Gerência Integrada de Redes e Serviços

Introdução

Há pouco tempo atrás, as operadoras de telecomunicações se preocupavam em aproveitar a máxima capacidade de transmissão dos sistemas existentes, se preocupando muito pouco com a qualidade do sistema e dos serviços prestados. Atualmente, este perfil vem mudando, ocorrendo o sacrifício da capacidade de transmissão de informações de usuário, em troca de uma capacidade transmissão de informações de gerência maior, para prover um serviço mais confiável e seguro, de qualidade indiscutivelmente maior.

Também podemos perceber que, devido à crescente digitalização da rede e o aumento da capacidade e confiabilidade dos sistemas, as empresas operadoras perdem um pouco o seu papel típico de atuação na arquitetura das redes e passa a se preocupar mais com os serviços fornecidos aos usuários, que passa a ser um fator diferencial de fundamental importância no mercado. Ocorre, desta forma, um crescimento muito grande em termos de criação, implantação e oferta de novos serviços, baseados na integração de áudio, dados, textos, imagens e vídeo, ou seja, multimídia. Como exemplo de alguns serviços emergentes, podemos citar os serviços de rede inteligente (RI), serviços em terminais de uso público, processamento digital de sinais de áudio e vídeo e os próprios serviços multimídia.

As redes de telecomunicações podem ser vistas, independente do tipo e dos equipamentos utilizados, como dividida em três níveis principais: aplicação, serviço e arquitetura .

A camada de aplicação é aquela empregada diretamente pelo usuário final. A camada de serviço deve ser projetada pelo provedor de rede para suportar todas as aplicações do usuário e a camada de arquitetura provê as soluções de engenharia que devem prover o transporte de qualquer tipo de serviço vendido pela operadora ao usuário. O serviço é normalmente designado como a facilidade que o provedor vende a seus clientes e tipicamente suporta várias aplicações.

A necessidade de qualidade, a diversificação e a complexidade cada vez maior destes serviços implica em uma necessidade tão vital quanto o próprio serviço: a sua gerência.

Dentro deste conceito de gerenciamento de redes de telecomunicações, começaram a surgir alguns sistemas de supervisão específicos para cada situação (por exemplo, gerenciamento de falhas, de tráfego) e para cada fabricante, ou seja, os chamados sistemas de gerência proprietários (figura 1.1).

Figura 1.1: Sistemas de Gerência Proprietários

Na figura acima, por exemplo, podemos ter os equipamentos como sendo várias centrais telefônicas de fabricantes distintos, cada uma com seu próprio sistema de gerência. As centrais são interligadas entre si, mas os sistemas de gerência são isolados.

Este tipo de sistema possui alguns problemas, como:

- a impossibilidade de interconexão entre sistemas de diferentes fabricantes devido ao uso de interfaces não padronizadas;
- 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 linguagem de comunicação homem-máquina;
- multiplicidade de base de dados: cada sistema tem a sua própria base de dados local, sendo necessário atualizar cada sistema isoladamente, o que acaba resultando em duplicidades e inconsistências.

Estes fatores acarretam em uma falta de integração entre processos que impossibita, por exemplo:

- obtenção de uma visão global do estado da rede e dos serviços;
- integração de forma automatizada das atividades operacionais;
- difusão de informações dos estados de circuitos e serviços de uma forma ampla;
- flexibilidade de roteamento na rede;
- operação e manutenção eficientes, etc.

Como conseqüência disto, temos:

- elevação do índice de falhas não detectadas;
- congestionamento na rede;
- falta de flexibilidade no roteamento;
- indicação múltipla da mesma falha;
- dados insuficientes para planejamento;
- deficiência de operação e manutenção;

Que acarretam em perda de ligações e de receitas, insatisfação do usuário e desperdício pelo aumento dos custos operacionais e investimentos extras.
Baseado nestes fatores, tem-se procurado uma solução para o problema da falta de integração entre os sistemas, que possibilite a Gerência Integrada de Redes e Serviços (GIRS), proposta pela TELEBRÁS, cujos conceitos se encontram na Prática 501-100-104.

Conceito de GIRS

De acordo com esta Prática, o conceito de GIRS é:
"o conjunto de ações realizadas visando obter a máxima produtividade da planta e dos recursos disponíveis, integrando de forma organizada as funções de operação, manutenção, administração e provisionamento (OAM&P) para todos os elementos, rede e serviços de telecomunicações"
A gerência deve ser integrada no sentido de:

- ser única para equipamentos semelhantes de fabricantes distintos;
- ser feita de maneira consistente pelos vários sistemas;
- ser feita desde o nível de serviço até o nível dos equipamentos;
- um operador ter acesso a todos os recursos pertinentes ao seu trabalho, independentemente do sistema de suporte à operação onde estes recursos estão disponíveis ou da sua localização geográfica;
- os sistemas se "falarem" de modo que as informações fluam de maneira automática.

A situação desejada, de uma maneira superficial, então, é a seguinte:

Figura 1.2: Gerência Integrada

Para se atingir este objetivo, é necessário, então:

- processos operacionais com fluxo contínuo;
- facilidades de reconfiguração em tempo real;
- dados 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.

Requisitos básicos

Para se chegar à integração das funções de gerência são necessários:

- elaboração de um modelo conceitual de operação, administração, manutenção e provisionamento baseado nos objetivos e estratégias da empresa; ·
- padronização dos modelos de informações de elementos de rede e serviços de telecomunicações;
- padronização das interfaces homem-máquina;
- automação de tarefas visando eficiência;
- flexibilidade de arquitetura;
- ambiente aberto, permitindo interconectividade e interoperabilidade;
- alta confiabilidade e segurança.

Objetivos Básicos
Integrando as funções de todas as camadas funcionais, podemos atingir alguns objetivos gerenciais, como:

- minimizar o tempo de reação a eventos da rede;
- minimizar a carga causada pelo tráfego de informações de gerenciamento;
- permitir dispersão geográfica do controle sobre os aspectos de operação da rede;
- prover mecanismos de isolação para minimizar riscos de segurança;
- prover mecanismos de isolação para localizar e conter falhas na rede;
- melhorar o serviço de assistência e interação com os usuários.

TMN - Telecommunications Management Network
Introdução
Estrutura Funcional
Areas Funcionais

Arquitetura TMN
Proximo Topico: Gerenciamento OSI
Volta ao Indice

Introdução

A TMN foi desenvolvida com o propósito de gerenciar redes, serviços e equipamentos heterogêneos, operando sobre os mais diversos fabricantes e tecnologias que já possuem alguma funcionalidade de gerência.
Desta forma, a idéia da TMN é proporcionar:

"uma arquitetura organizada, possibilitando a integração e interoperabilidade entre vários tipos de sistemas de operação e os equipamentos de telecomunicações , utilizando modelos genéricos de rede para a gerência, modelos genéricos de informações com interfaces e protocolos padronizadas"

de forma a criar um conjunto de padrões para administradores e fabricantes, utilizados no desenvolvimento e na compra de equipamentos de telecomunicações, e também no projeto da rede de gerência.
A TMN considera as redes e os serviços de telecomunicações como um conjunto de sistemas cooperativos e gerencia-os de forma harmônica e integrada. A figura 4.1 mostra o relacionamento entre a TMN e a rede de telecomunicações por ela gerenciada.
A TMN é, na realidade, uma rede de computadores utilizada para gerenciar uma rede de telecomunicações. A TMN interage com a rede de telecomunicações em vários pontos, através de interfaces padronizadas, podendo utilizar parte da rede de telecomunicações para realizar suas funções.
Algumas redes e serviços que podem ser gerenciadas pela TMN são:

- redes públicas e privadas, incluindo a RDSI, redes de telefonia móvel, redes privativas de voz e redes inteligentes;
- elementos de transmissão (multiplexadores, roteadores, cross-connects, equipamentos SDH);
- sistemas de transmissão analógica e digital baseados em cabos coaxiais, fibra óptica, rádio e enlace de satélite;
- mainframes, processadores front-end, controladoras remotas, servidores de arquivos, etc;
- redes locais, geográficas e metropolitanas (LAN, MAN e WAN);
- redes de comutação de circuito e pacotes;
- a própria TMN;
- terminais e sistemas de sinalização incluindo Pontos de Transferência de Sinalização (STP) e bases de dados em tempo real;
- serviços de suporte e teleserviços;
- sistemas de infra-estrutura e suporte, como módulos de teste, sistemas de energia, unidades de ar condicionado, sistemas de alarme, etc.

Figura 4.1: TMN e a rede de telecomunicações

4.2. Estrutura funcional

Descrever a estrutura funcional consiste em subdividir funcionalmente a gerência em níveis ou camadas que restringem as atividades de gerência contidas nelas, mas sendo possível a comunicação direta entre camadas não adjacentes.

Camada de Elemento de Rede

Corresponde às entidades de telecomunicações (software ou hardware) que precisam ser efetivamente monitorados e/ou controlados. Estes equipamentos devem possuir agentes para que possam fornecer as informações necessárias ao sistema de gerência, como coleta de dados de performance, monitoração de alarmes, coleta de dados de tráfego, etc.

Camada de Gerência do Elemento da Rede

Esta camada é responsável pelo gerenciamento dos equipamentos na forma de sub-redes, ou seja, pequenas partes da rede completa devem ser gerenciadas e suas informações sintetizadas para poderem ser aproveitadas pela Gerência de Rede do sistema, que tem assim a visão completa da rede.

Camada de Gerência de Rede

Esta camada gerencia o conjunto de elementos (sub-redes) como um todo, tendo uma visão fim-a-fim da rede. Para isso, ela recebe dados relevantes dos vários sistemas de Gerência de Elemento de Rede e ao processa-os para obter uma visão consisa da rede completa.

Camada de Gerência de Serviço

Esta camada relaciona os aspectos de interface com os clientes, e realiza funções como provisionamento de serviços, abertura e fechamento de contas, resolução de reclamações dos clientes (inclusive relacionados a tarifação), relatórios de falhas e manutenção de dados sobre qualidade de serviço (QoS).

Camada de Gerência de Negócio

É um ponto onde ocorrem as ações executivas, ou seja, é responsável pela gerência global do empreendimento. É neste nível em que são feitos os acordos entre as operadoras e onde são definidos os objetivos.

Áreas Funcionais

De forma a se englobar toda a funcionalidade necessária ao gerenciamento de uma rede de telecomunicações (planejamento, instalação, operação, manutenção e provisionamento), identificou-se cinco áreas funcionais:

- Gerenciamento de Desempenho
- Gerenciamento de Falhas
- Gerenciamento de Configuração
- Gerenciamento de Tarifação
- Gerenciamento de Segurança

Gerenciamento de Desempenho

O gerenciamento de desempenho envolve as funções relacionadas com a avaliação e relato do comportamento dos equipamentos de telecomunicações e a eficiência da rede. Estas funções se dividem basicamente em dois grupos:

- Medidas de Tráfego: capacitam o usuário a definir e controlar a entrega de relatórios de medidas de tráfego;
- Monitoração de Desempenho: informações que permitem ao usuário obter, avaliar e relatar parâmetros de desempenho da rede. Estas informações podem ser utilizadas como apoio ao diagnóstico de falhas, planejamento de rede e qualidade de serviço.

Algumas funções relativas ao gerenciamento de desempenho são:

- manter informações estatísticas;
- manter logs de históricos de estados;
- determinar a performance do sistema sob condições naturais e artificiais;
- alterar os modos de operação do sistema com o propósito de conduzir atividades de gerenciamento de desempenho.

Gerenciamento de Falhas

O gerenciamento de falhas engloba as funções que possibilitam a detecção, isolação e correção de operações anormais na rede de telecomunicações. As falhas impedem os sistemas de cumprir seus objetivos operacionais e podem ser transientes ou persistentes. As funções de gerenciamento de falhas podem ser divididas em:

- Supervisão de Alarmes: gerenciamento de informações sobre as degradações de desempenho que afetam o serviço;
- Teste: o usuário pode solicitar a execução de um teste específico, podendo inclusive estabelecer os parâmetros deste. Em alguns casos, o tipo e os parâmetros do teste podem ser designados automaticamente;
- Relatório de Problemas: utilizado para rastrear e controlar as ações tomadas para liberar alarmes e outros problemas.

Algumas funções do gerenciamento de falhas são:

- manter logs de erros;
- receber e agir sobre notificações de erros;
- rastrear e identificar falhas;
- gerar seqüências de testes de diagnóstico;
- corrigir falhas.

Gerenciamento de Configuração

O gerenciamento de configuração habilita o usuário a criar e modificar recursos físicos e lógicos da rede de telecomunicações. Suas funções são divididas em:

- Gerenciamento de Ordem de Serviço: possibilita a identificação e o controle do provisionamento de novos recursos necessários para a rede de telecomunicações. Uma Ordem de Serviço pode ser utilizada para solicitar novos recursos, físicos ou lógicos;
- Configuração de Recursos: funções que têm como finalidade possibilitar que os recursos da rede possam ser criados, roteados, controlados e identificados;
- Informação de Recursos: funções que têm por finalidade apresentar a lista de recursos alocados, verificar a consistência da informação e obter informação sobre os recursos disponíveis.

Algumas funções relativas ao gerenciamento de configuração são:

- setar parâmetros de controle de rotinas de operação do sistema;
- associar nomes aos objetos gerenciados e configurá-los;
- inicializar e deletar objetos gerenciados;
- coletar informações em tempo real a respeito de das condições atuais do sistema;
- obter avisos a respeito de modificações significativas nas condições do sistema;
- modificar a configuração do sistema.

4.3.4. Gerenciamento de Tarifação

O gerenciamento de tarifação provê um conjunto de funções que possibilitam a determinação do custo associado ao uso da rede de telecomunicações. Algumas funções associadas ao gerenciamento de tarifação são:

* informar aos usuários os custos associados aos recursos consumidos;

* habilitar limites de tarifação e setar agendamentos a serem associados com a utilização dos recursos; * combinar custos quando um requisito de comunicação exigir múltiplos recursos combinados.

4.3.5. Gerenciamento de Segurança

As principais funções que devem se encaixar no gerenciamento de segurança são:

- criação e controle de serviços e mecanismos de segurança;
- distribuição de informações relevantes à segurança;
- armazenamento de eventos relativos à segurança.

Arquitetura TMN

A arquitetura TMN tem por objetivo a coordenação do gerenciamento dos sistemas individuais a fim de se obter um efeito global na rede, com os seguintes requisitos:

- possibilitar várias estratégias de implementação e graus de distribuição das funções de gerenciamento;
- considerar o gerenciamento 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 adicionado e à outras admninistrações o acesso a informações e funções de gerenciamento.
- 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 gerenciamento de redes híbridas baseadas em tipos de equipamentos diversos;
- proporcionar flexibilidade na escolha do grau de confiabilidade / custo desejado para todos os componentes de gerenciamento de rede e para a rede como um todo.

A arquitetura geral TMN está estruturada em três arquiteturas básicas que podem ser consideradas separadamente no planejamento e projeto de uma TMN:

- **Arquitetura funcional:** descreve as funções de gerenciamento agrupadas em blocos funcionais através dos quais uma TMN pode ser implementada;
- **Arquitetura de informação:** fornece fundamentos para o mapeamento dos princípios de gerenciamento OSI em princípios TMN, baseado na abordagem orientada a objetos;
- **Arquitetura física:** baseada em blocos físicos, descreve interfaces e exemplos que constituem a TMN.

Estas arquiteturas podem ser consideradas como pontos de vista diferentes, que atuando em conjunto formam a TMN como um todo.

Figura 4.3: Arquiteturas TMN

Arquitetura funcional

Descreve a distribuição apropriada da funcionalidade (blocos funcionais) dentro da rede de gerência. Um bloco funcional ou agrupamento de funções gerais TMN é a base da arquitetura funcional. Através da ditribuição adequada dos blocos de função na rede pode-se implementar uma rede TMN de qualquer complexidade. A definição destes blocos funcionais e dos pontos de referência (fronteiras entre os blocos funcionais através das quais ocorrem as trocas de informações entre eles) entre os blocos, leva à especificação de interfaces padrões de TMN.

Blocos Funcionais

OSF - Bloco de Função de Sistemas de Suporte à Operações: processa informações de gerência com o propósito de monitorar, coordenar e controlar funções de telecomunicações, inclusive as próprias funções de gerenciamento (a própria TMN).
NEF - Bloco de Função Elemento de Rede: representa para a TMN as funções de telecomunicações e suporte requeridas pela rede de telecomunicações gerenciada. Essas funções não fazem parte da TMN, mas são representadas para ela através do NEF. A parte da NEF que representa as funções de telecomunicações para a TMN é parte da TMN, enquanto as funções de telecomunicações propriamente ditas não fazem parte da TMN. O bloco NEF possui, ainda, as seguintes funções:

- Função Entidade de Telecomunicações (TEF) - relativa aos processos de telecomunicações, como comutação e transmissão;
- Função Entidade de Suporte (SEF) - função relativa a equipamentos de suporte, como infra-estrutura e energia, comutação para proteção de canais de transmissão etc.

QAF - Bloco de Função Adaptador de Interface Q: conecta a TMN a entidades não TMN. Realiza uma adaptação entre um ponto de referência não TMN (por exemplo, interface proprietária) e um ponto de referência "Q3" ou "Qx" da TMN.
MF - Bloco de Função de Mediação: atua modificando a informação trocada entre a NEF ou QAF e a OSF, de acordo com as características da informação esperada por cada um deles. Os MF podem adaptar, armazenar, filtrar e condensar as informações.
WSF - Bloco de Função Estação de Trabalho: o bloco WSF provê os meios para o usuário interpretar e acessar as informações de gerenciamento, incluindo o suporte para interface homem/máquina (apesar desta não ser considerada parte da TMN).

Componentes Funcionais

Os componentes funcionais são as estruturas que compõem os blocos funcionais. Eles são descritos a seguir:

MAF - Management Application Function: a função de aplicação de gerência assume o papel de gerente ou agente, conforme a invocação feita, implementando efetivamente os serviços de gerenciamento. Recebe a denominação correspondente ao bloco de função a que pertence, ou seja, OS-MAF, NEF-MAF, MF-MAF e QAF-MAF.
MF-MAF - Mediation Function - Management Application Function: presente no MF para suportar os papéis de agente e gerente do MF.
OSF-MAF - Operations System Function - Management Application Function: presente para suportar funções de gerência das mais simples às mais complexas. Algumas das funções de gerência no OSF-MAF são correlação de alarmes, localização de problemas, estatísticas, análise de performance, etc.
NEF-MAF - Network Element Function - Management Application Function: presente no NE para suportar o papel de agente.
QAF-MAF - Q Adaptor Function - Management Application Function: presente no QA para suportar o papel de agente e gerente.
WSSF - WorkStation Support Function: necessário para a implementação da função WSF.
UISF - User Interface Support Function: transforma informações de usuário para o modelo de informações da TMN e vice versa, além de tornar o modelo de informação disponível em um formato visível na interface homem-máquina. A interface homem-máquina pode ser uma tela de uma estação, uma impressora ou outro dispositivo.
MIB - Base de Informação de Gerenciamento: repositório conceitual das informações de gerenciamento. Representa o conjunto de recursos gerenciados dentro de um sistema gerenciado. Sua estrutura de implementação não está sujeita a padronização dentro da TMN.
ICF - Interface Convergence Function: traduz o modelo de informação de uma interface para outra. Pode fazer alterações a nível sintático e/ou semântico, sendo obrigatório nos blocos MF e QAF.

MCF - Message Communication Function: associada com todos os blocos funcionais que possuem interface física, provê o meio para se trocar informações entre entidades pares através de uma pilha de protocolos. Esta pilha não precisa ser necessariamente uma pilha OSI de 7 camadas. Conforme o ponto de referência ao qual está associado, recebe a denominação MCFqx, MCFf ou MCFx.
DCF - Data Communication Function: provê funções de roteamento e interconexão, através da implementação das camadas 1 a 3 do modelo OSI.
DSF - Directory System Function: necessário para a implementação do serviço de diretório na TMN. O diretório utilizado na TMN é baseado na X.500. Note que existem várias recomendações do ITU-T sobre os diversos aspectos do serviço diretório. Um diretório contém informações sobre sistemas e quais associações podem ser feitas com estes, detalhes destas associações, detalhes de contexto de aplicações, detalhes de segurança, lista de objetos gerenciados, classes suportadas, etc..
DAF - Directory Access Function: necessário para acesso aos diretórios. É obrigatório no OSF e pode também ser necessário no WSF, MD, QAF e NEF, dependendo se eles se utilizam do serviço diretório.
SF - Security Function: necessário para prover segurança aos blocos funcionais. Os serviços de segurança são autenticação, controle de acesso, confidencialidade de dados e integridade de dados. Os detalhes acerca destes serviços de segurança são dados na X.800.

Pontos de Referência

Define os limites entre os serviços de dois blocos de função de gerência. A finalidade dos pontos de referência é identificar a passagem da informação entre blocos de função, permitindo acesso às informações trocadas entre estes blocos.
Existem três classes de pontos de referência:
a) classe q - entre OSF, QAF, MF e NEF
b) classe f - para ligação de estações de trabalho (ou WSF)
c) classe x - entre OSF´s de duas TMN´s ou entre uma OSF de uma TMN e um bloco funcional com funcionalidade equivalente de outra rede.
São definidas ainda outras duas classes de pontos de referência que não pertencem à TMN mas também são muito importantes:
d) classe g - entre a estação de trabalho e o usuário
e) classe m - entre QAF e entidades não TMN.

Função Comunicação de Dados (DCF)

Fornece os meios necessários para o transporte de informações entre os blocos funcionais da TMN. Pode prover roteamento, retransmissão e interfuncionamento de funções. A DFC fornece as funções das camadas 1, 2 e 3 do modelo OSI e pode ser suportada por diferentes tipos de subredes, como X-25, MAN, LAN, SSCC n° 7, RDSI ou SDH.

