Instituto de Computação
Universidade Estadual de Campinas

Gerência de Sistemas Distribuídos Heterogêneos: Facilidade de Monitorização em um ambiente CORBA

João Augusto Gomes de Queiroz

dezembro de 1997

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)
  Instituto de Computação — UNICAMP
- Prof. Dra. Noemi de La Rocque Rodriguez
  Departamento de Informática — PUC - Rio
- Prof. Dr. Paulo Lício de Geus
  Instituto de Computação — UNICAMP
- Prof. Dr. Luiz Eduardo Buzato (Suplente)
  Instituto de Computação — UNICAMP

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas, como requisito parcial para a obtenção do título de mestre em Ciência da Computação

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Queiroz, João Augusto Gomes de

Q32g Gerência de sistemas distribuídos heterogêneos: facilidade de monitorização em um ambiente CORBA / João Augusto Gomes de Queiroz -- Campinas, [S.P. :s.n.], 1997.

Orientador : Edmundo Roberto Mauro Madeira

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Gerência. 2. Sistemas operacionais distribuídos (Computadores). 3. Redes de computação. 4. Programação orientada a objetos. I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Gerência de Sistemas Distribuídos Heterogêneos: Facilidade de Monitorização em um ambiente CORBA

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por João Augusto Gomes de Queiroz e aprovada pela Banca Examinadora.

Campinas, 15 de dezembro de 1997

Edmundo R Mauro Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Tese de Mestrado defendida e aprovada em 15 de dezembro de 1997 pela Banca Examinadora composta pelos Professores Doutores

Noemi de la Rocque Rodriguez

Profª. Drª. Noemi de La Rocque Rodriguez

Prof. Dr. Paulo Lício de Geus

Edmundo R Mauro Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira

Resumo

Este trabalho apresenta a Facilidade de Monitorização, no contexto de uma Arquitetura de Gerência Integrada, e analisa a sua implementação. Descreve também uma proposta de extensão do Serviço de Eventos CORBA, motivado pela necessidade de se ter um modelo de monitorização assíncrono flexível, genérico e configurável. A Arquitetura é baseada na ODMA da ISO, cuja proposta é fundamentada no Modelo de Referência ODP, estendendo o padrão OSI/CMIP de gerência; e adota como infra-estrutura de suporte a CORBA, os CORBAservices e as CORBAfacilities, onde os objetos gerenciados são os componentes das aplicações e os elementos de suporte em um ambiente de processamento distribuído aberto. O protótipo implementa uma Base de Informação de Gerência, definida em IDL, para aplicações distribuídas CORBA, acessível por gerentes CORBA, com informações úteis para as áreas funcionais de desempenho e de contabilização. Os resultados observados com aplicações num ambiente heterogêneo, como o da Multiware, levam-nos à direção da necessidade de se adotar o conceito pleno de gerência integrada de sistemas distribuídos.

Abstract

The main goal of this work is to present a Monitoring Facility in the context of the Integrated Management Architecture and to analyze its implementation. It also describes an extension of the CORBA Event Service, motivated by the needs of asynchronous monitoring, designed to be flexible, generic and suitable to management. The Architecture is based on the ODMA/ISO, which is compliant with the RM-ODP to extend the OSI/CMIP management standard; uses the CORBA, the CORBAservices and the CORBAfacilities, where the target objects are the application components and the system support in an ODP environment. The prototype implements a Management Information Base defined in IDL for CORBA distributed application, accessible by CORBA managers, and provides management information for performance and accounting functional areas. The results observed from applications over a heterogeneous environment, like the Multiware platform, lead us to the need of a full integrated distributed system management.

À minha mulher Mônica e aos meus filhos
Igor e Eduardo pelo seu apoio e carinho.

Agradecimentos

Desejo expressar meus sinceros agradecimentos ao Professor Edmundo Madeira, meu orientador, pela confiança depositada no início deste trabalho, pela maneira cavalheira e serena que tem me orientado.

Agradeço aos meus colegas Flávio, Francisco, Geraldo e Nuccio pela amizade e o auxílio espontâneos.

Agradeço aos membros do "staff", professores do Instituto de Computação, por suas valiosas contribuições para a minha formação.

Aos meus amigos de Praça D'Armas: Nuccio Zuquello, Marques, Silva Roberto, Lúcio Dutra, Paulo Pagliusi e Francisco Vasconcellos.

À Diretoria de Telecomunicações da Marinha pela oportunidade concedida e ao Centro de Coordenação de Estudos da Marinha em São Paulo pelo apoio administrativo.

Não posso deixar de agradecer ao Clube do Café pelos momentos de descontração, principalmente durante a fase da implementação do protótipo.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AB	Architecture Board
ACSE	Association Control Service Element
ANSA	Advance Networked Architecture
API	Application Programming Interface
APM	Architecture Projects Management
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
AWT	Abstract Windowing Toolkit
B-ISDN	Broadband Integrated Services Digital Network
C4I	Command, Control, Communication, Computer and Information
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
CORBA	Common Object Request Broker Architecture
CORBAfacilities	Common Facilities Architecture
CORBAservices	Common Object Services Specification
CSCW	Computer Supported Cooperative Work
DCE	Distributed Computing Environment
DCF	Data Communications Functions
DCN	Data Communications Network
DES	Data Encryption Standard
DME	Distributed Management Environment
DN	Distinguished Name
DNS	Domain Name System
ESIOP	Environment-Specific Inter-ORB Protocol
FDDI	Fiber Distributed Data Interface
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Inter-ORB Protocol
HMA	Human Machine Adaptation
HTML	Hypertext Markup Language
ICF	Information Conversion Function
IDL	Interface Definition Language
IEEE	Institute of Eletrical and Eletronics Engineers
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
ITU-T	International Telecommunications Union
JDK	Java Development Kit
LLA	Logical Layered Architecture
MAF	Management Application Function
MCF	Message Communication Function
MD	Mediation Device

MD5	Message Digest 5
MF	Mediation Function
MIB	Management Information Base
MIT	Management Information Tree
MO	Managed Object
MOM	Monitor and Message-Oriented Middleware
NE	Network Element
NEF	Network Element Function
NMF	Network Management Forum
ODMA	Open Distributed Management Architecture
ODP	Open Distributed Processing
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
OS	Operations Systems
OSF	Operations System Function
OSF	Open Software Foundation
OSI	Open Systems Interconnection
otmsd	Orbix Talk Message Store Daemon
OTRMP	Orbix Talk Reliable Multicast Protocol
OTSFP	Orbix Talk Store-and-Forward Protocol
PDU	Protocol Data Unit
PF	Presentation Function
QA	Q Adapter
QAF	Q Adapter Function
RDN	Relative Distinguished Name
RFC	Request for Comments
RFP	Request for Proposals
RM-ODP	Reference Model for Open Distributed Processing
RMI	Remote Method Invocation
RMON-MIB	Remote Monitoring Management Information Base
ROSE	Remote Operation Service Element
RPC	Remote Procedure Call
SMI	Structure Management Information
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TMN	Telecommunications Management Network
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WS	Work Station
WSF	Work Station Function
WWW	World Wide Web

Capítulo 1

Introdução

As grandes transformações ocorridas no mundo, entre os séculos XIV e XVII, marcadas pelo Mercantilismo, como modelo econômico, e pelo Renascimento, no plano científico, cultural e artístico, têm explicações lógicas, embora nem sempre congruentes, nas diversas correntes filosóficas. O fato é que a humanidade deu um grande passo rumo ao desenvolvimento das idéias, dos costumes, da organização social e política e da configuração da ordem econômica que prevaleceria na Idade Moderna e, posteriormente, no mundo contemporâneo. Existe um consenso que um dos principais vetores dessas grandes transformações foi o advento do tipo móvel para impressão por Gutenberg em 1448. Vivemos, agora, uma transformação histórica comparável àquela época. A decadência dos sistemas centralizadores; a prevalência da democracia; o processo de globalização; a criação dos blocos econômicos como o Mercosul e o Alca; a exigência no universo empresarial de requisitos de qualidade e de produtividade; a necessidade de

um Estado eficiente; e a valorização da cidadania são alguns dos fenômenos que caracterizam a história contemporânea recente. Nesse contexto, acredita-se que as novas tecnologias da era da informação digital têm desempenhado a mesma missão da invenção de Gutenberg. Embora considere um pouco prematuro para concordar com tal comparação, ela reflete o impacto das novas tecnologias provocado nas relações sociais e econômicas. Tem sido freqüentemente comparada também à invenção da máquina a vapor, que marcou o começo da era industrial, motivando o início da nova era da informação. Esta era apresenta quatro características importantes: a descentralização, a globalização, a harmonização e a capacitação [Negroponte, 1995]. Particularmente, o efeito descentralizador manifesta-se com todo o vigor no comércio e na própria indústria de informática. Os ambientes de "mainframe", czar dos sistemas de gerência de informação que reinavam absolutamente, têm sido substituídos ou integrados por plataformas de sistemas distribuídos, em uma tendência que ficou conhecida como "downsizing" ou "rightsizing".

Os avanços tecnológicos recentes da Informática, com o desenvolvimento de microprocessadores com um poder cada vez maior, e das Telecomunicações, com a proposta de integração dos seus serviços oferecidos em uma rede digital de faixa larga, têm permitido a interconexão dos recursos computacionais das organizações e o acesso local e remoto aos seus Sistemas de Informação, em um processo irreversível de capilarização. Tais Sistemas de Informação são caracterizados pela heterogeneidade, sendo compostos basicamente por diversos elementos físicos de redes, por computadores de várias arquiteturas e por aplicações implementadas em diferentes linguagens de programação. Atualmente, as soluções adotadas são proprietárias, dependentes da tecnologia escolhida de "hardware" e de "software" e de seus fornecedores. Essa dependência restringe a implementação de simples mudanças ou de eventuais modernizações da infra-estrutura existente. A proposta atual é permitir a comunicação entre os componentes das aplicações dos Sistemas de Informações em ambientes distribuídos e heterogêneos, estendendo as suas fronteiras para além dos limites atuais impostos pelas linguagens de programação, pela arquitetura dos computadores, pelos sistemas operacionais e pelos protocolos de comunicação. Dessa forma, busca-se a interoperabilidade entre os componentes de tais sistemas.

(Bernstein, 1996) define interoperabilidade como a habilidade de um programa em um sistema ter acesso a programas e dados em um outro sistema. Por outro lado, os desenvolvedores de aplicações necessitam de interfaces de programação que ocultem a complexidade dos diversos e diferentes protocolos e sistemas operacionais, concentrando-se apenas nos requisitos significativos do projeto em si. É neste panorama que surgiu uma nova categoria de "software", conhecida como "middleware", com a função de conectar aplicações distribuídas e solucionar problemas de falta de transparência na comunicação entre os seus componentes e, ao mesmo tempo, permitir uma gerência simples e eficaz sobre todo o cenário hostil aos desenvolvedores, protegendo-os das complexidades dos protocolos de redes e dos sistemas operacionais. "Middleware" é uma camada de serviços que adota interfaces de programação e protocolos padronizados para resolver os problemas inerentes da distribuição e da heterogeneidade. A origem deste termo está na sua característica de estar acima dos sistemas operacionais e protocolos, e abaixo das aplicações distribuídas. Os "middlewares" podem ser classificados em quatro categorias: "Database Middleware", "Remote Procedure Calls" (RPC), "Object Request Broker" (ORB) e "Monitor and Message-Oriented Middleware" (MOM). Em todos os casos, um "middleware" permite a utilização simples e funcional de um conjunto de APIs (Application Programming Interfaces) que oferecem transparências de localização e de acesso; independência dos protocolos de redes; interação com outras aplicações de diferentes linguagens e sistemas operacionais. O aumento do uso de aplicações cliente/servidor, o crescimento da Internet e a incorporação destes produtos em ferramentas de desenvolvimento de sistemas são os principais fatores que estão impulsionando esta nova tecnologia.

O paradigma da orientação a objetos tem se mostrado uma ferramenta muito útil para o desenvolvimento de sistemas distribuídos, além de sua aplicabilidade ao desenvolvimento de componentes de software já existentes e testados, como ocorre em outras engenharias. A idéia de se modelar sistemas distribuídos como uma coleção de objetos interagindo é apropriada para integrar recursos em ambientes heterogêneos e distribuídos. Objetos oferecem um modelo natural para sistemas distribuídos abertos porque os componentes desses sistemas só podem se comunicar usando mensagens através de uma interface bem definida, adequando se às características de desacoplamento e extensibilidade. Neste contexto, uma forma de garantir a interoperabilidade entre objetos distribuídos é através da adoção de padrões de "juri", como o

Modelo de Referência ODP (Open Distributed Processing) dos organismos internacionais ISO (International Organization for Standardization) e ITU-T (International Telecommunications Union), e de "facto", como a arquitetura CORBA (Common Object Request Broker Architecture) do consórcio internacional OMG (Object Management Group), resultado de um consenso de mais de 700 empresas, centros de pesquisas e universidades. Suas propostas de "middleware" surgiram da necessidade de romper com o paradigma da arquitetura de distribuição em três camadas em que são delegadas às estações de trabalho o papel de interface com o usuário (primeira camada) e acesso (segunda camada) às aplicações servidoras (terceira camada) em máquinas mais robustas. Esses padrões formam o núcleo do projeto da plataforma "Multiware" (Loyolla, 1994), em desenvolvimento no Instituto de Computação e na Faculdade de Engenharia Elétrica e de Computação da Universidade de Campinas.

Tais propostas têm despertado a curiosidade da comunidade interessada em soluções de gerência tanto de elementos de redes quanto de sistemas de suportes e de aplicações distribuídas, motivada pelas restrições dos modelos atualmente existentes, SNMP (Simple Network Management Protocol) para a arquitetura Internet e CMIP (Common Management Information Protocol) para um ambiente OSI (Open Systems Interconnection), para se ter acesso aos recursos gerenciados. Embora a arquitetura de gerência TMN (Telecommunications Management Network) ofereça um modelo mais rico, sofre das mesmas restrições por ter incorporado o CMIP como protocolo de comunicação. Como exemplo disto, o Modelo de Referência ODP tem sido adotado como um meta-padrão para o desenvolvimento de padrões de gerência tanto em redes quanto em telecomunicações (Farooqui, 1997). Por outro lado, CORBA, além de oferecer uma plataforma distribuída aberta para aplicações, tem sido investigada como uma solução de interoperabilidade entre os domínios de gerência SNMP e CMIP (Soukoti, 1997).

O crescimento e a complexidade dos sistemas distribuídos têm tornado a sua gerência um tema de especial interesse para pesquisadores, desenvolvedores, vendedores e usuários. Tais ambientes são caracterizados principalmente por sua heterogeneidade, com uma diversidade muito grande de recursos de "software" e "hardware". As propostas de gerência nessa área devem abranger todos os componentes de tais sistemas com o propósito de entender as suas interações e correlações. Um sistema distribuído, como o ambiente da plataforma Multiware, consiste

tipicamente de um grande número de nós de diferentes arquiteturas, conectados por redes heterogêneas de comunicação, vários sistemas operacionais e muitos serviços de suporte, estabelecendo uma infra-estrutura para o desenvolvimento e a execução de aplicações distribuídas. Estas aplicações consistem de um conjunto de componentes cooperativos de "software" distribuídos. Como é adotado o paradigma da orientação a objeto para descrever e implementar este ambiente, seus elementos são objetos, com interfaces claramente definidas, cujos métodos podem ser invocados de qualquer nó do domínio estabelecido. Na arquitetura CORBA, os objetos residem em servidores, processos que oferecem um ou mais serviços, cujas interfaces são transparentemente chamadas por programas clientes. Portanto, os processos que implementam uma aplicação distribuída devem ser também considerados como recursos gerenciados.

O conceito de gerência pode ser definido como o conjunto de atividades, precauções e procedimentos estabelecidos para a supervisão e controle de um sistema gerenciado, garantindo a operação efetiva e eficiente de seus componentes, com o propósito de satisfazer as especificações e necessidades de seus usuários. O processo de gerência é composto por duas atividades essenciais: monitorização e controle (Sloman, 1993). A monitorização é feita através da observação e processamento dos dados sobre o comportamento dos componentes de um sistema gerenciado, permitindo o seu controle. O processo de gerência pode ainda ser ilustrado como uma ação contínua entre a entidade gerente e o sistema gerenciado em um ciclo fechado (Figura 1.1).

Em geral, a monitorização é responsável por prover informações de um sistema observado, úteis para o processo de tomada de decisões de gerência, com o propósito de controlar o comportamento de seus elementos. Em sistemas distribuídos, os problemas de tais atividades são ainda mais dificultados pelos aspectos da distribuição. Adicionalmente, o conceito de monitorização é diretamente oposto ao paradigma da encapsulação que protege o estado do objeto e os procedimentos associados da observação externa, escondendo detalhes internos que podem ser de interesse para o gerente.

Esta dissertação apresenta uma Facilidade de Monitorização de objetos CORBA, usando um ORB e o seu Serviço de Eventos, para as áreas funcionais de desempenho e de contabilização, no

contexto de uma arquitetura proposta de gerência integrada de sistemas distribuídos. Embora os objetos CORBA apresentem ambas as características de distribuição e encapsulação, este sistema apresenta uma maneira simples e modular de instrumentá-los e monitorá-los. A motivação deste trabalho residiu na necessidade de se investigar o uso de Modelo de Referência ODP e de CORBA como base para uma arquitetura de gerência aberta e distribuída, integrada com os modelos já existentes.

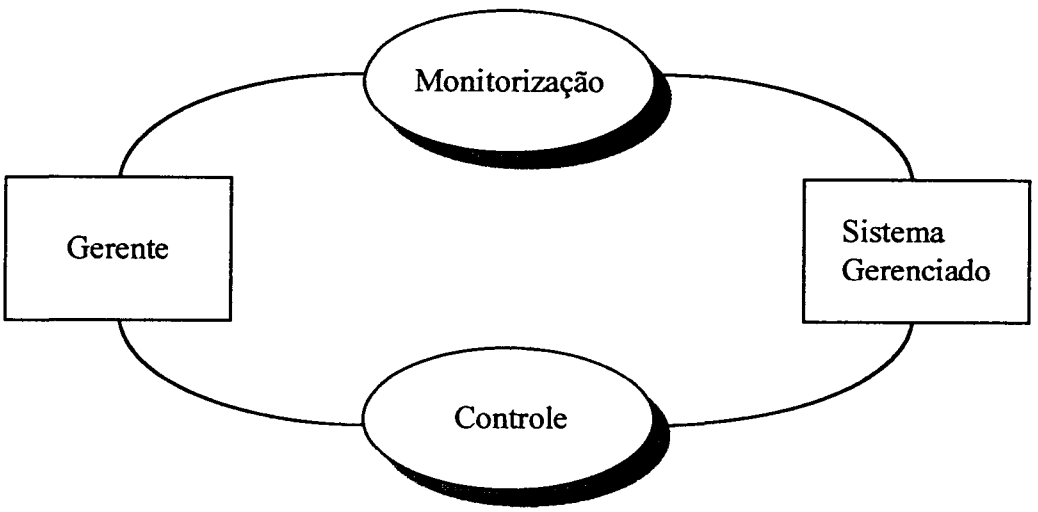

Figura 1.1 - Processo de gerência

O próximo capítulo apresenta os modelos atuais de gerência SNMP, CMIP e TMN. No capítulo 3, o Modelo de Referência ODP (ITU-T/ISO, 1995a) e CORBA (OMG, 1997c) são apresentados como modelos de gerência de sistemas distribuídos, por esta ser uma atividade essencialmente de processamento distribuído aberto. O capítulo 4 descreve a Arquitetura de Gerência de Sistemas Distribuídos proposta, baseada na ODMA (Open Distributed Management Architecture) (ITU-T/ISO, 1995b); na CORBA; nos CORBAservices (Common Object Services Specification) (OMG, 1997b); e nas CORBAfacilities (Common Facilities Architecture) (OMG, 1997a), propostos pela OMG e adotados no projeto da Multiware. No capítulo 5, os principais aspectos da modelagem e de projeto da Facilidade de Monitorização são apresentados. No capítulo 6, os aspectos de sua implementação e resultados são mostrados e discutidos, além da descrição dos trabalhos relacionados. O último capítulo apresenta a conclusão e propõe alguns trabalhos futuros.

Capítulo 2

Modelos de Gerência: SNMP, CMIP e TMN.

Este capítulo tem por objetivo descrever e discutir algumas características e limitações dos principais modelos existentes de gerência Os modelos SNMP, OSI/CMIP e TMN são analisados para se entender melhor as questões envolvidas na escolha de um modelo integrador, mais adequado para a gerência de sistemas distribuídos.

2.1 Modelo de Gerência SNMP

O SNMP (Case, 1990) foi introduzido no início de 1988 como o padrão Internet para a gerência e, também, como a solução para os problemas causados pelo rápido crescimento das redes TCP/IP (Transmission Control Protocol/Internet Protocol). Devido a sua simplicidade, traduzida em baixos custos de implementação, o SNMP se tornou um padrão de fato, presente na maioria das ferramentas comerciais disponíveis. A tarefa de gerência é executada por gerentes e agentes que atuam segundo um modelo cliente/servidor, onde o gerente desempenha o papel de um cliente e o agente atua como um servidor. Padronizado pelo IETF (Internet Engineering Task Force), o SNMP utiliza o termo objeto para representar algum recurso que precisa ser gerenciado. É importante observar que o modelo SNMP não é orientado, tampouco baseado, a objetos porque não suporta os conceitos relacionados ao paradigma.

O modelo de gerência SNMP é composto de três partes. A primeira parte é apresentada em um documento, conhecido como Estrutura de Informação de Gerência (SMI - Structure Management Information) (Rose, 1990), que identifica os tipos de dados que podem ser usados em uma Base de Informação de Gerência (MIB - Management Information Base), e especifica como os objetos em uma MIB são representados e nomeados. Tais objetos são definidos usando um pequeno subconjunto de ASN.1 (Abstract Syntax Notation One). A MIB é organizada em uma hierarquia, sendo que a estrutura da árvore básica é definida neste documento. A segunda parte corresponde à versão atual da MIB (McCloghrie, 1991), na qual os objetos são arranjados em dez grupos considerados essenciais e necessários em todas as implementações. A terceira parte consiste do protocolo de aplicação SNMP (Case, 1990) que é usado para regular a troca de informações de gerência entre gerentes e agentes. Todas as funções de gerência são mapeadas em operações de inspeção ou alteração de valores dos objetos gerenciados. O tipo de comunicação usado pelo SNMP não é orientado à conexão.

De acordo com o SMI, cada objeto é identificado unicamente pelo seu "OBJECT IDENTIFIER". Este identificador de objeto é um nó da árvore hierárquica de nomes, cuja estrutura é administrada de forma conjunta pela ISO e pela ITU-T. A sua estrutura define a forma de arranjo de objetos em grupos logicamente relacionados. Novos identificadores de

objetos para novas MIBs podem ser adicionados como folhas dessa árvore. Todo objeto é definido formalmente em ASN.1 com a declaração do seu tipo de dado e de seu relacionamento com outros objetos dentro da MIB. São usadas ainda macros ASN.1, "OBJECT-TYPE", para a declaração da sintaxe e da semântica dos objetos de uma forma sistemática. O SMI define apenas sete tipos de dados para os objetos: "INTEGER", "OCTET STRING", "OBJECT IDENTIFIER", "Counter", "Gauge", "TimeTicks" e "IpAddress" (Tabela 2.1). Os tipos estruturados ASN.1 "Sequence" e "SequenceOf" podem ser usados para a construção de tabelas bi-dimensionais de objetos escalares. Não existe nenhum tipo para manipular números reais, booleanos ou estruturas mais complexas. Não incorporada no SMI, o ASN.1 ainda prevê o tipo "threshold" para estabelecer limites para uma variável, com o propósito de gerar notificações caso estes valores sejam passados.

Tabela 2.1 - Tipos do Modelo de Informação SNMP

Tipos	**Descrição**
INTEGER	inteiro de 32 bits
Counter	inteiro positivo incremental de 32 bits
Gauge	inteiro positivo incremental e decremental de 32 bits
TimeTicks	inteiro positivo para a medida de intervalos de tempo, módulo 32, em 1:1000 segundo
OCTET STRING	uma seqüência de 0 ou mais octetos
IpAddress	endereço de 32 bits no formato especificado no protocolo IP
OBJECT IDENTIFIER	nome dos objetos gerenciados, sendo uma seqüência de até 128 inteiros positivos, conhecidos como sub-identificadores

Na sua arquitetura, um gerente se comunica com um agente usando o protocolo de aplicação SNMP, baseado normalmente em datagramas UDP (User Datagram Protocol), permitindo que envie PDUs (Protocol Data Units) dos tipos "GetRequest", "GetNextRequest" e "SetRequest", para leitura e alteração de uma lista de um ou mais objetos, e que ainda receba a PDU "GetResponse", com os valores dos objetos ou com um código de erro. A entidade agente processa as PDUs enviadas pelo gerente e pode ocasionalmente enviar para o gerente "Traps", notificações sobre um evento interno detectado (Figura 2.1). O protocolo SNMP não define primitivas de comando imperativas. Elas são implementadas indiretamente através da escrita de

determinadas variáveis do agente, com o uso da operação "SetRequest", que serão interpretadas pelo agente como uma medida a ser tomada.

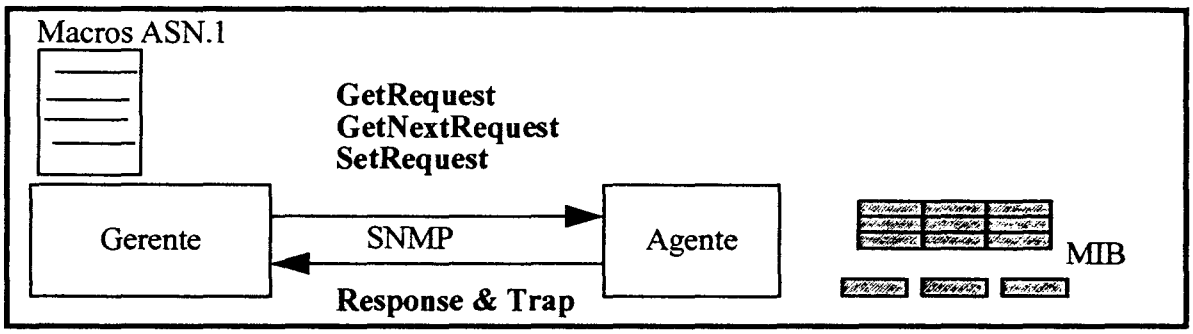

Figura 2.1 - Arquitetura SNMP

Embora o modelo de gerência SNMP tenha sido bem aceito, ele possui restrições que se tornaram evidentes à medida que os sistemas a serem gerenciados foram se tornando mais complexos. A limitação do seu modelo de informação, a falta de requisitos de segurança, a ausência de mecanismos de controle de concorrência, a falta de uma estratégia de gerência distribuída e a inexistência de funcionalidades mais elaboradas são algumas das suas críticas. Duas propostas estão sendo postas em prática para contornar as suas limitações. Uma delas consiste da substituição do protocolo SNMP pela sua segunda versão: SNMPv2 (Stallings, 1996). A outra alternativa busca a extensão do modelo na direção das funcionalidades do modelo OSI/CMIP, que será visto na próxima seção, através da RMON-MIB (Remote Monitoring Management Information Base) (Waldbusser, 1991; Hegering, 1994). Estas duas soluções são descritas nas próximas subseções.

2.1.1 Modelo SNMPv2

As limitações do modelo anterior motivaram as modificações introduzidas na nova versão que incluem o suporte à interação de um gerente com um outro, a extensão do modelo de informação, a definição de novas operações no seu protocolo e a incorporação de requisitos de segurança. SNMPv2 oferece tanto uma estratégia centralizada de gerência quanto distribuída. Nesta última opção, os sistemas podem desempenhar ambos os papéis de agente e de gerente.

Como agente, tal sistema poderá aceitar comandos de uma entidade superior em uma cadeia, na qual um gerente intermediário será capaz de prover informações resumidas dos agentes subordinados, armazenar localmente tais dados e emitir "Traps" para um gerente superior.

No seu modelo de informação, foram incorporados mais dois tipos de dados: "Unsigned32", para representação de inteiros de 32 bits sem sinal, e "Counter64", uma versão de 64 bits do tipo "Counter". Foram definidas ainda três diferentes extensões da MIB: SNMPv2 MIB, "Manager-Manager" MIB e "Party" MIB. A primeira define estatísticas de instrumentação, a descrição da configuração dos recursos, a geração de "Traps" pela própria MIB através de uma macro "NOTIFICATION TYPE" definida em ASN.1 e de uma nova variável "snmpSetSerialNo" para permitir o controle de concorrência de múltiplos "SetRequest". A segunda oferece suporte à distribuição da gerência e a terceira é definida para os novos requisitos de segurança.

As modificações no protocolo acrescentaram duas novas PDUs: "GetBulkRequest" para minimizar o número de troca de mensagens na leitura de variáveis, quando se deseja ter acesso simultâneo a diversos nós de uma MIB; e "InformRequest" para o envio de "Traps" entre gerentes. Foram ainda especificados o mapeamento do SNMPv2 em diferentes protocolos de transporte, embora o UDP ainda seja o de preferência do IETF. No aspecto de segurança, foram incorporados serviços de privacidade, autenticação e controle de acesso com o uso do protocolo simétrico de criptografia DES (Data Encryption Standard) e do protocolo de autenticação MD5 (Message Digest) para garantir a integridade das informações, autenticidade do destinatário e controle do tempo máximo de entrega das mensagens. Dessa forma, as primitivas podem ser apenas autenticadas, cifradas, porém não autenticadas, e simultaneamente cifradas e autenticadas. Contudo os esforços de padronização não entusiasmaram os usuários e vendedores de produtos de gerência, principalmente quanto aos requisitos de segurança especificados. No ano passado, o IETF baniu as especificações de segurança na mais recente versão dos documentos relativos ao SNMPv2 na falta de um consenso entre os membros do grupo de trabalho responsável (Stallings, 1996).

2.1.2 RMON-MIB

A primeira versão do RMON define uma MIB para ser executada em um agente, normalmente conhecido como monitor, que não necessita estar em contato ininterrupto com a aplicação de gerência, diminuindo não só a dependência como o tráfego de informações. Freqüentemente, estes monitores são dispositivos "stand-alones" e destinam uma grande parte de seus recursos computacionais para a tarefa de gerência. Vários são os recursos onde estes agentes podem ser instalados:

- interconectores de redes tais como pontes, roteadores e comutadores;
- máquinas hóspedes dedicadas ou não; e
- plataformas especialmente desenvolvidas para gerência.

A gerência de redes através destes monitores é feita através de operações "off-line" quando se deseja limitar a rotina de consulta entre a estação gerente e o monitor, da monitorização preemptiva se o monitor tiver recursos computacionais suficientes e não degradar o desempenho da rede para ela ser feita continuamente, da detecção e registro de problemas, da valorização dos dados coletados e de múltiplos gerentes de forma concorrente (Waldbusser, 1991).

A mais nova versão da RMON-MIB foi definida como um complemento da anterior, permitindo efetuar análises acima da camada de rede, através da leitura do cabeçalho do protocolo IP (Stalling, 1996), visualizando os protocolos acima da camada de rede. Esta nova fonte de informações permite ao gerente de rede monitorar o tráfego com um grande nível de detalhes, possibilitando a gerência do tráfego de protocolos de aplicação. Essa foi a maneira de se mudar o foco da gerência dos componentes de redes para os sistemas conectados a estes, definindo o conceito de gerência de sistemas.

