
        *  o valor do campo ativado para verificar se o pacote 'e         *
        *  suportado.                                                     *
        *                                                                *
        *****************************************************************/

#ifdef _HPUX_SOURCE
      feature_list[0].feature = MP_CMIS_PKG;
        feature_list[0].request= MP_ACTIVATE;

      mp_status = mp_negotiate (feature_list, workspace);

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                 (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_negotiate() failed: %s \n", err_buf);
          return(-1);
      }

        if (!feature_list[0].response) {
          printf("mp_negotiate() failed to activate CMIS package");
          return(-1);
      }
#endif

#ifdef _IBM_SOURCE
      printf("vou negociar o pacote\n");  /*usado para teste de
interoperabilidade*/
        mp_status = mp_version(workspace,feature_list);

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
```

```
            mp_error_message (mp_status, ERROR_BUFLEN,
                 (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_negotiate() failed: %s \n", err_buf);
          return(-1);
      }

        if (!feature_list[0].activated) {
          printf("mp_version() failed to activate CMIS package");
          return(-1);
      }
#endif
        /*****************************************************************
        *                                                                *
        * Use o comando mp_bind() para ligar ao o  postmaster. Como      *
        * este e' o gerente, pode-se usar MP_DEFAULT_SESSION  como       *
        * primeiro argumento. O postmaster retornara' uma bound          *
        * session como terceiro objeto.                                  *
        *                                                                *
        *****************************************************************/

#ifdef _HPUX_SOURCE
      /*mp_status = mp_bind(MP_DEFAULT_SESSION, workspace, &bound_session); */
#endif /* _hpux_source */


      /* crinado a session de uma forma diferente para ibm */
#ifdef _IBM_SOURCE
      /*om_ret = om_create(MP_C_SESSION, OM_TRUE, workspace, &session_ibm);
      if (om_ret != OM_SUCCESS) {
          printf ("om_create() failed: %d for session_ibm.\n",
                om_ret);

                mp_shutdown(workspace);
          return(-1);
      }*/

      /* inclusao de codigo para ibm */
      /*om_ret = om_put(session_ibm, OM_REPLACE_ALL, (OM_object)
publicSessionObj,
                    NULL, (OM_value_position) 0, (OM_value_position) 0);
      if(om_ret != OM_SUCCESS){
          printf("om_put() failed: %d \n", om_ret);
          om_delete(session_ibm);
          mp_shutdown(workspace);
          return(-1);
      }
      */
      /* modificacao feita para executar na ibm */
      /*    mp_status = mp_bind(session_ibm, workspace, &bound_session);*/

#endif /*_ibm_source */

      /*printf("vou fazer um bind\n"); *interoperabilidade */
      mp_status = mp_bind(MP_DEFAULT_SESSION, workspace, &bound_session);
```

```
      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                 (unsigned char *) &err_buf);
           (void)om_delete(mp_status);
          }
          printf("mp_bind() failed: %s \n", err_buf);
#ifdef _IBM_SOURCE
          /* om_delete(session_ibm);*/
#endif
          mp_shutdown(workspace);
          return(-1);
      }


      /*****************************************************************
       *                                                               *
       * A funcao omX_print() nao trabalha corretamente, a menos que  *
       * o workspace tenha sido criado.                                *
       * O resultado do cmis_get_arg vai ser impresso aqui.           *
       *                                                               *
       *****************************************************************/
#ifdef _HPUX_SOURCE
      /*omX_print (stdout, "CMIS_GET", workspace, get_arg);*/
#endif /*_hpux_source */

        /*****************************************************************
        *                                                               *
      * Cria um objeto context e inicializa-o.A opcao OM_TRUE faz com *
      * que o objeto seja criado com os valore default. A configuracao*
      * default para o 'atributo assincrono 'e que tenha falso, e *
      * assim, a interface opere sincronamente.                       *
        *                                                                *
        *****************************************************************/

        for(cont=0;cont<NUMGET;cont++){

        /*****************************************************************
        *                                                                *
        * O mp_get_req() 'e executado usando a session, o context e      *
      * o cmis get argument numa chamada sincrona (controlada pelo       *
      * objeto context).                                                 *
      *                                                                 *
        *****************************************************************/