Arquitetura Física

Define os blocos construtivos e as interfaces que permitem interligá-los. Estes blocos representam implementações físicas de funcionalidaades (blocos de funções) da TMN.
Os blocos construtivos da arquitetura física TMN são os seguintes:
Rede de Comunicação de Dados (DCN): é uma rede de dados que utiliza protocolos padronizados (deve, sempre que possível, seguir o modelo OSI) e permite a comunicação dos elementos de rede com os sistemas de suporte à operação. Pode ser composta de várias sub-redes de comunicação de dados, como X-25, RDSI, LAN, etc.
Elementos de Rede (NE): bloco que corresponde às entidades de telecomunicações (equipamentos ou facilidades) que são efetivamente monitorados e/ou controlados. É importante distinguir duas classes de funções que podem estar contidas numa NEF:

- funções de telecomunicações que estão diretamente envolvidas no processo de telecomunicações (comutação e transmissão);
- funções não diretamente envolvidas no processo de telecomunicações, como localização de falhas, bilhetagem, comutação, proteção e condicionamento de ar.

Sistema de Operação (OP): engloba as funções que permitem realizar o processamento e o armazenamento das informações relacionadas com a operação, a administração a manutenção e o provisionamento das redes e serviços de telecomunicações.
Dispositivo de Mediação (MD): é o bloco que age sobre as informações trocadas entre os NE e os OS, visando tornar a comunicação mais transparente e eficiente. Pode envolver várias categorias de processo:

- processos de conversão de informação entre diferentes modelos de informação;

- processos envolvendo interfuncionamento entre protocolos de alto nível;
- processo de tratamento de dados;
- processo de tomadas de decisões;
- processo de armazenamento de dados.

Estações de Trabalho (WS): é o bloco que engloba os recursos para o acesso de operadores aos blocos NE, OS e MD. Este terminal deve ser capaz de traduzir o modelo de informação usado na TMN, disponível no ponto de referência f, em um formato apresentável ao usuário, no ponto de referência g.
As funções das WS devem prover ao usuário do terminal as funções gerais para executar entrada e saída de dados. Geralmente incluem:

- segurança de acesso e login;
- reconhecimento e validação de entradas;
- formatação e validação de saídas;
- suporte para "menus", telas, janelas e paginação;
- acesso à TMN;
- ferramentas para modificação de lay-out.

Adaptador Q (QA): permite a interconexão de equipamentos ou interfaces não TMN às interfaces Qx ou Q3.

Protocolos de Comunicação para Interface Q3

A interface Q3 é aquela caracterizada por se localizar no ponto de referência q3, situado entre o Sistema de Suporte às Operações (OS) e os Elementos da TMN que realizam interface com ele.
Para que o transporte destas informações ocorra (através da DCN) de maneira satisfatória e eficiente, existem algumas famílias de protocolos para o transporte de dados padronizadas e recomendadas para a interface Q3. A figura abaixo nos mostra uma visão geral dos vários perfis (para as camadas inferiores do modelo OSI) recomendados. Basicamente, o que uma camada inferior deve fazer é prover suporte às camadas superiores. Recomenda-se, para o caso da interface Q3, que o conjunto de aplicações TMN com necessidade de protocolos similares sejam suportados por uma única seleção de protocolos para as camadas 4 a 7 do modelo OSI. Podem ser necessárias opções para as camadas 1 a 3, de modo a permitir o transporte mais eficiente em cada caso. Também é importante observar que a camada de aplicação (camada 7) é comum a qualquer família de protocolos, pois é a base para se assegurar a interoperabilidade.
Os perfis para as camadas inferiores são classificados em duas categorias: serviços orientados à conexão (CONS) e serviços não orientados à conexão (CLNS), sendo que estes podem assumir vários perfis (CONS1, CONS2, CLNS2, etc) conforme veremos a seguir.

Serviços Orientados à Conexão

4.4.2.2.1. CONS 1

Serviço Orientado à Conexão usando o Protocolo X.25.
É aplicado no ponto de referência entre a Rede Pública de Comutação de Pacotes e o OS/MD/QA/NE, que se comunicam com os OS's instalados na Rede Pública de Comutação de Pacotes ou na Rede Digital de Serviços Integrados (RDSI).
Os protocolos utilizados são os seguintes:

- Camada 1: X.27, X.21, X.21bis, V.II / V.35, V.28 / V.24
- Camada 2: X.25 LAPB [ISO 7776]
- Camada 3: X.25 PLP [ISO 8208]

4.4.2.2.2. CONS 2

Serviço Orientado à Conexão utilizando-se o serviço de suporte modo pacote sobre o canal D (16 kbps) da RDSI.
É aplicado no ponto de referência entre a RDSI e os OS/MD/QA/NE, que se comunicam com os OS's instalados na RDSI ou na Rede Pública de Comutação de Pacotes.
Os protocolos utilizados neste perfil são:

- Camada 1: I.430 (acesso básico - 2B+D), I.431 (acesso primário - 30B+D)
- Camada 2: Q.921 LAPD, X.25 LAPB [ISO 7776]
- Camada 3: Q.931, X.25 PLP [ISO 8208]

4.4.2.2.3. CONS 3

Serviço Orientado à Conexão utilizando-se o serviço de suporte modo pacote sobre o canal B (64 kbps) da RDSI.
Os protocolos utilizados neste perfil são:

- Camada 1: I.430 (acesso básico - 2B+D), I.431 (acesso primário - 30B+D)

- Camada 2: Q.921 LAPD, X.25 LAPB [ISO 7776]
- Camada 3: X.31, X.25 PLP [ISO 8208]

4.4.2.2.4. CONS 4

Serviço Orientado à Conexão que se utiliza o serviço de suporte modo circuito, de forma irrestrita, sobre o canal B (64 kbps) da RDSI.

- Camada 1: I.430 (acesso básico - 2B+D), I.431 (acesso primário - 30B+D)
- Camada 2: Q.921 LAPD, X.25 LAPB [ISO 7776]
- Camada 3: X.31, X.25 PLP [ISO 8208], modo circuito

4.4.2.2.5. CONS 5

Serviço Orientado à Conexão utilizando-se a MTP e o SCCP do SSCC#7.

Os protocolos utilizados neste perfil são:

- Camada 1: MTP (nível 1)
- Camada 2: MTP (nível 2)
- Camada 3: SCCP e MTP (nível 3)

4.4.2.2.6. CONS 6

Serviço Orientado à Conexão utilizando-se o protocolo X.25 sobre uma rede local

É aplicado ao OS/MD/QA/NE que está conectado ao ponto de referência numa rede local (LAN) orientado à conexão.

Os protocolos utilizados são:

- Camada 1: Sinalização Física
- Camada 2: LLC tipo 2 [ISO 8802.2] e MAC CSMA/CD [ISO 8802.3]
- Camada 3: ISO 8208, ISO 8801

4.4.2.3. Serviços Não Orientados à Conexão

4.4.2.3.1. CLNS 1

Serviço Não Orientado à Conexão usando rede local do tipo CSMA/CD

É aplicado no ponto de referência entre a Rede Local e os OS/MD/QA/NE, que se comunicam com os OS's instalados em Rede Local ou na PSPDN.

Os protocolos utilizados são:

- Camada 1: Sinalização Física
- Camada 2: LLC tipo 1 [ISO 8802.2] e MAC CSMA/CD [ISO 8802.3]
- Camada 3: ISO 8473 [ISO 8348 AD1] CLNS [ISO 8473 e ISO 8473 AD1]

4.4.2.3.2. CLNS 2

Serviço Não Orientado à Conexão utilizando-se IP (Internetwork Protocol) sobre o protocolo X.25.

É aplicado no ponto de referência entre a Rede Pública de Comutação de Pacotes e os OS/MD/QA/NE, que se comunicam com os OS's instalados em rede local.

- Camada 1: X.27, X.21, X.21bis, V.II/V.35, V.28/V.24
- Camada 2: X.25 LAPB [ISO 7776]
- Camada 3: IP ISO 8473 PLP, CCITT X.25 [ISO 8208]

4.4.2.4. Protocolos das Camadas Superiores

Para os protocolos das camadas superiores, deve-se considerar dois tipos de aplicação envolvidas: serviços transacionais e transferência de arquivos.

4.4.2.4.1. Camada de Transporte

Para uso sobre serviços de rede orientados à conexão

Os serviços de transporte para o serviço de rede orientado à conexão devem estar dentro do recomendado pela X.214 e pela ISO 8072 (dentro do que se aplica ao Serviço de Rede Orientado à Conexão). É também necessário que o protocolo de transporte esteja conforme a X.224 e a norma ISO 8073. Deve suportar as classes de serviços 0, 2 e 4, sendo que quando a 4 for exigida, deve-se suportar as classes 0 e 2.

Para uso sobre serviços de rede não orientados à conexão

Para os serviços de rede não orientados à conexão, o serviço de transporte deve estar de acordo com as normas ISO 8072 e ISO 8072/AD2. A operação do protocolo da camada de transporte sobre o serviço de rede não orientado à conexão, como descrito na norma ISO 8348/AD1, deverá usar os elementos da norma ISO 8073/AD2, operação classe 4, sobre o serviço de rede não orientado à conexão (observação: o suporte da operação classe 4 da norma ISO 8073/AD2 é obrigatório).

4.4.2.4.2. Camada de Sessão

Os serviços e protocolos da camada de sessão devem estar de acordo com as normas ISO 8326 e 8327. O protocolo da camada de sessão deve estar de acordo com a definição de protocolos da X.225 e da norma ISO 8327, sendo o suporte à versão 2 deste documento obrigatória. No caso de aplicações transacionais, as unidades funcionais requeridas são Kernel e Duplex. Já para aplicações envolvendo a transferência de arquivos, as unidades funcionais requeridas são Kernel, Duplex, Resynchronize e Minor Synchronize.

4.4.2.4.3. Camada de Apresentação

Os serviços da camada de apresentação devem estar em conformidade com o especificado nas normas X.216 e ISO 8822. É requerida a unidade funcional Kernel, e seus protocolos devem estar de acordo com os especificados na X.226 e na ISO 8823 (modo normal).

4.4.2.4.4. Camada de Aplicação

Vários ASE's são utilizados no perfil de protocolos da camada de Aplicação, dentre eles o ACSE, ROSE, CMISE e FTAM, já descritos no capítulo sobre Modelo OSI. Esta camada será mais detalhada no capítulo sobre Gerenciamento OSI.

4.4.2.5. Protocolos de Comunicação para Interface Qx

A interface Qx é caracterizada por aquela porção do modelo de informação que é compartilhada entre o dispositivo de mediação (MD) e os elementos de rede (NE). Desta forma, a interface Qx deve suportar a transferência de dados bidirecionais para o gerenciamento de sistemas de telecomunicações, sem se preocupar com a estrutura ou o significado das informações de gerenciamento transmitidas nem com a maneira na qual o gerenciamento é obtido como um resultado das trocas de protocolos de aplicação. A idéia aqui é definir os perfis dos serviços e dos protocolos das camadas, os elementos de serviço de aplicação e seus respectivos protocolos e a função de mapeamento dos serviços e dos protocolos devido à ausência das camadas subjacentes (apresentação, sessão e transporte).

A estrutura dos perfis de protocolos é mostrada na figura abaixo. Os serviços e protocolos de comunicação seguem o modelo de referência OSI. São definidos dois perfis de protocolos, A1 e A2, sendo que não existem as camadas de apresentação, sessão e transporte. Devido à eliminação destas camadas, existe a necessidade de se utilizar uma função de mapeamento da camada de aplicação para a camada de rede.

4.4.2.5.1. Perfil A1

Camada Física

A definição de serviços para a camada física está de acordo com a recomendação CCITT X.211. As classes de serviços que devem ser suportadas são as seguintes:

- transmissão síncrona;
- modo de operação half-duplex;
- topologia ponto-multiponto através de bus serial (ISO 8482);
- taxa de transmissão 19,2 kbps ou 64 kbps (velocidades menores podem ser utilizadas).

Além disso, deve fornecer facilidade de ativação e desativação da interface física e transmissão de dados.

Camada de Enlace

A definição do serviço da camada de enlace deve estar conforme a recomendação CCITT X.212. A classe de serviço é o modo orientado à conexão e o protocolo utilizado é o HDLC síncrono. Deve ser oferecido o serviço de estabelecimento, liberação e transferência de dados.

Camada de Rede

A definição dos serviços de rede no modo não orientado à conexão deverá estar de acordo com a ISO 8473. Este protocolo é definido para acomodar uma variedade de funções em diferentes configurações de sub-rede. A única diferença entre o protocolo da camada de rede do perfil A1 para o perfil A2 é que no perfil A2 não existe a necessidade de se utilizar a função de convergência especificada na ISO 8473 AD3.

A Função de Convergência deve estar de acordo com as normas ISO para o serviço subjacente previsto pela norma ISO 8473, sobre sub-redes que provêem o serviço de enlace de dados OSI nos modos orientados e não orientados à conexão. Os conjuntos de funções de convergência definidos são:

- com utillzação de sub-redes ISO 8802.2;
- com utillzação de sub-redes ISO 8208;
- com utillzação de sub-redes ISO 8886.

O procolo utilizado deve estar de acordo com a ISO 8473, mas existem ainda dois subconjuntos de protocolos: Protocolo Inativo de Camada de Rede (Inactive Network Layer Protocol) e o Protocolo sem Segmentação da Camada de Rede (Non-Segmenting Network Layer Protocol).

O Protocolo Inativo de Camada de Rede é utilizado quando se tem conhecimento de que os sistemas terminais estão conectados por uma única sub-rede e de que não é necessária nenhuma função de protocolo completo para prover o serviço de rede no modo não orientado à conexão.
O Protocolo Sem Segmentação é utilizado quando se tem conhecimento de que os sistemas terminais estão conectados em sub-redes distintas e de que o tamanho da unidade de dados é tal que não é necessária a segmentação.

4.4.2.5.2. Perfil A2

Camada Física

A definição dos serviços para a camada física do perfil A2 deve estar de acordo com o especificado na cláusula 6 da norma ISO 8802-3. A taxa de transmissão deve ser de 10 Mbps. O sistema de cabeamento poderá utilizar uma das três opções das listadas abaixo:

- Padrão IEEE 802.3 10 BASE 2;
- Padrão IEEE 802.3 10 BASE 5;
- Padrão IEEE 802.3 10 BASE T.

Camada de Enlace

A camada de enlace deve prover o serviço modo sem conexão e com reconhecimento. O método de acesso empregado deve ser o CSMA/CD. O Controle de Acesso ao Meio (MAC) deve estar de acordo com o especificado na norma ISO 8802.3. A definição do serviço modo LLC (Controle de Enlace Lógico) sem conexão com reconhecimento (tipo 3) deve estar de acordo com o especificado na norma ISO 8802.2/DAD2.

Camada de Rede

A camada de rede do perfil A2 deve possuir as mesmas características da camada de rede do perfil A1, com exceção da utilização da função de convergência.

Visão Global da Função de Mapeamento

Na interface Qx, as camadas de Transporte, Sessão e Apresentação não são especificadas. Em seu lugar, é definida uma Função de Mapeamento para prover o serviço requerido para a camada de aplicação sobre o serviço fornecido pela camada de rede. Nenhum protocolo para a Função de Mapeamento é definida.
A definição do serviço da Função de Mapeamento, a qual provê o serviço de apresentação requerido à camada de aplicação, deverá estar de acordo com a X.216. A Função de Mapeamento deve prover o serviço de apresentação P-DATA, com as primitivas de solicitação e indicação. Quando o ACSE for suportado na camada de aplicação, a função de mapeamento deve prover os serviços de apresentação P-CONNECT, P-REALEASE, P-U-ABORT E P-P-ABORT.

Camada de Aplicação

A camada de aplicação de gerenciamento de redes proverá o serviço do CMISE ao NM-ASE. Os elementos de serviço de aplicação requeridos para este serviço são o CMISE e o ROSE. Algumas aplicações podem requerer a adição do ACSE. As principais características dos elementos da camada de aplicação são idênticas às definidas para a interface Q3.

4.4.3. Arquitetura de Informação

4.4.3.1. Introdução

A arquitetura de informações descreve um modelo orientado a objeto para a modelagem da informação de gerência trocada entre blocos funcionais da TMN. Desse modo, a arquitetura de informação possui os fundamentos para a utilização dos princípios e conceitos do gerenciamento de sistemas OSI, como agente/gerente, domínios e conhecimento de gerenciamento compartilhado, necessários para a organização e o interfaceamento de sistemas de gerenciamento complexos.
Uma aplicação de gerência é uma atividade na qual ocorre um processamento de informações de forma distribuída entre dois ou mais processos cooperantes que trocam informações entre si. Esta troca de informações baseia-se em um sistema gerenciador (controle e monitoração) e um sistema gerenciado (recursos físicos ou lógicos). Para que haja possibilidade de troca de informações entre os dois sistemas (agente/gerente), existe a necessidade de uma visão compartilhada das informações de gerência trocadas e das regras de comunicação empregadas.
Para se garantir a perfeita operabilidade das comunicações agente/gerente, faz-se uso do modelamento das informações trocadas entre os sistemas em termos de objetos gerenciados. Um objeto gerenciado é uma abstração de um recurso físico ou lógico de um sistema gerenciado, definido através de suas características inerentes, ou atributos (ATTRIBUTES), operações de gerenciamento que suporta (ACTIONS), notificações que emite (NOTIFICATIONS) e do seu comportamento (BEHAVIOUR) diante de estímulos externos e internos.

O conjunto de todos os objetos de um sistema gerenciado, juntamente com suas propriedades (atributos, operações, notificações, etc), define a MIB (Management Information Base) do seu sistema.

4.4.3.2. Agentes e Gerentes

Conforme foi dito anteriormente, uma aplicação de gerência é baseada na troca de informações entre um agente e um gerente, sendo que cada um possui as seguintes características:

* agente: coleta informações relativas ao funcionamento dos objetos que gerencia, armazena estas informações na MIB e realiza operações de gerenciamento sobre estes objetos atendendo a solicitações enviadas pelo gerente
* gerente: coleta informações sobre os objetos gerenciados junto aos agentes, processa as informações e solicita aos agentes que executem as funções de gerenciamento a fim de controlar o funcionamento do objeto gerenciado

Como pode ser visto na figura 4.10, toda interação realizada entre agente e gerente é abstraída em termos de operações e notificações trocadas entre eles. Esta troca de operações e notificações é realizada sempre através do Serviço e Protocolo de Informações de Gerenciamento Comum (CMIS/CMIP), conforme mostrado na figura 4.11.

O CMIP comporta vários tipos de PDU's (Protocol Data Unit) que são mapeadas em operações equivalentes sobre os objetos gerenciados, os quais representam os recursos gerenciados. Estas PDU's são basicamente as seguintes:

M-GET: executa a leitura dos atributos de objetos gerenciados;
M-SET: executa a modificação dos atributos de objetos gerenciados;
M-ACTION: executa uma ação qualquer sobre um objeto gerenciado;
M-CREATE: cria uma instância de um objeto gerenciado;
M-DELETE: remove uma instância de um objeto gerenciado;
M-EVENT-REPORT: emite uma notificação sobre a ocorrência de um evento associado a um objeto gerenciado.

Além destas mensagens de protocolos, são definidas facilidades adicionais que permitem selecionar o grupo de objetos sobre o qual é aplicável uma dada operação. A facilidade denominada escopo nos permite selecionar um grupo de instâncias de objetos sobre os quais é realizada uma única operação. A facilidade de filtro, por sua vez, permite definir um conjunto de testes aplicáveis a um grupo de instâncias de objeto, anteriormente selecionado através da facilidade de escopo, de modo a extrair um subgrupo ainda menor sobre o qual deve ser efetuada uma operação de gerenciamento. Por último, há a facilidade de sincronização, que permite sincronizar várias operações de gerenciamento a serem realizadas sobre instâncias de objetos selecionadas através das facilidades de escopo e filtro.

4.4.3.3. Modelo de Informação

O modelo de informação é um recurso utilizado para descrever as informações (dados + comportamento) relativas ao sistema, independentemente do modo como estas são armazenadas ou utilizadas.

Os principais objetivos da utilização do modelo de informação como descritivo do sistema são:
identificar e descrever os dados necessários a um sistema de informação de uma forma clara e completa;
suportar o projeto de novos sistemas de informação;
suportar a integração de modelos de dados preservando a consistência dos mesmos;
assegurar uma arquitetura de dados que possibilite o desenvolvimento de sistemas integráveis ao invés de sistemas que duplicam os dados existentes;

O modelo de informação define a estrutura lógica da informação de gerência. A informação de gerência é modelada em termos de objetos gerenciados, seus atributos, operações de gerenciamento que suporta e as notificações que pode emitir. Também deve descrever os princípios de nomeação dos objetos e dos atributos, de forma que estes possam ser identificados e acessados pelos protocolos de gerenciamento [X.720].

Esta técnica de modelagem de informação pode ser utilizada para representar qualquer informação contida e/ou exteriorizada por um sistema de informação, por exemplo, para descrever a informação contida em uma base de dados.