2.2 Modelo de Gerência OSI/CMIP

O modelo de gerência OSI/CMIP foi desenvolvido para gerenciar sistemas abertos que seguem o modelo de referência OSI da ISO/ITU-T e especificado em um conjunto de documentos (BRISA, 1993). Os esforços de padronização se concentraram na definição:

- do Modelo de Organização, o qual estabelece a hierarquia entre sistemas de gerência, dividindo o ambiente gerenciado em domínios funcionais e administrativos;
- do Modelo de Informação com a estrutura da informação gerenciada, obedecendo ao paradigma da orientação a objeto;
- do Modelo Funcional que associa as diversas atividades de gerência às várias áreas funcionais de configuração, de falhas, de desempenho, de contabilização e de segurança; e
- do Modelo de Comunicação com o protocolo e suas primitivas.

O ambiente de gerência OSI/CMIP inclui também os conceitos de gerente, agente e objetos gerenciados (MO - Managed Objects). Um gerente pode obter informações atualizadas sobre os objetos gerenciados e controlá-los, enviando operações de gerência aos agentes. Um agente executa as operações sobre os objetos e ainda pode enviar ao gerente notificações emitidas por eles. Os objetos gerenciados são definidos em termos de seus atributos, de operações a que podem ser submetidos, de notificações que podem emitir para informar ocorrência de eventos internos, e de suas relações com outros MOs. Dentro deste contexto, uma MIB é definida como um conjunto de objetos gerenciados de um sistema aberto, no qual um MO representa a visão abstrata de um recurso real.

Para garantir a interoperabilidade entre diferentes sistemas de gerência de redes, é necessário ter uma visão comum da informação de gerência, implicando na definição de uma Estrutura de Informação de Gerência (SMI - Structure Management Information) que especifica o modelo de informação a ser adotado. Basicamente, a SMI faz uso do paradigma da orientação a objetos para a definição dos recursos gerenciados através de classes e introduz os conceitos de hierarquias de herança, de nomeação e de registros, usadas na identificação dos MOs. A hierarquia de herança,

também conhecida como hierarquia de classes, está relacionada com a especialização das propriedades das classes, através da extensão dos atributos, comportamentos, operações e notificações. A hierarquia de nomeação descreve as relações entre instâncias de objetos com seus respectivos nomes. A hierarquia de registro é usada para identificar de maneira universal os objetos, independentemente das outras duas hierarquias.

A definição de uma classe é feita de acordo com um "template", conhecido como GDMO (Guidelines for the Definition of Managed Objects), uma extensão das macros do ASN.1, usado para estruturar as definições das informações de gerência dos MOs e registrá-las na hierarquia de nomeação. Cada classe de objeto é caracterizada por um conjunto de pacotes (packages) que podem ser mandatórios ou condicionais. O pacote obrigatório é sempre definido quando uma instância de objeto é criada, enquanto que o condicional só estará presente se uma dada condição for satisfeita (Figura 2.2). Cada pacote, por sua vez, é constituído por um conjunto de atributos visíveis, pelas operações definidas sobre os atributos, pelas ações, pelas notificações que eles podem emitir e pelo seu comportamento.

```
<class label>        MANAGED OBJECT CLASS
[DERIVED FROM        <class label>
[ALLOMORPHIC SET     <class label>
[CHARACTERIZED BY    <package label>
[CONDITIONAL PACKAGES    <package label> PRESENT IF <condition-definitions>
[PARAMETERS          <parameters label>
REGISTERED AS        <object-identifier>
```

Figura 2.2 - Template em GDMO

Os objetos gerenciados que compõem uma MIB são arranjados em uma árvore (MIT - Management Information Tree), contendo instâncias dos MOs e estabelecem graus de hierarquia de nomeação. A identificação de um MO é através de um nome característico relativo (RDN - Relative Distinguished Name). O nome completo de uma instância (DN - Distinguished Name)

consiste da concatenação dos vários RDNs, começando pela raiz da árvore até a própria instância. O atributo DN é usado pelo protocolo de comunicação para identificar um nó da árvore e ter acesso à informação de gerência.

O Modelo Funcional estabelece os requisitos que devem ser satisfeitos pelas atividades de gerência, agrupando-as em cinco diferentes áreas funcionais: de configuração, de falhas, de desempenho, de segurança e de contabilização. Dentro de cada área funcional, foram desenvolvidos padrões de funções, incluindo requisitos, modelos e serviços que constituem processos de aplicação de gerência que utilizam os serviços oferecidos pela camada OSI de aplicação. A gerência de configuração tem por função a manutenção e monitorização da estrutura física e lógica da rede. A gerência de falhas é responsável pela manutenção e monitorização do estado de cada um dos objetos gerenciados e pelas ações necessárias para o restabelecimento das unidades com problemas. A gerência de desempenho preocupa-se com o desempenho da rede, mantendo registros e históricos dos estados para posterior análise. Finalmente, a gerência de contabilização se ocupa com a monitorização de quais recursos e de quanto desses recursos está sendo utilizado. Dessa forma, a ISO e a ITU-T definem as ferramentas para controlar as informações de gerência e os parâmetros dos objetos gerenciados e buscam resolver problemas de configuração da rede, falhas de componentes, níveis de desempenho alcançados, segurança e contabilização. Para atender aos requisitos dessas áreas funcionais foram definidas funções de gerência em quinze documentos da série ISO 10164-x (BRISA, 1993), apresentadas na tabela seguinte.

Tabela 2.2 - Funções de Gerência

Documentos	**Título**	**Área**
1	Função de Gerência de Objeto	Configuração
2	Função de Gerência de Estado	Configuração
3	Atributos para Representação de Relacionamentos	Configuração
4	Função de Relatório de Alarmes	Falhas
5	Função de Gerência de Relatório de Eventos	Falhas
6	Função de Controle de Log	Falhas
7	Função de Relatório de Alarme de Segurança	Segurança
8	Função de Registro para Auditoria de Segurança	Segurança
9	Função de Controle de Acesso	Segurança

10	Função de Medida de Contabilização	Contabilização
11	Função de Monitorização de Carga de Trabalho	Desempenho
12	Função de Gerência de Teste	Desempenho
13	Função de Sumarização	Desempenho
14	Categorias de Testes, Diagnósticos e Confiança	Falhas
15	Função de Escalonamento	Desempenho

A definição do padrão de comunicação entre agentes e gerentes é feita de forma análoga à dos protocolos do modelo OSI. Usam os serviço do CMISE (Common Management Information Service Element) para a troca de informação de gerência e o protocolo CMIP. O tipo de comunicação é orientado à conexão, sendo utilizadas as facilidades de comunicação da pilha de protocolo OSI. O elemento do CMIS utiliza outros elementos da camada de aplicação: o ACSE (Association Control Service Element) para gerenciar as associações gerente/agente e o ROSE (Remote Operation Service Element) para ações remotas. É importante observar que as operações realizadas sobre os objetos são definidas de maneira abstrata no SMI, de forma análoga às primitivas de serviços trocadas entre as camadas do modelo OSI. São definidos dois tipos de operações na interface do objeto, o primeiro se refere aos atributos dos objetos, e o segundo aos objetos como um todo. As operações do primeiro tipo são:

1. "Get": retorna o valor de um atributo, ou de uma lista de atributos, conforme solicitado. Portanto, esta operação pode atuar em uma sub-árvore inteira da MIT, diferentemente do SNMPv1 "GetRequest". Pode ainda estabelecer critérios ou filtros para selecionar entidades de atuação;
2. "Replace": substitui o valor de atributos especificados por valores fornecidos;
3. "Set": substitui o valor de atributos por valores especificados quando da definição dos objetos;
4. "Add": é utilizada no caso de atributos cujos valores são definidos como conjunto, permitindo acrescentar novos elementos a este; e
5. "Remove": é capaz de retirar elementos de atributos cujos valores são definidos como conjuntos.

Conseqüentemente, as operações que se aplicam a objetos como um todo são as que estão descritas a seguir. As suas semânticas são parte da definição das classes dos objetos gerenciados.

1. "Create": cria e inicia os valores dos atributos de um objeto.
2. "Delete": elimina um objeto em particular, removendo os seus atributos. Os efeitos desta operação sobre os elementos físicos correspondentes são localmente especificados; e
3. "Action": determina ao objeto gerenciado que realize uma ação específica e informe os seus resultados.

Tais operações de gerência são mapeadas na interface do objeto com o serviço de comunicação CMISE, para especificar o serviço e procedimento para a transferência das CMIP PDUs. O mapeamento pode ser observado na Tabela 2.3. A Figura 2.3 apresenta tal resultado integrado na arquitetura OSI/CMIP.

Tabela 2.3 - Mapeamento das Operações SMI para o serviço CMISE

SMI	CMISE
Create	M-CREATE
Delete	M-DELETE
Action	M-ACTION
Replace	M-SET
Add	M-SET
Remove	M-SET
Set	M-SET
Get	M-GET
Notification	M-EVENT-REPORT

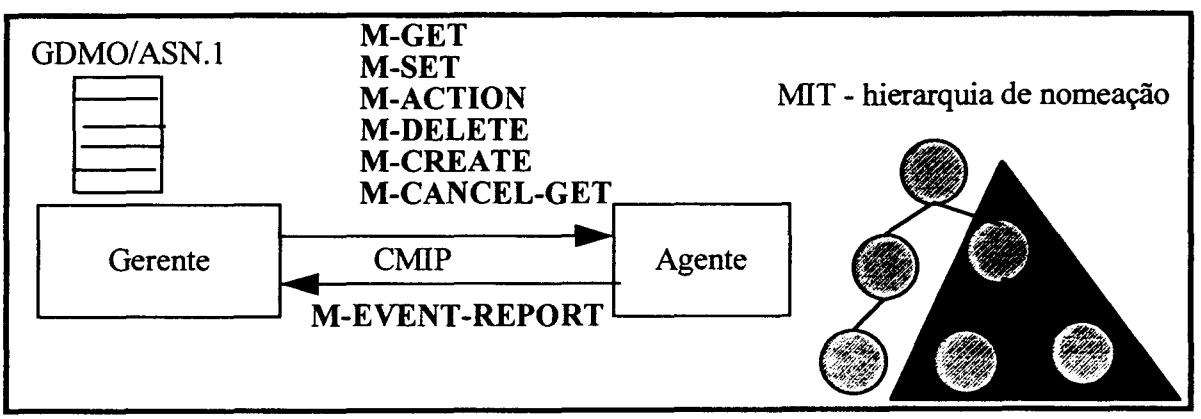

Figura 2.3 - Operações da Arquitetura OSI/CMIP

O modelo de gerência OSI/CMIP é mais poderoso do que o modelo SNMP, porém com um custo de implementação maior. As vantagens do primeiro sobre o segundo são presentes em praticamente todas as referências bibliográficas, destacando-se as seguintes (Gering, 1994):

- o conceito adotado de objeto gerenciado provê mecanismos para a encapsulação de maneira formal. O conceito de herança pode ser implementado usando GDMO para a definição de classes especializadas de suas super-classes, herdando as suas propriedades (atributos, ações, notificações e comportamentos) e adicionando outras. A forma de polimorfismo, conhecida como alomorfismo, permite que uma instância de objeto possa ser gerenciada como se fosse uma instância de uma outra classe;
- define um esquema de nomeação onde os nomes são únicos em todo o sistema (DN), independentes dos nomes das classes e compatíveis com o serviço de diretório OSI X.500;
- usa todo o conjunto de tipos definidos em ASN.1, diferentemente do SNMP que usa apenas uma parte;
- as operações definidas no protocolo CMIP permitem a criação e destruição dinâmica de instâncias de objetos;
- oferece um conjunto de funções de gerência para as áreas funcionais de desempenho, configuração, segurança, de falhas e contabilização; e
- o tipo de comunicação usado permite a troca de contexto e não limita o tamanho da informação de gerência.

Por outro lado, existem muitas críticas para o modelo por si só. As principais são (Sloman, 1995):

- os gerentes não podem enviar uma invocação diretamente para um objeto gerenciado, tendo que usar o protocolo CMIP para interagir com um agente. Os aspectos da orientação a objeto no modelo foram definidos após a especificação do protocolo CMIP, resultando em um modelo não tão elegante quanto a um genuinamente orientado a objetos;

- CMIP é implementado sobre uma pilha completa de protocolos OSI de comunicação, exigindo um ambiente operacional, muitas vezes, não disponível em situações de falhas;
- ASN.1 é uma notação muito poderosa, permitindo a definição de muitas construções que provavelmente nunca foram usadas em um documento GDMO;
- o esquema de nomeação é baseado em uma hierarquia de relacionamentos físicos, incompatível para gerenciar objetos que sejam capazes de migrar de um sistema para outro, como os agentes móveis; e
- a funcionalidade normal dos recursos e serviços são especificados e implementados diferentemente e separadamente da interface de gerência. Embora existam objetos gerenciados que representem separadamente dispositivos de "hardware", isto não é apropriado para objetos de "software".

Contudo, o seu modelo de informação é suficientemente genérico para representar uma grande variedade de recursos. Tal flexibilidade despertou o interesse na sua incorporação no modelo TMN para gerência de telecomunicações, descrito na próxima seção.

2.3 Modelo de Gerência TMN

O modelo de gerência TMN (Telecommunications Management Network) foi definido pela ITU-T em um conjunto de recomendações conhecido como série M.3000. A TMN é uma arquitetura que serve como um modelo genérico de rede de gerência de telecomunicações, possibilitando a monitorização e o controle de diversos equipamentos das redes de telecomunicações (Cohen, 1994). O seu conceito básico é fornecer uma infra-estrutura para interconectar diversos tipos de sistemas de suporte a operações (OS - Operations Systems) e equipamentos de telecomunicações para a troca de informações de gerência através de interfaces padronizadas, incluindo a definição de protocolos e mensagens. Neste contexto, a TMN foi planejada para gerenciar sistemas bastantes heterogêneos, que incluem: redes públicas e privadas, incluindo LANs, MANs, redes de telefonia móvel, redes virtuais e redes inteligentes; a própria

TMN; terminais de transmissão; sistemas de transmissão digital e analógica; "mainframes", processadores "front-end" e servidores de arquivos; PABX; "software" de telecomunicações etc (Brisa, 1994). A TMN considera as redes e os serviços de telecomunicações como um conjunto de sistemas cooperativos e os gerencia de forma integrada. A TMN é conceitualmente uma rede sobreposta que realiza a interface com uma rede de telecomunicações em vários pontos, com a finalidade de enviar e receber informações para controle de sua operação. A arquitetura geral da TMN está estruturada em três arquiteturas básicas que podem ser consideradas, separadamente, no planejamento e projeto de uma TMN. As arquiteturas básicas são: arquitetura funcional, arquitetura de informação e arquitetura física.

A arquitetura funcional descreve as funções de gerência agrupadas em blocos funcionais através dos quais uma TMN pode ser implementada. A definição destes blocos e dos pontos de referência entre os blocos leva à especificação de interfaces padrões TMN. Os blocos funcionais se comunicam através da Função de Comunicação de Dados (DCF - Data Communications Functions) que implementa as camadas 1 a 3 do modelo OSI. Os blocos funcionais da TMN são:

- Bloco funcional Sistema de Suporte a Supervisão (OSF - Operations System Function): processa informações relacionadas à gerência de telecomunicações com o objetivo de monitorar, coordenar e controlar as funções de telecomunicações;
- Bloco funcional Elemento de Rede (NEF - Network Element Function): comunica-se com a TMN para ser monitorado e controlado, provendo as funções de telecomunicações e de suporte que são requeridas pela rede de telecomunicações gerenciada;
- Bloco funcional Estação de Trabalho (WSF - Work Station Function): fornece meios para interpretar informações da TMN para os usuários, incluindo a interface homem/máquina;
- Bloco funcional Adaptador Q (QAF - Q Adapter Function): tem como função a conexão de entidades não TMN à rede TMN; e
- Bloco funcional Mediação (MF - Mediation Function): responsável pela compatibilização da informação trocada pelos blocos funcionais OSF e NEF ou OSF e QAF.

Cada um dos blocos funcionais é formado pelos componentes funcionais descritos a seguir:

- Função de Aplicação de Gerência (MAF - Management Application Function): tem a função de implementar o serviço de gerência, podendo ser mapeadas em gerente ou agente;
- Base de Informação de Gerência (MIB - Management Information Base): contém o conjunto de objetos gerenciados pelo sistema. A estrutura e a implementação da MIB não estão sujeitas à padronização TMN;
- Função de Conversão de Informação (ICF - Information Conversion Function): é a função responsável pela tradução entre os modelos de informação das diversas interfaces previstas na arquitetura TMN;
- Função de Apresentação (PF - Presentation Function): é responsável pela tradução de informações TMN para interfaces homem/máquina, e vice-versa;
- Função de Adaptação Homem/máquina (HMA - Human Machine Adaptation): responsável pela conversão do modelo de informação das MAFs para o modelo de informação apresentado pela TMN à PF, e vice-versa; e
- Função de Comunicação de Mensagem (MCF - Message Communication Function): esta função está associada aos blocos funcionais que possuem interface física, permitindo a conexão com a DCF.

As informações são trocadas entre os blocos funcionais através de pontos de referências. Dentro da arquitetura TMN, são definidas três classes de pontos de referência. Os pontos da classe "q" se situam entre os blocos OSF, QAF, MF e NEF. Os da classe "f" são para ligação de estações de trabalho (WSF). Os da classe "x" se localizam entre os blocos OSF de TMN diferentes ou entre um bloco OSF de uma TMN e um bloco funcional com funcionalidades equivalentes de outra rede. São padronizadas mais duas classes para troca de informações fora do escopo TMN: classe "g" entre estações de trabalho e usuários e classe "m" entre uma QAF e entidades gerenciadas não TMN.

A arquitetura de informação TMN incorpora o modelo de informação de gerência OSI/CMIP. Além disso, TMN acrescenta alguns conceitos de modo a permitir que o modelo atenda a outros requisitos. Um objeto gerenciado é uma abstração do recurso e representa suas propriedades, podendo também representar um relacionamento entre recursos ou uma combinação de recursos. É definido pelos seus atributos visíveis, pelas operações de gerência, pelo seu comportamento e pelas notificações emitidas. São definidos também os papéis de gerente e agente com uma relação de "muitos para muitos". O conceito de domínio é utilizado para agrupar objetos gerenciados em conjuntos, conhecidos como Domínios Gerenciais, particionando o ambiente de gerência em áreas funcionais, em estruturas organizacionais, ou de acordo com um critério geográfico ou de política de segurança. Cada domínio funcional é mapeado em um domínio gerencial sob o controle de uma função de sistema de suporte à operação (OSF). Com isto, é possível agrupar as OSFs segundo requisitos gerenciais, e também de acordo com níveis gerenciais.

A arquitetura física provê meios para a implementação dos blocos funcionais definidos pela arquitetura funcional. Os blocos que fazem parte da arquitetura física possuem blocos funcionais correspondentes na arquitetura funcional. Os blocos da arquitetura física são: Sistemas de Suporte à Operação (OS), Rede de Comunicação de Dados (DCN - Data Communications Network), Dispositivo de Mediação (MD - Mediation Device), Elementos de Rede (NE - Network Element), Adaptador Q (QA - Q Adapter) e Estações de Trabalho (WS - Work Station).

De forma a facilitar o entendimento da funcionalidade dos sistemas de gerência, a ITU-T definiu a Arquitetura Lógica em Camadas (LLA - Logical Layered Architecture) que consiste em estabelecer a arquitetura de gerência como uma série de camadas, utilizando uma abordagem recursiva para decompor uma atividade de gerência em uma série de domínios funcionais aninhados. Alguns dos aspectos mais importantes do processo de gerência são utilizados como critérios para agrupamento de OSFs segundo quatro camadas lógicas (Figura 2.4):

- Camada de Gerência de Elementos de Rede: nesta camada estão situadas as funções referentes à gerência de elementos de redes;

- Camada de Gerência de Rede: uma OSF desta camada tem como objetivo a gerência de uma rede como um todo. Oferece uma visão global da rede gerenciada, independente da tecnologia utilizada na sua implementação;
- Camada de Gerência de Serviços: as OSFs desta camada visam o conhecimento, a monitorização e o controle dos aspectos contratuais dos serviços oferecidos pelos clientes, constituindo-se do ponto de contato destes com os prestadores de serviço; e
- Camada de Gerência de Negócios: um dos objetivos das OSFs desta camada é a melhor utilização dos recursos sob o ponto de vista de negócios para buscar o maior retorno do investimento realizado.

O modelo TMN oferece um rico modelo de informação e a proposta de se ter uma visão integrada dos recursos gerenciados estruturados em domínios. Como TMN é baseado nos conceitos do modelo de gerência OSI, os comentários desfavoráveis da seção anterior também se aplicam a ele. Adicionalmente, não se concebe haver uma rede exclusiva para gerência quando se busca uma solução integrada através da chamada Rede Digital de Serviços Integrados de Faixa Larga (B-ISDN - Broadband Integrated Services Digital Network), baseada em redes ATM (Asynchronous Transfer Mode).

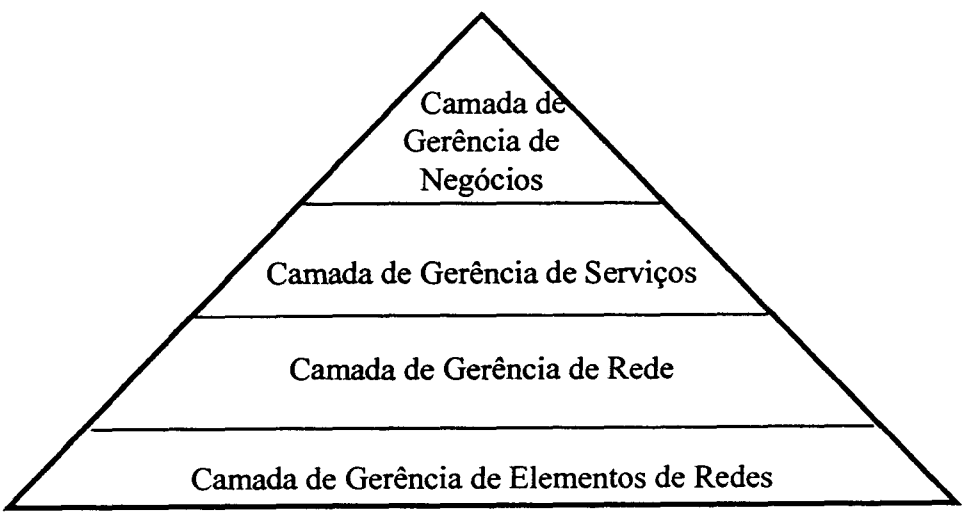

Figura 2.4 - Camadas da Arquitetura TMN

2.4 Conclusão

Os modelos e padrões atuais de gerência foram desenvolvidos para gerenciar componentes de redes, cujos serviços são especificados, projetados e implementados separadamente de sua funcionalidade normal. O modelo SNMP é usado para monitorar e controlar elementos Internet, como "gateways" e roteadores, através de aplicações em estações de gerência. Embora as suas necessidades de processamento e memória sejam baixas, as funções de "polling" e "trap" não são adequadas para a gerência de aplicações distribuídas. O modelo de gerência OSI/CMIP oferece uma abordagem mais sofisticada, usando conceitos de orientação a objeto como classes, polimorfismo e herança. Embora a linguagem GDMO corresponda a uma linguagem de definição de interface, como a CORBA IDL (Interface Definition Language), ela não oferece as facilidades de pré-compilação que gera automaticamente os "stubs" para empacotar e desempacotar parâmetros e de invocação dinâmica. O modelo TMN tem sido freqüentemente apontado como a solução mais adequada para a gerência de redes heterogêneas de comunicação de dados e de telecomunicações (Goulart, 1995), constituindo as redes B-ISDN. O seu modelo de informação é bastante abrangente permitindo representar os mais variados recursos. Além disso, a estrutura de gerência definida permite a definição de domínios, particionando os recursos e permitindo a interação entre eles. Contudo, sofre das mesmas restrições impostas pela adoção do protocolo CMIP na sua arquitetura. Tais padrões não reconhecem que o mesmo modelo usado para especificar, projetar e implementar sistemas distribuídos também pode ser usado para a gerência. Neste contexto, surgem propostas alternativas e soluções de interoperabilidade que serão descritas a seguir.

Capítulo 3

Modelos de Interoperabilidade de Gerência

De uma forma geral, as aplicações e as tarefas de gerência podem ser analisadas e agrupadas em cinco diferentes dimensões ortogonais (Hegering, 1994), ilustradas nos eixos da Figura 3.1. No primeiro eixo, as várias tarefas podem ser divididas em áreas de acordo com as funções que exercem, adotando-se as áreas funcionais de gerência OSI/CMIP: configuração, desempenho, de falhas, segurança e contabilização. Em uma outra dimensão, são consideradas as diferentes fases do ciclo de vida de um recurso gerenciado. Desde a fase de projeto, deve-se especificar requisitos de gerência para posterior implementação, garantindo a sua eficaz operação. No outro eixo, são apresentados os diferentes tipos de mídia, oferecendo serviços que precisam ser gerenciados. A seguir, são mostrados os tipos distintos de recursos gerenciados, desde os elementos de redes às aplicações, considerando os requisitos de gerência de uma organização. Finalmente, são abordadas as diferentes tecnologias usadas em redes locais, metropolitanas, geográficas,

corporativas e inteligentes. Segundo o autor referenciado, uma proposta de gerência integrada deve ser capaz de abranger uma grande porção do espaço multidimensional da figura e, se possível, em um ambiente heterogêneo e aberto. Tais requisitos têm despertado o interesse da adoção dos padrões de direito do Modelo de Referência ODP e de fato CORBA. Como exemplo disto, o RM-ODP tem sido adotado como um meta-padrão para o desenvolvimento de padrões de gerência em redes (ITU-T/ISO, 1995b) e em telecomunicações (TINA, 1995). Recentemente, o crescimento e sucesso do modelo integrador de objetos CORBA como uma solução de interoperabilidade tem motivado o mapeamento das especificações dos padrões OSI/CMIP e SNMP para CORBA IDL, através de adaptadores (Ban, 1996) e "gateways" (Soukoti, 1997), baseado na possibilidade de CORBA se tornar um padrão integrador de gerência .

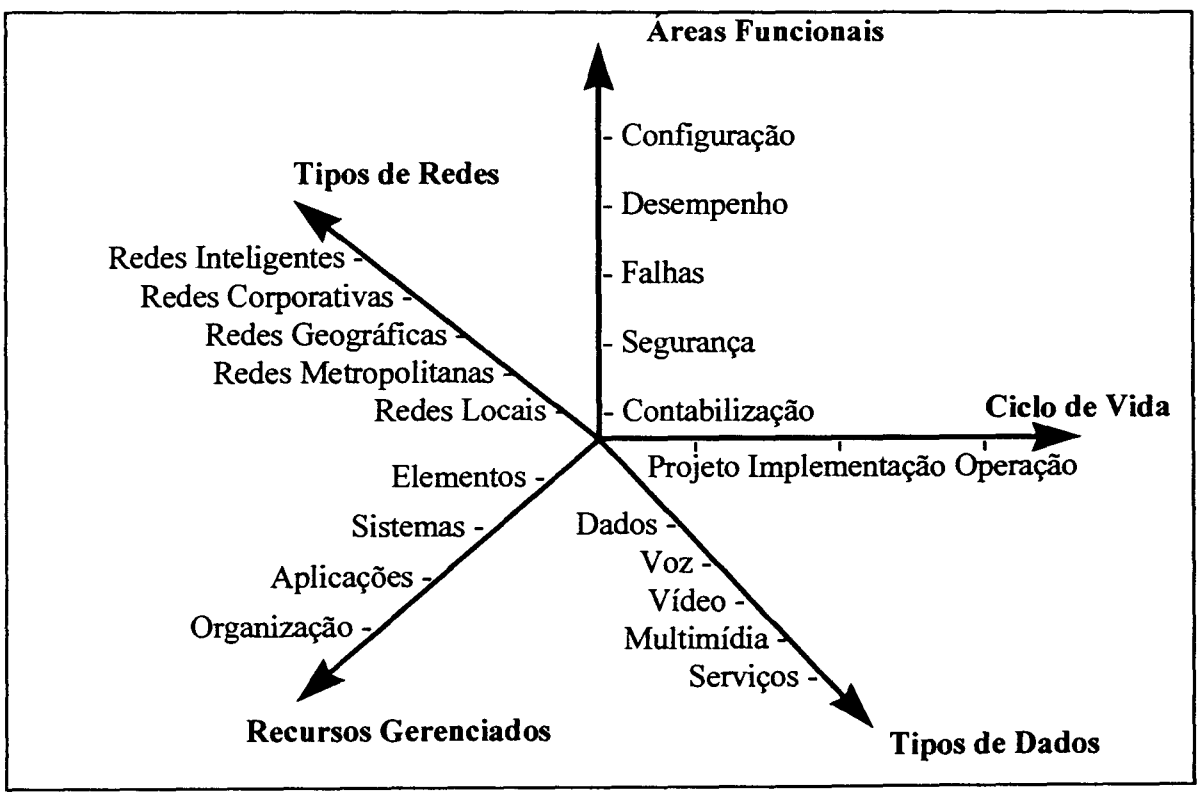

Figura 3.1 - Eixos ortogonais de gerência

Este capítulo tem por objetivo descrever e discutir algumas características dos principais modelos de interoperabilidade, introduzir novas propostas e apresentar os seus benefícios, principalmente

quando se deseja ter um modelo de gerência integrado. Os modelos são analisados para se entender melhor as questões envolvidas na escolha de um mais adequado para a gerência de sistemas distribuídos.

3.1 Modelo de Gerência baseado em RM-ODP

Como a gerência de sistemas distribuídos é essencialmente uma atividade de processamento distribuído, o RM-ODP (Reference Model for Open Distributed Processing) pode e deve ser usado para modelá-lo. O Modelo de Referência é um meta-padrão que suporta os conceitos de distribuição, de interoperabilidade, de transparências e de portabilidade de aplicações distribuídas, usando o paradigma da orientação a objetos.

O Modelo de Referência aborda os aspectos de distribuição de uma forma integrada, usando cinco pontos de vistas: de empresa, de informação, computacional, de engenharia e de tecnologia. Portanto, a complexidade da gerência é analisada sob estas óticas, em que cada uma apresenta uma abstração diferente do processo, em uma abordagem simultaneamente "top-down" e "bottom-up". Esta é uma maneira distinta de modelar o processo de gerência e mover da dimensão da gerência de redes para a de sistemas distribuídos integrados. O ponto de vista de empresa considera o sistema e o seu ambiente, focando nos propósitos, escopos e políticas da organização. O ponto de vista de informação se preocupa com a informação de gerência que necessita ser armazenada, processada e trocada entre os componentes do sistema. Ambas visões estabelecem especificações independentes da distribuição dos seus elementos. O modelo ODP computacional provê uma descrição orientada a objeto do sistema como um conjunto de componentes cooperativos, que potencialmente podem ser distribuídos. Sua linguagem oferece um conjunto de conceitos e regras para estruturar o sistema de gerência. O ponto de vista de engenharia estabelece um ambiente no qual as interações computacionais possam ser estabelecidas, descrevendo mecanismos, funções e transparências necessárias para suportá-las. São identificadas as transparências de localização, de relocação, de migração, de acesso, de falhas, de persistência, de replicação e de transação. O ponto de vista de tecnologia cobre os aspectos necessários para a implementação dos componentes identificados nas especificações dos outros

pontos de vistas. CORBA tem sido considerada como um padrão relacionado com RM-ODP por oferecer uma plataforma distribuída orientada a objetos, um dos componentes identificados no ponto de vista de engenharia (Farooqui, 1997).

A Figura 3.2 ilustra a adequação dos pontos de vistas ODP ao problema de gerência, desacoplando os requisitos das aplicações das peculiaridades da diversidade dos recursos gerenciados, onde as linhas horizontais representam os pontos de vista de empresa, de informação e de tecnologia, enquanto as verticais, os pontos de vista computacional e de engenharia. Este desacoplamento é muito útil quando se deseja ter um modelo de gerência integrado que permita o estabelecimento de especificações de alto nível, independentes das características dos objetos gerenciados.