        /*****************************************************************
      *                                                                  *
      * Antes do mp_get_req() vou comecar a marcar o tempo e depois    *
      * marco o tempo quando terminar                                    *
        *                                                                *
        *****************************************************************/
```

```
#ifdef _IBM_SOURCE
        /*criando o objeto que vai ser passado no mp_get_req, nao precisa!! */
        om_ret = om_create(MP_C_CMIS_GET_ARGUMENT, OM_TRUE, workspace,
&cmis_get_arg);

        if (om_ret != OM_SUCCESS) {
          printf ("om_create() failed: %d /n", om_ret);
            return(-1);
        }

        om_ret = om_put(cmis_get_arg, OM_REPLACE_ALL, get_arg , NULL,
                    (OM_value_position) 0, (OM_value_position) 0);

        if (om_ret != OM_SUCCESS) {
          printf ("om_create() failed: %d /n", om_ret);
            return(-1);
        }

      printf("vou fazer um get\n"); /*interoperabilidade */
      gettimeofday(&time1[cont],&tzp1);

        mp_status = mp_get_req( bound_session,
                                MP_DEFAULT_CONTEXT,
                                get_arg,  /*cmis_get_arg,*/
                          &result_obj,
                                &invoke_id[cont] );

      gettimeofday(&time2[cont],&tzp2);


        if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_get_req() failed: %s \n", err_buf);
          mp_unbind(bound_session);
          om_delete(bound_session);
          mp_shutdown(workspace);
          return(-1);
      }
      /* mandar imprimir o resultado  */
      /* if(cont == 9)
          omX_print( stdout, "RESULT", workspace, result_obj );*/

      /*  om_delete(priv_context); nao estou criando mais*/

     om_ret = om_get(result_obj, (OM_exclusions) 0, NULL, OM_FALSE,
(OM_value_position) 0,
            (OM_value_position) 0, &pub_result_obj, &total);
```

```
      if (om_ret != OM_SUCCESS) {
         printf ("om_create() failed: %d /n", om_ret);
         mp_unbind(session_ibm);
         om_delete(session_ibm);
         mp_shutdown(workspace);
         return(-1);
       }
      /*tentando imprimir o resultado do get */

      /* while(pub_result_obj->type != OM_NO_MORE_TYPES){
        switch(pub_result_obj->type){
          case MP_MANAGED_OBJECT_CLASS:
            printf("Managed Object Class \n");
            at_oid_to_str(pub_result_obj->value.object.object[1].value.string,
                          &pstr);
            printf("Class ObjId = %s\n",pstr);
            break;
          case MP_MANAGED_OBJECT_INSTANCE:
            printf("Managed Object Instance \n");
            break;
          case MP_CURRENT_TIME:
            printf("Current Time \n");
            break;
          case MP_ATTRIBUTE_LIST:
            printf("Attribute list \n");
            attribute_list =  pub_result_obj->value.object.object;
            while(attribute_list->type != OM_NO_MORE_TYPES){
            switch(attribute_list->type){
              case MP_ATTRIBUTE_ID:
                at_oid_to_str(attribute_list-
>value.object.object[1].value.string,
                              &pstr);
                printf("Attribute Id = %s\n",pstr);
                break;
              case MP_ATTRIBUTE_VALUE:
                if(attribute_list->syntax == OM_S_OBJECT)
                  printf("Attribute Value = %d \n",
                       attribute_list-
>value.object.object[2].value.enumeration);
                else
                  printf("else Attribute Value = %s \n",
                       attribute_list->value.string.elements
*object.object[2].value.integer*);
                break;

            } * fim do switch do attribute_list *
            attribute_list++;
            } * fim do while do attibute_list *
            break;
          case MP_GLOBAL_FORM:
            printf("GLOBAL FORM \n");
            break;

          default: ;