É interessante observar que a modelagem de informação acrescenta um nível a mais de abstração do que aqueles que trabalhamos normalmente no desenvolvimento de sIstemas de informação (projeto lógico e projeto físico). Isso implica em uma separação por "assuntos", ou seja, o modelo de

informação (que representa o conhecimento) é separado da representação lógica (esquemas relacionais ou hierárquicos), que por sua vez é separado da representação física. O modelo de informação deve abstrair das tecnologias de armazenamento e utilização da informação, assim como os programas devem abstrair das linguagens e arquiteturas de computadores.
O modelo de informação é baseado em objetos gerenciados, que são abstrações dos recursos (físicos ou lógicos) a serem gerenciados. As operações de gerenciamento a serem efetuadas sobre os recursos físicos, na realidade são realizadas sobre os objetos gerenciados. O efeito destas operações realizadas na MIB devem ser refletidas no recurso gerenciado (equipamento de telecomunicações), de forma que a operação de gerência seja efetuada completamente (tanto na MIB como no equipamento). É importante observar que se um recurso a ser gerenciado não for representado por um objeto, ele será invisível ao gerenciamento. A forma de interação entre o objeto gerenciado e seu recurso físico associado não faz parte do padrão.
Caracterizando-se os objetos gerenciados, veremos que existem vários objetos que compartilham as mesmas definições, mesmos atributos, etc. Estes objetos com características semelhantes podem ser agrupados em uma classe. As classes são definidas como uma coleção de pacotes (packages), cada um definido como uma coleção de atributos, operações, notificações e comportamento. Estes pacotes podem ser mandatórios ou condicionais, baseados em alguma condição de estado. Temos então que um objeto gerenciado é um caso particular de uma classe, ou seja, podemos dizer que um objeto é uma instância de uma classe.
A definição de uma classe consiste de:

- posição da classe de objetos gerenciados na hierarquia de herança;
- coleção de pacotes mandatórios de atributos, notificações e comportamento;
- coleção de pacotes condicionais de atributos, operações, notificações e comportamento, junto com a condição na qual cada pacote estará presente;
- estrutura do pacote (os atributos visíveis na fronteira do objeto, as operações aplicáveis a estes objetos, o comportamento deste objeto e as notificações que podem ser emitidas pelo objeto).

A forma como estas definições são dispostas (sintaxe) seguem os padrões GDMO (Guidelines for Definition of Managed Objects) e ASN.1 (Abstract Syntax Notation One).

4.4.3.4. Conhecimento de Gerência Compartilhado (SMK)

Cria um contexto para troca de informações entre um gerente (realiza operações) e um agente (envia notificações) da rede de gerência, de forma a permitir a interoperabilidade na comunicação dos sistemas numa aplicação de gerência, usando para tanto os mesmos protocolos e com um conhecimento comum sobre os objetos gerenciados.

4.4.3.5. Domínio de Gerência

Organização dos objetos gerenciados em conjuntos, divididos segundo critérios operacionais, tecnológicos, geográficos e até políticos.

5. Gerenciamento OSI

5.1. Introdução

O gerenciamento OSI é a base da arquitetura TMN, uma vez que a TMN é uma rede de computadores que interfaceia com uma rede de telecomunicações e troca informações relativas à gerência da rede de telecomunicações. Esta troca de informações , sendo feita por uma rede de computadores, segue o modelo aberto descrito pela ISO como modelo OSI (Open Systems Interconnection). Uma vez que o modelo OSI é utilizado, nada mais lógico do que utilizar os conceitos de gerenciamento OSI como base para o TMN.
O gerenciamento OSI é, então, caracterizado por três elementos básicos:

- gerentes
- agentes
- objetos gerenciados

Os gerentes, conforme já foi visto anteriormente, é a entidade que controla os objetos gerenciados, através de operações enviadas ao agente e recebe as notificações enviadas espontâneamente pelo agente.
O agente vai realizar as operações de gerência solicitadas pelo gerente sobre os objetos e transmitir as notificações emitidas pelos objetos ao gerente.

Um objeto gerenciado é a representação de um recurso a ser gerenciado. Por exemplo, uma placa ou um elemento de software é um recurso gerenciado e o "objeto placa" ou o "objeto software" é a representação deste recurso. Detalhes sobre os objetos gerenciados serão vistos no capítulo referente a modelo de informação.

5.2. Modelo de Gerência OSI

O gerenciamento OSI é dividido em três tipos:

- Gerenciamento de sistemas
- Gerenciamento de camada
- Operação de camada

A operação da camada gerencia uma única instância de comunicação em uma camada. Este é o gerenciamento que possui as menores exigências em termos de funções de apoio, já que não necessita de um protocolo específico de gerência (as informações de gerência são trocadas utilizando-se o protocolo normal de cada camada).
O gerenciamento de camada é realizado sobre objetos relacionados com as atividades de comunicação da mesma camada., através da utilização de protocolos de gerenciamento específicos e funções de apoio internas às camadas. Estes protocolos específicos não realizam serviços para as camadas superiores, sendo independentes dos protocolos de gerenciamento de outras camadas. O gerenciamento da camada exige a presença de funções de apoio em todas as camadas inferiores à camada envolvida (por exemplo, se a camada a ser gerenciada é a de rede, necessita-se das funções de suporte das camadas de enlace e física).
O gerenciamento de sistemas possui controle sobre quaisquer objetos pertencentes ao sistema através da utilização de um protocolo de gerenciamento de sistemas na camada de aplicação (normalmente o CMISE). Para realizar suas funções, necessita-se de funções de apoio nas sete camadas.
O modelo de gerenciamento OSI se baseia na idéia de que um SMAP (Processo de Aplicação de Gerência de Sistemas), que pode fazer o papel de agente ou gerente, se comunica com um SMAE (Elemento de Aplicação de Gerência de Sistemas) para realizar as suas funções de gerenciamento.
O SMAE é formado por um conjunto de ASE's (Elemento de Serviço de Aplicação) que provêem a infra-estrutura (na camada de aplicação) para o transporte de informações de gerência. Um ASE é uma entidade que realiza uma função (que pode ser comum - CASE - Elemento de Serviço de Gerenciamento Comum ou específica - SASE - Elemento de Serviço de Gerenciamento Específico). Os CASE realizam funções genéricas, que podem ser utilizadas por vários SASE's. Por exemplo, o ROSE - Elemento de Serviço de Operações Remotas, é um CASE utilizado pelo CMISE (que é um SASE). Desta forma, as entidades de serviço se completam para realizar as suas funções.
No nosso caso, o SMAE é composto pelos seguintes ASE's:

- SMASE - Systems Management Application Service Element
- CMISE - Common Management Information Service Element
- ACSE - Association Control Service Element
- ROSE - Remote Operations Service Element

O SMASE (Systems Management Application Service Element) é um ASE específico para gerenciamento. Ele provê vários serviços que estão disponíveis para o gerente da rede e para as aplicações (SMAP) que implementam as funções de gerência de rede.
O CMISE também é um ASE específico para gerência de rede. Ele define o serviço e os procedimentos usados para a transferência das CMIPDU's e provê um meio de troca de informações para as operações de gerenciamento.
O ACSE e o ROSE são ASE's comuns, ou seja, não são utilizados exclusivamente para gerenciamento. O ACSE (Association Control Service Element) é invocado quando se necessita estabelecer uma associação (obs: o termo associação é utilizado na camada de aplicação para definir o que chamamos de conexão nas outras camadas) e o ROSE (Remote Operations Service Element) para realizar a troca de informações entre sistemas remotos (como por exemplo, uma operação solicitada de um gerente para um agente). O estabelecimento de uma associação entre dois SMASE's é realizada através de uma negociação de contexto que indica o conhecimento inicial de gerenciamento compartilhado para aquela associação, incluindo os vários ASE's envolvidos.
O fluxo de dados de gerenciamento se estabelece, portanto, da seguinte forma:
1. uma aplicação de gerência invoca o CMISE (através de uma função do SMASE ou não) solicitando a execução de uma operação de gerência
2. o ACSE estabelece uma associação com a outra parte envolvida (por exemplo, o agente)
3. o CMISE monta a mensagem e, já que a associação já está estabelecida, a operação remota é realizada através do ROSE (a mensagem CMIP a ser transmitida corresponde a um dos parâmetros da operação)

4. o ROSE monta a mensagem e a transmite pela rede, utilizando o serviço da camada de apresentação e) o CMIP remoto desmonta a mensagem recebida e executa a operação solicitada sobre o objeto gerenciado
Estes conceitos do gerenciamento OSI são aplicáveis também no TMN, com algumas modificações em termos do modelamento de informação (MIB). No ponto de vista do funcionamento, ou seja, da forma como as operações são realizadas (do gerente para o agente), não há modificações. Isso quer dizer que tanto no gerenciamento OSI quando na TMN teremos a presença do SMASE, CMISE, ACSE e ROSE, trabalhando da mesma forma. Porém, os objetos definidos para representar recursos de uma rede OSI são diferentes dos objetos necessários para se representar uma rede de telecomunicações.
É importante ainda lembrar que as operações de gerência só são padronizadas na comunicação entre o gerente e o agente e que a implementação das operações nos recursos físicos está fora do escopo dos padrões, ou seja, a forma como um objeto é criado na MIB é padronizada (operação CREATE do CMIP), mas a forma como a criação do objeto acarretará na criação do recurso correspondente no seu equipamento é problema de implementação, fora do escopo dos padrões.

6. Modelo de Informação

6.1. Introdução

O modelo de informação é um dos aspectos mais importantes a se observar em uma TMN e é o ponto de partida para a implementação de uma rede consistente e confiável. O modelo de informação, na realidade, é o conjunto de objetos gerenciados que representam o equipamento (físico ou lógico) que vai ser gerenciado, mas modelado do ponto de vista da gerência que se quer realizar sobre ele. Isso tem a sua base na recomendação M.3020, que especifica uma metodologia para desenvolvimento de modelos de informação.
Antes de entrar no mérito da metodologia propriamente dita, vale a pena incluir aqui algumas definições importantes, como a definição de serviços e funções.

6.2. Serviços de Gerência

Um serviço é uma área de atividade de gerência que provê o suporte a um aspecto de operação. Os serviços, na realidade, são os objetivos a se alcançar com a implementação de uma TMN. Os serviços de gerência listados na M.3200 são:

- administração do usuário
- gerência de provisionamento da rede
- administração de contabilização, cobrança e tarifa
- qualidade de serviços e administração de desempenho de rede
- administração de análise e medições de tráfego
- gerência de tráfego
- roteamento e administração de análise de dígito
- gerência de manutenção
- administração da segurança
- gerência de logística

Os serviços de gerência são decompostos em grupos de conjuntos de funções, conjuntos de funções e funções. A definição de um serviço assegura que todas as funções necessárias à realização daquela atividade serão suportadas pela implementação.

Um grupo de conjunto de funções é uma parte menor do serviço TMN. O serviço de Gerência de Tráfego, por exemplo, visa a obtenção do maior número possível de chamadas completadas com sucesso através da maximização do uso de todos os equipamentos e facilidades disponíveis em qualquer situação de tráfego e ainda do controle de fluxo de tráfego para a utilização da capacidade máxima da rede. O serviço de Gerência de Tráfego possui como grupos de conjuntos de funções a Monitoração do Estado da Rede, Monitoração do Desempenho da Rede e Ações de Controle de Gerência de Tréfego.

6.3. Funções de Gerência

As funções de gerência, conforme definido na recomendação M.3400, são a menor parte do serviço percebíveis pelo usuário deste serviço. Uma função corresponde a uma série de ações sobre um objeto gerenciado a fim de realizar um serviço. As funções podem ser reutilizáveis, ou seja, serviços diferentes podem utilizar as mesmas funções para atingir objetivos diferentes. Um exemplo disto pode ser alguma função de monitoração de desempenho. Esta função pode, por exemplo, ser responsável pela coleta de dados de desempenho e, através de um processo qualquer, pode servir para a indicação de alarmes, quando o desempenho atingir um determinado nível. Portanto temos uma função que serve tanto para desempenho quanto para falhas.

6.3. Metodologia para Definição do Modelo de Informação

Um modelo de informação deve ser definido, então, de forma que os objetos gerenciados suportem os Serviços de Gerência definidos. Portanto podemos visualizar os passos para a definição de um modelo de informação:

Desta forma, o modelo será consistente e suportará os serviços definidos. Iremos agora analisar a forma como os objetos devem ser descritos para escrever o modelo de informação. Para isso, veremos todas as características que devemos definir para os objetos e a forma como eles devem ser descritos (GDMO e ASN.1).

6.4. Modelamento de Objetos

Um objeto gerenciado é uma representação para o sistema de gerência do recurso de telecomunicações a ser gerenciado. Um gerente não "enxerga" um recurso a ser gerenciado, a não ser que ele esteja representado por um objeto. O objeto gerenciado, então, não é apenas uma sintaxe representando um recurso, mas sim a definição da capacidade de gerenciamento do recurso. Utilizando-se técnicas de orientação a objetos, definimos classes, atributos, etc para modelar o recurso a ser gerenciado.
A modelagem orientada a objetos nos traz algumas vantagens e conceitos herdados da orientação a objetos, como por exemplo reutilização das classes, herança, encapsulamento, etc.
A orientação a objeto é extensível aos protocolos e serviços relacionados. O CMIP, por exemplo, utiliza os princípios de orientação a objetos, mas o banco de dados onde a MIB é implementada não precisa utilizar a orientação a objeto (geralmente é um banco de dados relacional). Um objeto gerenciado pode ser visto como uma esfera opaca recobrindo o recurso real, com uma "janela", através da qual a informação de gerenciamento pode passar, conforme pode ser visto na figura 6.3. Esta informação consiste em:

- operações de gerenciamento no objeto
- resultado das operações que retorna
- notificações que podem ser geradas espontaneamente pelo objeto gerenciado quando um evento ocorre no recurso

e corresponde aos elementos do serviço CMIS.

A figura 6.3 nos ilustra o conceito de encapsulamento, muito importante para análise orientada a objetos. A idéia de encapsulamento é justamente esconder os detalhes de implementação do objeto, permitindo o acesso a este apenas através de algumas funções bem definidas. De uma maneira simplista, pode-se fazer uma analogia com um programa de computador, por exemplo. Quando se faz um programa, o implementador escreve o código fonte do programa, onde são definidas todas as variáveis, funções, etc. Mas geralmente não se deseja que o usuário tenha contato com o programa a este nível. Gera-se então um executável, que irá esconder os detalhes de implementação do usuário, ou seja, irá permitir o acesso do usuário ao programa através de uma interface definida e controlada (desta forma podemos dizer que o arquivo executável é um encapsulamento do código fonte).
Algumas vantagens do encapsulamento são:

- protege as variáveis do objeto de serem corrompidas por outros objetos (proteção contra acesso não autorizado, valores inconsistentes, etc);
- esconde a estrutura interna do objeto de modo que a interação com este objeto seja relativamente simples e, à medida do possível, padrão.

É possível, dada a definição de um objeto gerenciado, que vários objetos satisfaçam as mesmas condições, até mesmo no mesmo sistema. Um exemplo disso é um log ou uma conexão. Vários logs de um sistema são gerenciados da mesma maneira, formando assim uma classe de objetos gerenciados, onde cada log em particular é uma instância desta classe.
Para se definir completamente uma classe, devemos caracterizar:

- as propriedades ou características visíveis na fronteira do objeto gerenciado, ou seja, os **atributos** e seus valores;
- as **operações de gerenciament**o aplicáveis à classe, sendo que algumas destas operações afetam apenas os atributos enquanto outras afetam o objeto como um todo;
- as **regras de filtragem** aplicáveis a filtros CMIS;
- o **comportamento** que exibe em resposta às operações de gerenciamento;
- as **notificações** que emite e as circunstâncias nas quais são emitidas;
- os **pacotes** que pode incluir;
- sua **posição na árvore de herança**;
- as **name bindings**, que definem o relacionamento entre as classes na árvore de contenção.

6.4.1. Hierarquia de Herança

Um conceito muito importante para o modelo de informação é o de Hierarquia de Herança, ou Hierarquia de Classes. Na hierarquia de herança, o conjunto de classes formam uma estrutura na qual existe o conceito de superclasses e subclasses. Uma subclasse herda todas as propriedades de sua superclasse, de maneira irrestrita, independentemente da necessidade ou não destas propriedades. Note que este conceito de superclasse e subclasse é relativo, pois uma classe pode ser subclasse de uma e superclasse de outra ao mesmo tempo.

A top é uma classe genérica definida na X.721, posicionada no topo da hierarquia.

A hierarquia de herança lida com especialização de classes e herança de atributos.

Para enxergar melhor o conceito de hierarquia de herança, tomemos um exemplo prático. Uma central telefônica CPA possui, para realizar a comutação, uma placa de comutação que possui um software baseado em tabelas. Vejamos a árvore de herança deste caso e depois a compararemos com a árvore de contenção.

Vamos supor, então, que as classes que estão nos quadrinhos cinzas foram as classes criadas por nós para atender ao modelo de informação desta central, derivadas das classes que estão nos quadrinhos brancos (que hipoteticamente já existem nos modelos genéricos). Supondo então, que as classes possuem como atributos:

Classe Software
- identificação
- linguagem

Classe Software de Comutação
- n. de tabelas de comutação
- versão

Classe Equipamento
- localidade

Classe Placas da Central
- tipo de barramento
- nível de tensão placa de comutação
- localização da placa
- estado operacional da placa

Classe Elemento gerenciado
- identificação

Classe Central
- prefixo
- localização

Classe Bastidor
- endereço

Tendo-se o conceito de que uma subclasse herda as características de sua superclasse, temos que os atributos da classe placa de comutação, por exemplo, passam a ser, além dos seus atributos originais (localização e estado operacional), os atributos herdados da classe placas da central (tipo de barramento e nível de tensão). Ou seja, se o gerente enviar ao agente um GET do valor do atributo tipo de barramento da placa de comutação, o agente vai consultar a classe placa de comutação, não vai achar o atributo, vai para a superclasse da placa de comutação (classe placas da central). Na classe placas da central o agente achou o atributo tipo de barramento da placa, lê o seu valor e manda a resposta ao gerente, como se o valor do atributo tivesse sido lido da classe placa de comutação. Portanto temos que a classe placa de comutação é uma especialização da classe placas da central.

A especialização de uma classe é conseguida através da:

- adição de novos atributos;
- extensão ou restrição da faixa de valores para um atributo existente;
- adição de novas operações e notificações às operações e notificações existentes;
- extensão ou restrição da faixa de valores de argumentos de operações e notificações existentes.

A hierarquia de herança se torna importante à medida em que facilita a reutilização dos atributos já definidos para as classes superiores. Desta forma, se ocorre a necessidade de se criar uma classe nova (que não existe nos modelos genéricos padronizados), deve-se sempre tentar criá-las especializando as classes padrões já existentes.

6.4.2. Hierarquia de Contenção

A Hierarquia de Contenção (ou Inclusão) refere-se, como o próprio nome diz, a uma relação onde um objeto (chamado objeto superior) contém outro objeto (denominado subordinado). Nota-se aqui a primeira e mais importante diferença entre as hierarquias de herança e contenção. Enquanto a hierarquia de herança diz respeito a classes de objetos, a hierarquia de contenção lida com instâncias das classes.
Esta relação de contenção pode ser utilizada para modelar as hierarquias existentes no mundo real, como por exemplo módulos, sub-módulos e componentes eletrônicos, ou no nosso exemplo da central telefônica, central, bastidor, placa de comutação, software de comutação. Isto significa que a central contém um bastidor que contém uma placa de comutação que contém um software de comutação (figura 6.6).
É importante distinguir bem o fato de que as duas hierarquias são, na realidade, duas formas diferentes de se enxergar a mesma coisa: o modelo de informação (na realidade, a relação entre as classes está definida em campos específicos do documento GDMO - detalhes nos ítens seguintes). Podemos até dizer, de uma maneira um pouco grosseira, que a árvore de inclusão se aproxima da realidade. Ela é dinâmica, ou seja, na medida em que os objetos vão sendo criados e deletados, eles aparecem e somem da árvore de inclusão.

Já a árvore de herança não tem esta característica dinâmica. Ela é criada para definir as características das classes e a estrutura de especialização delas e não muda, não importando quantas instâncias de cada classe existam no elemento gerenciado, e até mesmo não importando se existe alguma instância da classe. Por exemplo, pode não existir nenhuma instância da classe Placa de Comutação na minha central; por conseqüência, não existirá nenhuma Placa de Comutação na minha árvore de contenção (e também por conseqüência, não haverá nenhum Software de Comutação, pois uma placa de comutação contém um software de comutação, e se não existe a placa, não existirá o software). Mas na minha árvore de herança as classes Placa de Comutação e Software de Comutação existirão sempre.
A árvore de contenção também vai ser responsável pela formação do nome da instância do objeto gerenciado. O nome da instância é um DN (Distinguished Name), formado pela concatenação do RDN's (Relative Distinguished Name). Cada instância, quando criada, recebe um nome (id) para que possa ser identificada pelo protocolo e essa identificação se chama RDN. Para formar o nome completo da instância na árvore de contenção, ou seja, o DN, o processo é simplesmente concatenar os RDN's das classes superiores, começando pela mais alta.

Por exemplo, o DN do objeto Software de Comutação 2 é formado pelos RDN's de suas classes superiores, ou seja: Placa de Comutação 2, Bastidor e Central. Portanto, o DN do Software de Comutação 2 fica {1 2 3 4}. O Software de Comutação 1 ficaria com o DN={1 2 5 6}.