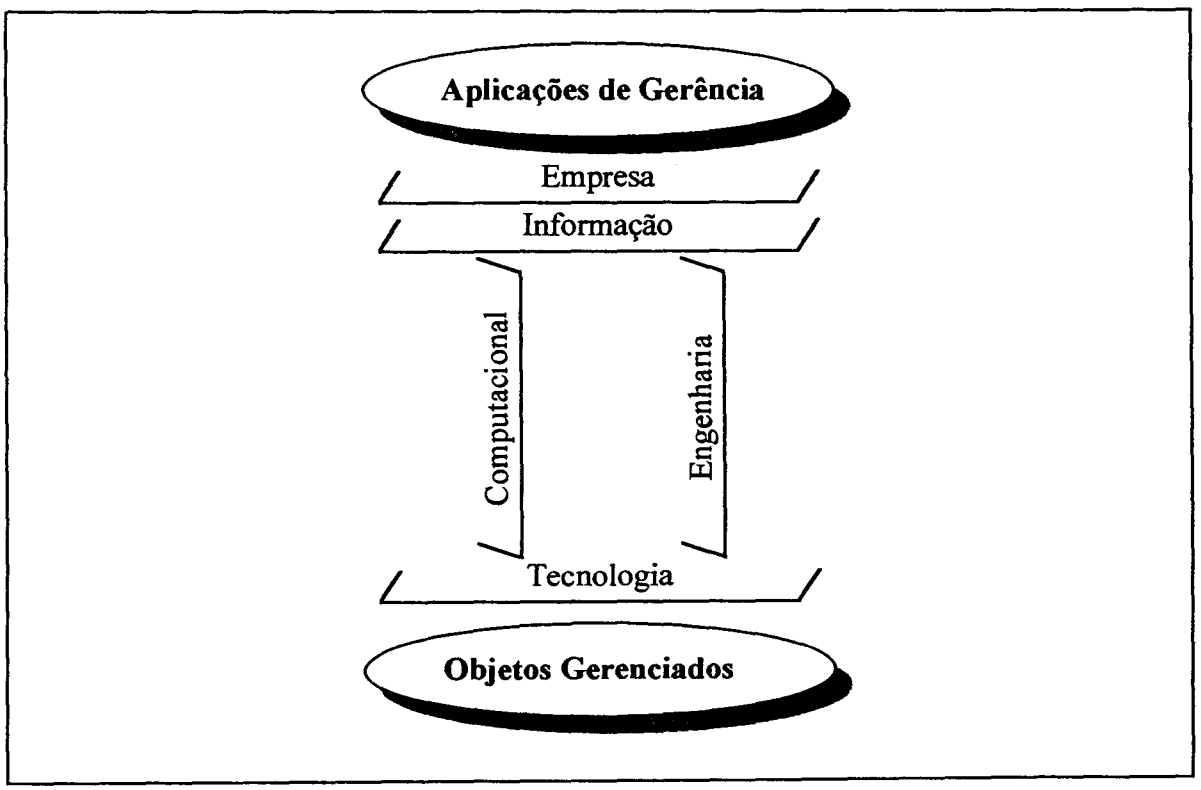

Figura 3.2 - Pontos de Vistas em ODP

Dessa forma, os pontos de vista computacional e de engenharia, representado pelas linhas verticais, fazem o elo e a distinção entre os requisitos de gerência e os seus aspectos de implementação. No primeiro, um sistema de gerência ODP é especificado como um conjunto de objetos computacionais, cujas interfaces podem ser tanto operacionais, de sinais ou do tipo "stream". Os objetos computacionais de gerência podem ter diferentes interfaces, cada uma representando uma visão diferente. Eles podem representar diferentes papéis nas suas interfaces: cliente ou servidor, produtor ou consumidor, gerenciado ou gerente. Um objeto gerenciado é uma entidade cujo comportamento pode ser monitorado e controlado por um objeto gerente. Suas múltiplas interfaces podem oferecer diferentes visões e prover informações para distintas áreas funcionais de gerência: desempenho, contabilização, segurança, falha e configuração. O objeto gerenciado pode receber invocações como um servidor ou ainda emitir notificações como cliente através de suas interfaces gerenciadas. O objeto gerente pode invocar operações como um cliente ou receber notificações através de suas interfaces de gerência. Por outro lado, a linguagem de engenharia contém os conceitos que descrevem a infra-estrutura necessária para suportar as interações entre os objetos das aplicações de gerência de forma transparente. Um objeto computacional pode corresponder a um ou mais objetos básicos de engenharia (BEO - Basic Engineering Object), agrupados em "clusters". Um ou mais "clusters" compõem uma cápsula que pode ser comparada a um processo, com o seu próprio espaço de endereçamento. Uma ou mais cápsulas povoam um nó sob o controle de seu núcleo que é responsável pelas funções de gerência do nó. Um sistema operacional (Kernel) é um exemplo de núcleo. Os serviços de um canal garantem as interações distribuídas transparentes entre BEOs. É constituído por três objetos de engenharia: "stub", "binder" e objeto protocolo. Esta estrutura é usada para prover interações transparentes entre objetos distribuídos nos papéis de cliente/servidor ou gerenciado/gerente. O "stub" é responsável pelo empacotamento de parâmetros em um "buffer" de comunicação. O objeto "binder" estabelece e mantém as interações distribuídas entre diferentes objetos, sendo o maior responsável pela transparência de localização. O objeto protocolo oferece o serviço de comunicação de suporte às interações remotas. Esta infra-estrutura estabelecida precisa também ser gerenciada. Uma entidade gerente deve monitorar e controlar as cápsulas, os objetos dos canais e os núcleos dos nós através de interfaces de gerência em uma visão integrada (Figura 3.3).

Como pôde ser lido, o modelo de objetos ODP pode ser aplicado como um meta-padrão para gerência. Tal possibilidade motivou a sua adoção na Arquitetura de Gerência Distribuída e Aberta (ODMA - Open Distributed Management Architecture) para redes de computadores, descrita a seguir.

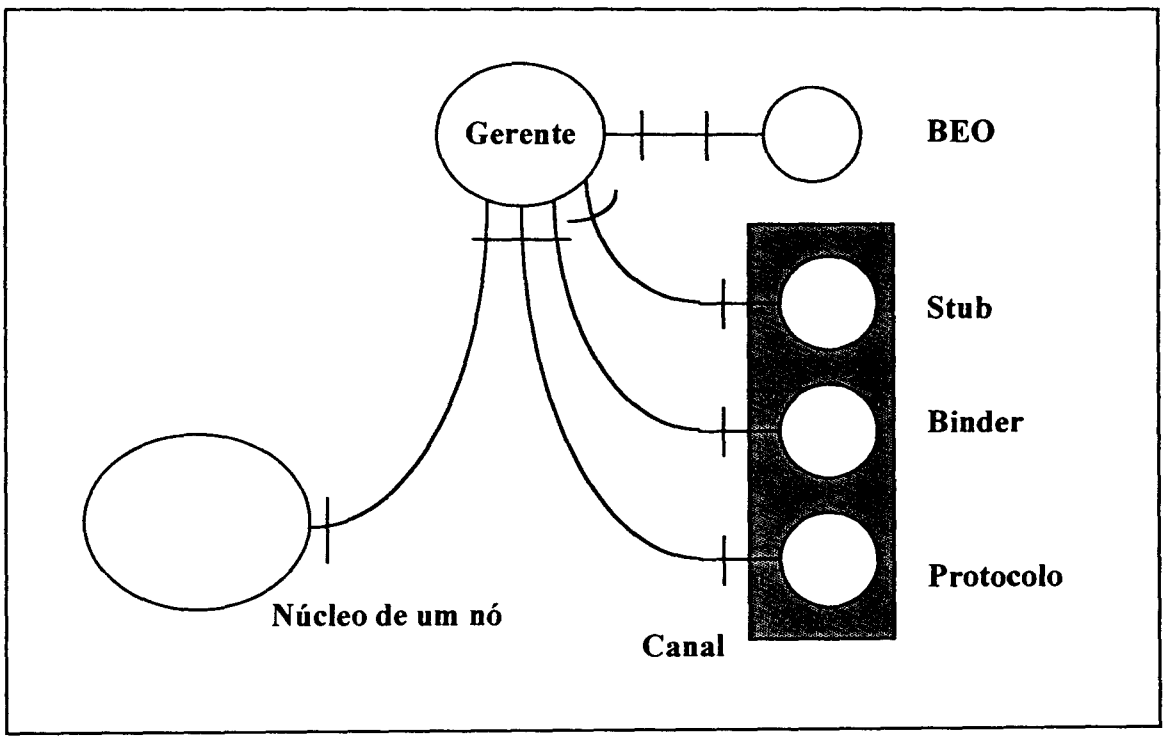

Figura 3.3 - Visão Integrada

3.1.1 Arquitetura ODMA

A necessidade de se integrar a gerência e o fato de se reconhecer que é um processo inerentemente distribuído motivaram o grupo de trabalho ISO:IEC JTC1:SC21 WG4 "Information Retrieval, Transfer and Management for OSI" da ISO a publicar o documento "Open Distributed Management Draft", cujo objetivo é oferecer uma arquitetura para a especificação e desenvolvimento de aplicações de gerência. Os seguintes princípios norteiam a sua padronização:

- o objetivo é prover uma arquitetura para gerência distribuída e aberta, conhecida por ODMA, para redes, sistemas e aplicações;
- esta arquitetura deve ser consistente com outras soluções identificadas para a gerência de sistemas distribuídos abertos, em particular para a gerência de aplicações;
- ODMA deve evoluir do atual modelo centralizado de gerência para um distribuído;
- a proposta não é substituir as atuais arquiteturas de gerência;
- ODMA deve ser compatível e consistente com o RM-ODP;
- os elementos básicos de ODMA são os gerentes, os recursos gerenciados e as relações entre eles; e
- a distinção entre os gerentes e os recursos é o papel desempenhado por eles nas suas relações.

Essa abordagem implica na distribuição das atividades de gerência, na gerência de aplicações distribuídas e na gerência dos recursos que podem ser distribuídos. ODMA é fundamentada no RM-ODP, adotando os pontos de vistas, as funções e as transparências ODP de forma seletiva. Os seus elementos básicos são gerentes, os recursos gerenciados e seus relacionamentos que podem ser representados por objetos ODP. ODMA deve ser consistente com outras soluções identificadas para a gerência de sistemas distribuídos abertos, em particular para a gerência de aplicações, a não ser que a arquitetura resultante se mostre inconsistente com o modelo RM-ODP.

Os itens abaixo descrevem o escopo do documento, limitando-se a:

- estabelecer uma infra-estrutura ODMA através da interpretação do RM-ODP para gerência;
- relacionar os conceitos do padrão de gerência OSI/CMIP com os do modelo ODP; e
- descrever as funções ODP, usando as técnicas da linguagem de definição de interface de CORBA IDL, como suporte.

Estão fora do escopo as descrições das linguagens dos pontos de vistas ODMA e de funções específicas, como as funções de gerência inter-domínio, resolvendo questões de interoperabilidade em diferentes domínios como o OSI/CMIP e CORBA IDL.

No ponto de vista de empresa ODMA, uma especificação deve definir contratos entre os objetos gerente e gerenciado. Segundo o RM-ODP, um contrato é um acordo que orienta parte do comportamento coletivo de um grupo de objetos. Cada objeto pode ter mais de um papel, dependendo apenas de suas interações. O agrupamento de objetos em comunidades é previsto para que se alcance um objetivo expresso através de um contrato. Uma comunidade é definida pelo seu objetivo, pela definição papel de cada objeto e pela política aplicada. Um política é definida como um conjunto de regras, expresso através de permissões, obrigações e proibições. O ponto de vista de informação ODMA referencia diretamente o RM-ODP. Todavia, no ponto computacional, o documento define as interfaces operacionais gerente e gerenciada, através das quais dois objetos podem interagir. Tais interfaces são especificadas por uma assinatura, composta pela indicação de seu papel e por suas operações e notificações. As operações são invocadas por uma interface gerente e recebidas por uma gerenciada. Notificações são emitidas por uma interface gerenciada para uma gerente. O documento ainda prevê que um objeto computacional pode ter múltiplas interfaces para diferentes propósitos. O conceito ODP de composição é usado para agrupar objetos computacionais ODMA. Um objeto composto exibe múltiplas interfaces, porém as interfaces entre os objetos computacionais dentro de um objeto composto não são visíveis externamente. No ponto de vista de engenharia ODMA, são identificados os mecanismos e as funções necessárias para suportar as interações distribuídas entre os objetos, além das transparências envolvidas. Refina o modelo de um canal ODP, composto pelos objetos "stub", "binder" e protocolo, identificando mais dois objetos funcionais: despacho de operações (operation dispatcher) e despacho de notificações (notification dispatcher). Os dois basicamente mantêm uma lista das interfaces gerenciadas no domínio de seu nó para operações e notificações respectivamente (Figura 3.4). O documento ainda identifica um objeto cumpridor de políticas (policy enforcer) para garantir que as regras especificadas para regular as interações entre os objetos ODMA não sejam violadas. Finalmente, o ponto de vista de tecnologia expressa como as especificações para um sistema ODMA são implementadas.

O Modelo de Referência ODP é adotado como um meta-padrão para o desenvolvimento de padrões para gerência de redes na arquitetura ODMA. Por outro lado, CORBA, além de oferecer uma plataforma distribuída aberta para aplicações, tem sido investigada como uma solução do ponto de vista de tecnologia ODP (Farooqui, 1997). Na próxima seção, CORBA será apresentada como um modelo integrador não só para sistemas distribuídos como também para gerência.

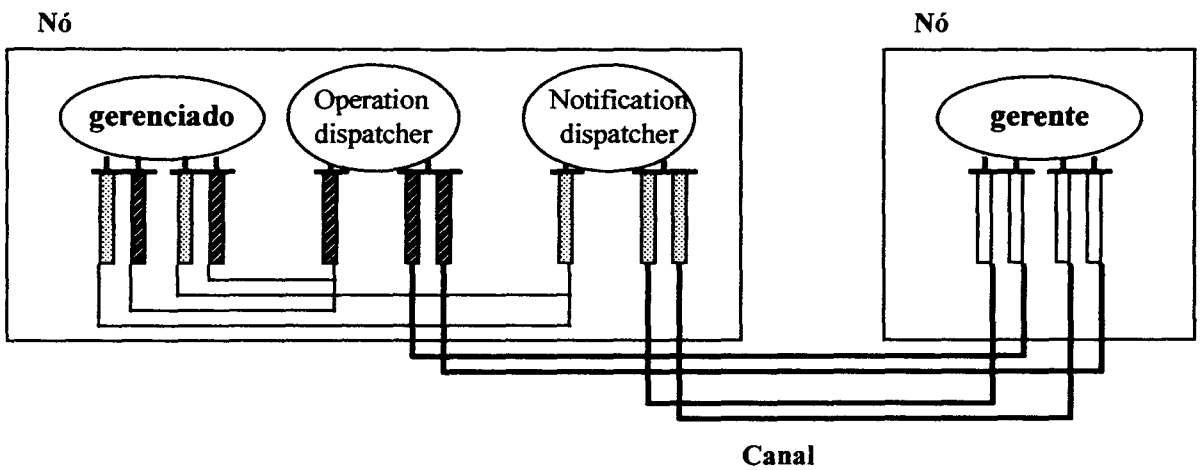

Figura 3.4 - Modelo de Engenharia ODMA

3.2 Modelo de Gerência CORBA

O paradigma da orientação de objetos tem se mostrado uma ferramenta muito útil para o desenvolvimento de sistemas distribuídos, além de sua aplicabilidade ao desenvolvimento de componentes de "software" já existentes e testados, como ocorre em outras engenharias. A idéia de se modelar sistemas distribuídos como uma coleção de objetos interagindo é apropriada para integrar recursos em ambientes heterogêneos e distribuídos. Objetos oferecem um modelo natural para sistemas distribuídos abertos porque os componentes desses sistemas só podem se comunicar usando mensagens através de uma interface bem definida, adequando-se às características de desacoplamento e extensibilidade.

No contexto de sistemas distribuídos abertos, torna-se necessário integrar objetos em processos diferentes e em máquinas distintas, usando-se o mecanismo, natural ao paradigma orientação a objeto, de troca de mensagens através de interfaces bem definidas. Na tentativa de se resolver os problemas de interoperabilidade de diferentes modelos de objetos, tem-se buscado uma solução que procura estender as fronteiras entre objetos para além dos limites tradicionais impostos pelas linguagens de programação, pelo espaço de endereçamento dos processos e, destacadamente, pelos sistemas operacionais das estações de trabalho e protocolos de rede. Portanto, a proposta destas arquiteturas é permitir a comunicação entre objetos em ambientes distribuídos e heterogêneos, através da padronização de um modelo de objetos integrador. A Figura 3.5 mostra uma arquitetura que adota um modelo integrador em que o mapeamento de tipos entre N diferentes modelos requer somente N mapeamentos. O mapeamento sem um modelo integrador exigiria N(N-1) mapeamentos, por causa da tradução de tipos de um modelo para todos os outros existentes. Por outro lado, a adição de um outro modelo exige apenas a definição de um único mapeamento, entre o novo e o integrador.

Esses modelos integradores devem, basicamente, tratar de dois tipos de problemas:

- identificar a compatibilidade entre tipos definidos em modelos diferentes e fazer o mapeamento entre eles; e
- compatibilizar o modelo de execução de objetos implementados em arquiteturas diferentes e que precisam interagir.

A definição de um modelo integrador deve ainda considerar os seguintes aspectos (Cerqueira, 1996):

- o conjunto de modelos a serem integrados, incluindo linguagens de diferentes paradigmas de programação, tanto orientadas a objetos quanto as que não obedeçam ao paradigma;
- a maneira como os modelos são integrados, adotando uma abordagem de união ou de interseção das características das linguagens integrantes;
- o grau de extensibilidade do modelo integrador. Ele deve prover mecanismos que lhe permitam incorporar novas características; e

- o grau de flexibilidade e transparência do ponto de vista dos modelos integradores, tanto para o desenvolvedor de um objeto quanto para o seu usuário cliente.

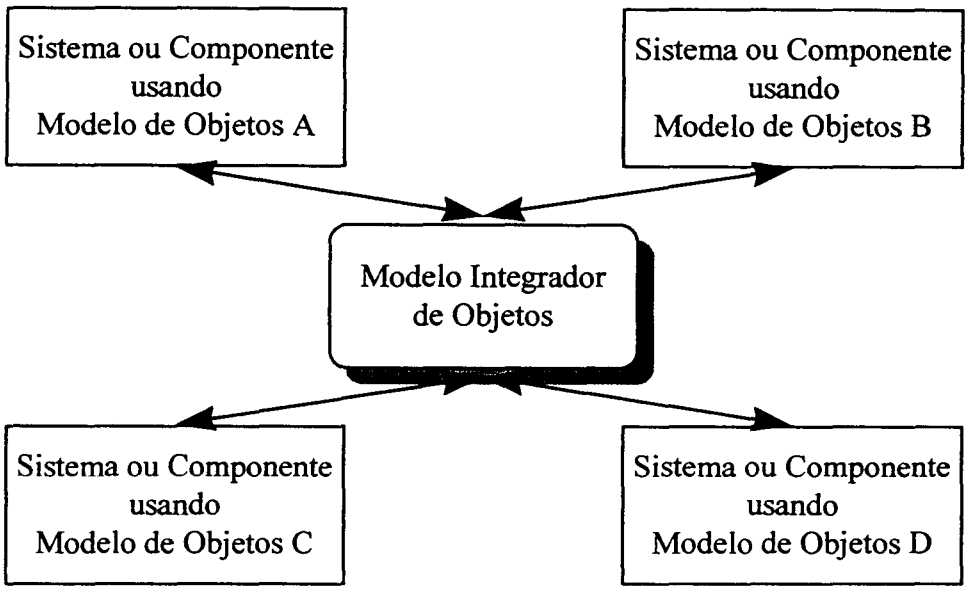

Figura 3.5 - Modelo Integrador

De uma maneira geral, todas as arquiteturas propostas para interoperabilidade procuram definir um modelo integrador através da padronização de seu modelo, de suas APIs e de seu protocolo, facilitando o desenvolvimento independente de componentes heterogêneos e distribuídos. Elas adotam uma linguagem de definição de interface (IDL - Interface Definition Language), estritamente declarativa, para descrever tipos de objetos que passam por um compilador que as traduz para adaptadores, chamados "stubs", em uma linguagem específica. Os "stubs" gerados podem ser assim ligados com os módulos clientes e servidores para compor a aplicação final. É com esta solução que a OMG definiu o padrão aberto CORBA, que já está em sua segunda versão e que será descrito a seguir.

3.2.1 Arquitetura CORBA

Na tentativa de se solucionar o problema de interoperabilidade em sistemas distribuídos abertos, foi desenvolvido o padrão CORBA pela OMG, um organismo internacional que envolve

praticamente todas as grandes empresas de software, tais como Apple, Digital, Hewlett-Packard, IBM, Microsoft e SunSoft. O seu modelo de objetos integrador recebe o nome de "Core Object Model" e define uma semântica básica para especificar as características visíveis de um objeto, de um modo padronizado e independente de implementação. Este modelo abstrato inclui um modelo formal de tipos, operações e subtipagem. A OMG define, a partir do "Core Object Model", um modelo concreto que oferece recursos voltados para o domínio de sistemas distribuídos. As suas principais particularidades são (OMG, 1997c):

- a definição da arquitetura de um "Object Request Broker", que é responsável pelos mecanismos de suporte a chamadas de métodos entre objetos distribuídos de forma transparente. A atual especificação da CORBA 2.0 padroniza o protocolo de interoperabilidade entre diferentes ORBs, denominado como GIOP (General Inter-ORB Protocol), e o seu mapeamento específico para TCP/IP, conhecido como IIOP (Internet Inter-ORB Protocol), ou ainda para um ambiente específico como o DCE (Distributed Computing Environment) através do protocolo ESIOP (Environment-Specific Inter-ORB Protocol); e

- especificação de uma IDL para a definição de tipos exportados por um ORB, além dos mapeamentos entre IDL e linguagens de implementação, tais como C, C++, Smalltalk, Java e Ada95. A CORBA 2.0 padroniza os mapeamentos entre C, C++ e Smalltalk. A sintaxe da OMG IDL é um subconjunto de C++ estendido com algumas definições que suportam a distribuição de objetos.

Essas duas particularidades permitem que CORBA possa ser definida como um barramento padrão de comunicação, onde componentes heterogêneos de "software" interagem entre si com transparências de localização e acesso em um ambiente aberto e não proprietário. Neste ambiente, os componentes distribuídos podem assumir dois papéis: de cliente e/ou de servidor. Para que um componente cliente possa fazer uma chamada a um outro no barramento, é necessário apenas que conheça a interface IDL do objeto CORBA servidor. O cliente enxerga o servidor sempre como local, mesmo que este seja remoto. Para isso, é necessário que os objetos servidores tenham interfaces especificadas em IDL e sejam implementados até mesmo em diferentes linguagens de programação, sistemas operacionais e arquiteturas de máquinas dos seus

prováveis clientes. Um objeto cliente realmente não precisa conhecer detalhes de implementação dos servidores.

A estrutura da ORB é responsável por receber requisições dos clientes através dos "stubs" ou de sua interface de invocação dinâmica; localizar e ativar, se necessário, a apropriada implementação de objetos CORBA; passar a chamada de método através dos esqueletos; e retornar as respostas, se necessário. A CORBA oferece uma plataforma de desenvolvimento e de execução de aplicações distribuídas com transparências de localização e acesso.

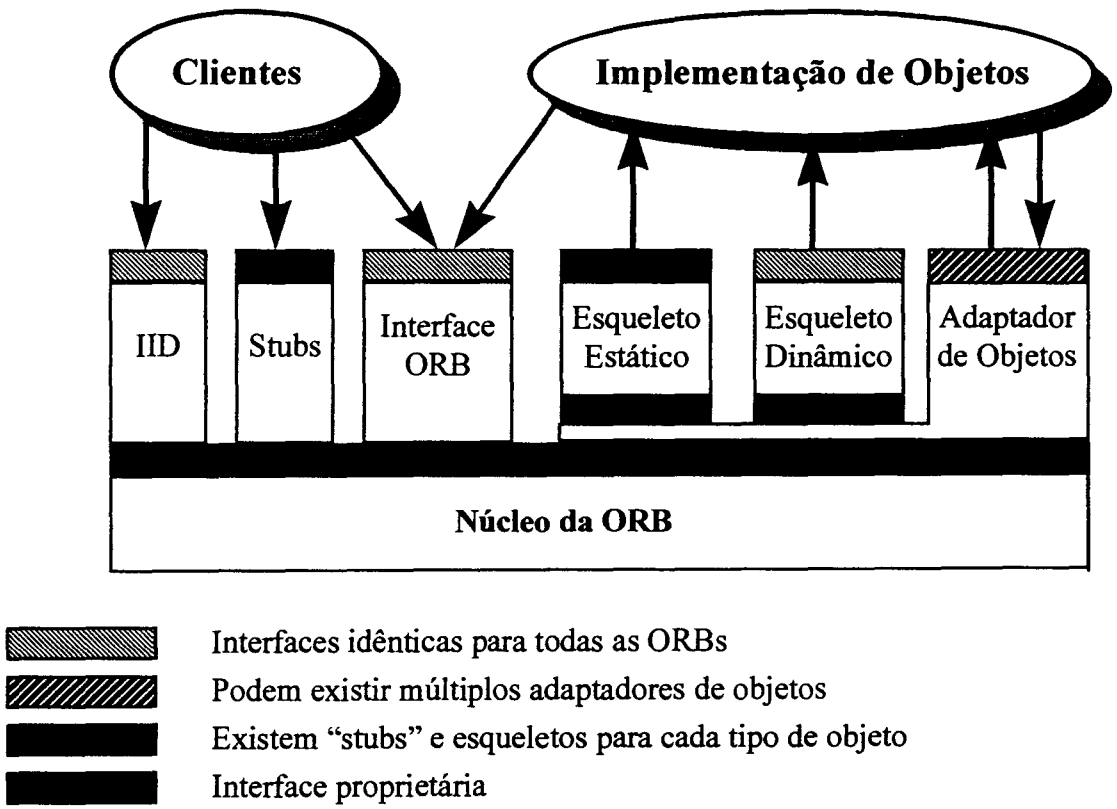

Figura 3.6 - Elementos da CORBA 2.0

A Figura 3.6 ilustra os elementos da CORBA, tanto do lado do cliente quanto do servidor. De uma forma mais didática, serão descritos a seguir os elementos do cliente separadamente do servidor. No lado do cliente, temos:

- os "Stubs" - responsáveis pela invocação de um cliente a um objeto remoto de forma transparente, usando a sintaxe natural de sua linguagem de implementação, escondendo as primitivas de chamadas de sistemas. Dessa forma, um programa cliente tem acesso a um método de um objeto remoto da mesma maneira que um outro residindo no seu próprio espaço de endereçamento. As rotinas dos "stubs" são geradas automaticamente no momento da compilação do arquivo com a descrição em IDL da interface do servidor;

- a Interface de Invocação Dinâmica (IID) - responsável por permitir que uma invocação a um servidor seja montada, sem o auxílio dos "stubs", por um programa cliente em tempo de execução. Para isto, é necessário especificar na invocação dinâmica o objeto CORBA, o método a ser executado e o conjunto de parâmetros. Tal recurso é necessário quando não se tem acesso aos arquivos com os "stubs" criados em tempo de compilação. Essa interface é de especial interesse para certos tipos de aplicações que precisam ter acesso a objetos remotos em tempo de execução, porém necessita do auxílio do Repositório de Interfaces, descrito a seguir;

- o Repositório de Interfaces - responsável pelo acesso à descrição das interfaces dos Objetos CORBA registrados e suas respectivas assinaturas com métodos e parâmetros. As suas APIs permitem o acesso, o registro e atualização destas informações, auxiliando a construção de invocações dinâmicas através da IID. Tais informações contidas no repositório podem ser consideradas como meta-dados dos objetos CORBA; e

- a Interface ORB - permite o acesso de clientes às funcionalidades não oferecidas pelas demais interfaces.

Do lado do objeto CORBA servidor, temos:

- os Esqueletos Estáticos - responsáveis por receber uma invocação remota a um objeto CORBA e, se necessário, enviar uma resposta de forma transparente, escondendo as primitivas típicas de entrada/saída. São similares aos "stubs" do cliente, sendo gerados automaticamente no momento da compilação do arquivo com a descrição IDL de cada método do objeto CORBA;

- o Esqueleto Dinâmico - responsável por oferecer um mecanismo de acesso a objetos em servidores que não possuem os correspondentes esqueletos estáticos. Tal recurso inspeciona os valores dos parâmetros da mensagem recebida para determinar o objeto destinatário e o correspondente método. Pode ser ainda usado para a implementação de pontes genéricas entre diferentes ORBs;

- o Adaptador de Objetos - responsável pelos serviços básicos de geração e interpretação de referência de objetos, e pela ativação e desativação de objetos CORBA e suas correspondentes implementações em servidores, de acordo com os modos previstos. Oferece ainda interfaces para o registro e consulta das implementações disponíveis de objetos CORBA, com o auxílio do Repositório de Implementação, descrito a seguir;

- o Repositório de Implementação - responsável pelo registro das implementações de objetos CORBA existentes em um domínio, com dados persistentes que permitem a sua ativação de acordo com os modos previamente estabelecidos. As suas interfaces não são alvo de padronização; e

- a Interface ORB - permite o acesso às funcionalidades não oferecidas pelas demais interfaces.

3.2.2 Linguagem IDL

A linguagem IDL é usada para descrever as interfaces dos objetos CORBA, sendo puramente declarativa. Uma definição IDL especifica cada atributo e parâmetro das operações de uma interface definida. Uma interface IDL oferece as informações necessárias para desenvolver clientes que usam remotamente as operações disponíveis. Tanto o cliente quanto a implementação de objetos não são escritos em IDL, mas em linguagens de programação para os quais foram definidos os mapeamentos dos conceitos em IDL. O mapeamento de um conceito em IDL para uma construção de uma linguagem destino dependerá das facilidades existentes.

IDL obedece às normas léxicas de C++, apesar de que algumas novas palavras chaves tenham sido introduzidas para dar suporte à distribuição. A gramática de IDL é um subconjunto de C++ com construções adicionais para os mecanismos de requisições de operações. Ela permite a

definição de tipos, de constantes, de exceções e de módulos, além de prover suporte aos recursos de pré-processamento para inclusão de arquivos e substituição de macros.

Toda declaração de interface contém um cabeçalho e um corpo. O cabeçalho é composto por um identificador que a nomeia e por uma especificação opcional de herança. O identificador define um nome do tipo interface, sendo uma referência para o objeto que suporte tal interface. O corpo de uma interface pode conter declarações : de constantes, de tipos, desde que não sejam novas interfaces, de exceções, de atributos, e de operações, incluindo o nome da operação, seu modo de execução, os tipos de seus parâmetros e do dado de retorno. A assinatura de uma interface é caracterizada por suas operações e atributos. Entretanto, uma definição de atributo é logicamente equivalente à declaração de um par de operação de acesso: uma para a leitura e outra para escrita.

Através do mecanismo de herança, uma interface pode ser derivada a partir de uma interface base, usando o operador de resolução de escopo "::". Uma interface pode ser derivada a partir de qualquer número de interfaces base e a ordem de derivação não é significativa. IDL ainda dá suporte a tipos de objetos e tipos de dados não objetos. Esses tipos não objetos podem ser divididos em tipos básicos: inteiros, números de pontos flutuantes, caracteres, booleanos, octetos e genérico "any"; e em tipos construídos: "array", "sequence", "string", "struct", "union" e "enum". IDL ainda oferece construções para nomear tipos de dados, semelhantes aos oferecidos pela linguagem C: "typedef", "struct", "union" e "enum". Uma seqüência (sequence) é um vetor unidimensional com duas características: um tamanho máximo, fixado em tempo de compilação, e um comprimento, determinado em tempo de execução. Um seqüência pode ser declarada, especificando ou não o seu tamanho máximo. O tipo "string" definido em IDL pode ser considerado como uma seqüência de caracteres, sendo mapeado para algum tipo da linguagem alvo.Os tipos de dados básicos do modelo CORBA são mostrados na Tabela 3.1, com a sua sintaxe em IDL e sua descrição. Da mesma forma, são apresentados os tipos estruturados na Tabela 3.2.