        } *fim do switch *
```

```
          pub_result_obj++;
        } *fim do while */

        om_delete(attribute_list); /* coloquei isto depois */

        om_delete(cmis_get_arg); /*coloquei isto depois */

        om_delete(result_obj/*[cont]*/);

        om_delete(pub_result_obj);

#endif   /* _ibm_source */

#ifdef _HPUX_SOURCE
      om_ret = om_create( MP_C_CONTEXT,
                          OM_TRUE,
                          workspace,
                          &priv_context);

        if (om_ret != OM_SUCCESS) {
          printf ("om_create() failed: %d /n", om_ret);
            return(-1);
        }
      gettimeofday(&time1[cont],&tzp1);

        mp_status = mp_get_req( bound_session,
                                priv_context,
                                get_arg,  /*get_arg,*/
                          &result_obj,
                                &invoke_id[cont] );

      gettimeofday(&time2[cont],&tzp2);


        if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
            (void)om_delete(mp_status);
          }
          printf("mp_get_req() failed: %s \n", err_buf);
          return(-1);
      }
      /* mandar imprimir o resultado  */
      /* if(cont == 9)
          omX_print( stdout, "RESULT", workspace, result_obj );*/

        om_delete(priv_context);

          om_delete(result_obj/*[cont]*/);
```

```
#endif /*_hpux_source */


        } /* fim do for para fazer NUMGET get's */

      som= ajuste(time1,time2,fpin);
      printf("\n media = %lu \n",som);
      fclose(fpin);

        /*****************************************************************
      *                                                                  *
      * se o codigo de retorno do mp_get_req() foi MP_SUCESS, entao      *
      * omX_print o confirmation object.                                 *
        *                                                                   *
        *****************************************************************/
#ifdef _HPUX_SOURCE
      /*    omX_print( stdout, "RESULT", workspace, result_obj[cont] );*/
#endif /*_hpux_source*/

        /*****************************************************************
        *                                                                  *
        * Unbind do postmaster.                                            *
        *                                                                  *
        *****************************************************************/

        mp_status = mp_unbind(bound_session);

      if (mp_status != MP_SUCCESS) {
          if (mp_status == MP_NO_WORKSPACE)
            strcpy (err_buf, "MP_NO_WORKSPACE");
          else if (mp_status == MP_INVALID_SESSION)
            strcpy (err_buf, "MP_INVALID_SESSION");
          else if (mp_status == MP_INSUFFICIENT_RESOURCES)
            strcpy (err_buf, "MP_INSUFFICIENT_RESOURCES");
          else {
            mp_error_message (mp_status, ERROR_BUFLEN,
                (unsigned char *) &err_buf);
           (void)om_delete(mp_status);
          }
          printf("mp_unbind() failed: %s \n", err_buf);
          return(-1);
      }

        /*****************************************************************
        *                                                                  *
        * Libero os objetos context e session criados, com o comando       *
        * om_delete().                                                     *
        *                                                                  *
        *****************************************************************/

        om_delete(bound_session);
      /*        for(cont=0;cont<NUMGET;cont++){
        om_delete(priv_context[cont]);
          om_delete(result_obj[cont]);
        }*/
        /*****************************************************************
        *                                                                  *
```

```
         * Agora libero o workspace com o comando mp_shutdown().          *
         *                                                                 *
         *******************************************************************/

       mp_shutdown(workspace);

       return(0);
}

/*funcao para fazer ajuste do tempo medido em microsegundos*/
/* Nao esta concluida, preciso fazer outros tipos de ajustes*/
long ajuste(struct timeval *temp1, struct timeval *temp2, FILE *fpin)
{
  int t=0,d[NUMGET],cont=0;

  for(cont=0;cont<NUMGET;cont++){

    t=temp2[cont].tv_sec - temp1[cont].tv_sec;
    /*    printf("\n temp1[%d]).tvusec= %lu (temp2[%d]).tvusec= %lu
",cont,temp1[cont].tv_usec,cont,temp2[cont].tv_usec);*/

    if(t==0){           /* os tempos estao na mesma casa dos segundos*/
      d[cont]=temp2[cont].tv_usec - temp1[cont].tv_usec;
    }
    else{          /*t>=1 */
      d[cont]= (t*1000000 + temp2[cont].tv_usec)- temp1[cont].tv_usec;
    }