6.5. Guia para Definição de Objetos Gerenciados (GDMO)

6.5.1. Classes

O gabarito das classes é formado por:
<class-label> MANAGED OBJECT CLASS
DERIVED FROM <class-label>, <class-label>;
CHARACTERIZED BY <package-label>, <package-label>;
CONDITIONAL PACKAGES <package-label> PRESENT IF condition-definition, <package-label> PRESENT IF condition-definition ;
REGISTERED AS object-identifier ;
Obs:

- [xxxxxxx] - pode ou não aparecer;

- [xxxxxxx]* - pode aparecer zero ou mais vezes;
- <xxxxxx> - string que deve ser substituída em cada instância;
- Letras maiúsculas representam palavras chaves que devem estar presentes em cada instância da template, a menos que esteja dentro de [];

onde:

- DERIVED FROM indica a superclasse ou superclasses da qual esta classe é herdada;
- CHARACTERIZED BY permite que um ou mais "packages" sejam incluídos na definição da Classe de Objeto;
- CONDITIONAL PACKAGES permite que um ou mais "packages" condicionais sejam incluídos;

Exemplo de gabarito de classe:

classeExemplo MANAGED OBJECT CLASS
DERIVED FROM "CCITT Rec. X.721 (1992) | ISO/IEC 10165-2 : 1992":top;
CHARACTERIZED BY pacoteExemplo1;
CONDITIONAL PACKAGES pacoteExemplo2 PRESENT IF "o recurso real suportar estes dados";
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) managedObjectClass(3) exampleClass(0)};

6.5.2. Pacote

O pacote é a coleção de atributos, notificações, operações e comportamento presentes no objeto. O pacote pode ser de dois tipos:

- condicionais: podem ou não estar presentes, de acordo com certas regras a serem seguidas pela instância;
- mandatórios: obrigatoriamente devem estar presentes em todas as instâncias da classe.

O gabarito de pacote é:

<pakage-label> PACKAGE
BEHAVIOR <behavior-definition-label>;
ATTRIBUTES <attribute-label> propertylist [,<parameter-label>*];
ATTRIBUTE GROUPS <group-label> <attribute-label>,
ACTIONS <action-label> <parameter-label>;
NOTIFICATIONS <notification-label> <parameter-label>;
REGISTERED AS object-identifier;

supporting productions

propertylist -> [REPLACE-WITH-DEFAULT] [DEFAULT VALUE value-specifier] [INITIAL VALUE value-specifier] [PERMITTED VALUES type-reference] [REQUIRED VALUES type-reference] [GET| REPLACE| GET-REPLACE] [ADD| REMOVE| ADD-REMOVE]

onde:

- BEHAVIOUR permite mostrar a semântica do "package" através de texto não processável;
- ATTRIBUTES lista todos os atributos que fazem parte do "package" (associado a cada atributo existem zero ou mais parâmetros);
- ACTIONS e NOTIFICATIONS incluem, por referência, ações e notificações que são parte do "package";

As propriedades que podem ser especificadas são:

- REPLACE-WITH-DEFAULT: o atributo tem um valor default que pode ser atribuído por operação de gerência;
- DEFAULT VALUE: especifica que o valor default deve ser usado na criação do "package", caso nenhum outro valor seja fornecido;
- INITIAL VALUE: valor obrigatório inicial;
- PERMITTED VALUES: restrições nos valores de atributos, expressos em ASN.1;
- REQUIRED VALUES: especifica os valores que o atributo deve ser capaz de tomar;
- GET/REPLACE/GET-REPLACE: operações permissíveis nos atributos de valor simples;
- ADD/REMOVE/ADD-REMOVE: operações permissíveis nos atributos de conjunto de valores.

Exemplo de gabarito de pacote:

pacoteExemplo2 PACKAGE
BEHAVIOUR exemploBehaviourClasse;
ATTRIBUTES objectName GET, qOS-Error-Cause GET, qOS-Error-Counter PERMITTED VALUES AttributeModule.QOSCounterRange REQUIRED VALUES AttributeModule.QOSCounterRange GET;
ATTRIBUTE GROUPS qOS-Group;
NOTIFICATIONS erroProtocolo;
REGISTERED AS { joint-iso-ccitt ms(9) smi(3) part4(4) package(4) examplePack2(1) };

6.5.3. Atributos

Os atributos são as características reais do recurso contidas no objeto. Cada atributo representa uma propriedade do recurso que o objeto representa, como a identificação, as características operacionais, os estados correntes, o endereço, etc.

Os atributos possuem regras de acesso (leitura, escrita, leitura-escrita) e regras com as quais ele pode ser localizado através de filtro de pesquisa.

O gabarito do atributo é o seguinte:

<attribute-label> ATTRIBUTE
DERIVED FROM <attribute-label> /WITH ATTRIBUTE SYNTAX type-reference;
[MATCHES FOR qualifer [, qualifer]*;]
[BEHAVIOR <behavior-definition-label>[,<behavior-definition-label>]*;]
[PARAMETER <parameter-label> [,<parameter-label>]*;]
REGISTERED AS object-identifier;

supporting productions
qualifier: EQUALITY / ORDERING / SUBSTRINGS / SET - COMPARISON / SET - INTERSECTION
onde:

- DERIVED FROM indica que a definição do atributo deriva de outra definição existente;
- WITH ATTRIBUTE SYNTAX especifica o tipo de dado ASN.1;
- MATCHES FOR define os tipos de testes que podem ser aplicados para um valor de atributo (filtragem para operações de gerência);
- BEHAVIOUR descreve, em texto, o comportamento do atributo e não inclui semântica processável por computador;
- PARAMETER permite que a definição do parâmetro de erro seja associada com o comportamento do atributo

Os qualificadores são:

- EQUALITY: um valor de atributo pode ser comparado por igualdade a um dado valor;
- ORDERING: um valor de atributo pode ser comparado a um dado valor para determinar qual é o maior valor;
- SUBSTRINGS: um valor de atributo pode ser testado pela presença de um "substring";
- SET-COMPARISION: um atributo de conjunto de valores pode ser comparado a um dado conjunto de valores para determinar a relação de superconjunto e subconjunto;
- SET-INTERSECTION: um atributo de conjunto de valores pode ser comparado a um dado conjunto de valores para determinar se existe uma intersecção não nula.

Exemplo de gabarito de atributo:

objetcName ATTRIBUTE
WITH ATTRIBUTE SYNTAX AttributeModule.ObjectName;
MATCHES FOR EQUALITY;
REGISTERED AS {joint-iso-ccitt ms(9) smi(3) part4(4) attribute(7) objectName(0) };

6.5.4. Notificações

As notificações são os eventos enviados espontaneamente do objeto gerenciado para o gerente para relatar a ocorrência de alguma coisa. Devido ao fato da notificação ocorrer sem que haja a solicitação do gerente, ela é considerada uma mensagem assíncrona. As notificações podem ser transmitidas ao gerente ou armazenadas em logs.

O gabarito de notificação é o seguinte:

<notification-label> NOTIFICATION
[BEHAVIOR <behavior-definition-label>[,<behavior-definition-label>]*
[PARAMETERS <parameter-label> [,<parameter-label>]*;]
[WITH INFORMATION SYNTAX type-reference [AND ATTRIBUTE IDS <field-name> <attribute-label>
[,<field-name> <attribute-label>]*] ;]
[WITH REPLY SYNTAX type-reference ;]
REGISTERED AS object-identifier ;

onde:

- BEHAVIOUR define o comportamento da notificação, os dados que devem ser especificados com a notificação, os resultados que a notificação pode gerar e seus significados;

- PARAMETERS identifica a informação do evento ou parâmetros de resposta do evento ou falhas de processamento associados com o tipo de notificação;
- WITH INFORMATION SYNTAX identifica o tipo de dados ASN.1 que descreve a estrutura de informação da notificação que é transportada no protocolo de gerência;
- WITH REPLY SYNTAX identifica o tipo de dados ASN.1 que descreve a estrutura da informação de resposta da notificação que é transportada pelo protocolo de gerência;

6.5.5. Comportamento

Os objetos gerenciados exibem certas características comportamentais, incluindo como o objeto reage a operações desenvolvidas nele e as restrições a seu comportamento, sendo que todas as instâncias da mesma classe exibem o mesmo comportamento.
O comportamento define:

- a semântica dos atributos, operações e notificações;
- a resposta a operações de gerência invocadas sobre os objetos gerenciados;
- as circunstâncias sobre as quais as notificações serão emitidas;
- as dependências entre os valores de atributos particulares;
- os efeitos de relacionamento nos objetos gerenciados participantes.

exemploBehaviourClasse BEHAVIOUR
DEFINED AS
" Descrição do comportamento da classe de objetos gerenciados, incluindo como os atributos adquirem certos valores e o que estes significam, quais as circunstâncias ocasionam a geração de notificações, etc. ";

6.5.6. Name Bindings

A template de Name Binding nos permite definir diversas relações de nomeção para as classes de objetos gerenciados para que possamos localizá-los na árvore de inclusão. Esta template indica qual atributo será utilizado como nomeador, para compor o DN da instância.
<name-binding-label> NAME BINDING
SUBORDINATE OBJECT CLASS <class-label> [AND SUBCLASSES];
NAMED BY SUPERIOR OBJECT CLASS <class-label> [AND SUBCLASSES];
WITH ATTRIBUTE <attribute-label>;
[BEHAVIOUR <behaviour-definition-label> [, <behaviour-definition-label>*];]
[CREATE [<create-modifier> [, <create-modifier], [parameter-label>]*;]]
[DELETE [<delete-modifier>], [<parameter-label>]*;]
REGISTERED AS object-identifier
Onde:
Create-modifier: WITH-REFERENCE-OBJECT | WITH-AUTOMATIC-INSTANCE-NAMING
Delete-modifier: WITH-REFERENCE-OBJECT | WITH-AUTOMATIC-INSTANCE-NAMING
Exemplo de template de Name Binding:
ExemploNameBinding NAME BINDING
SUBORDINATE OBJECT CLASS classeExemplo;
NAMED BY SUPERIOR OBJECT CLASS "CCITT Rec. X.721 (1992) | ISSO/IEC 10165-2 : 1992":system;
WITH ATTRIBUTE objectName;
BEHAVIOUR behaviourInclusao;
CREATE WITH-AUTOMATIC-INSTANCE-NAMING parmErroCreate;
DELETE DELETES-CONTAINED-OBJECTS;
REGISTERED AS {joint-isso-ccitt ms(9) smi(3) part4(4) nameBinding(6) exemploNB(0)};

6.5.7. Ações

A template de ação nos permite definir o comportamento e a sintaxe associada com um tipo de ação em particular. O comportamento especificará a funcionalidade da ação em termos de seu efeito sobre a classe de objeto gerenciado.
<action-label> ACTION
[BEHAVIOUR <Behaviour-definition-label> [, Behaviour-definition-label>*];
[MODE CONFIRMED;]
[PARAMETERS <parameter-label> [, <parameter-label>*];]
[WITH INFORMATION SYNTAX type-reference;]
[WITH REPLY SYNTAX type-reference;]
REGISTERED AS object-identifier;

Exemplo:
Activate ACTION
BEHAVIOUR behaviourActivate
MODE CONFIRMED;
WITH REPLY SYNTAX ActionModule.RespostaActivate;
REGISTERED AS {joint-isso-ccitt ms(9) smi(3) part4(4) action(9) activate(1) };

6.5.8. Operações de Gerência

As operações de gerenciamento podem ser de dois tipos:

- operações orientadas a atributos;
- operações sobre objetos gerenciados como um todo.

6.5.8.1. Operações Orientadas a Atributos

As operações orientadas a atributos são:

- GET ATTRIBUTE VALUE: operação de leitura de valor de atributos, onde os atributos a serem lidos são especificados na operação;
- REPLACE ATTRIBUTE VALUE: esta operação altera os valores dos atributos especificados com os novos valores conhecidos;
- SET WITH DEFAULT VALUE: operação para substituir o valor de alguns dos atributos do objeto gerenciado pelo seu valor default, definido como parte da especificação da classe de objeto em questão;
- ADD MEMBER: aplicável a atributos cujos valores são conjuntos acessíveis para escrita. Para cada conjunto especificado de valores de atributo, esta operação substitui os valores de atributos existentes pelo conjunto união do conjunto existente com o conjunto especificado nesta operação;
- REMOVE MEMBER: aplica-se a atributos cujos valores são conjuntos acessíveis para escrita. Para cada conjunto especificado de valores de atributo, esta operação substitui o conjunto existente de valores de atributo pelo conjunto diferença entre o conjunto já existente e o conjunto especificado nesta operação.

6.5.8.2. Operações Orientadas a Objetos

Estas operações aplicam-se a objetos gerenciados como um todo e seus efeitos geralmente não se limitam a modificar os valores dos seus atributos. As operações são:

- CREATE: requisita a criação e iniciação de um objeto gerenciado. Designa valores a todos os atributos do objeto criado, baseado na definição da classe do objeto e nas informações fornecidas na própria operação;
- DELETE: requisita que o objeto gerenciado remova a si mesmo. Caso este objeto possua outros objetos contidos nele ou possua relação com outros objetos, o seu comportamento depende da definição da classe à qual pertence;
- ACTION: requer que o objeto gerenciado execute a ação especificada e indique o seu resultado. A ação e a informação opcional associada são parte da definição da classe do objeto gerenciado.

6.6. Notação de Sintaxe Abstrata (ASN.1)

Devido à complexidade dos dados a serem manipulados e a diferença de representação destes nas diversas máquinas existentes levaram a criação de uma forma de representação de dados mais eficiente. Desta forma, criou-se duas formas de representação:

- Representação Abstrata: refere-se aos tipos dos dados e é descrito como uma "linguagem", independente da plataforma utilizada
- Representação Concreta: refere-se ao preenchimento dos dados com os seus valores devidos, através da utilização de octetos ou bits.

A ASN.1, portanto, é uma notação para sintaxe abstrata de dados, utilizada para definir os tipos de dados manipulados. Ela é uma notação bem completa, permitindo se representar tipos de dados bastante complexos de uma forma relativamente simples. Para a representação concreta, utiliza-se a sintaxe BER (Basic Encoding Rules).

6.6.1. Tipos ASN.1

Os tipos ASN.1 podem ser divididos em dois tipos: primitivos e construtores.
Os tipos primitivos, como o próprio nome sugere, representa diretamente os tipos simples que se quer manipular, como por exemplo um inteiro, um boolean, etc.

Tipos primitivos ANS.1:

- Integer - números inteiros
- Boolean - falso ou verdadeiro

- Real - números reais
- Bit string - strings binários
- Octet string - string de octetos
- Enumerated - inteiros enumerados
- Null - nulo

6.6.1.1. Integer

O INTEGER é um tipo de dado que pode assumir como valor os inteiros utilizados para contagem.
Numero ::= INTEGER
tipo num Numero ::= 100 -- valor atribuído a uma variável num do tipo Numero

6.6.1.2. Enumerated

O tipo ENUMERATED é uma lista de valores numerados, onde se pode assumir um destes valores.
Cor ::= ENUMERATED { azul (0), amarelo (1), vermelho (2) }
carro Cor::= azul -- ou 0

6.6.1.3. Real

Uma variável do tipo REAL pode assumir qualquer valor real, especificado através de um conjunto de três valores inteiros: mantissa, base e expoente. A mantissa e o expoente podem assumir qualquer valor, enquanto a base está limitada aos valores 2 ou 10.
TranscendentalNumbers ::= REAL
number TranscendentalNumbers::={27182818, 10, -7} --{mantissa, base, expoente}

6.6.1.4. Bit String

O tipo BIT STRING define uma seqüência de zero ou mais bits, sem restrição de tamanho.
BitNumber ::= BIT STRING
bitValue BitNumber ::= '10010'B
O BIT STRING também pode ser utilizado para listas de bits nomeados:
Services ::= BIT STRING { bina (0), sigame (1), conferencia (2), despertador (3) }
meuServico Services::={bina, conferencia, despertador} -- ou '1011'B

6.6.1.5. Octet String

Define uma seqüência de zero ou mais octetos, sem delimitação de tamanho. Pode ser usado para representar caracteres ou dados orientados a byte (os tipos string estão relacionados com este tipo, e serão vistos mais à frente).
UserName ::= OCTET STRING
initial UserName ::= "anon" -- ou '616E6F6E'H

6.6.1.6. Null

O tipo NULL representa uma situação de ausência de informação. Sua aplicação fica evidente quando se fala dos tipos construtores.

Tipos Contrutores

Baseados nos tipos primitivos podemos construir novos tipos mais complexos (chamados construtores), de forma que a abrangência da sintaxe se torne mais flexível e poderosa.:

- SET - coleção de ítens
- SEQUENCE - grupo ordenado de valores
- SET OF - coleção de ítens do mesmo tipo
- SEQUENCE OF - vetor de ítens do mesmo tipo
- CHOICE - permite a escolha de um tipo entre vários possíveis

Exemplo de um módulo ANS.1:

```
AttributeModule {joint-isso-ccitt ms(9) smi(3) part4(4) asn1Module(2) attributes(0)}
DEFINITIONS ::= BEGIN
ObjectName ::= GraphicString
QOSErrorCause ::= ENUMERATED { tempoRespostaExcessivo (0), tamanhoFilaExcedido (1),
larguraFaixaReduzida(2), taxaRetransmissaoExcessiva (3) }
QOSErrorCounter ::= INTEGER
QOSCounterRange ::= QOSErrorCounter {0 .. 4294967296} -- 32 bits
END
```

CORBA

The Common Object Request Broker :

Arquitetura e Especificacao (Introducao).

Ultimas modificacoes: 17 / 09 / 1997.

O objetivo deste documento eh apresentar uma *visao geral* [veja topico 4] da arquitetura comum do ORB do Object Management Group (OMG), para iniciantes no assunto, provendo um ponto de partida ao conhecimento de CORBA.

O que e CORBA?

CORBA (Common Object Request Broker Arquiteture) eh um padrao para objetos sendo desenvolvidos em sistemas distribuidos. A arquitetura deste padrao possibilita mecanismos pelos quais um objeto pode enviar requerimentos ou receber respostas, para/de outro objeto no sistema distribuido, de uma forma transparente como definido pelo ORB do OMG. O ORB sob tal arquitetura eh uma aplicacao que possibilita interoperabilidade entre objetos, construidos em (possivelmente) linguagens diferentes, executados em (possivelmente) diferentes maquinas em ambientes heterogenios distribuidos.

ORB (Object Request Broker) - **comercialmente** conhecido como CORBA- eh o 'coracao' da arquitetura que prove os mecanismos de comunicacao entre os objetos. O ORB fornece uma estrutura que permite objetos conversarem entre si, independente de aspectos especificos da plataforma e tecnicas usadas para implementa-los, ou seja, um cliente pode invocar, transparentemente, um metodo num objeto servidor, o qual pode estar na mesma maquina ou em qualquer lugar da rede. O ORB intercepta o requerimento invocado e fica responsavel em encontrar um objeto que implementara a operacao requerida, passar os parametros da invocacao ao objeto invocado e retornar o resultado ao cliente. O cliente nao precisa saber onde o objeto esta alocado , nao precisa saber qual a linguagem que implemta-o , seu sistema operacional ou qualquer outro aspecto do sistema que nao seja relacionado com a interface do objeto. Aplicacoes tipicas cliente/servidor usam seus proprios design ou um padrao reconhecido para definir o protocolo a ser usado entre os dispositivos. Definicoes de protocolos dependem da liguagem de implementacao , transporte na rede e uma duzia de outros fatores. ORBs simplificam este processo... Com um ORB, o protocolo de rede eh abstraido, o

usuario se preocupa apenas com a interface com os objetos, como se estes "fossem locais". Eh claro que os usuarios podem definir o protocolo de comunicacao entre seus objetos, durante a especificacao de suas interfaces. Seguir os padroes da arquitetura CORBA garante portabilidade e interoperabilidade de objetos sobre uma rede de sistemas heterogenios.

As Interfaces CORBA

CORBA eh baseada, dentre outros aspectos, na construcao e armazenamento de interfaces para os objetos. Estas interfaces sao armazenadas em repositorios de interfaces. No repositorio de interface os objetos podem fazer requerimentos para obter informacoes sobre determinada interface, ou mesmo criar uma nova interface nele [veja topico 9]. Como se pode perceber, eh desejavel que o leitor tenha um solido conhecimento da teoria da programacao objeto-orientada, devido ao forte relacionamento do ORB com objetos instanceados e com interfaces definidas para tais instanceas. Alguns aspectos da modelagem de objetos, relacionados com CORBA, podem ser encontrados em : O modelo de Objetos [veja topico 2].

As interfaces devem ser construidas para serem usadas no instanceamento de objetos, para controlarem o acesso aos objetos e controlar o relacionamento entre estilos de objetos (gerenciamento de classes). Por exemplo, alguem poderia construir uma classe A e uma classe B derivada de A, numa linguagem objeto-orientada. Na IDL esses mesmo alguem poderia especificar que uma interface A eh herdada por uma interface B e, assim, com o mapeamento de IDL para C++, por exemplo, alguem poderia ter um objeto do tipo B herdeiro de um outro do tipo A. A padronizacao da arquitetura da criacao de tais interfaces possibilita a criacao de aplicacoes que fazem instanceamento de objetos e manipulacao de requerimentos das operacoes contidas nos mesmos, como o ORB do OMG.

Seguindo uma gramatica definida na arquitetura CORBA, um desenvolvedor pode criar tais interfaces que podem ser usadas por qualquer orb num sistema. Em concordancia com tal gramatica, fica garantido que um objeto podera ser instanceado, com a interface em questao, pelo ORB e que o cliente que precisa do objeto vera', pela interface, as operacoes que espera encontrar - caso tenha conhecimento desta interface (Invocacao Estatica). E pode recorrer ao repositorio para saber quais operacoes sao fornecidas por esta interface (Invocacao Dinamica).