Tabela 3.1 - Tipos Básicos

Tipo	Sintaxe	Descrição
inteiro	short	número inteiro de 32 bits com sinal
inteiro	long	número inteiro de 64 bits com sinal
inteiro	unsigned short	número inteiro de 32 bits sem sinal
inteiro	unsigned long	número inteiro de 64 bits sem sinal
ponto flutuante	float	número de ponto flutuante de 32 bits (IEEE)
ponto flutuante	double	número de ponto flutuante de 64 bits (IEEE)
caracteres	char	
booleano	boolean	tipo booleano com valores "TRUE" e "FALSE"
octeto	octet	tipo de dado opaco de 8 bits que garante que nenhuma conversão será feita durante a sua transmissão entre os sistemas
genérico	any	tipo que pode representar qualquer tipo básico ou estruturado

Tabela 3.2 - Tipos Estruturados

Tipo	Sintaxe	Descrição
enumerado	enum <identifier>	lista ordenada de identificadores
estrutura	struct <identifier>	estrutura ordenada de pares
união	union <identifier>	um discriminante seguido de uma instância de um tipo apropriado para o discriminante
vetor	array	vetor de tamanho fixo de valores de um único tipo
seqüência	sequence <<type>, <const>>	vetor de tamanho variável de valores de um único tipo, sendo que o tamanho da seqüência é determinado em tempo de execução
vetor de caracteres	string	vetor de caracteres de tamanho variável

3.2.3 Serviços de Objetos e Facilidades Comuns CORBA

O cenário final, idealizado pela OMG, apresenta diferentes objetos CORBA, que vão além da simples interação transparente entre um e outro. A proposta é complementar as funcionalidades do barramento CORBA, como "middleware", e oferecer infra-estruturas (frameworks) para auxiliar a especificação e implementação de aplicações distribuídas, explorando as propriedades de reusabilidade e modularidade oferecidos pelo paradigma. A colaboração entre objetos CORBA é

definida no Modelo de Referência da Arquitetura de Gerência de Objetos (OMG, 1996b). A Figura 3.7 apresenta os componentes da definição.

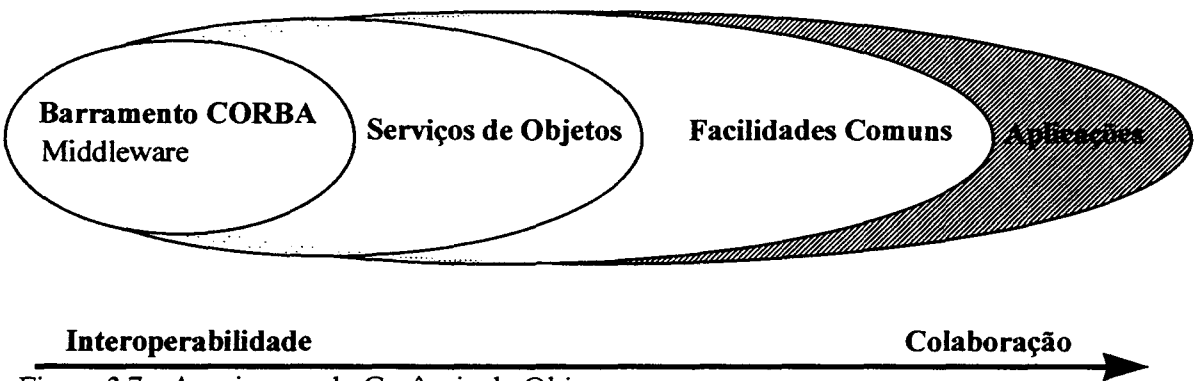

Figura 3.7 - Arquitetura de Gerência de Objetos

Além de CORBA, a OMG padronizou outros serviços de especial interesse para o desenvolvimento de aplicações distribuídas. Os Serviços de Objetos, conhecidos também como CORBAservices (OMG, 1997b), agrupam coleções de componentes definidos com interfaces IDL mais complexas com um complemento às funcionalidades de "middleware" do ORB. A partir dos CORBAservices já existentes, pode-se definir um novo objeto CORBA que usa as interfaces definidas em IDL do serviço desejado através de herança ou associação. Os seguintes serviços são de especial interesse para o desenvolvimento de aplicações distribuídas de gerência:

- Serviço de Nomes é empregado para resolução de nomes em um contexto de um ambiente ORB;
- Serviço de Eventos é concebido para desacoplar as comunicações entre objetos fornecedores que geram eventos e os consumidores que os recebem e os processam, usando os modelos "push" ou "pull";
- Serviço de Ciclo de Vida é usado para permitir as operações de criação, destruição, cópia e movimentação de objetos;
- Serviço de Tempo é desejável para se ter um tempo padrão acurado e preciso para sincronizar atividades distribuídas;

- Serviço de Transação para suportar múltiplos modelos de transações, inclusive do tipo aninhadas;
- Serviço de Controle de Concorrência para coordenar o acesso de múltiplos clientes a recursos compartilhados, resolvendo conflitos e evitando inconsistências, através do uso de trancas;
- Serviço de Relacionamentos permite que entidades (objetos CORBA) e seus relacionamentos com outros objetos sejam explicitamente representados. Objetos distribuídos não existem isoladamente, eles são conectados, podendo ser representados por grafos. Este serviço oferece uma interface com operações para percorrer o grafo de relacionamentos;
- Serviço de Persistência oferece um conjunto de interfaces para os mecanismos usados para gerenciar o estado persistente dos objetos CORBA, a partir dos quais pode-se reconstruir o estado dinâmico; e
- Serviço de Segurança controla o acesso às informações, autentica as partes envolvidas e cifra mensagens quando necessário.

As Facilidades Comuns são, por definição, coleções de objetos que proporcionam serviços para serem utilizados diretamente pelas aplicações, englobando funcionalidades mais ricas do que as dos Serviços de Objetos. São também conhecidas como CORBAfacilities e podem ser definidas como sendo especializações dos CORBAservices, herdando as suas interfaces. Elas ainda são divididas em duas categorias: horizontal e vertical (OMG, 1997a).

As facilidades horizontais são divididas em quatro tipos:

- Interface do usuário: são objetos de interface gráfica para a definição de um padrão para as aplicações;
- Gerência de Informações: permite compor documentos e trocar informações entre aplicativos, de modo semelhante ao que ocorre nos padrões OLE (Object Linking and Embedding) e OpenDoc;

- Gerência de Sistemas: define um conjunto de serviços para a monitorização e gerência de sistemas distribuídos. Esta facilidade é de especial interesse para o tema desta dissertação e será melhor descrita no próximo capítulo; e
- Gerência de Tarefas: implementa serviços de "workflow", agentes, transações longas e correio eletrônico.

As facilidades verticais provêem interfaces definidas também em IDL para segmentos especializados como a área financeira, saúde, transportes, seguros, etc.

Atualmente, a OMG tem abordado praticamente todo o espectro de tópicos relacionados à computação distribuída, como sistemas de tempo real, Internet, telecomunicações, sistemas financeiros, sistemas C4I (Command, Control, Communication, Computer and Information), comércio eletrônico, segurança, análise e projeto orientados a objeto, linguagens de programação, entre outros. Tais temas são discutidos através de listas de discussão e publicados através de RFPs (Request for Proposals) e RFCs (Request for Comments), disponíveis para os seus membros e, posteriormente, para o público. No início, as atividades técnicas do grupo estavam concentradas principalmente na especificação da CORBA através do grupo tarefa ORB. Contudo, à medida que a padronização da CORBA amadurecia, o foco de interesse foi desviado para os Serviços de Objetos e as Facilidades Comuns. Como conseqüência, os comitês técnicos foram divididos em dois:

- Comitê Técnico de Domínios, composto por forças tarefas para tratar de tecnologias orientadas para segmentos especializados como financeiro, de negócios, de telecomunicações e médico; e o
- Comitê Técnico da Plataforma, composto pelas forças tarefas com o propósito de acompanhar a evolução das especificações da CORBA com novos mapeamentos, Serviços de Objetos e Facilidades Comuns.

A integridade do Modelo de Referência é garantida por uma instância superior, conhecida como Quadro da Arquitetura (AB - Architecture Board) , composta por dez membros eleitos e um presidente, com o poder de rejeitar RFPs que não obedeçam ao modelo.

3.2.4 Motivação para o uso de CORBA em gerência

O paradigma da orientação a objeto sempre influenciou a padronização dos modelos de gerência, descritos no capítulo anterior, sendo um paradigma adequado tanto para a gerência de redes e de sistemas, quanto para aplicações distribuídas. A arquitetura CORBA para a distribuição de objetos tem sido adotada como um padrão aberto e não proprietário em um consenso das instituições membros e o seu uso tem sido efetivamente investigado no contexto de gerência (Feldkhun, 1997).

O modelo de informação oferecido por CORBA IDL é comparável a GDMO e superior ao do SNMP, permitindo a clara separação entre declaração de interface e a sua implementação. CORBA oferece o paradigma da orientação a objeto da mesma forma que os modelos OSI/CMIP e TMN com um maior grau de abstração. Da mesma forma que GDMO apresenta tal característica de separação, CORBA oferece uma solução muito mais simples e não requer que os desenvolvedores conheçam notações de sintaxe abstrata e esquema de codificação e decodificação. A sintaxe de IDL é basicamente uma extensão de C++, tendo sido especificada com o propósito de se definir facilmente o seu mapeamento para linguagens de implementação como C, C++, Smalltalk e Java. Uma outra motivação é a possibilidade de se usar ferramentas de desenvolvimento que suportam a tradução de modelos de objetos OMT (Rumbaugh, 1991) em interfaces IDL com compiladores que geram automaticamente os "stubs" e esqueletos dos objetos das aplicações.

Do ponto de vista de comunicação, CORBA oferece mais transparências com uma arquitetura mais elegante, sem a necessidade do agente intermediador SNMP/CMIP. Uma referência de objetos interoperável identifica unicamente um objeto CORBA. Uma plataforma ORB se encarrega de localizar a implementação de objetos, de sua ativação, do envio da invocação de métodos e das suas respostas. Uma vez conhecida a interface de um objeto servidor, um cliente CORBA pode ter acesso a seus métodos através das interfaces estáticas e dinâmicas. Chamadas estáticas usam os "stubs" gerados em tempo de compilação, a partir de descrições IDL dos objetos gerenciados a serem invocados. Chamadas dinâmicas utilizam a Interface de Invocação Dinâmica, que permite a um gerente construir e realizar chamadas a objetos gerenciados em tempo de execução através do Repositório de Interfaces. Uma das vantagens dessa abordagem é

o desenvolvimento de aplicações de gerência sob a forma de clientes CORBA, usufruindo das facilidades de verificação oferecidas pelo uso de uma IDL, ao invés de comunicação baseada diretamente em um protocolo de gerência. O uso deste repositório permite que se pense em fugir do modelo tradicional, onde a aplicação de gerência só pode oferecer ao usuário os serviços descritos na MIB, e onde o agente não tem como comunicar à aplicação de gerência alterações na sua funcionalidade. Pode-se ter acesso ao repositório através de uma ferramenta estilo "browser", e, baseando-se na informação coletada, gerar uma nova versão da aplicação de gerência (Rodriguez, 1997).

Recentemente, Java foi apresentada como uma nova plataforma de desenvolvimento de "software" independente de sistema operacional e processador, sendo amplamente difundida e rapidamente aceita (Flanagan, 1996). Tal sucesso se deve a sua similaridade com outras linguagens de programação já conhecidas e a sua rica biblioteca. A semelhança sintática com C e C++ é óbvia, enquanto o modelo de objetos e de execução foram herdados de Smalltalk e Simula67, facilitando o seu aprendizado. Sua especificação, tanto da linguagem de programação quanto do interpretador, é bastante simples, se comparada com C++. Por outro lado, suas bibliotecas de classes oferecem recursos para programação distribuída e concorrente, nos moldes exigidos para aplicações de gerência, sem deixar de lado requisitos básicos de interface gráfica e de mecanismos de entrada/saída. Embora o seu desempenho seja questionado por ser interpretada, existem soluções propostas de traduzir Java pré-compilado para código nativo da plataforma, através de compiladores sob-demanda. O seu interpretador, conhecido como Máquina Virtual de Java, executa programas Java pré-compilado em "byte-codes", permitindo múltiplas linhas de execução (threads) e oferecendo primitivas para sincronização entre elas. Tais características despertaram o interesses da OMG em propor o seu mapeamento para IDL e da comunidade curiosa no seu uso como tecnologia complementar na área de gerência.

A biblioteca Java "Remote Method Invocation" (RMI), presente na última versão disponível do Kit de Desenvolvimento Java (JDK - Java Development Kit), oferece classes que permitem o desenvolvimento de aplicações distribuídas, de forma razoavelmente simples e transparente (Sun, 1996). Na verdade, RMI é um protocolo conceitualmente parecido com o RPC, adaptado às características próprias de Java em que os objetos, no papel de servidor, ao receberem uma

requisição são serializados e passados por valor pela rede. Esta facilidade tem sido bastante discutida como uma solução alternativa à proposta aberta da OMG, podendo até mesmo concorrer com CORBA. Isto nos motivou a incluir Java RMI na Tabela 3.3, originalmente apresentada em (Orfali, 1996) e com mais algumas modificações. São resumidas as diferenças entre os modelos de gerência e os benefícios da adoção de CORBA ou Java como modelo integrador.

De uma forma geral, a motivação para o uso de CORBA como um modelo de gerência pode ser descrita pelos seguintes aspectos:

- a necessidade de se substituir o protocolo SNMP, criado em uma época em que as especificações de gerência eram restritas por causa da escassez de recursos de memória e de processamento. Ele não é apropriado para a gerência de aplicações;
- permite que os mesmos conceitos, técnicas e ferramentas usados para especificar, projetar e implementar sistemas distribuídos sejam também utilizados para a sua gerência;
- a definição de uma MIB em IDL permite que esta seja acessada transparentemente por um gerente CORBA em um ambiente de gerência distribuído e aberto;
- gerentes CORBA são capazes de ter uma visão uniforme dos recursos gerenciados através de uma interface de gerência em IDL, sejam eles componentes de redes, sistemas de suporte ou aplicações distribuídas. Este é um passo importante para o suporte ao conceito de gerência integrada;
- o modelo de objetos distribuídos de CORBA estende o modelo agente/gerente de CMIP e SNMP. Permite ainda que os objetos gerenciados possam chamar diretamente a entidade gerente, quando houver necessidade de reportar um evento significativo, além de oferecer a facilidade de invocação dinâmica;
- permite que os programadores não mudem o seu estilo original de programação, usando a sua linguagem de implementação favorita e conceitos que lhes são familiares como tratamento de exceções, ao invés de erros de protocolos; e

- as implementações CORBA em Java (Vogel, 1997) e o protocolo IIOP, já embutido no última versão do navegador da "Netscape", permitirão mais facilmente a integração das ferramentas de gerência com a tecnologia WWW (World Wide Web).

Tabela 3.3 - Comparação entre modelos

Característica	SNMP	SNMPv2	CMIP	Java RMI	CORBA
Nível de abstração	Baixo	Baixo	Baixo	Alto	Alto
Base Instalada	Grande	Pequena	Pequena	Muito Pequena	Muito Pequena
Objetos Gerenciados por Estação de Gerência	Pequena	Pequena	Grande	Grande	Grande
Modelo	Gerente e Agente	Gerente e Agente	Gerente e Agente	Serialização de Objetos	Objetos Distribuídos
Visão dos Objetos Gerenciados	Simples Variáveis organizadas em MIBs	Simples Variáveis organizadas em MIBs	Objetos com herança, definidos em MITs	Objetos com interface em Java, atributos e herança simples	Objetos com interfaces em IDL, atributos e múltipla herança
Independência de Linguagem	Sim	Sim	Sim	Não	Sim
Protocolo Interoperável	Não	Não	Não	Não	Sim
Interações Gerente/Agente	Polling. Raros Traps	Polling. Raros Traps	Eventos	Requisição & Resposta	Requisição & Resposta/ Eventos
Invocação Dinâmica	Não	Não	Não	Não	Sim
Ação explícita	Não	Não	Sim	Sim	Sim
Segurança	Não	Não	Sim	Sim	Sim
Interação entre Gerentes	Não	Sim	Sim	Sim	Sim
Transferência em Bloco	Não	Sim	Sim	Sim	Sim
Criação e Destruição	Não	Não	Sim	Sim	Sim
Modelo de	Datagrama	Datagrama	Sessão	RPC	ORB

Comunicação					
Órgão Padronizador	IETF	IETF	OSI	SunSoft	OMG

3.3 Conclusão

O RM-ODP provê uma infra-estrutura para aplicações distribuídas, capaz de ser usada para a integração dos modelos atuais de gerência. Contudo, a padronização está mais concentrada no desenvolvimento de sistemas distribuídos do que em sua gerência. Embora a técnica de modelagem do padrão OSI/CMIP não seja adequada para a gerência de sistemas distribuídos, a experiência obtida deve ser usada para a gerência de um sistema ODP. Os esforços da padronização ODMA estão tentando suprir essa deficiência e já incorpora CORBA em seu modelo.

O uso de CORBA como modelo integrador de gerência tem sido bastante discutida. Os mais tradicionais congressos na área já dedicam sessões exclusivas para o assunto. Contudo, a proposta não é substituir as arquiteturas existentes por CORBA, mas investigar a melhor forma de CORBA ser usada em conjunto com a atual infra-estrutura investida (Dittrich, 1997). Neste aspecto, o uso de Java e CORBA, como tecnologias complementares, poderá atender aos novos requisitos e especificações de gerência.

Embora tenha sido anunciado, durante o seu lançamento, que Java seria uma ferramenta de especial interesse na área de gerência de sistemas e sua biblioteca RMI soe como uma solução concorrente com CORBA, concordamos apenas com a primeira observação no atual estágio. Não acreditamos porque os objetos servidores Java sofrem de baixo desempenho devido às inerentes limitações de sua Máquina Virtual quando comparados com servidores CORBA. A resposta a uma chamada remota de métodos, no seu modelo de RPC, é antecedida pela serialização e a passagem dos objetos por valor pela rede, representando um custo muito alto. Além disso, não oferece interoperabilidade entre diferentes linguagens, restringindo o desenvolvimento dos objetos a uma única. RMI também não oferece nenhuma política de ativação de objetos como as disponíveis em CORBA. Além disso, CORBA ainda oferece os Serviços de Objetos e as Facilidades Comuns como um conjunto de APIs padrões para o

desenvolvimento de aplicações distribuídas interoperáveis. Na verdade, a própria SunSoft tem anunciado a disponibilidade do protocolo IIOP e do Serviço de Transações CORBA nas futuras versões do JDK.

Capítulo 4

Arquitetura de Gerência de Sistemas Distribuídos

Este capítulo descreve uma arquitetura proposta para gerência de sistemas distribuídos, refletindo o amadurecimento obtido com a leitura das principais referências sobre o assunto. Tal arquitetura pretende oferecer um conjunto de APIs em IDL para o desenvolvimento de aplicações distribuídas CORBA de gerência para os três domínios existentes: SNMP, CMIP e CORBA. Usa como infra-estrutura de suporte um ORB, os Serviços de Objetos, as Facilidades Comuns e os adaptadores SNMP/CMIP.

4.1 Introdução

A viabilidade de uso de um "middleware" tipo CORBA como tecnologia complementar para o desenvolvimento de ferramentas de gerência permite que se possa especificar arquiteturas livres do modelo restritivo agente/servidor dos paradigmas anteriores. Além disso, os serviços oferecidos pelos CORBAservices oferecem um conjunto de métodos já disponíveis e que, na sua ausência, precisariam ser especificados e desenvolvidos. A idéia de se adotar um "middleware" para gerência já havia surgido anteriormente com a proposta de especificação do Ambiente de Gerência Distribuída (DME - Distributed Management Environment) (Autrata, 1994), sobre o RPC/DCE (Distributed Computing Environment), padronizado por um consórcio, similar à OMG, conhecido como "Open Software Foundation". Tal proposta foi abandonada pelo fato do RPC da OSF não suportar o paradigma da orientação a objetos, tendo que incorporar um ORB e os CORBAservices no seu ambiente. Precisaria de um mapeamento específico estático e dinâmico de DCE IDL para CORBA IDL, introduzindo um custo adicional. O conceito de herança de interfaces CORBA IDL precisaria ser cuidadosamente tratado neste mapeamento. No lado do cliente, por não existir o conceito de invocação dinâmica em DCE, tanto as requisições CORBA estáticas quanto as dinâmicas, sintaticamente diferentes, teriam que ser convertidas em chamadas de procedimentos remotos. No lado do servidor, precisariam ser implementadas as políticas de ativação inexistentes em DCE. Mais ainda, o suporte a exceções em DCE não oferece a facilidade existente em CORBA de definição de estruturas pelos usuários.

A própria OMG manifestou o seu interesse em especificar uma infra-estrutura para gerência como uma das CORBAfacilities. Na revisão da Arquitetura das Facilidades Comuns de janeiro de 1995 (OMG, 1995), a proposta da Facilidade de Gerência de Sistemas era oferecer um conjunto de APIs para

- instrumentação de objetos CORBA;
- coleta de dados históricos em registros;
- especificação de níveis de qualidade de serviço;

- gerência de segurança;
- gerência de eventos baseada no Serviço de Eventos;
- escalonamento de tarefas; e
- configuração de instâncias.

Embora a proposta da OMG fosse muito genérica, apresentava o DME como um trabalho relacionado e incentivava a adoção das especificações dos Serviços de Objetos e das outras Facilidades Comuns, muitas delas ainda em fase de discussão. Tal documento foi a semente germinadora do tema desta dissertação.

4.2 Arquitetura de Gerência de Sistemas Distribuídos

A arquitetura descrita nesta seção é parte da plataforma Multiware (Loyolla, 1994), ilustrada na Figura 4.1, em um projeto que adota o RM-ODP e CORBA como padrões para se ter um ambiente distribuído, heterogêneo, aberto e não proprietário, assimilando as idéias de outros padrões existentes como ANSA/APM (Advance Networked Architecture/Archictecture Projects Management) e DCE/OSF . É sedimentada em três camadas de suporte a aplicações distribuídas: uma de Hardware/Software básico, a Middleware e a Groupware. A primeira camada é composta pelos sistemas operacionais e protocolos de comunicação. A camada Middleware é responsável por prover serviços de processamento distribuído aberto para a camada superior e para as aplicações distribuídas. É composta de uma ORB e de alguns serviços ODP, como o Trader (Lima, 1995), Suporte a Transação e a Grupo (Costa, 1996), e Qualidade de Serviço (Lima, 1997). Atualmente, a Middleware está sendo desenvolvida sobre a Orbix da Iona Technologies (Iona, 1995). Existe ainda uma subcamada de Processamento Multimídia que possibilita a troca de informações multimídia com uma qualidade de serviço especificada. Finalmente, aplicações CSCW (Computer Supported Cooperative Work) são suportadas pela camada Groupware.

O ambiente da plataforma Multiware consiste de um grande número de nós de diferentes arquiteturas, dispersos pelos dois laboratórios do Instituto de Computação e da Faculdade de Engenharia Elétrica e Computação, conectados por redes heterogêneas de comunicação, vários sistemas operacionais e muitos serviços de suporte. Tal infra-estrutura oferece recursos para o desenvolvimento e a execução de aplicações distribuídas. As soluções de gerência devem cobrir todos os componentes de um ambiente distribuído e heterogêneo como este, com o propósito de entender as suas interações e correlações. Os processos que implementam uma aplicação distribuída também devem ser considerados como recursos gerenciados. Como CORBA é adotada para implementar este ambiente, os elementos das aplicações são objetos, com interfaces claramente definidas em IDL, cujos métodos podem ser invocados remotamente. Tais objetos residem em servidores, processos que oferecem um ou mais serviços CORBA, cujas interfaces são transparentemente chamadas por programas clientes.

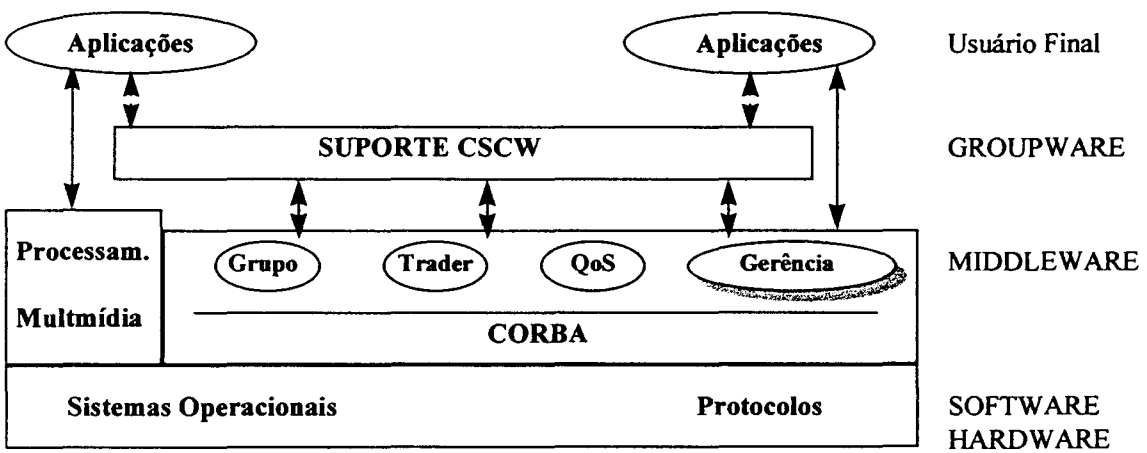

Figura 4.1 - Plataforma Multiware

A Arquitetura de Gerência de Sistemas Distribuídos proposta é baseada nos conceitos da ODMA, que adota o RM-ODP, e usa a CORBA, os CORBAservices e as CORBAfacilities como suporte, com a proposta de prover pelo menos a mesma funcionalidade do modelo de Gerência OSI/CMIP. Ela também foi influenciada pelos aspectos descritos em (Sloman, 1993; Sloman, 1995; Bauer, 1994).

(Sloman, 1993) sugeriu uma arquitetura, baseada em DME, com o propósito de incorporar as aplicações distribuídas como recursos gerenciados e enfatizar o uso de domínios para agrupar os objetos gerenciados e de um serviço para a especificação e manipulação de políticas para guiar o processo de decisão dos gerentes. Neste artigo, o autor referenciado menciona que as especificações OSI/CMIP dos serviços de domínios e de políticas eram confusas e complicadas. A composição de domínios OSI/CMIP é definida através de um critério de agrupamento, implicando na existência de objetos ativos que buscam recursos gerenciados que atendem ao critério estabelecido. O serviço OSI/CMIP de domínios também suporta o conceito de políticas dentro de sua especificação, contrariando os requisitos de modularidade e de flexibilidade. Segundo Sloman, tal fato é decorrente da necessidade de amadurecer o assunto antes de uma padronização. Comenta ainda que não havia nenhuma implementação disponível de DME e que seria necessário adotar adaptadores que traduzissem invocações DME em SNMP/CMIP. Por estas razões, o seu protótipo estava sobre uma plataforma da ANSA, orientada a objetos.

(Bauer, 1994) propôs uma arquitetura de referência de sistemas distribuídos que integra três classes de objetos gerenciados: elementos e serviços de redes de comunicação, sistemas operacionais e recursos de suporte, e os componentes das aplicações distribuídas de usuários. Enfatiza ainda que o desempenho de tais aplicações depende da adequada gerência dos três tipos de objetos em uma abordagem integrada. O âmago da arquitetura é um conjunto de serviços organizados como quatro subsistemas: monitorização, controle, configuração, e repositório. O serviço de monitorização é responsável em monitorar o comportamento dos objetos gerenciados, incorporando mecanismos de filtragem e de correlação de dados para diminuir o processamento final destes dados pelos gerentes. O serviço de controle é responsável em executar ações de controle a partir dos dados monitorados. O serviço de configuração estabelece a configuração inicial dos componentes de um sistema distribuído, acompanha as modificações e intervém, caso detecte a diminuição de desempenho. Finalmente, o serviço de repositório de informações ofereceria interfaces para acesso aos dados armazenados. Em um artigo posterior sobre a arquitetura (Rolia, 1995), os autores já incluíam CORBA e DCE como "middlewares" candidatas para suporte a sua arquitetura e destacavam a necessidade de se instrumentar os sistemas de suporte e os componentes das aplicações distribuídas para monitorização.

Baseado nestes aspectos acima discutidos, a arquitetura proposta pretende atender aos seguintes requisitos de projeto:

- abordar principalmente o domínio das aplicações CORBA, integrado com os sistemas de suporte subjacentes e os componentes de rede;
- usar o IIOP como protocolo de interoperabilidade de gerência e incorporar os Serviços de Objetos e as Facilidades Comuns. Este requisito exige o acompanhamento da evolução das especificações;
- permitir a monitorização, a execução de ações de configuração e de controle dos recursos gerenciados agrupados em domínios e sujeitos a políticas;
- oferecer uma interface gráfica comum e consistente, simplificando a interação do usuário;
- manter o custo de processamento e de consumo de memória o mais baixo possível; e
- atender às necessidades de interoperabilidade e escalabilidade, inerentes aos sistemas distribuídos.

Tal arquitetura é ilustrada na Figura 4.2. Não existe uma única aplicação monolítica, centralizada em uma plataforma, desempenhando todos os aspectos do processo de gerência. Existe um conjunto de aplicações cooperativas oferecendo uma interface única ao usuário, abordando as áreas funcionais de desempenho, de contabilização, de configuração, de falhas e de segurança. Elas habilitam uma visão logicamente centralizada do domínio gerenciado, independente da distribuição física de seus componentes. Há ainda um conjunto de Facilidades para Monitorização, para Controle, para Configuração, para o estabelecimento de Políticas e de Domínios, usadas como pequenas peças de um "lego" para o desenvolvimento das aplicações. O nível de suporte, composto por uma plataforma ORB com os CORBAservices e as CORBAfacilities, auxilia o desenvolvimento e implementação das aplicações e das facilidades. Mais ainda, este nível permite que os objetos gerenciados e gerentes sejam compilados em diferentes linguagens, desde que exista o correspondente mapeamento para CORBA IDL. O uso das APIs padrões dos CORBAservices e das CORBAfacilities ainda garante portabilidade e

reusabilidade de código, permitindo que os desenvolvedores manipulem objetos que são independentes do comportamento que herdam das classes.