  /* printf("\td[%d] = %lu \n",cont,d[cont]);*/
   /*  printf("temp1[%d]).t_sec= %lu (temp2[%d]).tv_sec= %lu t=%d\n
",cont,temp1[cont].tv_sec,cont,temp2[cont].tv_sec,t);*/
  }

  t=0;

  /*fprintf(fpin,"\n\t resultado do tempo medido\n");*/
  for(cont=0;cont<NUMGET;cont++){
    /*    fprintf(fpin,"\t d[%d]\t%lu \n",cont,d[cont]);*/
    fprintf(fpin,"\t%lu\n",d[cont]);
    t+=d[cont];
  }
  /*fprintf(fpin,"\t media = %lu \n",t/30);*/
  /*  fclose(fpin);*/
  return t/NUMGET;
}

FILE *AbreArq(char *argv1){

  FILE *fpin;

   if( (fpin = fopen(argv1,"a"))==NULL)
     printf("\n Nao consegui abrir arquivo %s\n",argv1);

  return fpin;
}

void FechaArq(FILE *fpin){
```

```
  fclose(fpin);
}

void PegaData(FILE *fpin){

  char *dat[30];
  FILE *p1;
  int cont;

      p1 = popen("date '+%c'","r");
      fprintf(fpin," \n");
      for( cont=0; cont<6; cont++){
           fscanf(p1,"%s ",&dat[cont]);
           /*printf("\ndat = %s  \n",&dat[cont]);*/
           fprintf(fpin," %s ",&dat[cont]);
            }
        pclose(p1);
}

FILE *iniciaArq(int argc, char **argv){

  FILE *fpin;
  /*  char Arqdefault[7]="result\0";  se na chamada do programa nao for dado
                          nenhum nome de arquivo, este e' usado
  char hostdefault[6]="urano\0";*/

  char *nome_arq="result\0";
  char *host="urano\0";
  int tam_mens=1000;
  char *opcao1="-h";
  char *opcao2="-a";
  char *opcao3="-t";
  int ii=1;

       if(!argc){
        fprintf(stderr,"argc=0\n",*argv);
        exit(-1);
      }
      /* printf("\n argc= %d\targv[1]= %s\targv[2]=%s \targv[3]=
%s\n",argc,argv[1],argv[2],argv[3]);*/
     for(ii=1 ; ii<argc-1 ; ii++){
      /*printf("\ni=%d",ii);*/

      if(strcmp(opcao1,argv[ii])==0){
          ii++;
          host=argv[ii];
          /*printf("\nhost= %s",host);*/
      }

      if(strcmp(opcao2,argv[ii])==0){
            ii++;
            nome_arq=argv[ii];
          /*printf("\tarq= %s",nome_arq);*/
      }

      if(strcmp(opcao3,argv[ii])==0){
```

```
        ii++;
        tam_mens= atoi(argv[ii]);
       /* printf("\ttam= %d",tam_mens);*/
     }
  }

   /*if(argc==1){*/
       fpin=AbreArq(nome_arq);
       fprintf(fpin,"\n tamanho da mensagem = %d",tam_mens);
       PegaData(fpin); /* coloca data e horario no arquivo de dados */
       fprintf(fpin,"\no nome do hostname do agente e' %s\n",host);
       /*  }*/

       /*  if(argc>1){
       fpin=AbreArq(argv[2]);
          fprintf(fpin," tamanho da mensagem = %s",argv[3]);
        pega a data e horario que esta' sendo executado o programa
          e coloca no arquivo que tem os dados
       PegaData(fpin);

         fprintf(fpin,"\no nome do hostname do agente e' %s\n",argv[1]);
       }*/
     return fpin;
}
```