Quando um cliente precisar de uma dada operacao, ele ira criar um requerimento, o qual eh tratado pelo ORB. O ORB faz chamadas ao adaptador de objetos e esse ultimo faz o instanceamento do objeto dinamicamente, se o objeto ainda nao existe, ou faz simples invocacao de metodos em objetos ja existentes. Pelo recurso de instanceamento dinamico, o cliente normalmente conhece a interface do objeto - pode ser que o cliente nao conheca a interface tambem, por exemplo quando o cliente recebe uma referencia do objeto no formato de string, - e pede ao ORB para

instancear este objeto. Com estes dados, o ORB chama a DIR (Dynamic Implementation Routine) que recebe os requerimentos passados para instancear objetos e basea-se nos parametros contidos nestes requerimentos, de forma que a DIR faz o instanceamento e invocacao (dinamica ou nao) do objeto correspondente ao requerimento. A invocacao dinamica faz-se necessaria quando interfaces de objetos nao sao conhecidas em tempo de compilacao, mas sim algumas assinaturas de alguns metodos conhecidas previamente de forma estatica ou por consultas ao Repositorio de interfaces. Quando as interfaces sao conhecidas em tempo de compilacao, o proprio cliente pode escolher uma interface e fazer sua invocacao, passando o valor de uma referencia ao objeto instanceado a partir desta interface. O servidor que instancia os objetos das interfaces conhecidas deve identificar cada objeto por uma referencia (geralmente uma grande string) que eh disponibilizada aos clientes, de alguma forma.

Topico 2 - O Modelo de Objeto

Ultimas Modificacoes: 19/08/97

Este documento descreve um modelo de objeto que molda a arquitetura CORBA. Mais informacoes sobre o Modelo de Objeto podem ser encontradas no *Object Management Architecture Guide* do Object Management Group.

Visao Geral

O modelo de objeto fornece uma representacao organizada dos conceitos de objeto e terminologia. O modelo de objeto do OMG eh abstrato, ja que ele nao eh entendido diretamente por uma dada tecnologia particular. O modelo de objeto descrito aqui eh o modelo concreto. O modelo concreto pode diferir do abstrato em varios aspectos:

- Ele pode elaborar o modelo de objeto abstrato tornando-o mais especifico, por exemplo, definindo a forma dos paramentros de um requerimento (request) ou a linguagem usada para tipos especificos
- Ele pode introduzir no modelo abstrato instancias especificas de entidades definidas pelo modelo, por exemplo, objetos especificos, operacoes especificas, ou tipos especificos
- Ele pode restringir o modelo abstrato eliminando entidades ou adicionando restricoes no seu uso.

Um sistema de objeto eh uma colecao de objetos que isola os clientes dos fornecedores de servicos por um encapsulamento de interface muito bem definido. Em particular, os clientes sao isolados das implementacoes dos servicos, da representacao de dados e do codigo executavel.
Esse modelo de objeto eh um exemplo de um modelo de objeto classico, onde um cliente envia uma mensagem para um objeto. Conceitualmente, o objeto interpreta a mensagem para decidir que servico executar. No modelo classico, uma mensagem identifica um objeto e zero ou mais parametros. Como na maioria dos modelos classicos de objeto, um primeiro parametro distinguivel eh requerido, o qual destingue qual a operacao a ser desenvolvida; a interpretacao da mensagem pelo objeto envolve o selecionamento de um metodo beseado numa operacao especifica. Operacionalmente, eh claro, o selecionamento de metodos poderia ser feiro pelo objeto ou pelo ORB.

Semantica

Um sistema de objetos fornece servicos para clientes. Um cliente de um servico eh qualquer entidade capaz de requerer os servicos.

Objeto

Um objeto eh uma entidade encapsulada e identificavel que prove um ou mais servicos que podem ser requeridos por um cliente.

Requerimento

Um requerimento eh um evento, i.e. , algo que ocorre num determinado momento. A informacao associada com o requerimento consiste de uma operacao, um identificador de objeto, zero ou mais paramentros, zero ou mais excecoes que podem ser levantadas, e um contexto opcional do requerimento.
Um *valor* eh algo que pode ser um paramentro legitimo em um requerimento. Um valor pode identificar um objeto. Um valor que identifica um objeto eh chamado de nome do objeto. Mais particularmente, um valor eh uma instancia de um tipo de dado da IDL OMG.
Uma *referencia a objeto* eh um nome de objeto que realmente denota um objeto particular. Especificamente, uma referencia a objeto ira identificar sempre o mesmo objeto, quando a referencia for usada em direrentes requerimentos no mesmo sistema ORB.
Um requerimento pode ter informacoes adicionais, que ficam definidas no *contexto* do requerimento.
Se um estado anormal ocorre devido a um requerimento, entao surge uma excecao que eh retornada ao cliente, esta excecao retornada pode ser uma excecao do systema ou definida pelo usuario.
Os parametros do requerimento sao identificados conforme a posicao dos mesmos. Um parametro pode ser de "input", "output", ou "input-output".

Criacao e Destruicao de objetos

Um objeto pode ser criado e destruido. Para um cliete, nao existe uma forma especial de criar ou destruir um objeto. Tudo que o cliete sabe eh se existe ou nao uma referencia para um objeto. Alem disto, um cliente pode desalocar uma dada referencia a um objeto, seja explicitamente ou de forma implicita usando as variaveis automaticas do CORBA (**<type>_var** como definido no *OMG C++ Language Mapping*).

Interfaces

Uma interface eh a descricao das possiveis operacoes fornecidas por um objeto. Tudo que um cliente precisa saber a respeito de um objeto esta contido na interface, i.e., o cliente sabe o que um objeto pode fazer, mas nao sabe como o objeto o faz. As interfaces sao especificadas na IDL OMG. As interfaces podem usar o mecanismo de heranca, de tal forma que os objetos podem ter multiplas interfaces. Cada interface herdada descrevera um subgrupo das operacoes contida na interface principal.

Operacoes

Uma operacao eh uma entidade que denota um servico que pode ser requerido. Uma operacao eh identificada por um identificador de operacao. Uma operacao nao eh um valor. A assinatura da operacao eh composta por:

- Especificacao dos parametros, que podem ser de entrada, saida, ou ambos.
- Especificacao do resulatado retornado pela operacao.
- Especificacao das excecoes que podem surgir com a operacao.
- Especificacao de um contexto adicional com informacoes que podem afetar a operacao.
- Especificacao da semantica da execucao que pode ser esperada com o requerimento da operacao.

Uma operacao eh tao generica, que ela pode ser desenvolvida uniformente por objetos de diferentes implementacoes. Para se alcancar tal generalidade, eh necessario descrever as interfaces sob a otica da IDL. A aparencia (assinatura) geral de uma interface, como descrita pela IDL, esta mostrada abaixo:

[oneway] <op_type_spec> <identifier>(param1,...,paraml) [raises (except1,...,exceptN)] [context(name1,...,nameM)]

Onde:

- O opcional **oneway** indica que a operacao sera executada uma vez, caso tuda corra bem , ou uma execao sera gerada. Espera-se a melhor performance por parte da operacao.
- O **<op_type_spec>** eh o tipo do valor retornado.
- O **<identifier>** prove o nome da operacao de tal interface.

- Os parametros da operacao usam flags com valores **in, out,** o**u inout** para indicar em qual direcao a informacao flui (com respeito ao objeto desenvolvendo a operacao).
- O opcional **raises** indica qual excecao, dentre as definidas, pode ser sinalizada para terminar um requerimento para esta operacao; se tal execao nao eh fornecida, nenhuma excecao para o usuario sera assinalada, nesta operacao.
- O opcional **context** indica qual informacao de contexto do requerimento sera avaliada na implementacao do objeto; nenhuma outra informacao de contexto eh requerida a ser transportada com o requerimento.

Parametros
Um parametro eh caracterizado pelo seu modo (**in, out, inout**) e seu tipo.
Resultado Retornado
O resultado retornado eh um parametro **out** distinguivel.
Excecoes
Uma excecao eh uma indicacao que uma operacao nao foi desenvolvida com sucesso. Uma excecao pode vir acompanhada com informacoes sobre essa excecao. Qualquer assinatura de interface ja inclui implicitamente as excecoes padroes "Standard Exceptions" [veja topico 6].
Contextos
Um contexto de 'request' fornece informacoes adicionais sobre a operacao requerida, que pode afetar a maneira com que um requerimento eh processado pela implementacao do objeto.
Atributos
Uma interface pode ter atributos. Um atributo eh logicamente equivalente a declaracao de funcoes de acesso: uma para retornar o valor da atributo e outra para atualizar o valor .
Um atributo pode ser somente de leitura. Neste caso somente uma funcao de retorno eh definida, a de leitura obviamente.

Implementacao de Objeto
O modelo de implementacao consiste de duas partes: o modelo de execucao e o modelo de construcao. O modelo de execucao descreve como os servicos sao desenvolvidos. O modelo de construcao descreve como os servicos sao definidos.

Topico 3 - Object Management Group

O Object Management Group, Inc. (OMG) eh uma organizacao internacional formada por mais de 500 membros, incluindo vendedores de sistemas de infornmacoes, desenvolvedores de softeware e usuarios. Fundada em 1989, A OMG promove a teoria e pratica da tecnologia Objeto-Orientada em desenvolvimento de softwares. O carater da organizacao incluie o estabelecimento das especificacoes de gerenciamento de objetos para promover um padrao comum para desenvolvimento de aplicacoes. Objetivos primordiais sao: reusabilidade, portabilidade, e interoperabilidade de softwares baseados em objetos, e em ambientes distribuidos e heterogenios. Em concordancia com tais especificacoes, sera possivel desenvolver um ambiente de aplicacoes heterogenio usando as principais plataformas de hardwares e sistemas operacionais.

Topico 4 - CORBA - *Visao Geral da Arquitetura.*

ORB

Um ORB nao precisa ser sempre o mesmo em varias implementacoes. O que eh importante sao as interfaces ORB. [veja o topico 10]. Qualquer implementacao de um ORB que forneca as interfaces apropriadas eh aceitavel. As interfaces sao organizadas em 3 categorias:

- 1- Operacoes que sao as mesmas para qualquer implementacao de ORB.
- 2- Operacoes que sao especificas para tipos especificos de objetos.
- 3- Operacoes que sao especificas a tipos particulares de implementacoes de objetos.

Mais de um ORB pode ser posto em funcionamento e, desta forma, podem haver varias referencias a um dado objeto. O cliente pode, entao, ter acesso a um objeto por diferentes referencias. Os ORBs devem ser capazes de distinguir suas referencias a objetos. Isto nao eh responsabilidade do cliente.

Linguagen de Definicao de Interfaces OMG

A *IDL OMG* especifica os objetos definindo a interface de cada um. Uma interface consiste em um conjunto de operacoes e parametros para aquelas operacoes, as excecoes que determinada operacao pode levantar, os atributos da classe a qual esta interface descreve, e o relacionamento entre classes. Com a *IDL* fica possivel a um objeto particular mostrar, a seus potencias clientes, que operacoes estao

acessiveis e como elas devem ser acessadas. Pelas definicoes na IDL eh possivel mapear objetos CORBA para linguagens de programacao particulares ou sistemas de objetos.

Object Adapters

Um object adapter [veja o topico 11] eh o meio primario em que a implementacao de um objeto acessa servicos fornecidos pelo ORB. Os servicos gerados pelo ORB por um adaptador inclui: geracao e interpretacao de referencias a objetos, invocacao de metodos, etc.

Repositorio de Interface

O repositrio de interface [veja o topico 9]eh um servico que prove objetos persistentes (durante o tempo de vida do repositorio de interfaces) que representam a informacao IDL numa forma disponivel em tempo de execucao. A informacao no repositorio de interface pode ser *usada pelo ORB para suprir requerimentos* . Superficialmente, usando a informacao no repositorio de interface, fica possivel a um programa invocar operacoes em um objeto cuja interface nao era conhecida em tempo de compilacao, i.e., ser capaz de determinar que operacoes sao validas no objeto e fazer uma invocacao nelas.

Repositorio de Implementacao

Eh o local onde sao guardadas informacoes sobre implementacoes de objetos dados.

A Estrutura de um Cliente

Um cliente de um objeto tem uma *referencia a um objeto* que refere-se a este objeto. O cliente pode entao, invocar operacoes nesta referencia de objeto, como se a referencia fosse o proprio objeto. Uma *referencia a um objeto* eh um token que pode ser invocado ou passado como parametro a uma invocacao num objeto diferente. Invocar um objeto involve especificar o objeto a ser invocado, a operacao a ser desenvolvida , e parametros a serm dados para a operacao e/ou parametros retornados da invocacao. No momento em que um ORB nao pode completar uma invocacao, uma excecao eh levantada.

Uma *referencia a um objeto* tambem pode ser convertida numa string que pode ser armazenada em qualquer arquivo ou comunicadas por diferentes meios e subsequentemente transformada novamente em *referencia a um objeto* pelo ORB que produziu a string. A representacao de uma *referencia* por uma string eh uma maneira muito flexivel de representacao, ja que uma string pode ser comparada, copiada e movida. Nunca existirao dois objetos identicos instanceados num sistema. Assim, cada objeto diferente deve ter sua propria referencia criada pelo ORB. O ORB cria tais strings usando metodos especiais que levam em consideracao dados como: IP da maquina (dominio), hora e data. O objetivo desta

consideracao eh criar strings sempre diferentes das ja criadas, i.e., evitar conflitos de strings. Tambem, sob este proposito, as string sao criadas com um tamanho muito grande. Quanto maior a string menor eh a probabilidade de conflitos.

Estrutura de uma implementacao de objeto

Muitas das implementacoes de objetos proveem seu comportamento usando facilidades como o ORB e adaptadores de objeto. Por exemplo, o *Basic Object Adapter* fornece algum dado persistente associado a um objeto, esta quantidade de dados (relativamente pequena) eh tipicamente usada para armazenar um identificador. A implementacao do objeto pode usar este identificador para instancear objetos persistentes armazenados em um servico de armazenagem da escolha de uma implementacao de objeto. Com esta estrutura fica possivel, nao somente para implementacoes diferentes de objetos usar o mesmo servico de objeto, mas aos objetos, escolher o servico que eh mais apropriado a eles.

A Estrutura de um Adaptador de Objeto

Esses adaptadores sao responsaveis pelas seguintes funcoes:

- Geracao e interpretacao de referencias a objetos.
- Chamada de metodos.
- Ativacao e desativasao de objetos.
- Mapeamento de referencias a objetos correspondentes.
- Registro de implementacoes.

Essas funcoes sao desenvolvidas usando-se o ORB Core. O object adapter define a maioria dos servicos do ORB que a implementacao de objeto depende. Se uma implementacao de objeto precisa de um dado valor no ato de sua invocacao, esse valor podera ser guardado juntamente com sua referencia. Dessa forma, por exemplo, ao usar a referencia, o ORB tambem tera outro dado a respeito do objeto criado. Se o ORB nao permitir esse tipo de coisa, entao o object adapter podera ser usado para guardar tal dado em seu proprio armazem de dados. Com object adapters, fica possivel para uma implementacao de objeto ter acesso a um servico que eh ou nao implementada no ORB Core. Se o ORB Core prove este servico, entao o object adapter simplesmente prove um interface para o mesmo, do contrario, o adaptador deve implementar o servico no topo do ORB Core.

Exemplo de Object Adapters:

- Basic Object Adapter: pode ser usado com a grande maioria de ORB com implementacoes convencionais.
- Object_Oriented Database Adapter: os objetos sao armazenados em um banco de dados que esta conectado ao adaptador. Desde de o OODB fornecea os metodos e armazenagem persistente, os objetos

podem estar registrados implicitamente e nenhum estado eh requerido no object adapter.

A Integracao de Sistemas de Objetos Externos.
A arquitetura comum do ORB (CORBA) eh planejada para permitir interoperabilidade com uma gama de sistemas de objetos.

Por existir varios sistemas de objetos, um desejo comum ira permitir aos objetos em tais sistemas serem acessiveis via ORB. Estes sistemas que sao ORBs, literalmente falando, podem ser conectados a outros ORBs atraves de mecanismos descritos mais adiante neste documento.

Topico 5 - IDL OMG - Sintaxe e Semantica

A Linguagem de defincao de interfaces (OMG IDL) eh a linguagem usada para descrever as interfaces que objetos clientes usam (chamam) e implemetacoes de objetos fornecem. Uma definicao de interface escrita em OMG IDL apresenta-se completa e especifica completamente os parametros de cada operacao. Uma interface OMG IDL fornece a informacao necessaria para desenvolver clientes que usam as operacoes daquela interface. Os clientes nao sao escrito em IDL, mas em outra linguagem, como C++, se o mapeameto de IDL para C++ estiver definido. Este mapeamento eh dependente de facilidades econtradas na linguagem do cliente. Por exemplo, uma excecao descrita na IDL pode ser mapeada para um extrutura numa linguagem que nao tem nocoes de execao, ou mapeada para uma excecao numa linguagem que tem esta nocao, por exemplo uma excecao de C++.
Como qualquer linguagem, a OMG IDL tem sua propria gramatica. Essa gramatica eh muito parecida com a gramatica do C++, mas com mais restricoes, pois a IDL eh uma linguagem de especificao e nao de implementacao. Veja a gramatica completa da IDL em [1] na pagina 3-9. A OMG IDL obedece as mesmas regras lexicas da linguagem C++, mesmo que algumas novas palavras chaves estejam presentes para tornar possivel os conceitos de distribuicao de objetos.
A gramatica da OMG IDL eh um subconjunto do padrao proposto ANSI C++, com construtores adicionais para suportar o mecanismo de invocacao de operacao. A OMG IDL eh uma linguagem totalmente declarativa, i.e., para ser usada na formulacao de declaracoes. IDL suporta a sintaxe C++ para constantes, tipos e

declaracoes de operacoes , mas nao inclui qualquer estrutura algoritmica ou variaveis.
Um arquivo com especificacoes de interfaces IDL deve ter a extensao ".idl".

Pre-Processamento
O pre-processamento OMG IDL, que eh baseado no pre-processamento ANSI C++, fornece subistituicao de macro, compilacao adicional, e inclusao de arquivos fontes. Em adicao, diretivas sao fornecidas pra controlar numeracao de linha usada em diagnosticos e para debugacao simbolica, para gerar uma mensagem de diagnostico com uma dada sequencia de tokens e para desenvolver acoes especificas de implementacao (o **#pragma**).

A seguir estao mostradas algumas passagens da gramatica OMG IDL e explicacoes a respeito. Para maiores detalhes veja [1].
Especificacao da IDL OMG
Uma especificacao IDL OMG consiste de uma ou mais definicoes de tipos, definicoes de constantes, definicoes de excecoes, ou definicoes de modulos. A sintaxe eh:

<specification>::= <definition>+
<definition> ::= <type_dcl>";"
| <type_dcl>";"
| <const_dcl>";"
| <except_dcl>";"
| <module> ";"

Declarecao de modulo

<module> ::= "modulo" <identifier> "{" <definition> "}"

Declaracao de uma interface
A definicao de uma interface concorda com a seguinte sintaxe:

<interface> ::= <interface_dcl>
| <forward_dcl>
<interface_dcl> ::= <interface_header> "{"<interface_body>"}"
<interface_dcl> ::= "interface" <identifier> [<inheritance_spec>]
<interface_body> ::= <export>*
<export> ::= <type_dcl>";"
| <const_dcl>";"
| <exept_dcl>";"
| <attr_dcl>";"
| <op_dcl> ";"

O *interface header* consiste de dois elementos:

- O nome da interface, que deve ser precedido pela palavra Interface. Este nome eh o identificador da interface.
- uma especificacao opcional de heranca.

Especificacao de heranca

<inheritance_spec>::= ";"<scoped_name>
{";"<scoped_name>}*
<scoped_name>::= <identifier>
| "::"<identifier>
| <scoped_name>"::"<identifier>

Cada **<scoped_name>** deve denotar uma interface definida previamente.

O corpo da interface

O corpo da interface consiste dos seguintes tipos de declaracoes:

- Declaracoes de constantes, que devem ser exportadas pela interface.
- Declaracoes de tipos, que sao os tipos exportados pela interface.
- Declaracoes de excecoes, que sao as estruturas de excecoes exportadas pela interface
- Declaracao de atributos, que especificam os atributos associados exportados pela interface.
- Declaracoes de operacoes, que especificam as operacoes que a interface exporta e o formato de cada uma, incluindo o nome da operacao, o tipo de dado retornado, o tipo de todos os parametros de uma operacao, excecoes legais que podem ser retornadas como resultado de uma invocacao, e informacoes contextuais que podem afetar o desenrolar de um metodo.

Forward Declaration- Declaracao Deferida.

Uma declaracao deferida declara o nome de uma interface sem defini-la. Isto permite a definicao de interfaces que referem-se uma a outra. A sintaxe consiste simplesmente na palavra chave **interface** seguida por um **<identifier>** que nomeia a interface.

Heranca

A heranca na IDL eh analoga a heranca de objetos na linguagem C++. Uma interface pode derivar de outra (a interface base). A interface derivada pode declarar novos elementos. Os elementos de uma interface base podem ser referidos como se eles fossem elementos da interface derivada. Para tal eh usado o operador de resolucao de nome "::".

Uma interface pode redefinir tipos, constantes, e nomes de excecoes que foram herdados.

Uma interface base pode ser uma base direta ou uma base indiereta, dependendo se a interface derivada foi derivada direta ou indiretamente da base em questao.
Uma interface pode ser derivada de qualquer numero de outras interfaces. Isto eh a heranca multipla. A ordem da derivacao nao eh significante.
Uma interface nao pode ser especificada como uma base direta mais que uma vez, mas pode ser declarada como base indireta mais que uma vez. Considere o seguinte exemplo:

```
interface A {...}
interface B : A {...}
interface C : A {...}
interface D : B, C {...}
```

Este relacionamento de interfaces eh legal.
No seguinte exemplo

```
const long L = 3;
interface A {
        void f (in float s[L]); //s tem 3 floats
    };
    interface B {
        const long L = 4;
    };
    interface C : B, A {}
```

QuaL eh a assinatura de f ? Em C a interface garante que eh : **void f(in float s[3]);** que eh identica aquela na interface A.