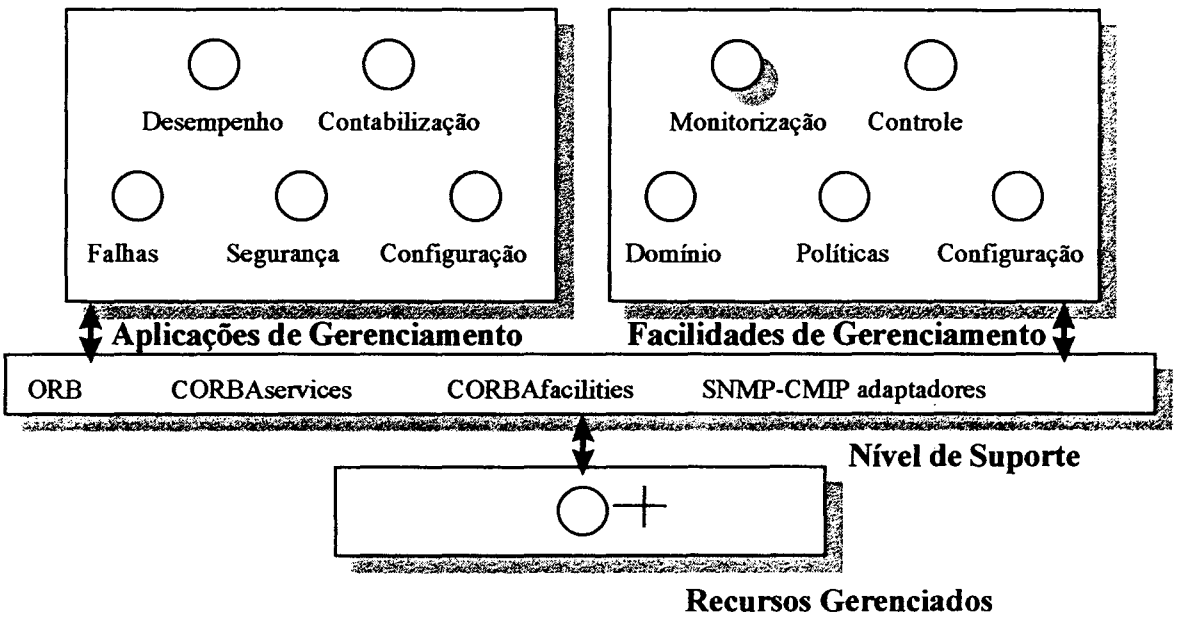

Figura 4.2 - Arquitetura de Gerência Integrada

A monitorização consiste na observação das atividades dos componentes dos sistemas distribuídos e da aquisição de dados estáticos e dinâmicos através de requisições e notificações. A Facilidade de Monitorização é definida como um processo dinâmico para aquisição, coleta e apresentação de informações sobre os recursos gerenciados, essencial ao processo de decisão. Um serviço de filtragem e de correlação de eventos é necessário para selecionar os dados coletados. A monitorização inclui as tarefas de geração, de processamento, de disseminação e de apresentação da informação monitorada (Mansouri-Samani, 1993). O Serviço de Eventos é usado para estabelecer comunicação assíncrona entre os objetos gerenciados e as entidades gerentes. Eventos podem ser gerados a partir de limites estabelecidos, usando os modelos "push" e "pull" através um canal de eventos. Os recursos do AWT (Abstract Windowing Toolkit) da linguagem Java são usados para criar apresentações dinâmicas e interativas das informações de gerência coletadas e filtradas, através de desenhos gráficos e componentes como listas, diálogos e botões.

A Facilidade de Controle desempenha ações para evitar a degeneração de um sistema distribuído, controlando o comportamento dos objetos gerenciados. Tais situações são estabelecidas a partir da informação do estado de um componente, exigindo ações reativas, preventivas e pró-ativas. Estas condições são normalmente detectadas pela Facilidade de Monitorização. Os seus principais clientes são a Facilidade de Configuração e as aplicações de desempenho. Um aspecto muito importante é garantir que somente as ações permitidas pela Facilidade de Políticas sejam executadas por gerentes ou por sistemas devidamente autorizados e autenticados.

A Facilidade de Configuração é capaz de estabelecer a configuração inicial de um sistema distribuído, acompanhar as mudanças e executar as modificações, quando necessárias. Em grandes sistemas, intervenções humanas são sujeitas a erros, podendo causar falhas. Tal facilidade é derivada do Serviço de Ciclo de Vida para permitir a criação, destruição, cópia e migração dos objetos gerenciados. O serviço de migração é desejável para migrar objetos para outros nós por motivos de balanceamento de carga ou para garantir qualidade de serviço.

Um aspecto importante é a especificação de quais operações de gerência um gerente pode executar através de permissões, obrigações e proibições com o objetivo de guiar o processo de tomada de decisão. Existe portanto uma necessidade de se especificar, representar e manipular políticas (Becker, 1993). A especificação começa com políticas abstratas que são refinadas, após muitas interações, em ações que possam ser interpretadas por sistemas computacionais (Marriot, 1994). A Facilidade de Políticas oferece operações para a criação e destruição de políticas, para ler e escrever seus atributos, para verificar políticas associadas a um domínio e determinar os domínios sujeitos a uma determinada política. A Facilidade de Domínios é usada para agrupar objetos sujeitos a uma política.

Em um ambiente ODP, existem múltiplas visões de gerência e diferentes limites de responsabilidades (Yemini, 1993). A gerência deve ser estruturada em domínios para dividir responsabilidade e autorização entre diferentes gerentes (Sloman, 1995). Tal estruturação pode refletir a conectividade física da rede, um domínio da plataforma ou ainda o organograma administrativo. Os domínios estabelecidos não encapsulam seus membros. Eles são entidades passivas que mantêm referências para seus membros que são especificados explicitamente e não em termos de um critério de agrupamento. A Facilidade de Domínio provê operações para a

escrita e leitura de atributos de um domínio, para a criação e destruição de domínios, para listar os seus componentes e inserir ou remover membros de um domínio.

O nível de suporte, composto por uma plataforma ORB, alguns CORBAservices e CORBAfacilities, permite o desenvolvimento de aplicações de gerência, simplificando a implementação e garantindo interoperabilidade a nível de código. O Serviço de Tempo é desejável para se ter um tempo padrão acurado e preciso para sincronizar atividades de Gerência e mensagens. O Serviço de Nome é empregado para resolução de nomes em um contexto de gerência. O Serviço de Eventos é concebido para desacoplar as comunicações entre objetos gerenciados que geram eventos e os gerentes que os recebem através do Canal de Eventos, usando os modelos "push" ou "pull". O Serviço de Ciclo de Vida é usado para permitir as operações para criação, destruição, cópia e movimentação de objetos. O Serviço de Concorrência para coordenar o acesso de múltiplos clientes a recursos compartilhados, resolvendo conflitos e evitando inconsistências, através do uso de trancas (locks). O Serviço de Persistência oferece um conjunto de interfaces para os mecanismos usados para gerenciar o estado persistente dos objetos gerenciados, a partir dos quais pode-se reconstruir o estado dinâmico. Finalmente, o Serviço de Segurança controla o acesso às informações de gerência, autentica as partes envolvidas e cifra mensagens quando necessário.

Em resposta ao interesse manifestado em se ter uma Facilidade Comum de Gerência de Sistemas, o consórcio de empresas X/Open Company Limited, liderados pela Tivoli Systems, apresentou uma RFC (X/Open, 1995) com uma proposta de um conjunto de APIs para gerência. A RFC propõe oito módulos com interfaces IDL para:

1. SysAdminTypes.idl - definição de novos tipos e estruturas de dados para gerência;
2. SysAdminExcept.idl - definição de novas exceções de gerência;
3. Identification.idl - definição de métodos para a identificação única do objeto gerenciado;
4. SysAdminLifeCycle.idl - definição de métodos para a localização de objetos;
5. ManagedSets.idl - definição de métodos básicos para agrupamento de objetos em conjuntos, similares ao conceito de domínios;

6. ManagedInstances.idl - definição de serviços de suporte a gerência de instâncias através da especialização do Serviço de Ciclo de Vida;
7. PolicyRegions.idl - definição de métodos para a manipulação de políticas atribuídas a um conjunto de objetos; e
8. Policies.idl - definição de métodos para a inicialização, validação, incorporação e remoção de objetos de políticas.

Embora a própria X/Open reconhecesse que o documento oferecia apenas um subconjunto de todas as facilidades necessárias para o desenvolvimento de aplicações de gerência e tenha previsto a necessidade de identificá-las e incorporá-las ao original, tal procedimento não foi cumprido. Tal proposta foi aceita e incorporada como uma das CORBAfacilities (OMG, 1997a). As principais restrições da RFC são as seguintes:

- não ter incorporado o Serviço de Eventos para a monitorização assíncrona dos objetos gerenciados pelos gerentes. Caso houvesse ocorrido isto, teriam detectado com antecedência a necessidade de se estender tal Serviço com mecanismos de filtragem e correlação de eventos e requisitos de persistência para realmente ser útil para gerência;
- não ter especificado quais são os mecanismos usados para que um objeto CORBA apresente uma interface de gerência, além de sua interface normal;
- não ter especificado a instrumentação dos objetos CORBA; e
- não ter previsto a necessidade de adaptadores CORBA que traduzissem invocações CORBA em SNMP/CMIP.

As três primeiras observações são decorrentes do escopo deste tema de dissertação que se concentrou na especificação e na implementação da Facilidade de Monitorização em um ambiente CORBA, necessitando definir interfaces de gerência, instrumentar as implementações de objetos e estender o Serviço de Eventos. A última surge da necessidade da coexistência desejável e inevitável de CORBA com os modelos atuais de gerência, estendendo o seu domínio original. Recentemente, CORBA foi adotada pelo "Network Management Forum" (NMF) no seu "Management System Framework", uma infra-estrutura para o desenvolvimento de aplicações de gerência, como uma solução de interoperabilidade entre sistemas de gerência. Nesse sentido, o

trabalho conjunto do NMF com a própria X/Open resultou no mapeamento sintático das especificações dos padrões OSI/SNMP para CORBA IDL, permitindo a implementação de "gateways" (Soukoti, 1997) entre os paradigmas. Tal proposta será comentada na seção seguinte.

4.3 Adaptadores CORBA, SNMP e CMIP

A necessidade de se ter mecanismos que ofereçam interoperabilidade entre sistemas de gerência baseados em diferentes tecnologias motivou a formação de um grupo de trabalho, patrocinado pela X/Open e pelo fórum NMF e conhecido como "Joint Inter Domain Management". O seu foco de atenção está sobre a questão de interoperabilidade entre três tecnologias chaves: SNMP, CMIP e CORBA. SNMP tem uma grande base instalada na área de gerência de redes de computadores; CMIP é usado em gerência de redes de telecomunicações através do padrão TMN; e CORBA tem sido reconhecido como um padrão emergente em sistemas distribuídos orientados a objetos. Para se almejar tal interoperabilidade, são necessárias:

- a Tradução de Especificação para estabelecer um mapeamento sintático entre os modelos de informação; e
- a Tradução de Interação para prover um mecanismo para a conversão de protocolos.

Dessa forma, objetos em um domínio podem ser representados no outro domínio e as interações podem ser orientadas pelo domínio de escolha. Um objeto CORBA deve ser capaz de interagir com um objeto definido em GDMO como se estivesse no domínio CORBA, idealmente sem ter que saber que o objeto alvo está em um outro domínio diferente. Naturalmente, o reverso é desejável: um gerente OSI deve ser capaz de gerenciar objetos CORBA como se eles estivessem definidos em GDMO (Figura 4.3).

A Tradução de Especificação cobre o processo pelo qual as especificações são traduzidas de um domínio para outro, necessitando de algoritmos que traduzam definições GDMO/ASN.1 em interfaces CORBA IDL, SNMP MIBs em IDL e vice-versa. A Tradução de Interação cobre o processo pelo qual as interações de um domínio são traduzidas em outras correspondentes do

outro modelo. Um "gateway" deve ser capaz de receber uma CMIP PDU e transformá-la em uma ou mais requisições/respostas CORBA. Os seguintes princípios de tradução foram assumidos:

- embora ASN.1 permita a notação de construções difíceis de serem mapeadas em IDL, sendo que muitas delas não sejam realmente usadas para a definição de um objeto GDMO, a tradução deve oferecer um mapeamento tão completo quanto possível; e
- explorar e adotar, sempre que possível, os CORBAservices, particularmente os Serviços de Eventos e Serviço de Nomes.

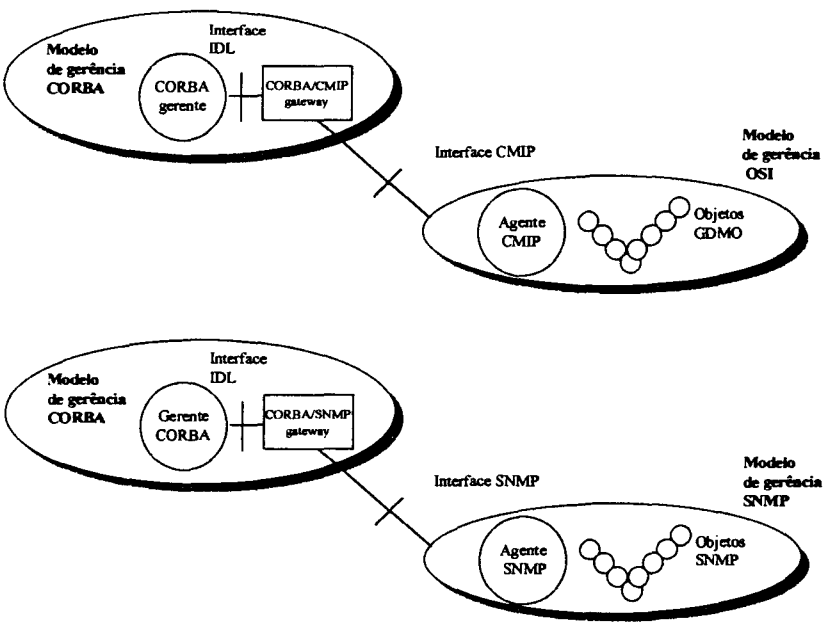

Figura 4.3 - Interoperabilidade entre Modelos de Gerência

4.4 Conclusão

O objetivo da arquitetura de gerência de sistemas distribuídos descrita é oferecer uma infra-estrutura e um conjunto de serviços úteis para o desenvolvimento de aplicações de gerência em um ambiente CORBA, com o uso dos adaptadores SNMP/CMIP, em uma proposta integrada. Os componentes da arquitetura são objetos CORBA, acessíveis por operações ORB como protocolo de gerência. Contudo, tal sugestão não pretende esgotar o assunto, pois os seus requisitos devem ser sempre revistos em função da evolução das necessidades inerentes do tema e das mudanças que possam ocorrer em seu nível de suporte. Embora a proposta da X/Open como uma infra-estrutura para o desenvolvimento de aplicações de gerência tenha sido aprovada recentemente pela OMG, existe a necessidade de absorver a especificação de outros importantes CORBAservices, como o Serviço de Eventos, e de outras CORBAfacilities. Esta observação é decorrente da especificação e da implementação da Facilidade de Monitorização, descrita nos capítulos seguintes. As outras Facilidades previstas são temas para trabalhos futuros. Existe uma base sedimentada de pesquisa sobre a manipulação de políticas e de domínios (Becker, 1993; Marriot, 1994), necessitando adaptar tais conceitos com o nível de suporte da nossa arquitetura. A Facilidade de Configuração é uma proposta de extensão do Serviço de Ciclo de Vida, precisando analisar as interfaces do módulo "ManagedInstances" da proposta da X/Open. Finalmente, a Facilidade de Controle precisa ser cuidadosamente investigada para se ter um conjunto de APIs úteis para a tomada de ações reativas, preventivas e pró ativas.

Capítulo 5

Facilidade de Monitorização

Este capítulo apresenta os principais aspectos considerados na modelagem e projeto da Facilidade de Monitorização, no contexto da arquitetura descrita no capítulo anterior, utilizando os recursos oferecidos por um ORB e pelo Serviço de Eventos.

5.1 Introdução

Em um sistema de gerência, os gerentes influenciam o comportamento dos objetos gerenciados através de invocação de operações. A decisão de quais operações precisam ser invocadas é tomada a partir da monitorização síncrona e assíncrona dos recursos gerenciados. O

comportamento destes é representado pelos seus atributos monitorados sincronamente e pelos eventos emitidos assincronamente. Considerando a estrutura de comunicação do modelo OSI/CMIP, verifica-se que a maior parte de sua funcionalidade reside no agente, oferecendo serviços de mapeamento de protocolo, empacotamento de dados e acesso remoto. Os serviços de comunicação são prestados geralmente pelo CMISE que usa o protocolo CMIP para invocação de operações, resolução de nomes e disseminação de eventos. Em um ambiente CORBA, a maioria destes serviços já é oferecida por um ORB. O uso de CORBA como plataforma de gerência resulta em uma arquitetura mais simples, onde as entidades gerentes e objetos gerenciados com interfaces em IDL continuam a desempenhar o mesmo papel, porém já não é mais necessário o agente (Figura 5.1).

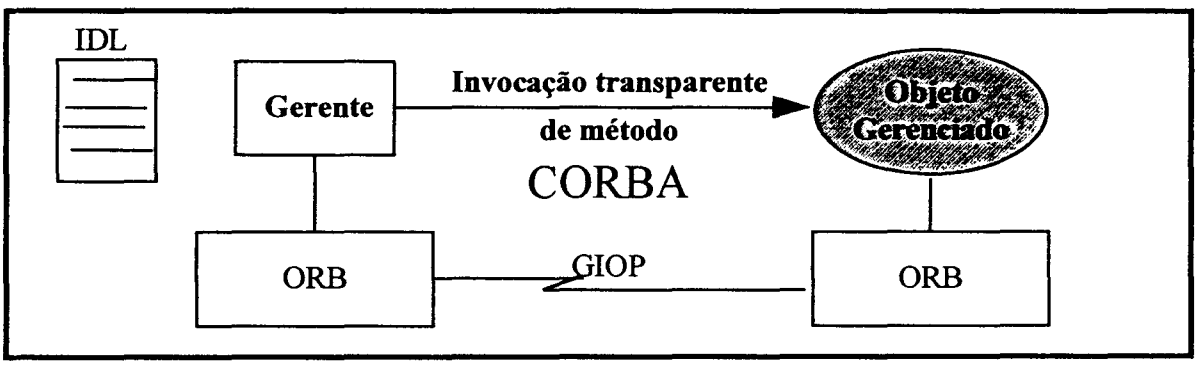

Figura 5.1 - CORBA como um ambiente de gerência.

Entretanto, a invocação remota de um método em um ORB resulta na execução síncrona da chamada, necessitando de uma extensão para suportar um modelo assíncrono de comunicação. Neste modelo assíncrono, o comportamento de um objeto gerenciado é monitorado através de um conjunto de eventos primitivos sobre as atividades mais elementares dos recursos. A quantidade e a freqüência de geração destes eventos podem ser muito elevadas para serem processadas pelas aplicações gerentes que estão interessadas em apenas um subconjunto dos eventos. Portanto, existe a necessidade de se filtrar e correlacionar esses eventos primitivos em eventos compostos. Este pré-processamento reduz o número de mensagens de notificações recebidas pelos gerentes. Em ambientes distribuídos orientados a objetos, os objetos que executam o pré-processamento de eventos podem estar separados dos gerentes e dos objetos

gerenciados, e ainda em diferentes localizações. O pré-processamento de eventos ainda pode ser aplicado recursivamente de forma que eventos compostos podem ser criados a partir de outros compostos.

Em uma plataforma ORB, uma invocação de um método resulta na execução síncrona de uma operação por um objeto CORBA, onde tanto o cliente quanto o servidor têm que estar presentes. CORBA oferece o Serviço de Eventos para comunicação assíncrona entre objetos através de um objeto mediador: o canal de eventos. Este serviço é considerado como uma extensão opcional de um ORB. As semânticas deste serviço deixam livres os objetos CORBA das semânticas restritivas do modelo síncrono de um ORB. O Serviço de Eventos será melhor descrito na próxima seção. A seguir, será feita uma análise da adequação do seu emprego em um ambiente CORBA de gerência na Seção 5.3. Na Seção 5.4, será apresentado um modelo modular, flexível e genérico de monitorização. Este modelo se mostrou útil tanto para a versão síncrona da Facilidade de Monitorização, descrita na Seção 5.5, quanto para a assíncrona, discutida na Seção 5.6. Finalmente, será feita uma breve conclusão na última seção.

5.2 Serviço de Eventos

A OMG definiu o Serviço de Eventos como um conjunto de interfaces em IDL que permite a comunicação entre objetos através de eventos. Essa funcionalidade assíncrona é oferecida externamente pelo Serviço de Eventos (OMG, 1997b), baseado no paradigma "publish/subscribe", em que um objeto publica o seu evento em um quadro de avisos e outros registram seu interesse em recebê-lo. A especificação define para os objetos os papéis de:

- fornecedor - objeto que gera eventos; e
- consumidor - objeto que recebe eventos.

Os eventos são gerados pelos fornecedores e recebidos pelos consumidores, usando os modelos de comunicação "push" e "pull". O modelo "push" permite a um fornecedor de eventos de forma ativa iniciar a transferência para um consumidor. Por outro lado, o modelo "pull" permite

a um consumidor de eventos requisitar de forma ativa eventos de um fornecedor. Portanto, os modelos determinam se a iniciativa da requisição de eventos é do fornecedor (push) ou do consumidor (pull). Inicialmente, a correspondência entre os papéis de cliente/servidor e de fornecedor/consumidor pode parecer confusa, considerando os dois modelos de início de comunicação de eventos. No modelo "push", o fornecedor envia requisições para o consumidor. Neste caso, o fornecedor é cliente e o consumidor é servidor. No modelo "pull", o consumidor requisita eventos de um fornecedor. Agora o consumidor é o cliente do fornecedor (Figura 5.2).

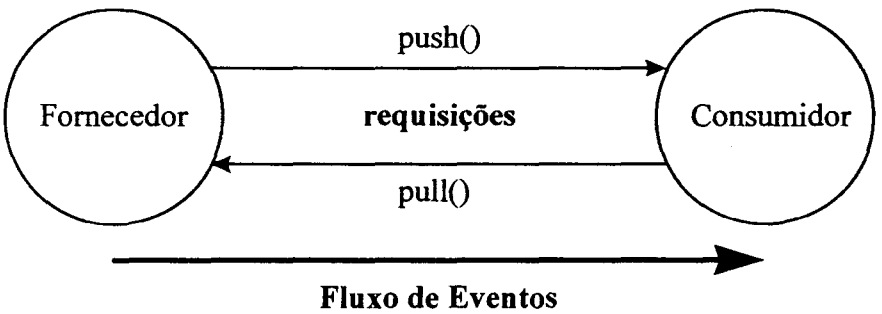

Figura 5.2 - Modelos "Push" e "Pull"

Essa forma de comunicação assíncrona direta, estabelecida pelo Serviço de Eventos, raramente será usada, porém serve para auxiliar a sua compreensão. A especificação define um estilo indireto de comunicação assíncrona que adota uma outra entidade: o Canal de Eventos. Um canal de eventos é um objeto CORBA com uma interface IDL definida para permitir a comunicação assíncrona indireta entre um fornecedor e um consumidor, tanto no modelo "pull" como no "push". Através de um canal, um fornecedor envia eventos sem conhecer a existência dos possíveis consumidores. Um único canal pode simultaneamente ter mais do que um fornecedor comunicando de forma indireta com múltiplos consumidores. Um canal nesta configuração pode ser considerado com um repetidor de "multicasts"

Além dos modelos "push" e "pull", são definidas ainda duas formas ortogonais de comunicação assíncrona na especificação. As comunicações podem ser baseadas em tipos ou genéricas, tanto no "push", quanto no "pull". São baseadas em tipos quando são realizadas através de operações

IDL e os eventos são passados através de parâmetros com tipos CORBA definidos. São genéricas quando as operações são chamadas com parâmetros do tipo CORBA "any".

5.2.1 Comunicação Assíncrona através de Canais de Eventos

Para a comunicação assíncrona indireta, um canal de eventos é introduzido entre um ou mais fornecedores e consumidores, desempenhando um papel de mediador entre os dois e gerenciando referências de objetos. Apresenta-se tanto como um consumidor "proxy" para os fornecedores quanto como um fornecedor "proxy" para os consumidores. Através de um canal, um fornecedor envia eventos sem conhecer a existência dos possíveis consumidores que os receberão. Ainda mais, um fornecedor pode enviar eventos no modelo "push" para o canal e um consumidor recebê-los no modelo "pull". Um determinado fornecedor ou consumidor pode ainda usar simultaneamente mais de um canal de eventos.

Embora a invocação direta sem o canal de eventos não seja de muito interesse para gerência, a descrição de suas interfaces é útil como uma introdução para a compreensão da comunicação indireta. Um consumidor no modelo "push" deve possuir a interface IDL "PushConsumer" para receber eventos de um fornecedor. Um fornecedor envia eventos através da invocação da operação "push" do consumidor com o evento como parâmetro. A interface ainda tem a operação "disconnect_push_consumer" para encerrar a comunicação e liberar recursos associados. Do outro lado, um fornecedor neste modelo pode opcionalmente ter a interface "PushSupplier" com a operação "disconnect_supplier" para informar ao consumidor a desconexão. No modelo "pull", um fornecedor deve implementar a interface "PullSupplier" para transmitir eventos para um consumidor através da operação bloqueadora "pull" ou da não bloqueadora "try_pull", além de possuir o método "disconnect_pull_supplier" para encerrar a comunicação assíncrona. Um consumidor pode ainda implementar a interface "PullConsumer" com a operação "disconnect_pull_consumer" para a sua desconexão. É importante ressaltar que estas interfaces podem ser implementadas diretamente sobre um ORB, sem nenhum serviço adicional.

Para a comunicação indireta através de um canal de eventos, a especificação define as interfaces de "PushSupplier" e "PullSupplier" para os objetos fornecedores; de "PushConsumer" e "PullConsumer" para os consumidores; e "ProxyPushConsumer", "ProxyPullConsumer", "ProxyPushSupplier" e "ProxyPullSupplier" para os canais. Estes diferentes papéis podem ser visualizados na Figura 5.3.

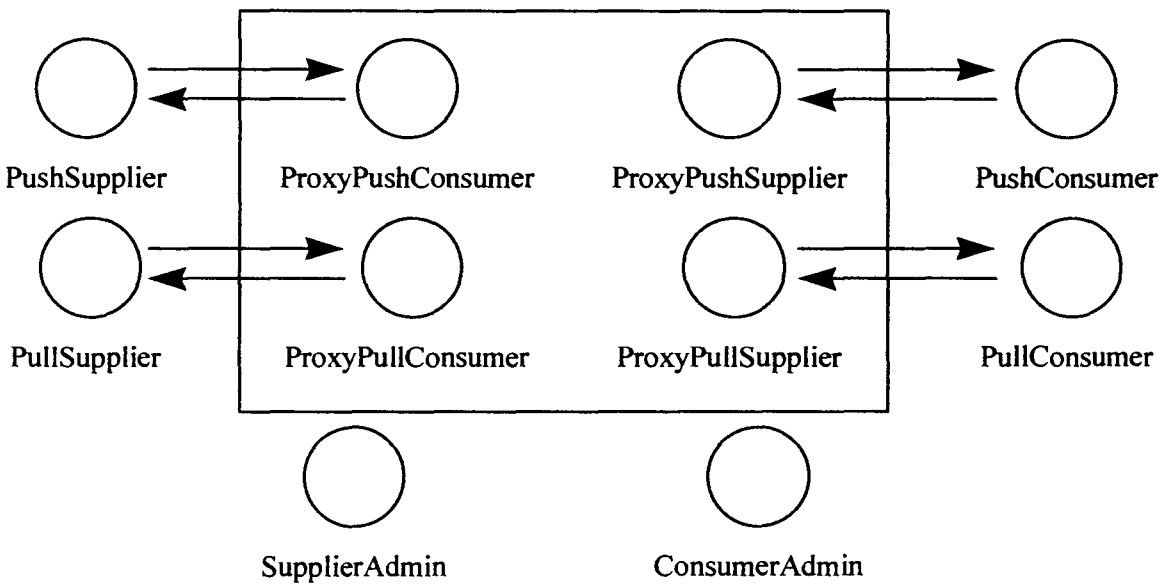

Figura 5.3 - Fornecedores e consumidores com um canal de eventos

Um canal de eventos é criado incrementalmente através da adição de fornecedores e consumidores. O primeiro passo para a conexão de um objeto consumidor ou fornecedor é obter uma referência de objetos para um objeto de administração de canal. A interface "Event_Channel" define três operações administrativas:

1. "ConsumerAdmin for_consumers()" - para a adição de consumidores;
2. "SupplierAdmin for_suplliers()" - para a adição de fornecedores; e
3. "destroy()" - para a destruição do canal.

Qualquer objeto que possua uma referência de objeto com a interface "Event_Channel" pode executar a operação "for_consumers()" para a conexão de consumidores no canal, retornando uma outra referência com a interface "ConsumerAdmin". A operação "for_suppliers()" permite a

conexão de fornecedores ao canal, retornando uma referência de objeto do tipo "SupplierAdmin". As interfaces "ConsumerAdmin" e "SupplierAdmin" definem objetos administrativos que retornam objetos "proxy" consumidores e fornecedores, permitindo que um canal se comporte como um fornecedor ou consumidor nos modelos "push" ou "pull". As interfaces em IDL são descritas a seguir:

```
interface ConsumerAdmin {
      ProxyPushSupplier obtain_push_supplier();
      ProxyPullSupplier obtain_pull_supplier();
};

interface SupplierAdmin {
      ProxyPushConsumer obtain_push_consumer();
      ProxyPullConsumer obtain_pull_consumer();
};
```

Portanto, um fornecedor no modelo "push" envia eventos para um canal que deve se comportar como um objeto consumidor "ProxyPushConsumer". Para isto ocorrer, o fornecedor deve cumprir os seguintes passos:

1. obter uma referência de objetos de um canal de eventos;
2. chamar a operação administrativa "for_suppliers()" para retornar um objeto do tipo "SupplierAdmin";
3. enviar uma requisição para o objeto administrativo para retornar o objeto "ProxyPushConsumer"; e
4. se conectar opcionalmente ao objeto "ProxyPushConsumer" do canal de eventos.

Um consumidor no modelo "push" deve implementar a interface "PushConsumer" com a operação "push" que será chamada pelo "ProxyPullConsumer" no canal de eventos. Para receber eventos, a aplicação consumidora deve:

1. obter uma referência de objetos de um canal de eventos;
2. chamar a operação administrativa "for_consumers()" para retornar um objeto do tipo "ConsumerAdmin";

3. enviar uma requisição para o objeto administrativo para retornar o objeto "ProxyPushSupplier"; e
4. se conectar obrigatoriamente ao objeto "ProxyPushSupplier" do canal de eventos.

Um fornecedor no modelo "pull" deve implementar a interface IDL "PullSupplier" com as operações "pull()", "try_pull()" e "disconnect_pull_supplier()" que serão chamadas pelo "ProxyPullConsumer" no canal de eventos. Para se conectar com o canal de eventos, o fornecedor deverá cumprir os seguintes passos:

1. obter uma referência de um canal de eventos;
2. chamar a operação administrativa "for_suppliers()" para retornar um objeto do tipo "SupplierAdmin";
3. enviar uma requisição para o objeto administrativo para retornar o objeto "ProxyPullConsumer"; e
4. se conectar obrigatoriamente ao objeto "ProxyPushConsumer" do canal de eventos.

Após estes quatro passos, o fornecedor está pronto para enviar eventos através das operações bloqueadora "try()" e não bloqueadora "try_pull().

Finalmente, um consumidor no modelo "pull" recebe eventos através das chamadas "pull" e "try_pull" do canal de eventos que se comporta como um fornecedor "proxy". Para isto ocorrer, é necessário que o consumidor:

1. obtenha uma referência de objetos de um canal de eventos;
2. chame a operação administrativa "for_consumers()" para retornar um objeto do tipo "ConsumerAdmin";
3. envie uma requisição para o objeto administrativo para retornar o objeto "ProxyPullSupplier"; e
4. se conecte opcionalmente ao objeto "ProxyPullSupplier" do canal de eventos.

Um canal de eventos pode ainda permitir a comunicação assíncrona entre fornecedores e consumidores de um mesmo modelo ou de diferentes modelos. Se os objetos consumidores e fornecedores são do mesmo modelo, esta combinação é conhecida como canônica. Se são de diferentes modelos, a combinação é denominada como híbrida. Na combinação híbrida de fornecedor "push" e consumidor "pull", os dois objetos são clientes do canal de eventos. Por

outro lado, na combinação de fornecedor "pull" e consumidor "push", o canal de eventos é cliente destes dois objetos, necessitando que implementem a correspondente interface em IDL (Figura 5.4).

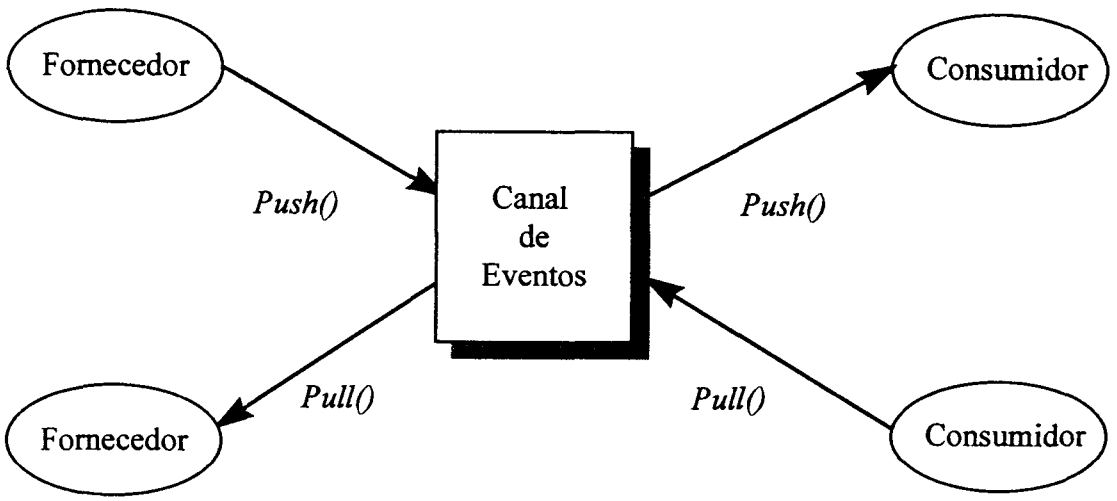

Figura 5.4 - Modelos de Comunicação Assíncrona com o Canal de Eventos

Análise do Serviço de Eventos

Como foi lido na seção anterior, o modelo assíncrono de comunicação é oferecido, opcionalmente e externamente a um ORB, pelo Serviço de Eventos, com o canal de eventos desempenhando um papel de mediador entre fornecedores e consumidores. As semânticas "push" e "pull" deixam livres os consumidores e fornecedores das semânticas restritivas do modelo síncrono de um ORB.

Particularmente para aplicações de gerência, onde um objeto gerenciado é potencialmente um fornecedor e as gerentes são consumidores de eventos, a especificação deste serviço também não é suficiente. Examinando a Função de Gerência de Relatório de Eventos do modelo OSI/CMIP, em que as notificações são processadas e a partir delas são formados relatórios de eventos, os quais são distribuídos pelos Discriminadores de Repasse de Eventos para os destinatários

previamente determinados, observa-se a necessidade de se adotar um modelo similar. Tal função tem como objetivos (BRISA, 1993):

- definir um serviço de controle de relatórios de eventos que permita selecionar os relatórios que devem ser enviados a um gerente em particular;
- determinar os destinatários para os quais os relatórios de eventos devem ser enviados;
- especificar um mecanismo para controlar o repasse de relatórios que permita, por exemplo, suspender e retomar a transmissão de tais relatórios;
- possibilitar que um sistema de gerência modifique as condições usadas na emissão de tais relatórios; e
- designar endereços alternativos para que os relatórios de eventos possam ser encaminhados em caso de indisponibilidade do endereço primário.

É importante ressaltar que o modelo OSI/CMIP define uma notificação como a informação emitida por um objeto gerenciado relativo a um evento interno ocorrido, formando os relatórios de eventos. Como a OMG usa o termo evento para expressar o mesmo conceito, será adotada a mesma terminologia. Os objetos gerenciados geram notificações que são encaminhadas ao módulo de Detecção e Processamento de Eventos. Tais notificações são processadas e a partir delas são formados os relatórios de eventos potenciais, os quais são distribuídos aos diversos Discriminadores de Repasse. Estes têm a função de selecionar quais eventos devem ser repassados a um destino particular. As condições a serem satisfeitas para que um relatório de evento em potencial possa ser enviado são especificadas através de atributos que atuam como um mecanismo de filtragem. Os seguintes atributos podem ser especificados:

- classe do objeto gerenciado;
- instância do objeto gerenciado;
- tipo de evento; e
- atributos de um dado tipo de evento.

Em um ambiente CORBA, é possível implementar uma função similar, usando apenas a especificação original do Serviço de Eventos. Os canais de eventos podem ser encadeados para a criação de uma cadeia de filtragem de eventos de forma que os consumidores somente recebam eventos que eles realmente estejam interessados. Contudo, esta solução não soa como adequada porque a cadeia de filtros aumenta o número de nós que uma mensagem precisa atravessar entre fornecedores e consumidores, significando um custo adicional, muitas vezes inaceitável. Caso um consumidor não esteja interessado em determinados tipos de eventos, deverá implementar uma rotina própria de filtragem para descartá-los, representando um desnecessário desperdício de recursos de comunicação e de processamento. A quantidade e a freqüência de geração de tais eventos são muito elevadas para serem processadas pelas aplicações gerentes que estão interessadas em apenas em um subconjunto dos eventos.

Portanto, existe a necessidade de se filtrar, combinar e correlacionar eventos primitivos em compostos. Este pré-processamento reduz consideravelmente o número de mensagens recebidas pelo gerente. O pré-processamento de eventos pode ser aplicado recursivamente de tal forma que eventos compostos são criados a partir de outros compostos. A função de filtragem, definida pelo Serviço de Notificações de TINA-C (Telecommunications Information Networking Architecture Consortium) (OMG, 1996a), é baseada no tipo do evento ou nos seus dados adicionais, como o tipo de objeto que gerou o seu evento, a hora em que o evento foi gerado e a identidade do objeto fornecedor. Em sistemas distribuídos orientados a objetos, os objetos que executam esse pré-processamento podem estar separados dos gerentes e dos recursos gerenciados, residindo provavelmente em localizações diferentes (Mansouri-Samani, 1994). Como o Serviço de Eventos adota o canal de eventos para a comunicação assíncrona indireta entre fornecedores e consumidores, o canal de eventos pode ser estendido para que incorpore mais esta funcionalidade de pré-processamento de eventos, por ser um objeto mediador presente entre objetos gerentes e gerenciados.

O modelo OSI/CMIP ainda especifica a Função de Controle de "Log" com o propósito de preservar informações em recursos modelados por "logs", sobre eventos que possam ter ocorrido ou sobre operações executadas (BRISA, 1993). O "log" é um repositório de registros de "logs" que contém informações que devem ser armazenadas. Essas informações são derivadas de

relatórios de eventos recebidos, de eventos internos ou de PDUs. O serviço de controle de "log" deve ser flexível de forma a permitir a seleção de registros que devem ser preservados e ser capaz de modificar os critérios usados na preservação destes registros. Um sistema externo deve ainda poder determinar se algum registro foi perdido, recuperá-lo e eliminá-lo, se necessário.

Examinando a funcionalidade OSI/CMIP de "log" e a especificação do Serviço de Eventos, verificamos que não existe nenhuma menção na padronização de um mecanismo similar que permita o armazenamento e a recuperação das informações dos consumidores e fornecedores registrados e dos eventos manipulados pelo canal de eventos. Considerando que o número de conexões e desconexões de consumidores e fornecedores é bem menor que a quantidade de eventos tratados pelo canal, as informações sobre os objetos registrados podem ser facilmente guardadas através de suas referências de objetos. Tais referências podem ser convertidas para um formato do tipo "string", através da operação CORBA "object_to_string()", no momento do registro dos fornecedores e consumidores, e armazenadas pelo próprio canal de eventos. Por outro lado, o armazenamento dos próprios eventos exige que o canal guarde uma cópia de cada evento recebido, até que seja repassado para os consumidores interessados. É necessário um protocolo confiável de comunicação em grupo entre o canal e seus consumidores. Contudo, este protocolo não é trivial de se implementar e o seu desempenho é significantemente menor quando comparado com o protocolo não-confiável IP Multiponto. Um outro problema relacionado com o armazenamento de eventos no canal é o fato de poderem ser do tipo genérico CORBA "any", podendo representar tanto um tipo comum das linguagens de programação, quanto um tipo definido pelo usuário. Alguns ORBs oferecem extensões proprietárias para o armazenamento e recuperação do tipo "any" na tentativa de solucionar este problema. Diversas funções de gerência necessitam preservar tais informações sobre eventos que possam ter ocorrido, podendo mudar essa necessidade de tempos em tempos.

Portanto, existe a necessidade de se estender a especificação do Serviço de Eventos CORBA e se verificar a sua adequação como suporte para a Facilidade de Monitorização, cujo modelo genérico será apresentado na seção seguinte. Os itens descritos abaixo resumem a nossa motivação em modificar tal serviço para que atenda às exigências de gerência:

- o Serviço de Eventos oferece diferentes modelos de transmissão de eventos, "push" e "pull", úteis para um grande espectro de aplicações. Contudo para aplicações de gerência, o modelo canônico "push" é o mais adequado para a comunicação assíncrona entre os objetos gerenciados, no papel de fornecedores de eventos, e os gerentes, no papel de consumidores, simplificando o processo de assinatura;

- as semânticas padronizadas do Canal de Eventos determinam que todos os eventos sejam repassados para todos os consumidores conectados. Portanto, cada gerente, no papel de consumidor, deverá implementar rotinas de filtragem para descartar eventos que não sejam de seu interesse. Tal interesse poderia ser especificado no momento de seu registro e assinatura no Canal de Eventos;

- os gerentes só tomam decisões após a chegada de múltiplos eventos de um conjunto determinado de objetos gerenciados, dependendo de um mecanismo adicional de correlação de eventos. A padronização não oferece tal facilidade no seu Canal de Eventos. Como no item anterior, os gerentes serão responsáveis por mais esta tarefa; e

- a falta de requisitos de persistência no Canal de Eventos, tanto de eventos quanto dos registros dos fornecedores e consumidores, reduz a sua possibilidade de emprego em um ambiente de gerência.

5.4 Modelo da Facilidade de Monitorização

Para se especificar um sistema de monitorização flexível, genérico e eficiente, é necessária uma abordagem modular. A arquitetura proposta, derivada do modelo ODP (Hoffner, 1994), é composta dos módulos de geração, observação, coleta, armazenamento, processamento e apresentação (Figura 5.5).

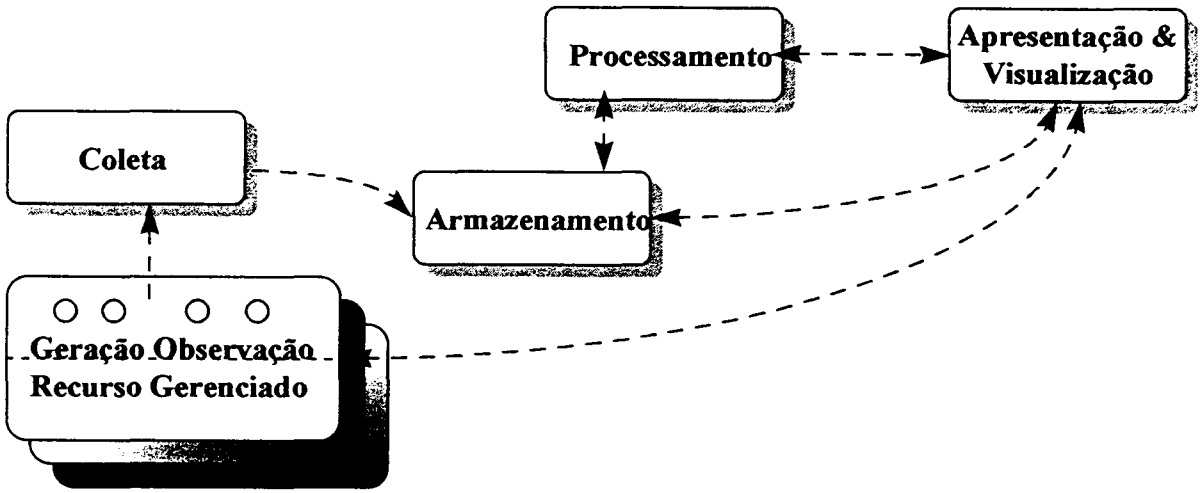

Figura 5.5 - Modelo de Monitorização

O módulo de geração é responsável pela aquisição de dados brutos de monitorização como resultado de um importante evento interno do recurso gerenciado. Esta atividade é conhecida em gerência como instrumentação. Usando o conceito da orientação a objeto, a instrumentação é desempenhada por objetos sensores especializados para a geração de dados de gerência, estando localizados no próprio recurso gerenciado. Existe um custo de processamento associado à instrumentação, devendo ser o menor possível, pois pode até mesmo afetar o desempenho funcional do recurso gerenciado. Este compromisso sugere uma maneira de se ativar e desativar a instrumentação de acordo com as necessidades da monitorização.

A observação oferece uma interface para que os dados instrumentados sejam acessados, podendo ainda oferecer um ponto de controle para a instrumentação, ativando-a ou desativando-a se necessário. Em um ambiente de gerência CORBA, tal interface será descrita em IDL com atributos que representem o estado do recurso gerenciado; e com operações que definem o seu comportamento de gerência. A definição de um atributo em IDL é logicamente equivalente à declaração de um par de métodos de acesso: um para leitura e outro para escrita. Permite ainda que o acesso remoto à interface de gerência do recurso gerenciado de forma transparente, tanto através dos "stubs" quanto pela Interface de Invocação Dinâmica.

O módulo de coleta adquire os dados de vários objetos gerenciados instrumentados, residentes em diferentes localizações, permitindo as atividades subseqüentes de armazenamento e processamento. O módulo de armazenamento se responsabiliza pela preservação e recuperação dos dados de gerência processados ou não. O módulo de processamento é necessário para transformar, combinar, correlacionar e filtrar diferentes dados de gerência, levando à atribuição de um novo significado aos dados originais. Este módulo permite que a quantidade de dados enviados para uma aplicação gerente seja reduzida, aumentando o valor semântico destes dados. Incentiva-se ainda que tal atividade também seja incorporada nos módulos anteriores, sem comprometer a sua funcionalidade original. Esta atividade é dependente da área funcional e do escopo da monitorização.

Os componentes do módulo de apresentação oferecem interfaces gráficas ao usuário para a visualização das informações coletadas e processadas. Dados observados dos recursos gerenciados podem ser diretamente apresentados sem nenhum tipo de processamento. Os recursos gráficos de Java, através do seu AWT (Abstract Windowing Toolkit), oferecem as facilidades necessárias para este módulo. Ainda mais, "Applets" em Java, embutidos em uma página HTML (Hypertext Markup Language) e disponíveis em um servidor WWW, permitem ao gerente navegar num sistema de gerência integrado, usando as ferramentas disponíveis, sem a necessidade de se instalar, na estação de gerência, o "software" correspondente de apresentação. Qualquer atualização em tal "software" só será implementada na cópia existente no servidor "Web". Os "applets" permitem ainda a visualização dos dados sobre os recursos gerenciados em qualquer ponto remoto da Internet, considerando as restrições e medidas de segurança necessárias. Em geral, a execução dos métodos de uma aplicação AWT requer métodos de inicialização, de desenhar, parar e destruir objetos gráficos.

Tal modelo descrito se mostrou suficientemente genérico tanto para a versão síncrona quanto para a assíncrona da Facilidade de Monitorização. Na primeira, estamos interessados em explorar o uso de um ORB como protocolo de gerência para o acesso a objetos gerenciados em um ambiente CORBA. Na variante assíncrona, investigamos o uso do canal de eventos de forma a incorporar os módulos de coleta, processamento e armazenamento. Embora esta abordagem

modular apresente módulos distintos de coleta, armazenamento e processamento, eles podem estar presentes em um único processo sem perda desta estruturação.

5.5 Facilidade de Monitorização Síncrona

A versão síncrona da Facilidade de Monitorização, em um ambiente de gerência CORBA, usa todo o potencial de comunicação oferecido por um ORB. As interações entre os gerentes e os recursos gerenciados serão basicamente através de requisições e respostas, usando o protocolo IIOP, desde que pelo menos os objetos gerenciados possuam uma interface de gerência em IDL.

Considerando o modelo descrito na última seção, os módulos de geração e observação residem na própria implementação de objetos CORBA. O módulo de apresentação é tipicamente um cliente CORBA, que envia requisições para os outros módulos e recebe as correspondentes respostas para serem visualizadas através de um recurso gráfico. Os módulos de coleta, de processamento e armazenamento se comportam tanto como clientes, enviando chamadas de métodos, quanto como servidores, recebendo outras. Este papel já foi denominado como "serviente" (Iona, 1995).

Inicialmente, deseja-se uma maneira de se obter dados de gerência das implementações de objetos instrumentadas. Esses dados são gerados por sensores que podem ser criados no próprio espaço de endereçamento dos processos cliente/servidor. Tais sensores são objetos especializados e incorporados unicamente para a aquisição de dados de interesse, evitando que não acrescentem um custo adicional muito alto. Dados dinâmicos são obtidos por sensores, como contadores e temporizadores, instanciados para adquirir uma determinada métrica. Tais sensores devem ser instalados em pontos de interesse, onde ocorram transições de eventos significativos. Um outro tipo de sensor, conhecido como composto, é definido para retornar atributos específicos como a identificação do recurso gerenciado, o seu estado operacional e outros. O primeiro passo em direção à implementação do nosso protótipo foi investigar uma maneira de se instrumentar as implementações de objetos CORBA.

Como a proposta deste trabalho é gerenciar aplicações distribuídas CORBA para as áreas funcionais de desempenho e contabilização, motivado por freqüentes questões surgidas durante a operação de um sistema distribuído, foi preciso definir métricas de desempenho. A maioria das métricas obtidas para a gerência de desempenho é também usada para contabilização (Hegering, 1994). Essas questões buscam identificar as possíveis causas de um excessivo tempo de resposta de uma aplicação e os usuários responsáveis pelo consumo de recursos em um ambiente heterogêneo como o da Multiware. O nosso foco principal está nas interações operacionais entre objetos nos papéis de cliente e de servidor. Portanto, as métricas de desempenho escolhidas basicamente foram tempo de resposta, vazão e utilização (Rolia, 1993), traduzidas para o ambiente CORBA. Tal tradução foi baseada na especificação, apresentada em (Friedrich, 1995b), para monitorização de aplicações em um ambiente DCE, exigindo o uso de contadores, temporizadores e sensores do tipo composto. Tais métricas podem ser obtidas tanto no espaço de endereçamento de um processo cliente quanto de um servidor. Do ponto de vista do cliente, as métricas definidas foram: o número de requisições enviadas, o tempo de empacotamento (marshalling) dos parâmetros pelos "stubs", o número de repostas recebidas, o tempo de desempacotamento (unmarshalling) dos parâmetros e o correspondente tempo de resposta. Da perspectiva do servidor, elas são: o número de requisições recebidas, o tempo de desempacotamento dos parâmetros, o tempo de residência local de processamento, o número de respostas enviadas, o tempo de empacotamento dos parâmetros, a utilização e a vazão. A instrumentação calcula ainda dados estatísticos como os valores máximos e mínimos, o somatório simples e o somatório dos quadrados para computar a média e a variância da amostra durante um intervalo de tempo. Os serviços oferecidos pelos elementos da aplicação distribuída devem ser monitorados e caracterizados separadamente. Outras informações são também necessárias para os propósitos de desempenho e contabilização. Esses dados podem ser visualizados na Tabela 5.1, onde se pode verificar os métodos usados para a instrumentação de clientes e servidores em um ambiente CORBA. Como se pode notar, apenas o método "get_principal()" é disponível na especificação da arquitetura CORBA, as demais são específicas da plataforma Orbix (Iona, 1995).

Tabela 5.1 - Atributos da Interface

Dados	Método	Classe	CORBA/Orbix
nome da operação	operation()	CORBA::Request	Orbix
nome da implementação	myImplementationname(); myServer()	CORBA::BOA; CORBA::ORB	Orbix
identificador	myMarkerName()	CORBA::BOA	Orbix
modo de ativação	myactivationmode()	CORBA::BOA	Orbix
máquina hóspede	myHost()	CORBA::ORB	Orbix
usuário cliente	get_principal()	CORBA::BOA	CORBA
pid	getpid()	"Unix system call"	--

Tais atributos foram especificados em uma interface IDL "Server_Managed_Object", mostrada logo abaixo, para que um componente servidor de uma aplicação distribuída CORBA a implemente. A interface apresenta uma estrutura "Instrument_Data" com os atributos e uma operação "listInstrument_Data()" para retornar a estrutura. Tal interface é conceitualmente equivalente a uma MIB SNMP/CMIP, gozando de todas as vantagens que CORBA oferece para gerência.

```
interface Server_Managed_Object {
     struct Instrument_Data {
          enum      state      {idle,      unmarshalling_request,      processing,
     marshalling_response, committing,};
          readonly attribute string operation_name;
          readonly attribute string implementation_name;
          readonly attribute string marker_name;
          readonly attribute string activation_mode;
          readonly attribute string hostname;
          readonly attribute string principal;
          readonly attribute long number_of_request;
          readonly attribute double unmarshalling_time;
          readonly attribute long number_of_response;
          readonly attribute double marshalling_time;
          readonly attribute double throughput;
          readonly attribute double utilization;
          readonly attribute double residence_time;
```

```
    };
    void listInstrument_Data(out Instrument_Data);
};
```

A Figura 5.6 mostra o esquema estático do ponto de vista ODP de informação, usado para especificar a versão síncrona da Facilidade de Monitorização, apresentando as hierarquias de classe, de acordo com a notação gráfica da Técnica de Modelagem de Rumbaugh (Rumbaugh, 1991). Como pode ser observado, a Facilidade é composta pelas classes "Componentes_Aplicação", "Coletor", "Processamento" e "Apresentação". Neste modelo, a classe "Processamento" faz os cálculos estatísticos, através de suas operações, sobre os dados coletados pela classe "Coletor". Os componentes da aplicação são um agregado de classes "Cliente" e "Servidor". Estas duas são constituídas por classes "Sensor" e "Observação". A classe "Coletor" está associada a um ou mais objetos da classe "Observação". Estas classes são relacionadas através de uma associação de um para muitos. A classe "Sensor" possui métodos que os ativam e desativam, permitindo que sejam usados quando necessário. Da mesma forma, a classe "Observação" tem métodos para a ativação e desativação da instrumentação no componente da aplicação. Através do mecanismo de herança, um sensor pode ser especializado em "Temporizador", "Contador" e "Composto", com métodos específicos para cada tipo de instrumentação. Da mesma forma, a classe "Processamento" está associada a um ou mais objetos da classe "Coletor". A classe "Apresentação" é povoada por classes especializadas "Gráficos" e "Diálogos"; e associada à classe "Processamento".

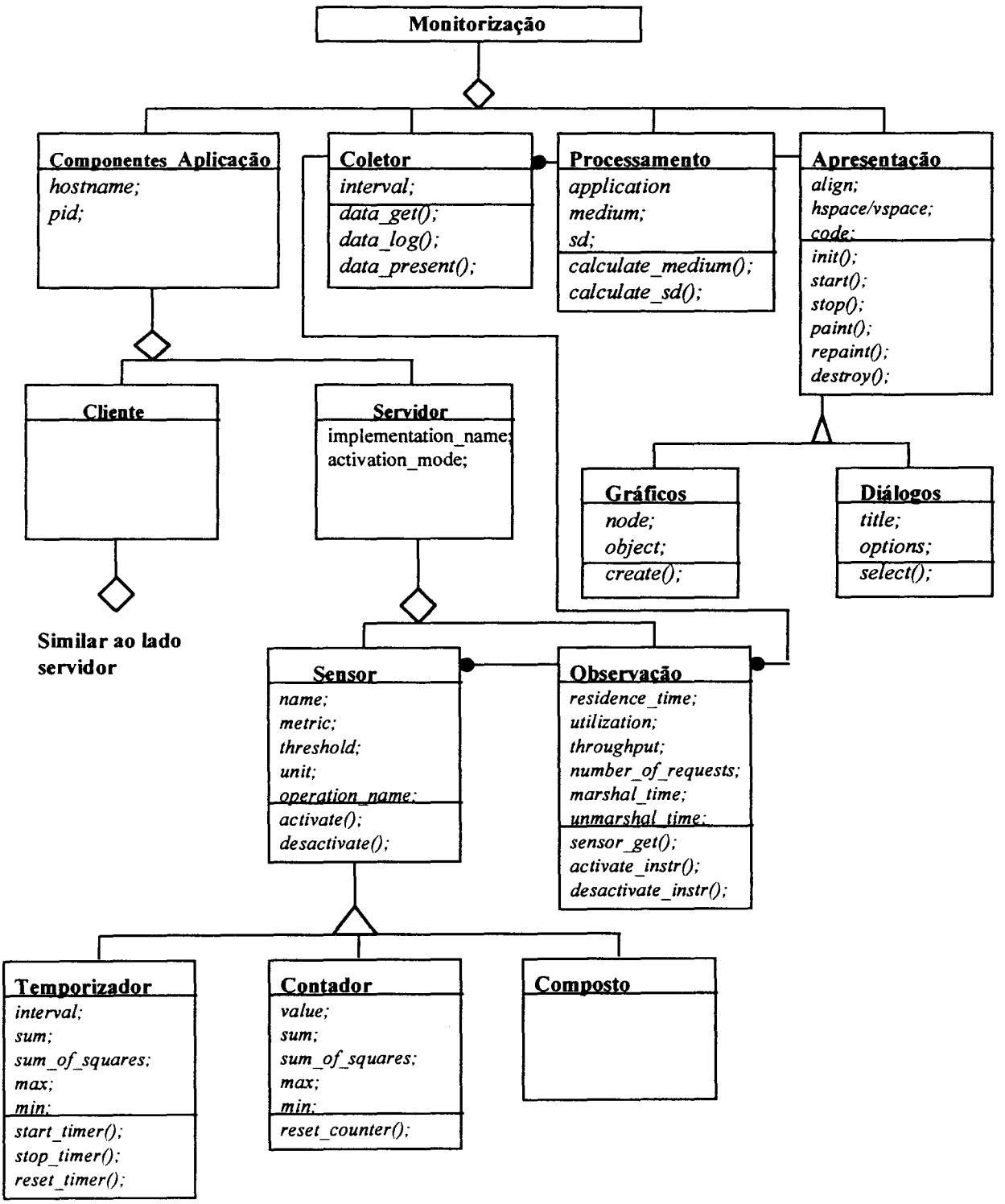

Figura 5.6 - Esquema Estático

5.6 Facilidade de Monitorização Assíncrona

Enquanto na seção anterior exploramos o uso de um ORB como protocolo de gerência para a monitorização dos componentes das aplicações distribuídas, esta parte apresenta a versão assíncrona da Facilidade de Monitorização. As interações entre os gerentes e os recursos gerenciados serão basicamente através de eventos. A proposta é estender o canal de eventos de forma a incorporar os módulos de coleta, processamento e armazenamento.

Considerando o modelo descrito na seção 5.4, os módulos de geração e observação continuam a residir na própria implementação de objetos CORBA. Serão fornecedores de eventos. O módulo de apresentação desempenhará o papel de consumidor de eventos. Os módulos de coleta, processamento e armazenamento são incorporados ao canal de eventos, comportando-se tanto como consumidor quanto como fornecedor. Contudo, esta incorporação requer algumas mudanças nominais para que a arquitetura final do canal de eventos seja compatível com a descrição da especificação do Serviço de Eventos. Portanto, o foco principal desta dissertação está sobre esta extensão.

Neste modelo, o canal de eventos estendido desempenha o mesmo papel de mediador do Serviço de Eventos especificado. Adotamos apenas o modelo "push" por satisfazer ao paradigma de gerência em que os objetos gerenciados enviam notificações de forma ativa para os gerentes. Externamente, o canal de eventos oferece duas interfaces administrativas "ConsumerAdmin" e "SupplierAdmin" que permitem a adição de fornecedores e consumidores a um canal. Os métodos dessas interfaces retornam objetos "proxy" consumidores (ProxyPushConsumer) e fornecedores (ProxyPushSuppliers), permitindo que um canal se comporte como um consumidor ou fornecedor no modelo "push". Internamente, o Canal é compreendido por vários módulos que encapsulam tarefas independentes (Figura 5.7):

- Módulo "ProxyPushConsumer";
- Módulo "ProxyPushSupplier";
- Módulo de Assinatura e Filtragem;

- Módulo de Armazenamento; e
- Módulo de Correlação de Eventos.

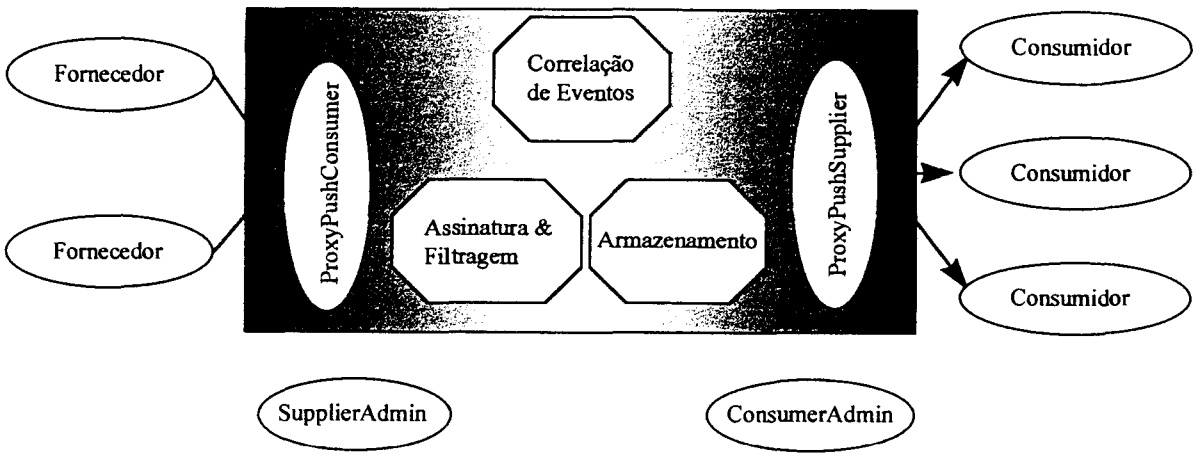

Figura 5.7 - Extensão do Canal de Eventos

A interface do Módulo "ProxyPushConsumer" é similar à interface de mesmo nome do Serviço de Eventos na qual os fornecedores usam para a conexão com o Canal de Eventos. Contudo, é necessário estender a interface "ProxyPushConsumer" de modo que os fornecedores possam especificar os tipos de eventos que eles geram. Com esta informação, o módulo de Assinatura e Filtragem pode compor estruturas de dados para a consulta pelos consumidores registrados. De forma similar, a interface "ProxyPushSupplier" é também estendida para que os consumidores possam especificar os tipos que desejam receber.

O módulo de Assinatura e Filtragem estende as interfaces originais para permitir o registro de consumidores interessados em um subconjunto de eventos. Este módulo permite que uma aplicação gerente (consumidora) possa se registrar e determinar que tipo de informação deseja receber através de critérios estabelecidos. Ele dispõe de filtros e funções que selecionam as notificações que devem ser repassadas para os consumidores que previamente se registram para receber determinados tipos de eventos. Tanto a filtragem quanto a correlação de eventos requer um sistema bem definido de tipos de eventos que estabeleça a identidade do remetente, o seu tipo, o dado associado e a hora em que o evento foi gerado. A solução adotada consiste na

definição em IDL de um módulo com uma estrutura para expressar a mesma semântica, descrita logo abaixo.

```
module EventTypes    {
          typedef CosNaming::NameComponent name;
          typedef CosTime::UTCtime event_time;
          typedef string event_type;
          struct Event    {
               event_type type;
               name origin;
               object objref;
               event_time time_of_occurrence;
               };
     };
```

De forma similar, porém ortogonal, (Schade, 1997) propõe uma classe abstrata em CORBA IDL a partir da qual todos os eventos são derivados. A sua solução não é adequada pelo fato de que CORBA atualmente não suporta a passagem de objetos por valor.