Declaracao de Tipo
A OMG IDL fornece construcoes para nomear tipos de dados; isto eh, a IDL fornece, como a linguagem C, declaracoes que associam um *identifier* com um tipo de dados. A OMG IDL usa a palavra chave **typedef** para associar um nome com um tipo de dado; um nome tambem pode ser associado a um tipo de dado via as declaracoes **struct, union e enum;** a sintaxe eh:

```
<type_dcl> ::= "typedef" <type_declarator>
                | <struct_type>
                | <union_type>
                | <enum_type>
    <type_declarator> ::= <type_spec> <declarators>
```

Cada tipo de dado OMG IDL eh mapeado para um tipo de dado nativo na linguagem destino do mapeamento. Entretanto, durante este mapeamento, erros de conversao podem ocorrer, durante o desenvolvimento de uma invocacao de operacao. O mecanismo de invocacao pode levantar uma excecao se uma tentaiva

de conversao para uma valor ilegal ocorrer. Em tais situacoes sao levantadas excecoes padroes.

Tipos Construidos

Os tipos construidos sao:

<constr_type_spec> ::= <struct_type>
| <union_type>
| <enum_type>

Mesmo sendo sintaticamente possivel gerar especificacoes recursivas de tipos em OMG IDL, tal recursao eh semanticamente restringida. A unica forma permitida de especificacao de tipos recursiva eh atraves do uso de tipos templates **sequence.** Por exemplo, a seguinte passagem de interface eh legal:

```
struct foo{
        long value;
        sequence<foo>chain;
    }
```

Tipos Template

Um tamplete eh formado sob a seguinte sintaxe:

<template_type_spec>::= <sequence_type>
| <string_type>

Para declarar uma sequencia de tipos vem:

<sequence_type> ::= "sequence""<"<simple_type_spec>":"
<positive_int_const>">"
| "sequence""<"<simple_type_spec> ">"

Como visto acima, pode estar presente ou nao o declarador do tamanho da sequencia de tipos. No caso do declarador de tamanho estar presente, a sequencia e chamada de "bounded sequence". Um tipo sequencia pode ser usado como um parametro para outra sequencia de tipo. Veja:

```
typedef sequence<sequence<long> >Fred;
```

Isto declara Fred como sendo uma sequencia de tamanho indefinido de uma outra sequencia de tamanho indefinido de Long. Repare que os dois "> >" devem estar separados por pelo menos um espaco em branco, para nao serem considerados como um token ">>".

Declaracao de Excecao

A declaracao de excecao permite a declaracao de estruturas de dados que podem ser retornadas para indicar que uma condicao excepcional ocorreu durante o desenvolvimento de um requerimento. A sintaxe para isto eh:

<except_dcl>::= "exception"<identifier>"{"<member>*"}"

Quando uma excecao eh retornada como resultado de um requerimento, esta eh identificada pelo seu nome, i.e., identificador de excecao (<identifier>).

Se uma excecao eh declarada com membros, um programador tera acesso aos valores daqueles membros quando uma excecao for levantada. Se nenhum membro for especificado, nenhuma informacao adcional sera acessivel quando a excecao for levantada.
Um conjunto de Standard Exceptions [veja topico 6] eh definido e corresponde aos erros que podem ocorrer em tempo de execucao de requerimento.

Declaracao de Operacao
Declaracoes de operacoes em IDL OMG sao similares a declaracao de funcoes em C. A sintaxe eh:

<op_dcl> ::= [<op_attribute>] <op_type_spec> <identifier>
<parameter_dcls> [<raises_expr>] [<context_expr>]
<op_type_spec>::= <param_type_spec>
| "void"

Uma declaracao de operacao consiste de:

- Um atributo de operacao, opcional, que especifica qual semantica de comunicacao o sistema deve fornecer quando a operacao eh invocada.
- O tipo do resultado retornado. Operacoes que nao retornam qualquer resultado devem especificar o tipo **void** como retorno.
- Um identificador que nomeia a operacao no escopo da interface na qual eh declarada.
- Uma lista de parametros que denota zero ou mais declaracoes de parametros para a operacao.
- Uma ou mais excecoes opcionais, das quais uma podera ser levantada, como resultado retornado pela operacao.
- Uma expressao opcional de contexto que indica quais elementos do contexto do cliente podem ser consultados pelo metodo que implementa a operacao, e por conseguinte, podem modificar o comportamento da implementacao do objeto.

Raises Expressions
A expressao *raises* especifica que excecoes podem ser levantadas como resultado da invocacao de uma operacao. Segue abaixo a sintaxe para tais especificacoes:

<raises_expr> ::= "raises""("<scoped_name>{","<scoped_name>}*")"

O <scoped_name> deve ser de uma excecao previamente definida.
Expressoes de Contexto
Uma expressao de contexto especifica que elementos do contexto do cliente podem afetar a maneira como o requerimento eh processado pelo objeto. A sintaxe eh:

<context_exp>::= "context""("<string_literal>{","<string_literal>}*")"

Modulo CORBA

Com o proposito de previnir que os nomes definidos na especificacao CORBA colidam com nomes na linguagem de programacao e outros sistemas de software, todos os nomes definidos em CORBA sao tratados como se eles fossem definidos em um modulo nomeado CORBA. Entretanto, numa especificacao OMG IDL, palavras chaves tais como Object nao deve ser precedidas com um prefixo "CORBA::" Outros nomes de interfaces como TypeCode nao sao palavras chave OMG IDL e devem ser referenciadas pelo seu nome de escopo completo (i.e., CORBA::TypeCode) como uma especificacao OMG IDL.

Considere o seguinte exemplo:

```
interface A {
exception E {
long L;
};
void f() raises (E);
};
interface B : A {
void g() raises(E);
};
```

Neste exemplo, a excecao eh conhecida pelos nomes globais **::A::E** e **::B::E**.

Topico 6 - Standard Exceptions - Excecoes Padroes

Aqui estao presentes as excecoes padroes definidas para o ORB.

Esses identificadores de excecoes podem ser retornados como resultado de qualquer invocacao de operacao. *Standard Exceptions* nao precisam ser listadas na expressao **raises.** Existem muitas excecoes possiveis de serem levantadas (excecoes padroes) e desta forma seria muito descomodo e complexo manipular todas elas. Entao sao criadas classes de excecoes padroes. Por exemplo, uma invocacao de operacao pode falhar em muitos pontos dirferentes devido a

inabilidade de alocar memoria dinamica. Melhor que enumerar varias excecoes diferentes correspondentes aos diferentes meios que tais falhas causam uma excecao, eh levantar uma unica excecao correspondente ah falha de alocar memoria dinamica. Cada excecao padrao inclui um *minor code* para designar a subcategoria da excecao; o assinalamento de valores para o *minor code* eh deixado para cada implementacao de ORB.
Cada excecao padrao inclui um codigo '**completion_status'** que tem um dos seguintes valores *{COMPLETED_YES, COMPLETED_NO, COMPLETEDE_MAYBE}*. Esses valores tem os seguintes resultados:
COMPLETED_YES : a implemetacao do objeto conseguiu completar a operacao invocada antes da execao ter sido levantada.
COMPLETED_NO: a implemetacao do objeto nao conseguiu completar a operacao invocada antes da execao ter sido levantada.
COMPLETED_MAYBE: o estado final apos a execucao da operacao eh indeterminado.

Definicoes das Excecoes Padroes

```
#define ex_body {unsigned long minor; completion_status completed:}
enum completion_status {COMPLETED_YES, COMPLETED_NO,
COMPLETED_MAYBE};
enum exception_type {NO_EXCEPTION, USER_EXCEPTION,
SUSTEM_EXCEPTION};
  exception UNKNOWN                    ex_body; // 1
  exception BAD_PARAN                  ex_body; // 2
  exception NO_MEMORY                  ex_body; // 3
  exception IMP_LIMIT                  ex_body; // 4
  exception COMM_FAILURE               ex_body; // 5
  exception INV_OBJREF                 ex_body; // 6
  exception NO_PERMISSION              ex_body; // 7
  exception INTERNAL                   ex_body; // 8
  exception MARSHAL                    ex_body; // 9
  exception INITIALIZE                 ex_body; // 10
  exceptionNO_IMPLEMENT                ex_body; // 11
  exception BAD_TYPECODE               ex_body; // 12
  exception BAD_OPERATION              ex_body; // 13
  exception NO_RESOURCES               ex_body; // 14
  exception NO_RESPONSE                ex_body; // 15
  exception PERSIST_STORE              ex_body; // 16
  exception BAD_INV_ORDER              ex_body; // 17
  exception TRANSIENT                  ex_body; // 18
```

```
  exception FREE_MEM                    ex_body; // 19
  exception INV_IDENT                   ex_body; // 20
  exception INV_FLAG                    ex_body; // 21
  exception INTF_REPOS                  ex_body; // 22
  exception OBJ_ADAPTER                 ex_body; // 23
  exception DATA_CONVERSION             ex_body; // 24
  exception OBJECT_NOT_EXIST            ex_body; // 25
  exception BAD_CONTEXT                 ex_body; // 26
```

1) the unknown exception
2) an invalid parameter was passed
3) dynamic memory allocation filure
4) violated implementation limit
5) communication failure
6) invalid object reference
7) no permission for attempted op.
8) ORBinternal error
9) error marshalling param/result
10) ORB initialization failure
11) operation implementation unavailable
12) bad typecode
13) invalid operation
14) insufficient resources for req.
15) response to req. not yet available
16) persistent storage failure
17) routine invocations out of order
18) transient failure - reissue request
19) cannot free memory
20) invalid identifier syntaxe
21) invalid flag was specified
22) error accessing interface repository
23) failure detected by object adapter
24) data conversion error
25) non-existent object, delete reference
26) error processing context object

Inexistencia de Objeto
Essa excecao padrao de sistema eh levantada sempre que uma invocacao a um objeto deletado eh desenvolvida.

Topico 7 - DII -Dynamic Invocation Interface - Interface de Invocacao Dinamica

---

Visao Geral

A ORB DII permite criacao dinamica e invocacao de requerimentos a objetos. Um cliente usa este tipo de interface para enviar um requerimento para um objeto obtem **as mesmas semanticas** que um cliente usa nas operacoes robustas geradas da especificacao de tipo.

O ORB aplica o principio de encapsulamento para os requestes, ou seja, para um cliente um requerimento consiste apenas em uma *object reference*, uma operacao, uma lista de paramentros e nada mais.

Na DII, os parametros de um *request* sao fornecidos como elementos de uma lista. Cada elemento eh uma instancia de um **NamedValue.** Os parametros devem ser fornecidos na mesma ordem como aqueles definidos para a operacao no Repositorio de Interface, ou no arquivo IDL que o especifica.

Todos os tipos definidos nesta pagina sao parte do modulo CORBA e, entao, quando referenciados na CORBA IDL [veja topico 5] os nomes de tipos devem ser prefixados por "CORBA::".

Estruturas de dados comuns

O tipo **NamedValue** eh um tipo familiar na OMG IDL. Este tipo pode ser usado como um tipo de paramentro diretamente ou como um mecanismo para descrever argumentos para um requerimento. O tipo **NVList** eh um pseudo-objeto usado para a construcao de lista de paramentros. Os tipos sao descritos em OMG IDL e C, respectivamente, como:

```
typedef unsigned long Flags;
struct NamedValue{
        identifier name; // argunment name
        any argument; //argument
        long len; //length/count of argument value
        Flags arg_modes; //argument mode flags
    };
    CORBA_NamedValue *Corba_NVList; /* C */
```

O **NamedValue e o NVList** sao usados num requerimento de operacao para descrever argumentos e valores retornados. Eles tambem sao usados no contexto de rotinas de objetos para passar lista de nomes de propriedades e valores. A **NVList** so pode ser criada com a operacao **create_list** do ORB.
O campo **arg_modes** pode ter os seguintes valores:
CORBA::ARG_IN o valor associado eh um argumento de *input*
CORBA::ARG_OUT o valor associado eh um argumento de *output*
CORBA::ARG_INOUT o valor associado e um argumento de *in/outpit*
Uso de Memoria
Cada argumento retornado (**out_argument**) associado com a **NVList** eh deletado por conta do ORB, quando o mesmo nao eh mais necessario. Se o programador nao associa os argumentos de retorno com a NVList, ele tera que liberar a memoria de cada um deles por conta propria, usando **CORBA_free()**

Request Operations
As operacoes *request* sao definidas em termos do *Request pseudo-object.*

```
module CORBA{
    interface Request{
            Status add_arg(
                    in Identifier name, //argument name
                    in TypeCode arg_type, //argument datatype
                    in void *value, //argument value to be added
                    in long len, //length/count of argument value
                    in Flags arg_flags //argument flags
                );
                Status invoke (
                    in Flags invoke_flags //invocation flags
                );
                Status delete ();
                Status send(
                    in Flags invoke_flags//invocation flags
                );
                Status get_response (
                    in Flags response_flags // response flags
                );
        };
};
```

Argumentos podem ser associados com a *request* pela especificacao deles na **create_request** ou adcionado-os via chamada a **add_arg.**

Create_request

Status create_request (
in Context ctx //context object for operation
in Identifier operation. //intended operation on object
in NVList arg_list, //args to operation
inout NamedValue result, //operation result
out Request request, //newly created request
in Flags req_flags //request flags
);

A operacao **create_request** eh desenvolvida na referencia do objeto que sera invocado. Essa operacao cria um requerimento ORB.
As propriedades **context** associadas com a operacao sao passadas para a implememtacao do objeto. A implememtacao do objeto nao pode modificar a informacao de contexto passada a ela.
O resultado da operacao eh posto no **result** depois que a invocacao eh completada.

Invoke

Status invoke(
in Flags invoke_flags //invocation flags
);

Essa operacao chama o ORB, o qual desenvolve um metodo apropriado. Se o metodo retorna com sucesso, seu resultado eh colocado no argumento **result** especificado no **create_request.**

Delete

Status delete();

Essa operacao deleta a *request*. Qualquer memoria associada com a *request* eh tambem liberada.

Send

Status send(
in Flags invoke_flags //invocation flags
);

Send inicia a operacao conforme de Request. Diferente de **invoke, send** retorna o controle para o chamador sem esperar a operacao terminar. Para derterminar quando a operacao esta completa, o chamador deve usar a operacao **get_response** ou **get_next_response .**

Get_response

Status get_response(
in Flags response_flags //response flags
);

get_response determina se um requerimento foi terminado. Se o flag **RESP_NO_WAIT** eh 'setado', a operacao **get_response** retorna imediatamente mesmo que o *request* ainda esteja em progresso. Por outro lado, **get_response** espera ate que o requerimento tenha terminado antes de retornar, caso o flag acima nao tenha sido setado.
O seguinte flag eh definido para **get_response:**
CORBA::RESP_NO_WAIT que indica que o chamador nao quer esperar por uma resposta.

Operacoes com Lista
Nesta secao estao descritas operacoes com objetos **NVList.**

```
interface NVLIST{
    Status add_item(
            in Identifier item_name, //name of item
            in TypeCode item_type, //item datatype
            in void *value, //item value
            in long value_len, //length of item value
            in Flags item_flags //item flags
        );
        Status free();
        Status free_memory();
        Status get_cout(
            out long count //number of entries in the list
        );
};
```

Create_list
Essa operacao, a qual cria um pseudo-objeto, eh definida na interface ORB e eh mostrada a seguir:

```
Status create_list(
    in long count, //number of items to allocate for list
    out NVList new_list //newly created list
                );
```

Essa operacao aloca uma lista de tamanho especificado, e a limpa para uso inicial. Com a rotina **add_item,** itens podem der adicionados a lista. Uma **NVList** so pode ser alocada usando-se **create_list.**
Add_item

```
Status add_item (
    in Identifier item_name, //name of item
    in TypeCode item_type, //item datatype
    in void *value, //item value
```

in long value_len, //length of item value
in Flags item_flags //item flags
);

Essa operacao adiciona um novo item a lista indicada. Este item eh adicionado no final da lista.

Free

Status free();

Essa operacao libera a estrutura lista e qualquer memoria associada (uma chamada implicita para a operacao **free_memory** eh feita.).

Free_memory

Status free_memory();

Essa operacao libera qualquer memoria para argumento alocado dinamicamente associado com a lista. A estrutura da lista em si nao eh liberada.

Get_count

Status get_count(
out long count //number of entries in the list
);

Essa operacao retorna o numero total de itens alocados para a lista.

Create_operation_list

Essa operacao, a qual cria um pseudo-objeto, eh definaida na interface ORB.

Status create_operaction_list(
in OperationDef oper, // operation
out NVList new_list //argument definitons
);

Essa operacao retorna uma **NVList** inicializada com as descricoes de argumentos para uma dada operacao. A informacao eh retornada numa forma que pode ser usada em *Dynamic Invocation requests*. Os argumentos sao retornados na mesma ordem em que eles definidos para a operacao. A operacao **free** eh usada para liberar a informacao retornada.

Objetos Contexto

Um objeto contexto contem uma lista de propriedades, cada uma consistindo de um nome e uma valor string associado com tal nome. Por convencao, o contexto representa informacoes relacionadas com o cliente, ambiente do mesmo, ou circunstancias de um requerimento que sao inconvenientes para passar como parametros.

Propriedades contexto podem representar uma fracao do ambiente do cliente ou uma fracao do ambiente da aplicacao, que pode ser mostrada ao ambiente do servidor.

O contexto associado com uma operacao particular eh passado como um parametro distinguivel, permitindo ORBs particulares tomarem vantagem das

propriedades do contexto, por exemplo, usando os valores de certas propriedades para influenciar o comportamento de um metodo, etc..

```
modulo CORBA {
    interface Context{
            Status set_one_value(
                    in Identfier prop_name, // property name to add
                    in string value //property value to add
                );
                Status set_value (
                    in NVList values //property values to be changed
                );
                Status get_value(
                    in Identifier start_scope, //search scope
                    in Flags op_flags, //operation flags
                    in Identifier prop_name, //name of property(s) to retrieve
                    out NVList values //requested property(s)
                );
                Status delete_values(
                    in Identifier prop_name //name of property(s) to delete
                );
                Status create_child(
                    in Identifier ctx_name, //name of context object
                    out Context child_ctx //newly created context object
                );
                Status delete(
                    in Flags del_flags //flags controlling deletion
                );
        };
};
```

Get_default_context

Essa operacao, a qual cria um pseudo-objeto Contexto, eh definida na interface ORB.

```
Status get_defult_context(
    out Context ctx //context object
            );
```

Essa operacao retorna uma referencia para um objeto contexto default de processo.

Set_one_Value

Essa operacao atualiza uma unica propriedade de objeto contexto. Atualmente, somente propriedades do tipo string sao suportadas pelo objeto contexto.

Topico 8 - DSI - Dynamic Skeleton Interface

A **interface do esqueleto dinamico** (DSI) eh um meio para entregar *requests* do ORB a uma implementacao de objeto que, em tempo de compilacao, nao tem nenhum conhecimento sobre o tipo de objeto que ela mesma implementa . Isto eh, um cliente faz um requerimento a um objeto, e a implementacao deste objeto por nao saber como executar este request, consulta a interface do esqueleto dinamico (interface para compor a interface do objeto apropriado), entao, cria o objeto apropriado para executar o request. Assim, o cliete obtem o resultado da operacao pedida na invocacao, atraves de um objeto instanceado de uma interface construida dinamicamente. Tudo isto ocorre sem o cliete usar a DII [veja o topico 7]. A DSI eh o lado do servidor analogo ao lado DII do cliente. Da mesma forma que a implementacao de um objeto nao pode saber se o cliente fez a invocacao usando DII ou tipos especificos robustos de objetos, o cliente que faz a invocacao nao pode determinar se a implememtacao esta usando esqueleto de tipos especificos ou DSI para conectar a implementacao ao ORB.

Visao Geral
A ideia basica da DSI eh implementar todos os requerimentos de objetos particulares usando-se sempre uma mesma implementacao, isto eh, para varias necessidades a objetos diferentes eh usada sempre uma dada rotina que consegue causar o instanceamento de tais objetos. Entao, o ORB pode acessar varios objetos diferentes atravez de uma mesma implementacao de objeto, e esta implementacao por sua vez, acessa um repositorio de interfaces [veja o topico 9] e para montar um esqueleto de interface dinamicamente. O ORB usa essa rotina de implemetacao dinamica (DIR), como o ponto de acesso commum a varios objetos diferentes. `A DIR sao passados todos os parametros explicitos da operacao desejada pelo cliente e a operacao que foi requerida. A DIR, entao, pode usar o objeto invocado, seu adaptador e o repositorio de interface para aprender mais sobre o objeto particular e sua invocacao.

Topico 9 - O Repositorio de Interface

O repositorio de interface eh um componente do ORB que fornece armazenamento persistente das definicoes de interfaces (durante o tempo de vida do repositorio - em algumas implementacoes de ORB) - ele gerencia e fornece acesso a uma colecao de definicoes de objetos espscificados na OMG IDL [veja o topico 5].

Visao Geral

Para um ORB processar *requests* corretamente, ele deve ter acesso aas definicoes de objetos a serem manipulados. As definicoes de objetos podem ser acessiveis ao ORB por duas maneiras:

1. Incorporando as definicoes proceduralmente dentro de rotinas robustas (informacoes conhecidas em tempo de compilacao.
2. Como objetos acessados atraves do repositorio de interface acessivel dinamicamente.

Em particular, o ORB pode usar definicoes de objetos mantidas no Repositorio de Interface para interpretar e manipular os valores fornecidos no *request*.

O Escopo de um Repositorio de Interface

O repositorio de interfaces pode armazenar valores constantes, que podem ser usados em outras definicoes de interfaces ou para a conveniencia de programadores. Ele tambem pode armazenar typecodes, que sao valores que descrevem um tipo em termos de estrutura.

O repositorio de interface usa modulos como um meio de agrupar as interfaces e como meio de navegacao atraves de tais grupos usando nomes. Modulos podem conter constantes, typedefs, excecoes, definicoes de interfaces e outros modulos. Este repositorio eh o conjunto de objetos que representam a informacao no mesmo. Ha operacoes que atuam em sua estrutura aparente de objetos. Tambem ha operacoes que extraem informacao em uma forma eficiente, obtendo um bloco de informacoes que descreve uma interface inteira ou uma operacao inteira.

Um ORB pode ter acesso a diferentes repositorios de interfaces. Como mostrado na seguinte figura, a mesma interface **Doc** esta instalada em dois diferentes repositorios, um na SoftCo, Inc., que vende *Doc objects,* e uma na Customer, Inc., que compra *Doc Objects* da *SoftCo.* A *SoftCo* da' `a interface em questao o identificador de seu repositorio (repository id) . O cliente tambem tem sua interface **Doc** com o mesmo numero repository id do repositorio da SoftCo., de tal forma que o ORB do cliente sabera que estas interfaces sao a mesma e podera estabelecer comunicacao entre ORBs.

```
module softco{
    interface Doc id 123{
            void print ();
        };
};
```

```
module testfirst{
    module softco{
            interface Doc id 123{
                    void print ();
                };
        };
};
```

Estrutura e Navegacao.

Ha 3 maneira para localizar uma interface no Repositorio de interface.

1. Obtendo um objeto **InterfaceDef** diretamente do ORB.
2. Navegando atraves do espaco de nomes de modulos usando uma sequencia de nomes.
3. Localizando o objeto **interfaceDef** que corresponde a um particular identificador de repositorio.

Obter um objeto **InterfaceDef** diretamente eh util quando um objeto eh encontrado e seu tipo nao era conhecido em tempo de compilacao. Usando-se a operacao **get_interface()** numa *referencia a objeto,* fica possivel recuperar a informacao do repositorio de interface sobre o objeto.

Navegar no espaco dos nomes de modulos eh util quando a informacao sobre uma interface particular nomeada eh necessaria. Iniciando-se no modulo **root** do repositorio, fica possivel obter entradas por nome.

Localizar o objeto **InterfaceDef** pelo ID eh util quando procurando por uma entrada num repositorio de interface que corresponde a outro. Um ID deve ser globalmente unico. Usando o mesmo ID em dois repositorios, fica possivel obter o ID para uma interface em um repositorio, e entao obter informacao sobre aquela interface pelo outro repositorio que pode estar mais perto ou conter informacao adicional sobre a tal interface.

Interfaces do Repositorio de Interface

Varias interfaces abstratas sao usadas como interfaces bases para outros objetos no IR.

Um conjunto comum de operacoes eh usado para localizar objetos no Repositorio de Interfaces. Essas operacoes sao definidas nas interfaces abstratas **IRObject, Container, e Contained** descritas a seguir. Todos os objetos IR sao herdados do **IRObject,** o qual fornece uma operacao para identificacao do tipo atual de objeto. Objetos que sao containers (contedores) herdam operacoes de navegacao da interface abstrata **Container.** Objetos que sao contained (contidos) por outros objetos herdam operacoes de navegacao de interfaces **Contained.**

A interface **IDLType** eh herdada por todos os objetos IR que representam tipos IDL, incluido interfaces, typedefs, e tipos anonimos (como **SequenceDef** e **ArrayDef**), que nao sejam tipos primitivos IDL. A interface **TypedefDef** eh herdada por todos os tipos nomeados que nao **InterfaceDef**.
Esses tipos abstrados comentados acima nao sao instanciaveis.

Definicoes de tipos

Varios tipos sao usados atraves das definicoes das interfaces IR

```
module CORBA{
    typedef string identifier;
    typedef string ScopedName;
    typedef string RepositoryId;
    enum DefinitionKind {
        dk_none, dk_all, dk_Attribute, dk_Constant, dk_Exception ,
        dk_Interface,dk_Module, dk_Operation, dk_Typedef,
        dk_Alias, dk_Struct, dk_Union, dk_Enum, dk_Primitive,
        dk_String, dk_Sequence, dk_Array, dk_Repository
      };
};
```

Identifier sao simples nomes que identificam modulos, interfaces, constantes, typedefs, exceptions, attributes, and operations.

Um **DefinitionKind** identifica o tipo de um objeto IR.

IRObject

A interface **IRObject** representa a interface mais generica (analoga ao objeto TObject mais generico que existe na Object-Pascal do Delphi) da qual todas as outras interfaces sao derivadas .

```
module CORBA {
    interface IRObject {
        // read interface
        readonly attribute DefinitionKind def_Kind;
        // write interface
        void destroy ();
      };
};
```

Read Interface

O atributo **def_kind** identifica o tipo da definicao.

Write Interface

A operacao **destroy** anula a existencia do objeto. Se o objeto eh um **Container, destroy** eh aplicada em todos os objetos que ele contem.

Contained

A interface **Contained** eh herdada por todas as interaces do Repositorio de Interfaces que sao contidas por outros objetos IR. Todos os objetos, exceto o objeto root (**Repository)** e definicoes de anonimos (**ArrayDef, StringDef, e SequenceDef),** e tipos primitivos sao contidos por outros objetos.

```
module CORBA{
    typedef string VersionSpec;
    interface Contained : IRObject {
            //read/write interface
                attribute Repositoryid id;
                attribute identifier name;
                attribute VersionSpec version;
            //read interface
            readonly attribute Container defined_in;
            readonly attribute ScopedName absolute_name;
            readonly attribute Repository containing_repository;
            struct Description{
                DefinitionKind kind;
                any value;
            };
            Description describe ();
            //write interface
            void move(
                in Container new_container,
                in Identifier new_name,
                in VersionSpec new_version
            );
        };
};
```

Read Interface

Um objeto que eh contido por outro objeto tem um atributo **id** que identifica-o globalmente e um **nome** que o identifica dentro do **Container. Version** serve para identificar o objeto daqueles com o mesmo nome. O atributo **defined_in** indica em qual **Container** um objeto **Contained** esta definido. O atributo **absolute_name** eh um **ScopedName** que identifica um objeto **Contained** unicamente dentro do **Repository.**

A operacao **describe** retorna uma estrutura contendo informacao sobre a interface.

Write Interface

Atualizar o atributo **id** causa a mudanca do identificador glabal dessa definicao. Um erro eh retornado se o **id** especificado ja existe no repositorio deste objeto. Os atributos **name e absolute_name** sao analogos ao **id** quanto a mudanca de valor destes.

A operacao **move** automaticamente remove esse objeto de seu container corrente e adiciona-o no container especificado em **new_container.** O novo cantainer deve estar no mesmo repositorio e nao deve haver um objeto com o mesmo nome ja contido no novo container. O atributo **name** eh ,entao, mudado para **new_name** e o atributo **version** eh mudado para **new_version**.

Container

A interface **Container** eh usada para formar uma hierarquia de recipiente no Repositorio de Interface. Um **Container** pode conter qualquer numero de objetos derivados da interface **Contained**. Todos os **Container**s, exceto o **Repository,** tambem sao derivados do **Contained.**

```
module CORBA{
    typedef sequence<Contained> ContainedSeq;
    interface Container : IRObject {
            //read interface
            Contained lookup (in ScopedName Search_name);
            ContainedSeq contents(
                    in DefinitionKind limit_type,
                    in boolean exclude_inherited
                );
                ContainedSeq lookup_name(
                    in identifier search_name,
                    in long levels_to_search,
                    in DefinitionKind limit_type,
                    in boolean exclude_inherited
                );
                struct Description {
                    contained contained_object;
                    DefinitionKind kind;
                    any value;
                };
                typedef sequence<Description> DescriptionSeq;
                DescriptionSeq describe_contents (
                    in DefinitionKind limit_type,
                    in boolean exclude_inherited,
                    in long max_returned_objs
```

```
);
// write interface
MduleDef create_module(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version
);
ConstantDef create_constant(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType Type,
    in any value
);
StructDef create_struct(
    in RepositoryId id,
    in identifier name,
    in VersionSpec version,
    in StructMemberSeq members
);
UnionDef create_union(
    in RepositoryId id,
    in identifier name,
    in VersionSpec version,
    in IDLType discriminator_type,
    in UnionMemberSeq members
);
EnumDef create_enum(
    in RepositoryId id,
    in identifier name,
    in VersionSpec version,
    in EnumMemberSeq members
);
AliasDef create_alias(
    in RepositoryId id,
    in identifier name,
    in VersionSpec version,
    in IDLType original_type
);
InterfaceDef create_interface(
```

```
                in RepositoryId id,
                in identifier name,
                in VersionSpec version,
                in InterfaceDefSeq base_interfaces
            );
    };
};
```

Read Interface

A operacao **lookup** localiza uma definacao nesse container dado um nome com escopo definido usando as regras de scopo OMG IDL.

A operacao **contents** retorna a lista de objetos diretamente contidos por ou herdados pelo objeto. Essa operacao eh iniciada no Repository Object. Um cliente usa essa operacao para tomar conhecimento de todos so objetos contidos pelo repositorio, contidos por modulos com o Repositorio, etc.

limite_type : tem o proposito de indicar ate que tipo de objetos devem ser retornados.

exclude_inherited: se verdadeira, indica que a pesquisa de **lookup** deve procurar, tambem, nos objetos herdados.

A operacao **lookup_name** eh usada para localizar um objeto atraves de seu nome.

levels_to_search : com levels_to_search = -1 => **lookup** procura no objeto corrente e em todos os outros objetos contidos. Com levels_to_search = 1 => **lookup** procura no objeto corrente.

A operacao **describe_contens** eh uma combinacao de **contents e describe.** Para cada objeto retornado por **contents**, uma descricao do objeto tambem eh retornada.

Write Interface

A interface **Container** fornece operacoes para criar ModuleDefs, ConstantDefs, StructDefs, Uniondefs, EnumDefs, AliasDefs e InterfaceDefs como objetos contidos. O atributo **definid_in** de uma deifinicao criada com qualquer uma dessas operacoes eh inicializado para identificar o **Container** no qual a operacao eh invocada.

IDLType

A interface **IDLType** eh uma interface abstrata herdada por todos os objetos IR que representam tipos OMG IDL.

```
module CORBA{
    interface IDLType : IRObject{
            readonly attribute TypeCode type;
        };
```

};

O atributo **type** descreve o tipo definido por um objeto derivado de um **IDLType.**

Repository

Esta eh uma interface que fornece acesso global ao Interface Repository. O objeto **Repository** pode conter constantes, typedefs, exceptions, interfaces, e modulos. Ja que ela eh derivada de **Container**, eh possivel procurar qualquer definicao por **nome** ou **id.**

Pode haver mais que um Interface Repository em um particular ambiente ORB.

```
module CORBA {
    interface Repository : Container {
            //read interface
            Contained lookup_id (in RepositoryId search_id);
            PrimitiveDef get_primitive (in PrimitiveKind kind);
            //write interface
            StringDef create_string (in unsigned long bound);
            SequenceDef create_sequence(
                    in unsigned long bound,
                    in IDLType element_type
                );
                ArrayDef create_array(
                    in unsigned long length,
                    in IDLType lement_type
                );
        };
};
```

Read Interface

A operacao **lookup_id** eh usada para localizar um objeto em um **Repository** dado seu **RepositoryId.** Se o objeto procurado nao esta la, uma referencia a objeto = **nil** eh retornada.

A operacao **get_primitive** retorna uma referencia para um **PrimitiveDef** com o atributo kind especificado.

Write Interface

As operacoes **create_<type>** criam novos objetos definindo tipos anonimos. Como essas interfaces nao sao derivadas de **Container**, eh responsabilidade do chamador invocar **destroy** no objeto retornado.

ModuleDef

Um **ModuleDef** pode conter constants, typedefs, exceptions, interfaces e outros objetos modulo.

```
module CORBA{
    interface ModuleDef : Container, Contained {
    };
    struct ModuleDescription{
            identifier name;
            RepositoryId id:
            RepositoryId defined_in;
            VersionSpec Version;
        };
};
```

A operacao **describe** herdada para um objeto **ModuleDef** retorna um **ModuleDescription.**

ConstantDef Interface

Um objeto **ConstantDef** define uma constante nomeada.

```
module CORBA {
    interface ConstantDef : Contained {
            readonly attribute TypeCode type;
            attribute IDLType type_def;
            attribute any value;
        };
        struct ConstantDescription {
            identifier name;
            RepositoryId id;
            RepositoryId defined_in;
            VersionSpec version;
            TypeCode Type;
            any value;
        };
};
```

Read Interface

O atributo **Type** especifica o **TypeCode** descrevendo o tipo da constante. O tipo da constante deve ser um dos tipos simples (long, short,float,char,string,octet,etc.). O atributo **type_def** identifica a definicao do tipo da constante.

A operacao **describe** para um objeto **ConstantDef** retorna um **ConstantDescription.**

TypedefDef Interface

TypedefDef eh uma interface abstrata usada como uma interface base para todos objetos nomeados *non-objects* (estruturas, unioes, enumeracoes, e

alias). A interface **TypedefDef** nao eh herdada pelas definicoes de objetos para tipos privados ou anonimos.

```
module CORBA{
    interface TypedefDef: Containde, IDLType{
    };
    struct TypeDescription{
            identifier name;
            RepositoryId id;
            RepositoryId defined_in;
            VersionSpec version;
            TypeCode type;
        };
};
```

A operacao herdada **describe** para as interfaces derivadas de **TypedefDef** retorna um **Typedescription.**

Veja tambem *StructDef, UnionDef, EnumDef, AliasDef, PrimitiveDef, StringDef, SequenceDef, ArrayDef, ExceptionDef e AttributeDef* na pagina 6-19 de [1].

OperationDef

Uma **operationDef** representa a informacao necessaria para definir uma operacao de uma interface.

```
module CORBA{
    enum OperationMode {OP_NORMAL, OP_ONEWAY};
    enum ParameterMode {PARAM_IN, PARAM_OUT,
    PARAM_INOUT};
    struct ParameterDescription{
            Identifier name;
            TypeCede type;
            IDLType type_def;
            parameterMode mode;
        };
        typedef sequence<ParameterDescription> ParDescriptionSeq;
        typedef Identifier ContextIdentifijer;
        typefdef sequence<ContextIdentifier> ContextIdSeq;
        Typedef sequence<ExceptionDef> ExceptionDefSeq;
        typedef sequence<ExceoptionDescription> ExcDescriptionSeq;
        interface OperationDef: Contained{
            readonly attribute TypeCode result;
            attribute IDLType result_def;
            attribute ParDescriptionMode params;
```

```
            attribute OperationMode mode;
            attribute ContextIdSeq contexts;
            attribute ExceptionDefSeq exceptions;
        };
        struct OperationDescription {
            identifier name;
            RepositoryId id;
            RepositoryId defined_in;
            VersionSpec version;
            TypeCode result;
            OperationMode mode;
            ContxtIdSeq contexts;
            ParDescriptionSeq parameter;
            ExcDescriptionSeq exceptions;
        };
};
```

Read Interface

O atributo **result** eh um tipo **TypeCode** descrevendo o tipo do valor retornado pela operacao. O membro **mode** indica que um parametro eh in, out, ou inout. O **modo** da operacao pode ser oneway (i.e., nemhum output eh retornado) ou normal. O atributo **contexts** especifica a lista de identificadores de contexto que aplicam-se a operacao.

InterfaceDef

Um objeto **InterfaceDef** representa uma definicao de interface. Ele pode conter constantes, Typedefs, exceptions, operacoes e stributos.

```
module CORBA{
    interface interfaceDef;
    typedef sequence<InterfaceDef> interfaceDefSeq;
    type sequence<RepositoryId> RepositoryIdSeq;
    type sequence<OperationDescription> OpDescriptionSeq;
    typedef sequence<AttributeDescription> AttrDescriptionSeq;
    interface InterfaceDef : Container, Contained, IDLType {
            //read/write interface
            attribute InterfaceDefSeq base_interface;
            // read interface
            boolean is_a(in RepositoryId interface_id);
            struct FullInterfaceDescription{
                    Identifier name;
                    RepositoryId Id;
                    RepositoryId defined_in;
```

```
            VersionSpec version;
            OpDescriptionSeq operations;
            AttrDescriptionSeq operations;
            RepositoryIdSeq base_interface:
            TypeCode type;
        };
        FullInterfaceDescription describe_interface ();
        //write interface
        AttributeDef create_attribute(
            in RepositoryId id,
            in Identifier name,
            in versionSpec version,
            in IDLType type,
            in AttributeMode mode;
        );
        OperationDef create_operation(
            in RepositoryId id,
            in Identifier name,
            in versionSpec version,
            in IDLType type,
            in OperationMode mode,
            in ParDescriptionSeq params,
            in ExceptionDefSeq exceptions,
            in ContextIdSeq contexts
        );
    };
    struct InterfaceDescription{
        Identifier name;
        RepositoryId id;
        RepositoryId defined_in;
        VersionSpec version;
        RepositoryIdSeq base_interfaces;
    };
};
```

Read Interface

O atributo **base_interfaces** lista todas as interfaces das quais esta eh herdada. A operacao **is_a** retorna **true** se a interface na qual foi invocado a operacao eh identica ou herda, direta ou indiretamente, da interface identificada por esse parametro **interface_id.** Caso contrario, retorna **false**.

RepositoryIds

RepositoryIds sao valores que podem ser usados para estabelecer a identificacao da informacao no repositorio. Um **RepositoryId** eh representado como uma string, permitindo programas armazenar, copiar, e comparar eles sem recorrer a uma estrutura de valor.

Veja TypeDef na pagina 6-33 de [1.]

Veja a gramatica OMG IDL completa para *Interface Repository* na pagina 6-41 de [1].

Topico 10 - A Interface ORB

A interface ORB eh uma interface daquelas funcoes do ORB que nao dependem de qual adaptador de objeto esta sendo usado. Essas operacoes sao sempre as mesmas para todos os ORBs e todas as implementacoes de objetos, e podem ser executadas por clients de objetos ou implementacoes. Algumas dessas operacoes parecem estar no ORB, outras parecem estar nas *referencias a objetos*. Ja que as operacoes ditas aqui sao implementadas pelo proprio ORB, elas nao sao, de fato, operacoes nos objetos - por isso sao chamadas de *pseudo objetos*.

A interface ORB tambem define operacoes para criar listas e determinar o contexto default usado na DII [veja o toico 7].

Convertendo *Referencias a Objetos* em Strings

Pelo fato de uma referencia a objeto ser opaca e ter a capacidade de diferir de ORB a ORB, a referencia a objeto em si nao eh um valor conveniente para armazenar referencias a objetos num armazenamento persistente ou comunicar referencias por meios diferentes de invocacoes. Dois problemas devem ser resolvidos: permitir uma referencia a objeto ser convertida em um valor que um cliente pode armazenar de alguma forma, e certificar-se que tal valor podera ser convertido novamente na referencia a objeto apropriada.

Uma referencia a objeto pode ser traduzida para um string pela operacao **object_to_string.** O valor pode ser armazenado ou comunicado de qualquer maneira que tambem seja possivel a uma string. Subsequentemente, a operacao **string_to_object** aceitara uma string produzida pela operacao **object_to_string** e retornara a referencia ao objeto correspondente.

```
module CORBA {
interface ORB{
    string object_to_string (in Object obj);
    Object string_to_object(in string str);
    Status create_list(
    in long count,
```

```
    out NVList new_list
    );
    Status create_operation_list(
    in OperationDef oper,
    out NVList new_list
    );
    Status get_default_context( out Context ctx);
};
};
```

Para garantir que um ORB ira entender a forma string de uma referencia a objeto, a operacoe **object_to_string** ORB deve ser usada para produzir um string.

Operacoes do ORB para objetos

Ha algumas operacoes que podem ser feitas em qualquer objeto. Essas operacoes sao implemetadas diretamente no ORB.

```
module CORBA {
    interface Object{
            ImplementationDef get_implementation ();
            InterfaceDef get_interface ();
            boolean is_nil();
            Object duplicate ();
            void release();
            boolean is_a(in string logical_type_id);
            boolean non_existent();
            boolean is_equivalent(in Object other_object);
            unsigned long hash(in unsigned long maximum);
            Status create_request(
            in Context ctx,
            in Identifieroperation,
            in NVList arg_list,
            inout NamedValueresult,
            out Requestrequest,
            in Flags req_flags
            );
        };
};
```

Uma operacao na referencia a objeto, **get_ interface,** retorna um objeto no repositorio de interface, [veja o topico 9] o qual fornece informacao de tipo que pode ser util a um programa. Uma operacao no objeto chamada

get_implementation retornara um objeto em um repositorio de implementacao que descreve a implemntacao do objeto.
Duplicando e liberando copias de referencias a objetos
Object duplicate();
void release();
Se mais que uma copia de referencia a um objeto eh necessaria, o cliente pode criar um **duplicate**. Note que a implementacao do objeto nao eh envolvida nesta duplicacao, e que a implementacao nao pode distinguir se a original ou a duplicata foi usada em um particular request.
Qunado uma referencia a um objeto ja nao eh mais necessaria por um programa, o armazenamento da mesma pode ser liberado pelo uso da operacao **release();** Aqui, tambem, a implementacao do objeto nao eh envolvida e qualquer outra referencia a tal objeto e o proprio nao sao afetados.
Referencias Nil
Uma referencia a objeto que tem valor OBJECT_NIL denota nenhum objeto. Uma referencia a objeto pode ter o valor testado quanto a isto pela operacao **is_nil.** A implememtacao do objeto nao eh envolvida em tal teste.
Operacao de checagem de equivalenvia.
Uma operacao esta definida para facilitar a manutencao de segurnaca de tipo para referencias a objetos sobre o escopo de um ORB, isto tem grande importancia em CORBA *Language Bindings* para linguagens que nao suportam **dynamic-cast**.
boolean is_a (in string logical_type_id);
The **logical_type_id** eh um string denotando um identificador de tipo compartilhado (RepositoryId). A operacao retorna **true** se o objeto eh realmente uma instancia de tal tipo, incluido os tipos ancestrais ao tipo do objeto.
Hashing
Referencias a objetos sao associadas com identificadores internos no ORB que podem ser acessados indiretamente por aplicacoes usando a operacao **hash().** O valor de tais identificadores nao mudam durante o tempo de execucao do objeto. Observe que dois objetos com valores **hash** diferentes sao garantidamente diferentes, enquanto que dois objetos com valores **hash** identicos nao sao garantidamente identicos, devido a colisao entre os valores **hash**.
Teste de Equivalencia
A operacao **is_equivalente()** eh usada para determinar se duas referencias a objetos sao equivalentes. Se o retorno eh **true** entao o objeto em questao eh igual ao objeto passado como paramentro.
Se duas referencias a objetos sao identicas, elas sao equivalentes. Duas referencias a objeto diferentes que referem ao mesmo objeto sao equivalentes.