O módulo de armazenamento é flexível de forma a permitir a seleção dos registros, suspender ou retornar a atividade de "logging", modificar os critérios na preservação dos eventos e recuperar notificações preservadas. O módulo de armazenamento permite ainda que os eventos de um fornecedor sejam armazenados seqüencialmente antes de enviá-los para os consumidores.

Embora a filtragem seja considerada um tipo de correlação de eventos (Meira, 1997), a proposta inclui um outro módulo de Correlação de Eventos que implementa mecanismos que interpretam e analisam os eventos, levando em consideração um conjunto de critérios preestabelecidos, ou definidos dinamicamente em função do processo de gerência. Tais mecanismos permitem a especificação das condições que devem ser satisfeitas pelos eventos, antes que sejam repassados a um consumidor particular. Deseja-se reduzir a quantidade de eventos transferidos ao consumidor, aumentando o conteúdo semântico dos eventos resultantes ou compostos. A proposta é adotar um conjunto de operadores que atuam sobre eventos primitivos, como operandos. Essas expressões permitem a criação de eventos compostos com maior conteúdo semântico, da mesma forma que Mansouri-Samani apresentou em (Mansouri-Samani, 1997). Tais operadores podem ser vistos na Tabela 5.2. Os operadores &, ! e + podem ser considerados como básicos.

Tabela 5.2 - Operadores de correlação de eventos

Operador	Significado	Exemplo
&	AND lógico	e1 & e2
|	OU lógico	e1 | e2
;	seqüência	e1 ; e2: e1 seguido por e2
!	exclusão	e1 ; e2 ! e3: e1 seguido por e2 sem e3
+ <período de tempo>	retardo	e1 + <100>: define um retardo para o evento ser repassado para os seus consumidores

Infelizmente, não é possível assumir que a ordem em que os eventos chegam a um canal de eventos é a mesma que eles foram enviados. Portanto, é preciso reordená-los para realmente se ter um quadro consistente do comportamento dos recursos monitorados. Mansouri-Samani considerou como pré-requisito a existência de um relógio global sincronizado. Um dos elementos da estrutura "Events", descrita anteriormente, traz esse dado de tempo, de acordo com a especificação do Serviço de Tempo CORBA, permitindo que os eventos sejam comparados.

A precedência de operadores determina a ordem na qual uma expressão, com mais de um operador, é avaliada. Em alguns casos, pode determinar o resultado da combinação dos operadores. Operadores de mais alta precedência são avaliados primeiro. Operadores de mesma precedência são avaliados da esquerda para direita, baseado na ordem em que aparecem na expressão. Pode-se ainda alterar a ordem na qual as expressões são avaliadas, colocando aquelas que se quer avaliar primeiro entre parênteses, sendo uma medida muito útil onde a precedência não é muito clara e tornado a sua leitura mais fácil. A Tabela 5.3 mostra uma proposta de precedência dos operadores, classificados na seqüência descendente. Por exemplo, na expressão "e1 + 100 & e2 | e3", o operador de retardo é avaliado primeiro do que o lógico. Por outro lado, em "e4 + 100 & (e5 | e6)", o operador entre parênteses é considerado antes do que os outros.

Tabela 5.3 - Precedência de Operadores

Operador	Observações
()	parênteses
+	retardo

;	seqüência
!	exclusão
&	AND
|	OR

5.7 Conclusão

Este capítulo apresentou os principais aspectos considerados na modelagem e projeto da Facilidade de Monitorização, discutindo os recursos oferecidos pela arquitetura CORBA. O uso de um ORB como plataforma de gerência resulta em uma arquitetura mais simples, onde as entidades gerentes e objetos gerenciados com interfaces em IDL continuam a desempenhar o mesmo papel, porém já não é mais necessário o agente. Por outro lado, uma invocação de um método remoto implica na execução síncrona de uma operação por um objeto CORBA, onde tanto o cliente quanto o servidor têm que estar obrigatoriamente presentes. As semânticas do Serviço de Eventos com o seu canal de eventos deixam livres os objetos CORBA das restrições do modelo síncrono de uma plataforma ORB.

Contudo, a especificação de tal serviço necessita ser estendida para que atenda aos requisitos de gerência. O modelo canônico "push" é o mais adequado para a comunicação assíncrona entre os objetos gerenciados, no papel de fornecedores de eventos, e os gerentes, no papel de consumidores, simplificando o processo de assinatura. As semânticas padronizadas determinam que todos os eventos sejam repassados para todos os consumidores conectados, não oferecendo nenhum dispositivo que permita que os consumidores registrem o interesse em um subconjunto de eventos. Não existe também nenhum mecanismo adicional de correlação de eventos que aumente o seu conteúdo semântico. A ausência de requisitos de persistência no canal de eventos diminui ainda mais sua possibilidade de emprego em um ambiente de gerência.

O modelo da Facilidade de Monitorização proposto é composto pelos módulos de geração, observação, coleta, armazenamento, processamento e apresentação. Este modelo se mostrou suficientemente genérico tanto para a versão síncrona quanto para a assíncrona. Na primeira variante, os módulos de geração e de observação se localizam na própria implementação de objetos CORBA, enquanto o módulo de apresentação é tipicamente um cliente. Os módulos de

coleta, de processamento e armazenamento se comportam tanto como clientes, enviando chamadas de métodos, quanto como servidores, recebendo outras. Na versão assíncrona, os módulos de coleta, processamento e armazenamento são incorporados ao canal de eventos, comportando-se tanto como consumidor quanto como fornecedor. Para atender estes requisitos e ser compatível com a especificação do Serviço de Eventos, o canal estendido adotou internamente as funcionalidades de assinatura e filtragem, de armazenamento e de correlação de eventos. Os aspectos de implementação serão discutidos no próximo capítulo.

Capítulo 6

Aspectos de Implementação

Este capítulo trata dos principais aspectos de implementação do modelo descrito anteriormente e expõe alguns resultados obtidos, além de apresentar algumas características da implementação de um ORB e de um Serviço de Eventos que influenciaram algumas decisões nesta fase.

6.1 Introdução

Atualmente, a plataforma Multiware está sendo desenvolvida em estações IBM RS/6000 e Sun Sparc, com os sistemas operacionais AIX, SunOS e Solaris, conectados por redes "Ethernet", "Fast Ethernet" e FDDI (Fiber Distributed Data Interface). Este ambiente está disperso em dois

laboratórios, inter-conectados por um enlace de 10 Mbits. Usa a Orbix (Iona, 1995) e OrbixWeb (Iona, 1997), com os respectivos mapeamentos de IDL para C++ e Java, como implementações de um ORB. Portanto, a Facilidade de Monitorização deve ser portável e interoperável em tais sistemas heterogêneos.

Este capítulo está estruturado da seguinte forma: a próxima seção faz uma descrição das características das plataformas Orbix e OrbixWeb, enquanto a Seção 6.3 apresenta os principais aspectos de implementação da versão síncrona da Facilidade de Monitorização. A seguir, é descrito o serviço de eventos OrbixTalk da Iona na Seção 6.4. A Seção 6.5 analisa a implementação dos módulos que estendem o canal de eventos, usando os recursos do OrbixTalk. Na Seção 6.6, são apresentados os principais trabalhos relacionados. Finalmente, são discutidas algumas conclusões na Seção 6.7.

6.2 Plataforma Orbix

CORBA oferece uma infra-estrutura de comunicação que permite que os objetos de uma aplicação distribuída sejam vistos como componentes heterogêneos de "software", conectados a um barramento padrão de comunicação e identificados por uma referência única de objetos, desde que suas interfaces obedeçam à especificação da IDL. As plataformas Orbix (Iona, 1995) e OrbixWeb (Iona, 1997), com os respectivos mapeamentos de IDL para C++ e Java, são as duas implementações de um ORB usadas neste trabalho. Embora elas tenham mapeamentos distintos, grande parte desta descrição é válida para as duas.

Nos dois ORBs, um objeto CORBA, com a sua interface declarada em IDL, deverá ser implementado por uma classe nativa de uma das linguagens. Esta classe deverá implementar os métodos correspondentes a cada um dos atributos e operações especificados na interface IDL. Instâncias dessas classes podem ser acessadas de qualquer nó, onde esteja configurada uma das plataformas. Esses objetos residem em processos servidores Orbix em uma máquina hóspede. Um servidor pode ter coletivamente diferentes objetos CORBA com interfaces IDL distintas. Um servidor pode ainda ter objetos auxiliares, que não tenham a interface IDL e implementados

em C++ ou em Java, sendo invisíveis aos clientes remotos. O termo servidor, usado em (Iona, 1995; Iona, 1997), não pode ser confundido com um dos dois papéis (cliente/servidor) que um objeto CORBA pode desempenhar. Conceitualmente, um servidor Orbix corresponde à definição de uma implementação de objetos CORBA.

Tipicamente, as duas plataformas consistem de um compilador IDL, responsável pelo respectivo mapeamento para uma das linguagens, de um conjunto de bibliotecas em C++ ou em Java que serão incluídas nos componentes das aplicações, e de processos permanentemente ativos que auxiliam a conexão remota destes componentes. Tais processos são conhecidos como "orbixd" e são responsáveis por localizar e ativar, se necessário, os processos servidores. Residem obrigatoriamente nas máquinas onde se queira que os componentes servidores das aplicações sejam ativados. São freqüentemente comparados ao processo UNIX "inetd" que permite a localização e ativação de processos servidores TCP/IP, como Telnet e FTP, em portas de comunicação conhecidas. Um cliente tem acesso transparente aos objetos CORBA, conhecendo apenas a sua interface IDL, através dos "stubs" e esqueletos. (Figura 6.1).

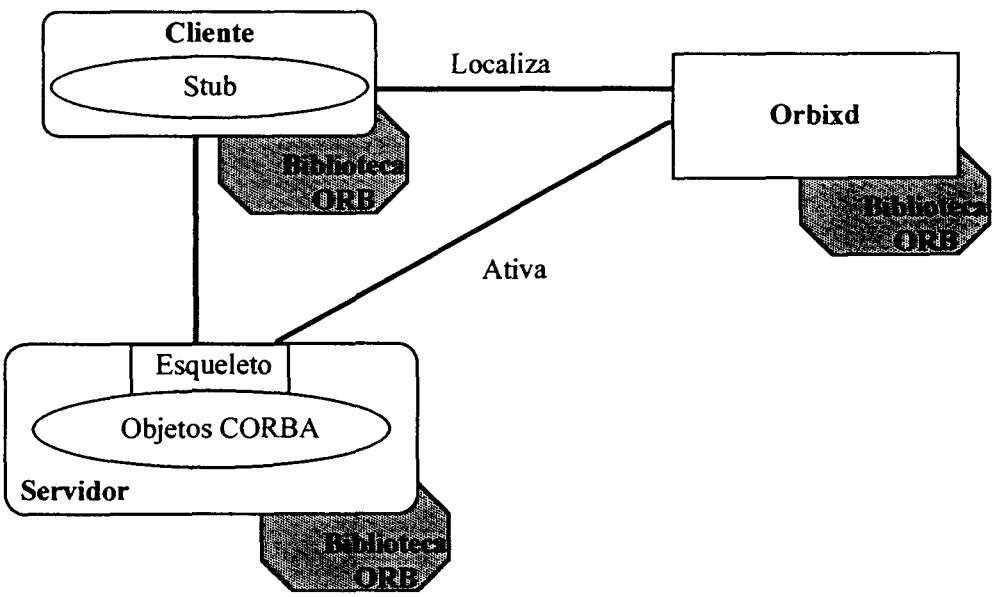

Figura 6.1 - Componentes da Orbix e OrbixWeb

Em Orbix, uma referência de objetos identifica, de forma única, um objeto remoto em um espaço de endereçamento diferente do cliente, seja em um outro nó ou na própria máquina. A referência de objeto é composta pelo:

- identificador único do objeto, conhecido como "marker";
- o nome da implementação de objetos; e
- o nome da máquina onde está sendo executada a respectiva implementação de objetos.

Um cliente pode se conectar a um objeto CORBA, especificando apenas o seu identificador, através do método proprietário "bind()" que retorna a referência de objetos para ser usada na invocação de métodos remotos, usando a sintaxe normal de chamada da linguagem. A localização física do objeto é transparente para o programador. No espaço do cliente, a referência de objeto é visto como um objeto "proxy" para o outro remoto. Todas as requisições são enviadas através deste "proxy", sem o conhecimento de sua existência. Os objetos "proxy" executam o código gerado automaticamente pela compilação do arquivo com a especificação da interface IDL. O único procedimento que um desenvolvedor deverá tomar será a inclusão deste código, disponível em um ou mais arquivos, no cliente. Caso ele não tenha acesso a estes arquivos em tempo de compilação, poderá ainda montar uma requisição em tempo de execução, usando as chamadas de métodos da Interface de Invocação Dinâmica e do Repositório de Interfaces. Esta requisição dinâmica é criada através das operações "invoke()" e "get_response()" da interface CORBA "Request", mapeada para uma classe de mesmo nome em uma das duas linguagens.

O Repositório de Implementações da Orbix permite o registro de servidores com um ou mais objetos CORBA, associando o seu nome registrado com a classe que o implementa e permitindo o uso de um dos modos de ativação. Os servidores registrados podem ser ativados em três modos. No modo compartilhado, é alocado apenas um processo para cada servidor registrado. Este processo é ativado pelo "orbixd" para a execução do código de um método do objeto CORBA residente no servidor. Permanece ativo ou é desativado, caso não receba nenhuma outra invocação durante um período de tempo pré-determinado. No modo não compartilhado, é ativado um processo diferente para cada objeto CORBA chamado. No terceiro modo, conhecido como chamada por método, a Orbix ativa um processo para cada chamada individual de método

de um objeto CORBA. O registro de um servidor no Repositório de Implementações é feito através do utilitário "putit", criando um arquivo nomeado com a concatenação da designação do servidor e do sufixo "imp". Tal arquivo contém os dados necessários para a ativação do servidor em um dos modos especificados com o utilitário.

Os seguintes passos são necessários para se escrever uma aplicação cliente/servidor, usando a Orbix:

1. definir as interfaces dos objetos CORBA em IDL;
2. compilar essas interfaces;
3. implementar essas interfaces com as classes na respectiva linguagem, inserindo código para as operações definidas originalmente em IDL. Nesta fase, podem ser criadas outras classes ou usar bibliotecas existentes na linguagem nativa para auxiliar o seu desenvolvimento;
4. escrever uma classe servidora que cria instâncias dessas classes;
5. registrar o nome desta classe servidora no Repositório de Implementação, usando o utilitário "putit"; e
6. escrever o código do cliente, usando o método "bind()", criando instâncias dos objetos CORBA e fazendo chamadas remotas, da mesma forma que faria para se ter acesso a um objeto local.

Os arquivos com a especificação IDL das interfaces dos objetos CORBA devem ser compilados, tanto para verificar a sua sintaxe quanto para mapeá-la para uma das linguagens de programação. O compilador IDL gera um conjunto de arquivos com código e estrutura de dados para permitir o acesso de um cliente ao respectivo objeto, e um outro conjunto para ser usado na sua implementação pelo programador. O exame de um destes arquivos permite que se verifique que cada atributo da interface IDL é mapeado para um método de leitura e para um outro de escrita, enquanto cada operação IDL é traduzida diretamente para um método em C++ ou em Java. Todos estes métodos tem suporte para tratamento de exceções, permitindo que sejam capturados os erros de comunicação com os objetos distribuídos. Pode ainda declarar novas exceções ou ainda estender as disponíveis. O compilador ainda gera código de classes para os dois modos de implementação de um objeto CORBA, sendo escolhida apenas uma delas. A escolha dos modos "BOAImpl" e "Tie" afetam o desenvolvimento dos componentes das aplicações, pois o primeiro

usa o mecanismo de herança para implementação dos servidores, precisando conhecer os detalhes da respectiva classe. Por este motivo, é preferível a escolha do segundo modo de forma direta.

É necessário ainda escrever uma classe servidora com poucas linhas de código para instanciar os objetos CORBA e fazer uma chamada à operação "impl_is_ready()", indicando a sua disponibilidade na rede, dentro do escopo do método "main()". O código do processo cliente é desenvolvido, necessitando apenas da descrição da interface IDL do objeto e de alguns dos arquivos gerados pelo compilador. É caracterizado pela presença do método "bind() que cria um objeto "proxy" do tipo da classe do servidor e que o liga à implementação de forma transparente. Caso não tenha acesso a estes arquivos, poderá ainda consultar o Repositório de Interfaces e usar o recurso da Interface de Invocação Dinâmica para montar uma requisição ao objeto desejado.

6.3 Aspectos de Implementação da Facilidade de Monitorização Síncrona

Nesta fase de implementação, estávamos interessados principalmente em conceber uma forma de instrumentar os componentes de uma aplicação distribuída para a coleta de métricas de desempenho e de contabilização, dispondo-as em uma interface IDL. Esta solução deveria ainda respeitar o conceito de encapsulação que protege o estado interno dos objetos de uma aplicação, além de não sobrecarregar estes componentes, afetando o seu desempenho original. O estudo da plataforma Orbix, originalmente adotada pela "Multiware" revelou que os filtros de processos, existentes nesta implementação, poderiam ser úteis para implementar os objetos sensores da instrumentação.

Os filtros de processos, disponíveis na antiga Orbix1.3, na atual Orbix2.1 e na recente OrbixWeb2.0 com o mapeamento IDL/Java, permitem que um código adicional seja executado antes ou depois do código normal de uma operação chamada. Sugere-se ainda que eles possam ser usados como mecanismos de auxílio para autenticação, para criação de múltiplas linhas de execução (lightweight thread) e para monitorização (Iona, 1995). São capazes de monitorar todas

as requisições e respostas no espaço de endereçamento de um cliente ou de um servidor. Um filtro no lado cliente pode ainda adicionar dados para o "buffer" de saída de uma requisição, de tal modo que possam ser recuperados no filtro correspondente do lado servidor. Similarmente, um filtro pode adicionar dados a uma reposta. Tais filtros se mostraram capazes de desempenhar o comportamento dos sensores, especificados no capítulo anterior, podendo ainda ser instalados em cadeia. Tais mecanismos, não previstos na especificação CORBA, têm a característica modular exigida pela instrumentação.

Dessa forma, os sensores implementados são derivados da superclasse "CORBA::Filter", permitindo a instrumentação em dois pontos de uma requisição e em outros dois pontos de uma reposta, antes e depois do empacotamento (marshalling) e desempacotamento (unmarshalling) dos parâmetros, tanto no espaço do processo cliente quanto do servidor. Foram estendidos os métodos originais da classe nestes pontos para a aquisição das métricas especificadas. Como pode ser observado na Figura 6.2, os sensores implementados calculam as métricas definidas do ponto de vista do cliente: o número de requisições enviadas, o tempo de empacotamento dos parâmetros pelos "stubs", o número de repostas recebidas, o tempo de desempacotamento dos parâmetros e o tempo de resposta. Da perspectiva do servidor, podem medir o número de requisições recebidas, o tempo de desempacotamento dos parâmetros, o tempo de residência, o número de respostas enviadas, o tempo de empacotamento dos parâmetros, a utilização e a vazão.

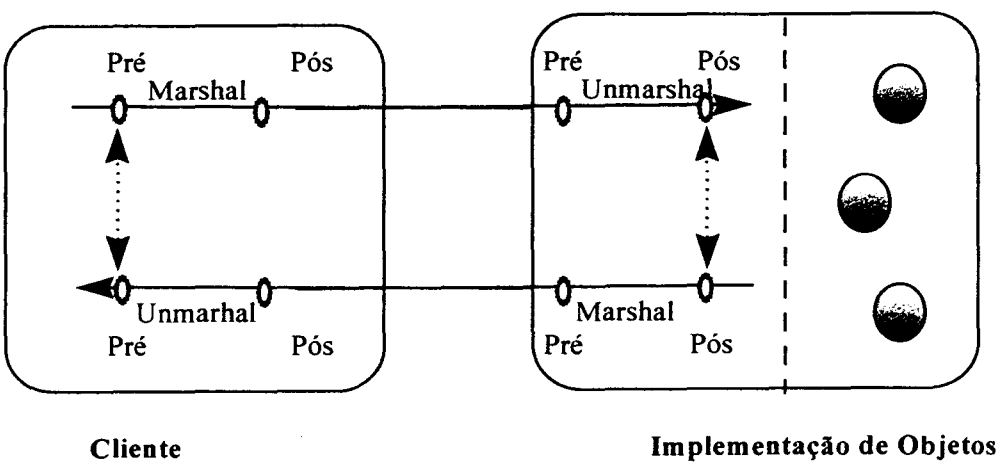

Figura 6.2 - Pontos de monitorização dos filtros como sensores

Cada métrica desta pode ser computada através da inserção de código específico nos métodos originais dos filtros:

- "outRequestPreMarshal()" - ação disponível nas requisições enviadas antes do empacotamento dos dados;
- "outRequestPostMarshal()" - ação disponível nas requisições enviadas depois do empacotamento dos dados;
- "inRequestPreMarshal()" - ação disponível nas requisições recebidas antes do desempacotamento dos dados;
- "inRequestPostMarshal()" - ação disponível nas requisições recebidas depois do desempacotamento dos dados;
- "outReplyPreMarshal()" - ação disponível nas respostas enviadas antes do empacotamento dos dados;
- "outReplyPostMarshal()" - ação disponível nas respostas enviadas depois do empacotamento dos dados;
- "inReplyPreMarshal()" - ação disponível nas respostas recebidas antes do desempacotamento dos dados; e
- "outReplyPostMarhal()" - ação disponível nas respostas recebidas depois do desempacotamento dos dados.

Cada um destes métodos acima tem como parâmetro um objeto da classe "CORBA::Request", permitindo que os detalhes de uma requisição sejam monitorados como o objeto CORBA alvo da invocação, através de "Request::target()", e o nome do método chamado através de "Request::operation()". O construtor "Filter()" da classe "CORBA::Filter", ao ser chamado, adiciona na cadeia de filtros de processos o objeto recém criado. Instâncias de "CORBA::Filter" não podem ser criadas diretamente, pois o método construtor não permite, necessitando que seja especializada por uma subclasse.

No lado servidor, cinco sensores foram instanciados, três como temporizadores, um como contador e um outro como composto para retornar dados adicionais. Um temporizador foi instalado para medir o tempo de resposta do servidor e os outros dois para calcular os tempos de empacotamento e desempacotamento. Foi ainda definida uma classe auxiliar, com a funcionalidade de um relógio, para oferecer um serviço local de tempo. Da mesma forma, cinco sensores foram instalados em cadeia no lado cliente: três temporizadores, um contador e um composto (Figura 6.3).

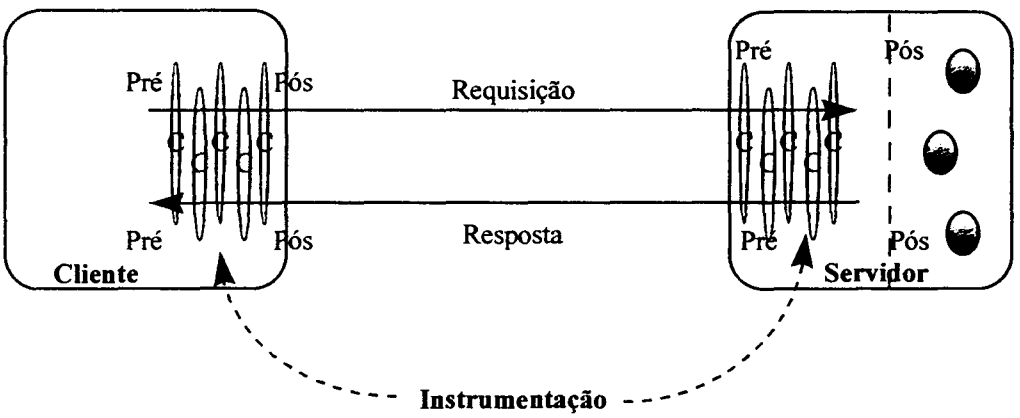

Figura 6.3 - Cadeia de sensores

O tempo de resposta de um cliente é calculado por um temporizador, instalado no seu espaço de endereçamento, com o disparo do relógio por uma requisição enviada, sendo parado pela resposta esperada do servidor. No próprio escopo do temporizador, já são calculados o somatório e quadrado dos somatórios para o cálculo dos valores médios e desvio padrão de uma amostra. O código em C++ desse temporizador é mostrado a seguir:

```
// este sensor mede o tempo resposta do cliente
class Response_timer :: Filter {
//   instanciação do relógio
     Clock* cl1;
// estrutura de dados
     Instrument_Data i_d;
public:
//  uma requisição recebida dispara o relógio
```

```
     unsigned char outRequestPreMarshal (CORBA::Request &r, Environment &) {
          cl1.start_timer();
          return 1;
     };
//   uma resposta enviada pára o relógio
     unsigned char inResponsePosMarshal (CORBA::Request &r,  Environment &) {
          cl1.stop_timer();
          i_d.residence_time = cl1.diff_time;
          i_d.sum_residence_time += i_d.residence_time;
          i_d.sum_of_sq_residence_time += i_d.residence_time**2;
          delete[] cl1;
     return 1;
};
}
```

Inicialmente, um temporizador foi instanciado no lado cliente para calcular o tempo de resposta entre uma requisição enviada e uma resposta recebida, visando controlar o impacto da instrumentação inserida no servidor. Essa medida foi muito útil para medir o custo adicional da instrumentação em todas as fases de sua implementação. Verificamos que o impacto da instalação dos filtros é proporcional ao custo do código incluído na instrumentação. A característica modular de poder instanciá-los em cadeia permite o controle deste custo. Nós ainda notamos que os maiores tempos de resposta são de requisições para o "orbixd" que manipula as referências de objetos CORBA. Contudo, tais valores são dependentes de fatores estáticos e dinâmicos dos nós, de seus sistemas operacionais e da própria rede de comunicação. A monitorização em um ambiente, como o da Multiware, permitiu confirmar estas observações, levando-nos à conclusão da necessidade de integração do conceito de gerência. Um alto valor de tempo de resposta pode ser ocasionado tanto pelos próprios componentes da aplicação, quanto pela própria máquina ou pela rede.

Paralelamente, verificou-se a possibilidade de que algumas informações monitoradas poderiam ser úteis para a área funcional de configuração, como o número da porta de escuta, o seu modo de ativação e o identificador. Para a área de segurança, o nome do usuário cliente poderia ser monitorado como o responsável pela chamada. Os atributos da interface IDL de um servidor instrumentado, especificada no capítulo anterior, apresenta apenas os dados essenciais para o nosso propósito.

O ambiente heterogêneo da Multiware permitiu a monitorização de aplicações instrumentadas em diferentes configurações. A Tabela 6.1 apresenta um exemplo de amostragem de algumas métricas coletadas em dois servidores diferentes. Em uma das colunas, são mostradas as médias dos tempos de residência dos servidores. Contudo, estes valores compõem apenas uma pequena fração do tempo de resposta de um cliente. Por exemplo, os valores de média e desvio padrão do tempo de resposta do cliente foram 5,24 mili-segundos e 0,81, enquanto os valores do tempo de residência no servidor foram apenas 0,36 mili-segundos e 0,12 respectivamente. O custo de comunicação degrada o tempo de resposta do cliente, porém não afeta o tempo de residência do servidor, devido basicamente ao processamento do método remoto chamado.

Tabela 6.1 - Amostragem

Nome do Server	**Máquina**	**pid**	**Utilização**	**Tempo de Residência**	**Vazão**	**Operação mais requisitada**
matrix	marumbi	4483	0.11	0.32	7.8	set
matrix	marumbi	4483	0.13	0.35	8.0	set
grid	trancoso	1181	0.21	0.29	9.8	get
grid	trancoso	1181	0.20	0.30	8.0	get

A Tabela 6.2 mostra as diferenças entre os tempos de resposta no cliente e o de residência, devido ao processamento local no servidor, com os valores da média e desvio padrão, em três situações distintas: o cliente e o servidor na mesma máquina, em diferentes máquinas do mesmo laboratório, e em diferentes laboratórios. Estes resultados pretendem isolar a participação do custo de comunicação no cômputo do correspondente tempo total de resposta, em diferentes configurações.

Tabela 6.2 - Diferenças de Tempos de Respostas

Localização	**Mesma máquina**	**Diferentes Máquinas**	**Diferentes LAN's**
Média	4.18	5.01	8.53
D.P.	1.09	1.15	3.39

6.3.1 Interface Gráfica

Embora originalmente não tenha sido especificada nenhuma interface gráfica para a visualização dos dados, as similaridades de Java com C++ e de OrbixWeb com Orbix despertaram o interesse no uso desta tecnologia. Inicialmente, pensávamos em desenvolver "applets", que pudessem ser embutidos em páginas HTML e dispostos em servidores WWW, podendo ser futuramente integrados com outras ferramentas de gerência de redes e de sistemas em um navegador. O processo de decisão de um gerente seria simplificado pela interação com os "applets" de forma bastante intuitiva. Dessa forma, um gerente poderia navegar através de "applets", visualizando outros dados de interesse como carga de CPU, atividades de disco e de paginação dos nós sobre os quais os componentes das aplicações CORBA estão sendo executados, e monitorando os elementos do subsistema de comunicação. Qualquer manutenção seria feita em um único arquivo que substituiria o original no servidor WWW. Entretanto, os navegadores e o interpretador Java impõem restrições aos "applets" que impedem a abertura de conexões com outras máquinas, além daquela de onde ele foi carregado. Nesta época, a forma de contornar essas dificuldades ainda não estava muito clara, agravadas pelo fato de desejar que o "applet" fosse essencialmente um cliente "OrbixWeb". Dessa forma, o "applet" poderia enviar requisições remotas, usando a sintaxe natural da linguagem de chamada de métodos, mascarando o uso dos métodos das classes "Socket" e "DatagramSocket" da biblioteca "java.net" de Java. A solução foi desenvolver uma aplicação gráfica, sem as características de um "applet".

A idéia básica do AWT é que uma janela Java é um conjunto de componentes aninhados, começando desde a janela mais externa até o menor componente da interface gráfica. Tais componentes podem ser janelas, barras de menus, botões, campos de textos e recipientes (containers), que por sua vez podem conter outros componentes. Esse agrupamento de componentes cria uma hierarquia que determina a disposição dos itens na tela, a ordem em que eles são pintados e como os eventos são passados de um componente para outro. Os principais componentes do AWT são:

- "containers": são componentes genéricos que podem conter outros componentes, incluindo outros "containers". A forma mais comum é o painel, que representa um recipiente que pode ser exibido na tela. Os "applets" são uma forma de painel;

- quadros - são simples superfícies para desenho;
- componentes de construção da janela - incluem janelas, molduras, barras de menu e diálogos; e
- outros componentes - esses componentes podem ser rótulos, botões, caixas de seleção, botões de opção, menus de opção e campos de texto, elementos típicos de uma interface com o usuário.

As classes dentro da biblioteca AWT são escritas e organizadas para refletir a estrutura abstrata dos recipientes e dos outros componentes. A raiz da hierarquia de herança da maioria dos componentes AWT é a classe abstrata "Component", que fornece métodos para uma exibição básica e para manipulação de eventos. As classes "Container" para recipientes; "Canvas" para quadros; "Checkbox" para caixas de seleção; "Choice" para menus de opção, "TextComponent" para áreas e campos de texto; "List" para listas de rolagem; "Label" para rótulos; "Scrollbar" para barras de rolagem; e "Button" para botões são subclasses diretas de "Component". As classes "Panel" e "Window" são herdeiras da classe pai "Container", sendo objetos que podem conter outros componentes AWT.