Inicializacao do ORB e do OA e Referencias Iniciais

Antes que uma aplicacao possa entrar no ambiente CORBA, ela deve primeiro:

- Ser inicializada em um ambiente ORB e object adapter [veja o topico 11] (BOA e OA).
- Obter referencias aos pseudos-objetos ORB e OA (incluindo BOA) - e algumas vezes a outro objetos - para uso futuro de operacoes ORB e OA.

CORBA V2.0 fornece operacoes para inicializar aplicacoes e obter apropriadas referencias a objetos:

- operacoes que fornecem acesso ao ORB. Essas operacoes residem no modulo CORBA, mas nao na interface ORB.
- operacoes que fornecem acesso ao Basic Object Adapter (BOA) e outro adaptadores de objetos (OAs). Essas operacoes residem na interface ORB.
- operacoes que fornecem acesso ao Repositorio de Interface. Essas operacoes residem na interface ORB.

Inicializacao ORB

Quando uma aplicacao requer um ambiente corba, eh necessario um mecanismo para obter as referencias a pseudo-objetos ORB e OA. Isto serve para dois propositos: Primeiro, inicializar uma aplicacao dentro de um ambinte ORB e OA. Segundo, retornar as referencias a pseudo-objetos ORB e OA aa aplicacao para futuras operacoes ORB e OA. A operacao de inicializacao de ORB e OA deve ser ordenada com a inicializacao do ORB ocorrendo antes do OA.

Inicializacao BOA e OA

Um ORB pode ter zero ou mais *object adaptors* associados com ele. Servidores devem ter uma referencia ao OA com o proposito de acessar sua funcionalidade. O unico object adapter definido na CORBA eh o Basic Object Adapter (BOA). Entretanto, outros tambem podem ser mencionados, como o Library Object Adapter (LOA). Dada uma referencia ORB, uma aplicacao deve ser capaz de inicializar a si propria em um ambiente OA e obter a referencia do pseudo objeto OA pelo ORB.

Topico 11 - BOA - Basic Object Adapter

Um adaptador de objeto eh uma interface primaria que uma implementacao usa para acessar funcoes do ORB. O Basic Object Adapter (BOA) eh uma interface com a funcao de instancear e dar suporte a uma ampla variedade de objetos. O BOA inclui interfaces convenientes para a geracao de referencias a objetos,

registro de implementacoes que consistem em um ou mais programas, ativacao de implementacoes, e autenticacao de *requests*.
As seguintes funcoes sao fornecidas pelo BOA:

- Geracao e interpretacao de *object references*.
- Ativacao e desativacao de implemetacoes
- Ativacao e desativacao de objetos individuais
- Invocacao de metodos pelos esqueletos.

O BOA suporta implememtacoes de objetos que sao construidas de um ou mais programas. O BOA ativa e se comunica com esses programas usando facilidades do sistema operacional que nao sao parte do ORB. Portanto o BOA requer alguma informacao que nao eh portavel por heranca .
O mecanismo para a amarracao de programas ao BOA e ao ORB nao esta especificado porque isto eh dependente de linguagem e do sistema. Se assume que o BOA pode conectar os metodos ao esqueleto por alguma maneira, quando a implememtacao eh compilada, instalada ou ativada, etc. Subsequentemente aa ativacao, o BOA pode fazer chamadas em rotinas da implememtacao e a implementacao pode fazer chamadas ao BOA.

[figura 1]
Esta figura mostra a estrutura basica do BOA, e algumas das interacoes entre o BOA e uma implementacao de objeto. O BOA ira iniciar um programa para fornecer uma implementacao de objeto (1). A implementacao do objeto notifica ao BOA que sua inicializacao foi terminada (a impl. do objeto esta pronta) e que ele esta pronto para receber requerimentos (2). Quando a implementacao de objeto recebe o primeiro requerimento, o objeto eh ativado (3). Em subsequentes requerimentos, o BOA chama o metodo apropriado usando o esqueleto por interface (4). Em varios momentos, a implementacao pode acessar servicos BOA tais como criacao de objetos, desativacao, e assim por diante (5).
O BOA exporta operacoes que sao acessadas por implementacoes de objetos. O BOA tambem chama a implementacao de objeto sob algumas circunstaceas.

A interface BOA
Em pratica, o BOA eh comumente implementado parcialmente como um componente e parcialmente como uma biblioteca na implementacao de objeto. O componente separado eh necessario para fazer uma ativacao quando a implementacao nao esta presente. A parte biblioteca eh necessaria para estabelecer a ligacao dos metodos com o esqueleto. A exata particao da funcionalidade entre estas partes eh dependente de implementacao. Quando uma implementacao de objeto eh invocada, algumas operacoes sao satisfeitas pela bilblioteca, outras pelo servidor externo, e algumas pelo ORB Core.
module CORBA {

```
interface InterfaceDef; //from Interface Repository
interface ImplementationDef; //from Implementation Repository
interface Object; //an Object reference
interface Principal; //for the authentication service
typedef sequence <octet, 1024> ReferenceData;
interface BOA {
        Object create(
            in ReferenceData id,
            in InterfaceDef intf,
            in ImplementationDef impl
        );
        void dispose (in Object obj);
        ReferenceData get_id (in Object obj);
        void change_implementation(
            in Object obj,
            in ImplementationDef impl
        );
        Principal get_principal(
            in Object obj,
            in Environment ev
        );
        voi set_exception(
            in exception_type major, //no, user,or system_exception
            in string userid, // exception type id
            in void *param //pointer to associated data
        );
        void impl_is_ready(in ImplementationDef impl);
        void deactivate_impl (
        in ImplementationDef impl);
        void obj_is_ready(in Object obj, in ImplementationDef impl);
        void deactivate_obj (in Object Obj);
    };
};
```

Os requerimentos de uma implementacao para o BOA sao dos seguintes tipos:

- Operacoes para criar ou destruir referencias a objetos, ou manipular a informacao que um BOA mantem para uma referencia a objeto.
- Operacoes associadas com um *request* particular.
- Operacoes para manter o registro dos objetos ativos e implemetacoes.

Os Requerimentos do BOA sao dos seguintes tipos:

- Ativar um implementacao

- Ativar um objeto
- Desenvolver uma operacao.

Registro das implementacoes

O BOA espera informacoes descrevendo as implementacoes que serao armazenadas em um Repositorio de Implementacao. Existem objetos de interface OMG IDL chamdos **ImplementationDef,** que capturam essas informacoes. O repositorio de implementacao pode conter informacao para depuracao, administracao, etc. Note que o repositorio de implementacao eh distinto do repositorio de interfaces, mesmo que, de fato, eh possivel implementa-los juntos. O repositorio de interfaces contem informacoes sobre as interfaces. O ORB Core pode ou nao pode fazer uso do repositorio de interface ou do repositorio de implementacao, mas o BOA usa esses objetos para associar referencias a objetos com suas interfaces e implementacoes.

Ativacao e Desativacao de Implementacoes

Existem dois tipos de ativacoes que um BOA necessita desenvolver como parte da invocacao de operacao. O primeiro eh a *ativacao de implementacao,* a qual ocorre quando nenhuma implementacao para um objeto esta disponivel para manipular um requerimento. O segundo eh a ativacao de objeto, a qual ocorre quando nenhuma instancia de objeto esta disponivel para manipular o requerimento.

A ativacao de uma dada implementacao requer a coordenacao entre o BOA e os programas contendo a implementacao. Este documento usa o termo *servidor* como a entidade executavel separadamente que o BOA pode 'startar' em um sistema particular.

Uma *activation policy* descreve as regras que uma dada implementacao segue quando ha multiplos objetos ou implementacoes ativas. Ha quatro *policies* que todas as implementacoes de BOA suportam para a ativacao de implementacoes:

- Uma *sharede server policy,* onde multiplos objetos ativos, de uma da implementacao, compartilham o mesmo servidor.
- Uma *unshared server policy,* onde somente um dado objeto, de uma implementacao, em cada momento pode estar ativo em um servidor.
- Uma *server-per-method policy,* onde cada invocacao de um metodo eh implementada por um servidor separado sendo iniciado, e quando o metodo termina, o servidor eh terminado .
- Uma *persistent server policy,* onde o *server* eh ativado por algo fora do BOA. Um *server* persistente eh considerado como sendo compartilhado por multiplos objetos ativos.

Os tipos de ativacao de implementacao estao ilustrados na figura seguinte:

[figura 2]

A eh um servidor compartilhado, onde o BOA inicia um processo e entao esse processo registra-se com o BOA. B eh um caso de um servidor persistente, que eh

muito semelhante ao anterior, mas ele apenas registra-se com o BOA, sem esse ultimo ter que iniciar um processo. Um servidor nao compartilhado esta ilustrado no caso C, onde o processo foi iniciado pelo BOA e pode manter somente um objeto. Um servidor por metodo no caso D faz com que cada invocacao a um metodo seja feita iniciando-se um processo.

Shared Server Activation Policy

Em um servidor compartilhado, multiplos objetos podem ser implementados pelo mesmo programa. Este eh o tipo mais comum de servidor. O servidor eh ativado assim que um requerimento eh feito em qualquer objeto implementado pelo servidor. Quando o servir esta inicializado, ele notifica sua prontidao ao BOA chamando a rotina **impl_is_ready.** Subsequentemente, o BOA ira entregar requerimentos ou ativacoes de objetos para qualquer objeto implementado pelo servidor. O servidor permanece ativo e ira receber requerimentos ate que ele chame **deactive_impl.** O BOA nao ira ativar outro servidor para esta implementacao se um ja estiver ativo.

Um objeto permanece ativo enquanto o seu servidor esta ativo, a menos que o servidor chame **deactivate_obj** para o objeto.

Unshared Server Activation Policy

Esse tipo de servidor eh conveniente se o proposito de um objeto eh encapsular uma aplicacao ou se o servidor requer exclusivo acesso a um recurso tal como uma impressora. Um novo servidor e iniciado em qualquer momento que um *request* eh feito a um objeto que ainda nao esta ativo, ate mesmo se um servidor para outro objeto com a mesma implementacao estiver ativo.

Server-per-Method Activation Policy

Nesta tatica, um novo servidor eh sempre iniciado a cada vez que um requerimento eh feito. O servidor 'roda' somente durante a execucao do metodo particular. Varios servidores para o mesmo objeto ou para o mesmo metodo de um mesmo objeto podem estar ativos num mesmo momento. Pelo fato de um novo servidor ser iniciado a cada requerimento, nao eh necessario aa implementacao notificar ao BOA quando um objeto esta pronto ou desativado.

Pesistent Server Activation Policy

Servidores persistentes sao aqueles ativados por meios fora do BOA. Tais implementacoes notificam ao BOA que elas estao disponiveis usando a **impl_is_ready**. Uma vez que o BOA tem conhecimento do servidor persistente, ele trata este servidor como sendo um servidor compartilhado. Se nenhuma implementacao esta pronta quando um requerimento chega, um erro eh retornado para aquele requerimento.

Geracao e Interpretacao de Referencias a Objetos

Referencias a objetos sao geradas pelo BOA usando o ORB Core quando requerido por uma implementacao. O BOA e o ORB Core trabalham juntos para

associar alguma informacao com uma referencia particular a um objeto. Mais tarde, essa informacao eh usada pela implementacao durante a ativacao de um objeto. Note que esta eh a unica informacao que uma implementacao pode usar portavelmente para distinguir diferentes referencias a objetos. A operacao BOA usada para criar uma nova *object reference* eh:

Object create(
in ReferenceData id,
in InterfaceDef intf,
in ImplementationDef impl
);

O **id** eh uma informacao imutavel de identificacao, escolhida pela implementacao em tempo de criacao de objeto, e nunca muda duranrte a vida do objeto. O **intf** eh o objeto Repositorio de Interface que especifica o conjunto completo de interfaces implementadas pelo objeto. O **impl** eh o objeto Repositorio de Implementacao que especifica a implementacao a ser usada pelo objeto. Note que a propria referencia a objeto eh opaca e pode ser diferente em diferentes ORBs. Somente a implementacao pode interpretar normalmente o valor do **id.** A operacao para obter o **id** eh um operacao BOA:

ReferenceData get_id(in Object obj);

Eh possivel a uma implementacao associada com uma referencia ser mudada. Isso ira fazer com que novos requerimentos sejam manipulados conforme a informacao na nova implementacao. A operacao para mudar a implementacao eh uma operacao BOA:

void change_implementation(
in Object obj,
in ImplementationDef impl
);

Se uma referencia a um objeto eh copiada, toda as copias tem o mesmo **id, intf e impl.**

Uma implementacao pode descartar um objeto que ela tinha criado chamando o BOA para invalidar a respectiva referencia. A implementacao eh responsavel por desalocar quaisquer outras informacoes sobre o objeto. Depois que um **dispose** eh feito, o ORB Core e o BOA ajem como se o objeto nao existiu no passado.

Armazenamento Persistente

Objetos (ou mais precisamente, referencias) sao feitos persistentes pelo BOA e pelo ORB Core, de tal forna que um cliente que tem um referencia a um objeto pode usa-la em qualquer momento sem se preocupar, mesmo se a implementacao foi desativada ou o sistema foi "reestardado".

Para este proposito, o BOA fornece uma pequena quantidade de armazenamento para um objeto no valor **id.** Na maioria dos casos, esse armazenamento eh

insuficiente e inconveniente para o estado completo do objeto. Entao, a implementacao fornece e gerencia tal armazenamento de informacao, usando o **id** como uma chave para localizar a informacao correta em alguma forma de armazenamento de informacoes. Por exemplo, o **id** pode conter o nome de um arquivo, ou a chave para um DB que mantem o estado persistente.

O Exemplo "Hello World".

Aqui esta presente um exemplo que mostra como um cliente, que deseja uma execucao de uma determinada operacao, pode tirar proveito do ambiente CORBA. A vantagem aqui demostrada fica mais evidente quando o cliente necessita de uma operacao que pode ser executada em um sistema distribuido, com o cliente desconhecendo onde ela sera executada (em qual maquina). O exemplo usado aqui eh o famoso 'Hello Word', que mostrara como um metodo que imprime 'hello' pode ser usado por um cliente que o necessita mas desconhece como instancear diretamente um objeto que forneca tal operacao.

Suponha um sistema 'rodando' numa maquina A de uma rede, como um protocolo de rede desconhecido. Suponha, tambem, que existe um ambiente CORBA nesta rede. Um exemplo de ORB, para suportar um ambiente CORBA pode ser o OmniBroker versao 1.0 [2]. Nosso exemplo ira supor o uso deste ORB. Ja que existe a arquitetura CORBA, um repositorio de interface esta presente com a seguinte interface:

```
// IDL
interface Hello {
        void hello();
}
```

A interface mostrada esta construida sob a gramatica da IDL e deve estar num arquivo com terminacao '.idl', no nome. Por exemplo, Hello.idl. Esta interface deve ser conhecida pelo sistema cliente, no caso de invocacao nao dinamica, o qual fica sabendo que pode haver um objeto, instanceado em algum lugar, por exemplo numa maquina B, que implemente a operacao desejada hello(). Eh o servidor que se encarrega de construir a operacao hello(). No caso do [2], as interfaces IDL sao convertidas para codigo C++, para que o servidor possa implementar a operacao da interface. Assim, usando o comando *idl Hello.idl* os seguintes arquivos sao automaticamente criados:

Hello.h, Hello.cpp, Hello_skel.h e Hello_skel.cpp.

O unico arquivo interessante, do ponto de vista do usuario do [2], eh o **Hello_Skel.h.** Esse arquivo deve ser incluido na implementacao da classe.

Implementando o Servidor

Para implementar o servidor, nos precisamos definir uma classe para a interface Hello definida no paragrafo previo. Para isso, nos criamos uma classe **Hello_impl** que eh derivada da classe esqueleto **Hello_skel,** definida no arquivo **Hello_skel.h.**

```
// C++
#include <Hello_skel.h>
class Hello_impl : public Hello_skel
{
public:
        Hello_impl();
        virtual void hello();
    };
```

A implementacao para essa classe eh:

```
//C++
#include <CORBA.h>
#include <Hello_impl.h>
Hello_impl::Hello_impl()
{
}
void Hello_impl::hello()
{
        cout << "Alo Alo mundao!" << endl;
    }
```

A classe Hello_impl eh salva num arquivo chamado Hello_impl.h e sua implementacao eh salva num arquivo chamado Hello_impl.cpp. Agora precisamos escrever o programa principal para implementar o servidor:

```
//C++
#include <CORBA.h>
#include < Hello_impl.h>
#include <fstrem.h>
int main(int arg, char* argv[], char*[])
{
CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
CORBA_BOA_VAR boa = orb->BOA_init(argc, argv);
hello_var p = new Hello_impl;
CORBA_String_var s = orb->object_to_string(p);
const char* refFile = "Hello.ref";
ofstream out(refFile);
out << s << endl;
out.close();
boa->impl_is_ready(CORBA_implemetationDef::_nil());
```

```
}
```

Esta implementacao eh salva num arquivo chamado S**erver.cpp.**

A primeira coisa que deve ser feita, no servidor, eh inicializar o ORB e o BOA:

```
CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
CORBA_BOA_VAR boa = orb->BOA_init(argc, argv);
```

Os parametros argc e argv sao passado para main quando o programa eh inicado. Esses parametros podem ou nao podem ser usadas. Isso depende da implementacao do CORBA.

Em segundo, uma instancia do objeto conveniente eh criada:

```
hello_var p = new Hello_impl;
```

Hello_var, como qualquer tipo _var, eh um apontador "esperto", i.e. p ira liberar o objeto criado pelo *new hello_impl* automaticamente quando p sair do escopo.

O cliente deve ser capaz de acessar a implementacao do objeto. Isso pode ser feito salvando-se uma referencia a objeto "stringfield" num arquivo a qual pode ser lida pelo cliente e convertida para uma referencia novamente. A operacao *object_to_string()* converte uma referencia a objeto CORBA em sua representacao string:

```
CORBA_String_var s = orb->object_to_string(p);
const char* refFile = "Hello.ref";
ofstream out(refFile);
out << s << endl;
out.close();
```

Finalmente, o servidor deve entrar em seu evento loop o qual , agora, fica pronto para receber requerimentos. Isso eh feito por :

```
boa->impl_is_ready(CORBA_implemetationDef::_nil());
```

Implementando o Cliente

Escrever o cliente requer menos trabalho que escrever o servidor, desde que o cliente, neste exemplo, consiste somente de uma funcao main().

```
//C++
#include <CORBA.h>
#include <Hello.h>
#include <fstream.h>
int main(int argc, char* argv[], char*[])
{
CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
CORBA_BOA_VAR boa = orb->BOA_init(argc, argv);
const char* refFile = "hello.ref";
ifstream in(refFile);
char s[1000];
in >> s;
CORBA_Object_var obj = obj->string_to_object(s);
```

```
    assert(!is_nil(obj));
    Hello_var hello = Hello::_narrow(obj);
    assert(!is_nil(hello));
    hello->hello();
    }
```

Essa implementacao eh salva num arquivo chamado **Client.cpp.**

Primeiro, eh necessario iniciar o ORB e o OBA .

O proximo passo eh transformar a string novamente a uma object reference. A operacao _narrow(), entao, gera um objeto **Hello** .

Finalmente, a operacao Hello pode ser usada pelo cliente.

Conclusoes:

O servidor deve ser iniciado antes do cliente, ja que o cliente precisa de uma string que eh gerada pelo servidor.

Se voce implementar esse exemplo com o servidor numa maquina A e o cliente em maquina B, entao, de alguma forma, voce tera que disponibilizar a string criada ao cliente, antes de iniciar o cliente. Voce pode iniciar o servidor e copiar a string para a maquina onde estara o cliente e, entao , iniciar o cliente.

A primeira vista, esta pode ser uma maniera muito complicada para se obter um Hello. Por outro lado, pense nos beneficios que isto pode oferecer. Voce pode iniciar o cliente e o servidor em diferentes pares de maquinas , com os mesmos resultados exatamente. Voce nao precisa se preocupar com metodos especificos de plataforma, relacionados com a comunicacao do cliente e servidor, e/ou nao precisa se preocupar com o protocolo usado. Existe um ORB para a plataforma de sua escolha.