A disposição dos componentes AWT em uma interface gráfica é determinada pela ordem em que são adicionados ao recipiente que os contém e pelo tipo de gerenciador de apresentação que está sendo utilizado para planejar a tela. O gerenciador determina como as partes da interface serão divididas e como os componentes serão apresentados. O AWT oferece cinco gerenciadores: "FlowLayout", "GridLayout", "GridBagLayout", "BorderLayout" e "CardLayout", que são escolhidos através do método "setLayout()". Uma vez definido o gerenciador, a ordem da adição dos componentes também influenciará a apresentação final. A classe "Flowlayout" é a mais básica na qual os componentes são adicionados um por vez, linha por linha. Se um componente não cabe dentro de um linha, ele é colocado na próxima linha. Os gerenciadores de grade "GridLayout" e "GridBagLayout" oferecem um controle maior sobre a localização dos componentes de um painel, dividindo a área de apresentação em um número especificado de linhas e de colunas de uma grade. Cada componente é adicionado em uma célula da grade, iniciando da linha superior e progredindo da esquerda para direita. A classe "GridBagLayout" se diferencia de "GridLayout" por permitir que as variáveis, que determinam a posição de um

componente dentre de um recipiente, sejam definidas através de um objeto do tipo "GridBagConstraints". A classe "BoderLayout" permite a disposição dos componentes em um local, onde a sua posição é especificada como uma indicação geográfica: norte, sul, leste, oeste e centro. Em um gerenciador "CardLayout", os componentes são exibidos seqüencialmente, sendo utilizados para produzir um visualizador de "slides", onde cada cartão pode ter o seu próprio "layout".

Para que uma interface gráfica seja interativa ao usuário, é preciso ainda que sejam definidos eventos associados aos componentes pressionados, inseridos ou selecionados. Portanto, existe o método "action()" na classe abstrata "Component" que intercepta uma ação de qualquer componente da interface, com dois argumentos passados. O primeiro é um objeto da classe "Event" que representa o evento em si. Todos os eventos gerados são instâncias desta classe, que contém informações sobre onde e quando o evento ocorreu, o tipo de evento e outras informações. O segundo parâmetro de "action()" depende do tipo de componente que está gerando o evento, sendo um argumento arbitrário da classe "Object", uma espécie de meta classe que é a raiz de todas as classes existentes em Java. Para tratar com os diferentes componentes e as ações que eles possam gerar, é necessário verificar o tipo de objeto que chamou o evento, dentro do corpo do método "action()". Esse objeto é armazenado na variável de instância "target", precisando ainda utilizar o operador "instanceof" para verificar qual é o tipo de componente responsável pelo evento:

```
public boolean action (Event evt, Object arg) {
     if (evt.target instanceof Button)
     //verifica se o evento foi gerado por um objeto da classe Button
}
```

Finalmente, os dados instrumentados são visualizados através de um cliente Java, no ambiente da OrbixWeb, através de uma interface gráfica mostrada na Figura 6.4. O painel é composto basicamente de duas listas de rolagem, gerenciadas pelo "BorderLayout", e de um recipiente que contém um menu de opção e um botão, dispostos pelo "GridBagLayout". Acima das listas, existem ainda dois rótulos que servem para indicar o conteúdo das duas listas. O evento

provocado pela seleção de uma opção do menu está associado com a lista da esquerda, enquanto o fato de pressionar o botão da direita está relacionado com o que será apresentado na outra lista. O menu de opção "Máquina" permite a escolha de um nome DNS (Domain Name System) de uma das máquinas do domínio a ser monitorado. Este domínio é facilmente configurado através da facilidade do localizador, existente na Orbix, que estabelece uma relação de prioridade de ativação dos servidores de acordo com uma relação de nomes das máquinas escolhidas. Após a escolha de uma máquina, o evento associado a esse componente apresenta na lista à esquerda os detalhes de todas as implementações de objetos que estão sendo executadas naquele nó. Estes dados são obtidos através do método "IT_daemon::listActiveServers()", disponível na interface IDL do processo "orbixd", retornando uma estrutura de dados do tipo "serverDetails". A seleção combinada de um determinado servidor, apresentado na lista à esquerda, e do botão "Medida" dispara uma requisição remota "Server_Managed_Object::listInstrument_Data()" para a interface IDL do servidor instrumentado, que devolve uma estrutura "Instrument_Data" apresentada na lista à direita. Estas interações podem ser vistas na Figura 6.5.

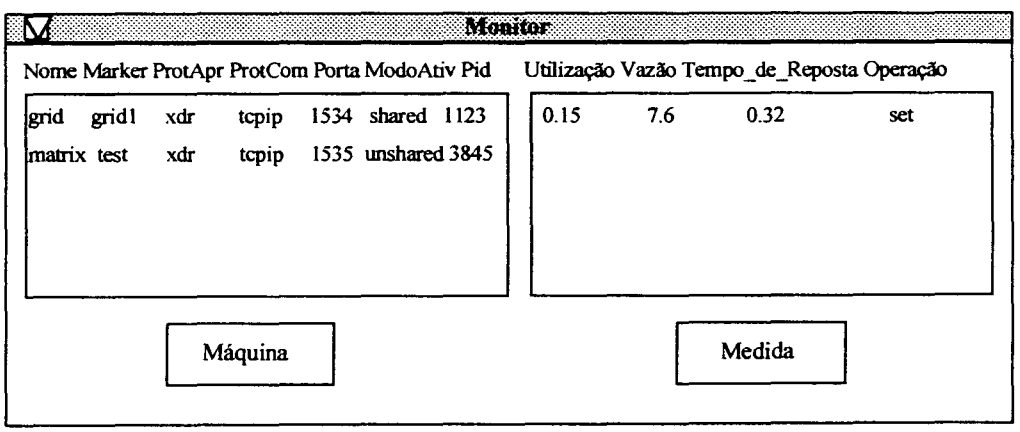

Figura 6.4 - Interface Gráfica em Java

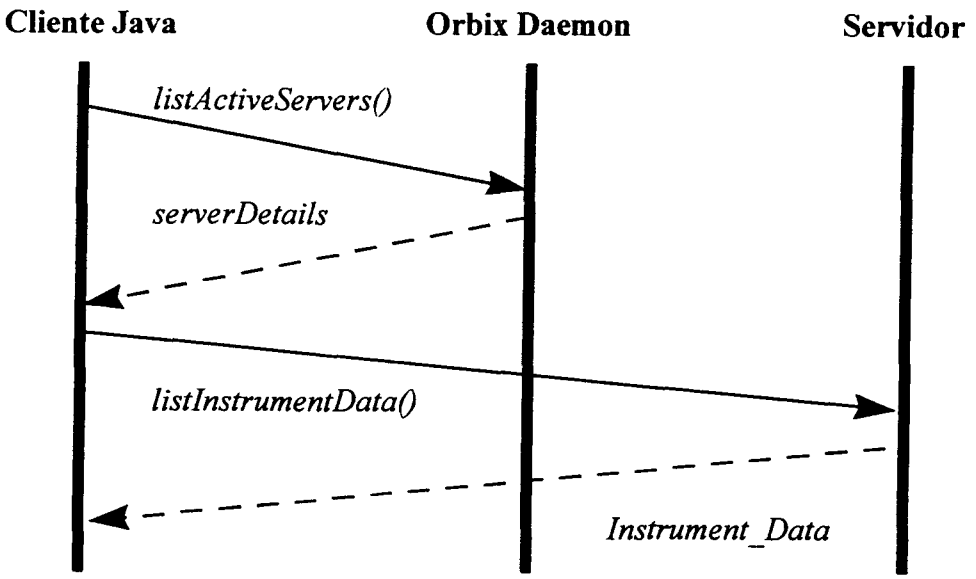

Figura 6.5 - Interações dinâmicas

Serviço OrbixTalk

A implementação da especificação do Serviço de Eventos pela Iona, OrbixTalk, é oferecida, como uma extensão de sua plataforma "Orbix", para permitir a comunicação entre clientes e os objetos remotos através de mensagens. A documentação (Iona, 1996) especifica os papéis de falador (talker) e de ouvinte (listener) para os objetos que respectivamente geram mensagens e que as recebem. O fluxo de mensagens é unidirecional, de um ou mais faladores para um ou mais ouvintes. Portanto, uma única mensagem pode simultaneamente ter mais de um falador para mais de um ouvinte. De forma geral, qualquer número de faladores podem enviar mensagens através do mesmo fluxo para uma outra quantidade de ouvintes, sem que nenhum deles tenha conhecimento da existência do outro. A correspondência entre os papéis de cliente/servidor e de falador/ouvinte é direta: o falador é cliente do ouvinte que recebe as mensagens e as processa. De antemão, pode-se afirmar que estes papéis são similares aos objetos fornecedores e consumidores no modelo "push", terminologia adotada na padronização da OMG, descrita no capítulo anterior. Essa é a primeira diferença desta implementação para o serviço originalmente especificado..

Na OrbixTalk, os eventos são enviados para objetos através de mensagens. Eventos de um tipo particular são identificados por um "TopicName" e um ouvinte especifica quais são os eventos de seu interesse, usando esta identificação. Essa é a segunda diferença da padronização da OMG. Dessa forma, a implementação permite que o conteúdo semântico dos eventos seja organizado em uma estrutura hierárquica de tópicos, cada um identificado por um "TopicName". Uma aplicação cliente pode simplesmente determinar que tipo de informação está interessada e informar isto, usando "TopicName". O formato de um tópico é semelhante a uma URL (Uniform Resource Locator), usada na tecnologia WWW. A primeira parte, o prefixo, identifica o protocolo de comunicação. O protocolo de transporte usado pela OrbixTalk é o "OrbixTalk Reliable Multicast Protocol" (OTRMP). O resto do nome é hierarquicamente organizado, permitindo uma grande flexibilidade na administração do espaço de nomes. Como exemplo, apresentado na documentação, pode-se estruturar um tópico que identifica uma teleconferência, organizada pela Iona e prevista para as 10:00 horas:

```
"otmrp//iona/teleconf/10.00AM"
```

Para que uma aplicação envie um evento para um "TopicName", deverá criar um objeto "proxy" e registrá-lo como falador. Todas as comunicações são feitas através do "proxy" registrado como um falador que usa a camada de transporte de OrbixTalk. Dessa forma, os objetos faladores não precisam se preocupar com quem receberá estas informações sobre o evento. Os ouvintes determinam que tipo de informação desejam receber. Se houver múltiplos ouvintes de um determinado tópico, todos eles receberão a mesma informação.

OrbixTalk usa nomes hierárquicos para identificar grupos de informação. Esses grupos são traduzidos para "IP Multicast Groups" que são endereços IP classe D. Portanto, existe um processo, conhecido como "Directory Enquiries" para executar esta tradução. Este servidor é usado apenas uma vez para ajustar a comunicação inicial entre um falador e um ouvinte.

A Figura 6.6 ilustra a arquitetura de OrbixTalk, mostrando um falador, um ouvinte e o processo "OrbixTalk Directory Enquiries Daemon", conhecido como "oad". Inicialmente, um falador deseja enviar mensagens de um determinado tópico, precisando verificar se já houve tradução prévia para um endereço multiponto. Se não houver, uma requisição é enviada para o processo "oad", solicitando o número IP para o nome do tópico. O processo "oad" consulta uma tabela

interna e retorna o número IP, se já houver o mapeamento, ou cria uma nova entrada com um número não utilizado ainda. Dessa forma, o objeto falador pode enviar mensagens para qualquer ouvinte interessado naquele tópico. Por outro lado, o ouvinte faz uma chamada ao "oad", registrando o seu interesse em um determinado tópico, recebendo o correspondente número IP classe D. A diferença fundamental desta forma de troca de mensagens e de uma invocação na plataforma Orbix é que o falador desconhece a existência ou o número dos objetos ouvintes.

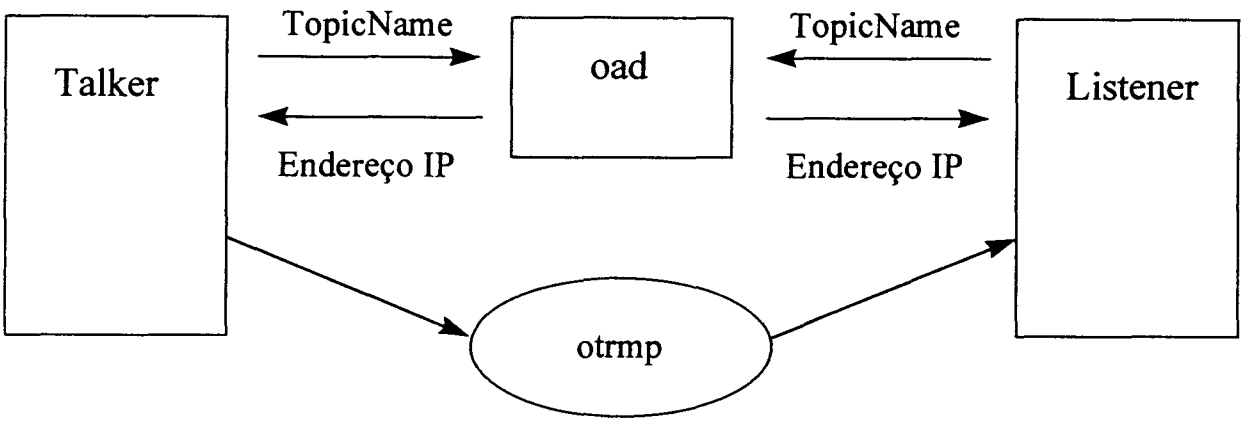

Figura 6.6 - Arquitetura OrbixTalk

OrbixTalk oferece ainda uma versão persistente do seu protocolo, permitindo o armazenamento das mensagens e a sua recuperação e conhecido como "OrbixTalk Store-and-Forward Protocol" (OTSFP). Este modo tem as mesmas características anteriormente descritas, porém envolve um outro processo que armazena as mensagens em disco seqüencialmente, antes de serem enviadas para os ouvintes. Cada mensagem especifica um tópico e, após ser salva, é enviada para os ouvintes. Esta é a terceira diferença da padronização da OMG. O processo, denominado como "OrbixTalk Message Store Daemon" (otmsd), acusa o recebimento de cada mensagem recebida de um objeto falador. Caso não receba a confirmação, o falador continuará tentando enviar a mensagem em intervalo de tempo configurável. Após ela ter sido preservada em memória secundária, o "otmsd" envia a mensagem, usando o protocolo OTRMP sem aguardar retorno nenhum do ouvinte. O "otmsd" envia ainda periodicamente a informação do número da última mensagem enviada. Caso o ouvinte detecte a falta de uma mensagem, ele pode solicitar a sua recuperação.

A Figura 6.7 apresenta os componentes da versão persistente da arquitetura OrbixTalk. O objeto falador deve identificar-se com um nome de aplicação persistente único e ter um registro local próprio, usado para armazenar os números de seqüência das mensagens efetivamente enviadas para o "otmsd". Para usar o protocolo OTSFP, ele ainda deve especificá-lo como prefixo no formato URL do "TopicName". O falador usa o próximo número seqüencial, guardado em um registro local persistente, para enviar uma mensagem. Ao recebê-la, o "otmsd" a armazena em disco e guarda em memória primária um índice para acesso. Cada mensagem recebida provoca uma confirmação. Após isto, a mensagem pode ser finalmente enviada para os processos destinatários. Os ouvintes preservam ainda em um registro local o número da última mensagem recebida. Em caso de falhas de um ouvinte, esta informação pode ser enviada para o "otmsd" que providencia a recuperação e o reenvio de todas as mensagens com o número de seqüência maior.

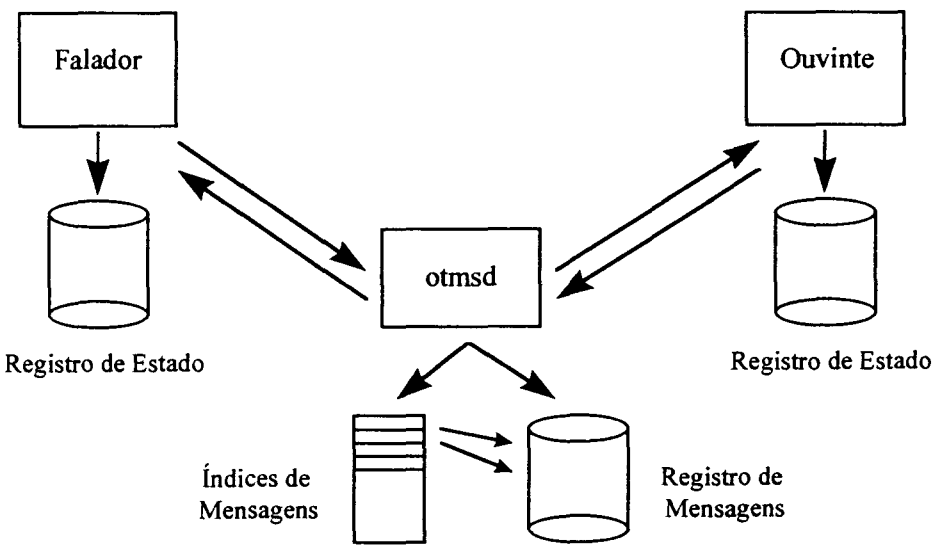

Figura 6.7 - Versão Persistente da Arquitetura OrbixTalk

6.4.1 Detalhes do Protocolo OTMRP

O protocolo IP aceita "multicast", usando endereços classe D de 32 bits (Tanenbaun, 1996). Os quatro bits mais significativos são obrigatoriamente "1110" que identificam o endereço Internet

como endereço de grupo. Os demais 28 bits são usados para identificação de grupos. Portanto, os números IP classe D variam de 224.0.0.0 a 239.255.255.255. Quando um processo envia um pacote para um endereço classe D, é feita uma tentativa de entregá-lo a todos os membros do grupo endereçado, mas não há qualquer garantia que isso realmente aconteça. Estão previstos dois tipos de endereços de grupo: endereços permanentes e temporários. Grupos permanentes têm endereços do mesmo tipo, não precisando serem definidos. Para eles, são reservados os números na faixa de 224.0.0.0 a 224.0.0.255. Por outro lado, grupos temporários são criados antes de serem usados.

OrbixTalk usa o protocolo de transporte UDP sobre IP Multiponto para compartilhar informações. Todavia, esse protocolo não é confiável, sendo incorporados mecanismos ao OTRMP que garantem que as mensagens de um determinado falador sejam efetivamente enviadas para todas as aplicações ouvintes na seqüência correta. As mensagens, maiores do que um pacote UDP, são fragmentadas antes de serem enviadas e remontadas pelos ouvintes. OrbixTalk estabelece um número seqüencial para cada mensagem antes de sua fragmentação. Os fragmentos são enviados e armazenados em memória para retransmití-los, caso seja necessário. A aplicação ouvinte coleta todos fragmentos de uma determinada mensagem e as remonta. Periodicamente, um falador envia uma mensagem com a indicação do número seqüencial da última mensagem enviada, para que os ouvintes verifiquem se houve perda e solicitem o reenvio.

6.5 Análise da Implementação da Facilidade de Monitorização Assíncrona

As três diferenças de OrbixTalk da padronização da OMG, apontadas na seção anterior, são decorrentes da necessidade de se estender o Serviço de Eventos CORBA original. Elas permitem ainda a implementação de muitos dos pré-requisitos estabelecidos na proposta, descrita no capítulo anterior, de se ampliar o canal de eventos, com mecanismos que ofereçam filtragem, persistência e correlação de eventos de maneira modular.

A proposta é implementar uma classe "Extended_Event_Channel", herdando os métodos da classe "OrbixTalk", especializando-a. Os principais métodos estão relacionados na Tabela 6.3. As três primeiras operações são do serviço original. As quatro seguintes do módulo "Subscribe" permitem a gerência do registro de consumidores e fornecedores. Finalmente, o método "setPersistentAppName()" define um identificador único para controle de persistência, tanto para o falador quanto para o ouvinte.

Tabela 6.3 - Métodos usados

Método	**Módulo**
initialise()	
processEvent()	
addTimerEvent()	
removeTimerEvent()	
registerTalker()	Subscribe
registerListener()	Subscribe
unregister()	Subscribe
isRegistered()	Subscribe
setPersistentAppName()	Log

O módulo de Assinatura e Filtragem é implementado usando as facilidades do protocolo OTRMP. Ao fazer a assinatura, uma aplicação gerente determina os eventos de interesse através de um "TopicName", no formato URL em que a primeira parte identifica o protocolo e o sufixo especifica a hierarquia de tópicos. Neste caso, o prefixo pode ainda ser omitido por ser o protocolo padrão. Como o escopo deste trabalho é a gerência de aplicações CORBA para as áreas funcionais de desempenho e de contabilização, pode-se estruturar este tipo de informação de forma hierárquica, utilizando até mesmo dados obtidos com a instrumentação através dos sensores implementados a partir de CORBA::Filter:

```
"management/performance/CORBA_server/grid/number_of_users"
"management/performance/CORBA_server/grid/most_request_operation"
"management/performance/CORBA_server/grid/principal"
```

Estes formatos URLs especificam eventos de uma implementação de objetos CORBA "grid", no papel de falador/consumidor, que possam ser de interesse para gerentes que se livram das semânticas restritivas do modelo síncrono. Esta forma de estruturação se mostrou muito interessante por permitir diversas maneiras de filtrar as informações de gerência através de

diferentes URLs. A interface IDL deve ser especificada e implementada da mesma forma que é feito quando se usa apenas a Orbix. Uma aplicação consumidora deve então:

- inicializar um objeto OrbixTalk com o método "initialise()";
- criar objetos ouvintes com o operador Java/C++ "new", passando o "TopicName" do evento como parâmetro para o método construtor;
- registrá-los com o método "registerListener()", sendo enviada uma requisição para o "oad" para obter o endereço "multicast"; e
- aguardar mensagens para o tópico especificado, através de invocações remotas, ao chamar o método "processEvents()". É uma chamada similar ao "impl_is_ready()", existente na Orbix, para que uma aplicação de objetos passe a receber invocações remotas para os objetos CORBA.

Dessa forma, a implementação do protocolo OTRMP trata e processa todas as mensagens recebidas pelo ouvinte, através de requisições IDL.

Por outro lado, uma aplicação remetente deve:

- inicializar um objeto OrbixTalk com o método "initialise()";
- criar objetos faladores com o operador "new";
- registrá-los com o método "registerTalker()", passando o "TopicName" como parâmetro, sendo enviada uma requisição para o "oad" para obter o endereço "multicast". É uma chamada similar ao "bind()", existente na Orbix, para que a aplicação cliente se ligue à implementação de objetos através do protocolo IIOP. Todavia, o seu efeito provoca o encaminhamento dinâmico de todas as invocações para o protocolo OTRMP. Existe ainda o método "unregister()" que serve para cancelar o registro e reverter o destino das invocações para o IIOP; e
- enviar mensagens para o tópico especificado ao chamar o método "processEvents()".

Do lado do falador, o protocolo OTRMP deve tratar as requisições para enviar os fragmentos da mensagem, verificar o sincronismo das mensagens e enviar periodicamente mensagens de informação com o número seqüencial. OrbixTalk ainda oferece uma classe abstrata "TimerEvent" que define uma interface para eventos que necessitem ser enviados periodicamente. O seu construtor tem um parâmetro para se especificar a periodicidade do evento, em mili-segundos. O seu método "fired()" permite a inclusão de código para tratar esses tipos de eventos quando o prazo estipulado expira. A chamada ao método "addTimerEvent()" reinicia o ciclo do evento periódico.

O módulo de Armazenamento é implementado usando as facilidades do protocolo OTSFP. Ao fazer a assinatura, uma aplicação gerente, no papel de ouvinte, determina os eventos de gerência de seu interesse através de um "TopicName", no formato URL, identificando obrigatoriamente o prefixo. É necessário ainda especificar, através da chamada ao método "setPersistentAppName()", um nome único para a aplicação gerente, para controle de seu estado persistente, no seguinte formato exemplo:

```
"//Management/manager"
```

Portanto, uma aplicação gerente deve ainda:

- inicializar um objeto OrbixTalk com o método "initialise()";
- criar objetos ouvintes com o operador Java/C++ "new", passando o "TopicName" do evento como parâmetro para o método construtor;
- registrá-los com o método "registerListener()", e
- aguardar mensagens para o tópico especificado, através de invocações remotas, ao chamar o método "processEvents()". É uma chamada similar ao "impl_is_ready()", existente na Orbix, para que uma aplicação de objetos passe a receber invocações remotas para os objetos CORBA.

De forma análoga, um recurso gerenciado, no papel de falador, deve também especificar, através da chamada ao método "setPersistentAppName()", um nome único para identificar o seu registro de mensagens, no mesmo formato exemplificado anteriormente. Deve ainda:

- inicializar um objeto OrbixTalk com o método "initialise()";
- criar objetos faladores com o operador "new";
- registrá-los com o método "registerTalker()", passando o "TopicName" como parâmetro; e
- enviar mensagens para o tópico especificado ao chamar o método "processEvents()".

Dessa forma, em caso de falhas, um gerente ao se recuperar pode consultar o seu índice de notificações recebidas e solicitar o reenvio dos eventos não recebidos. As características de persistência oferecidas por OrbixTalk permite que objetos gerentes solicitem a recuperação das mensagens preservadas dos objetos gerenciados de uma forma muito simples e eficiente. Permite ainda que seja feita uma auditoria nos registros armazenados, muito útil para se apurar eventos anteriores.

A implementação do módulo de Correlação de Eventos seria a principal contribuição nesta fase, porém não faz parte da proposta desta dissertação, ficando para um trabalho futuro. A idéia inicial seria combinar eventos, especificados através de URLs, usando os operadores lógicos, de exclusão, de retardo e seqüencial, descritos no capítulo anterior. Seriam usados para que as aplicações gerentes, no papel de ouvintes, consigam aumentar o conteúdo semântico dos eventos de seu interesse. As propostas de implementação de uma funcionalidade similar (Mansouri-Samani, 1997) e (Schade, 1997), incorporada ao canal de eventos, podem contribuir para este trabalho futuro.

6.6 Trabalhos Relacionados

Esta seção agrupa os trabalhos relacionados em dois parágrafos, o primeiro para a versão síncrona e o seguinte para a parte assíncrona da Facilidade de Monitorização. Esta divisão reflete a principal diferença desta dissertação para os demais descritos: aborda a monitorização, usando os dois paradigmas de requisição/resposta (Queiroz, 1997c) e de eventos (Queiroz, 1997a), disponíveis em um ambiente de gerência CORBA, no contexto de uma arquitetura integrada.

Friedrich et all. em (Friedrich, 1995a) descrevem uma arquitetura e apresentam um protótipo para monitorização de aplicações baseadas em DCE. Embora mostre informações que auxiliem a entender o comportamento operacional de DCE, tais dados são também úteis para gerência de desempenho. (Schade, 1996) propõe um método para integrar uma aplicação distribuída em um ambiente de gerência, baseado em uma interface "Management" em IDL, extensão da CORBA IDL, e uma biblioteca de instrumentação para suportar o desenvolvimento de aplicações capazes de serem gerenciadas. (Üslander, 1996) também apresenta uma arquitetura para a gerência de aplicações CORBA, integrada com os atuais padrões de gerência de redes e de sistemas. (Chadha, 1996) apresenta uma forma de gerenciar aplicações CORBA, a partir de uma plataforma de gerência OSI/CMIP, monitorando simples atributos dos componentes. Este trabalho se diferencia por apresentar uma maneira modular de instrumentar implementações de objetos CORBA e de monitorá-los, para as áreas funcionais de desempenho e de contabilização, no contexto maior de uma arquitetura de gerência integrada, discutindo aspectos de implementação e apresentando resultados.

Recentemente, o grupo de trabalho de Telecomunicações (telecom) da OMG emitiu uma RFP de um Serviço de Notificações (OMG 1997e) que pretende estender o Serviço de Eventos para suportar o processamento de eventos. Tal RFP incentiva que as extensões já existentes sejam apresentadas através de RFCs. Em (Mansouri-Samani, 1997), os autores apresentam uma linguagem interpretada que permite a especificação de eventos compostos em termos de eventos primitivos através de operadores de composição em um sistema distribuído fracamente acoplado. (Schade, 1997) sugere que uma linguagem de especificação de eventos, derivada do trabalho de (Mansouri-Samani, 1997), pode ser aplicada na extensão do Serviço de Eventos CORBA. Ambos não apresentam nenhuma análise de implementação em um Serviço de Eventos disponível, limitando-se a abordar apenas a questão de correlação de eventos de forma conceitual. Este trabalho se distingue por este fato. Harrison et all. em (Harrison, 1997) apresentam uma extensão do Serviço de Eventos CORBA para satisfazer requisitos de Qualidade de Serviço em sistemas distribuídos de tempo real. Ainda que tenha uma aplicação final distinta e apresente módulos de filtragem, de "dispatching" e de correlação de eventos, as nossas propostas são similares pelo fato de se preocuparem em usar o Canal de Eventos como mediador.

6.7 Conclusão

Este capítulo descreveu os principais aspectos da implementação das versões síncronas e assíncrona da Facilidade de Monitorização, usando os recursos disponíveis da plataforma Orbix e do serviço OrbixTalk.

O primeiro passo foi instrumentar os clientes e servidores CORBA com sensores, habilitando a sua monitorização, usando a semântica CORBA de requisição/resposta. O protótipo implementa uma MIB definida em IDL, acessível por objetos CORBA, com informações de gerência úteis para as áreas funcionais de desempenho e de contabilização. Tais informações são ainda visualizadas em uma aplicação gráfica em Java, como um componente cliente no ambiente da OrbixWeb. Estima-se ainda que o número potencial de sensores usados possa ser maior ainda, na medida que se deseje adquirir mais dados, porém o custo da instrumentação deve ser avaliado. A característica de poder instalar os filtros em cadeia permite o controle modular do impacto desta instrumentação. Os resultados observados com aplicações num ambiente heterogêneo, como o da Multiware, levam-nos à direção da necessidade de se adotar o conceito de gerência integrada de sistemas distribuídos.

OrbixTalk garante a comunicação assíncrona e a ordenação da seqüência de eventos entre fornecedores/faladores e consumidores/ouvintes, através de seu protocolo confiável OTRMP. O Canal de Eventos é implementado de forma descentralizada através de um conjunto de "proxies" cooperativos no ambiente de Orbix, um para cada objeto conectado ao canal. Dessa forma, o fornecedor envia eventos usando o protocolo assíncrono, sem conhecimento da existência do consumidor. Portanto, não existe nenhum processo intermediário, centralizado entre fornecedores e consumidores. Todavia, oferece um serviço de eventos persistente que usa um processo mediador para armazenar as mensagens enviadas, antes de serem entregues para os destinatários. A implementação dos módulos de Assinatura e Filtragem e de Armazenamento usam estas características disponíveis.

As versões síncrona e assíncrona da Facilidade de Monitorização descrita constituem a principal diferença deste trabalho, quando comparado com os demais descritos na seção anterior, por usar tanto o paradigma de requisição/resposta quanto de eventos, disponíveis em um ambiente de

gerência CORBA. A experiência obtida com CORBA tem mostrado que os esforços de padronização da OMG também devem ser concentrados em gerência. A maioria dos mecanismos e operações, utilizados na instrumentação e na extensão do canal de eventos, só são disponíveis nas implementações da Iona. Elas não são parte da especificação CORBA. A proposta da X/Open (X/Open, 1995), descrita no Capítulo 4 e aprovada pela OMG, não especifica nenhum mecanismo para que um objeto CORBA apresente uma interface de gerência e para sua instrumentação. Contudo, os benefícios de se usar uma plataforma aberta, distribuída e orientada a objetos são maiores do que estas restrições.

Capítulo 7

Conclusão

Esta dissertação apresenta uma Facilidade de Monitorização, no contexto da Arquitetura de Gerência Integrada e no âmbito da plataforma Multiware. Esta arquitetura é baseada na ODMA, cuja proposta é fundamentada no Modelo de Referência ODP, estendendo o padrão OSI/CMIP de gerência; e adota como infra-estrutura de suporte a CORBA, os CORBAservices e as emergentes CORBAfacilities, onde os objetos gerenciados são os componentes das aplicações e os elementos de suporte em um ambiente de processamento distribuído aberto.

O modelo descrito da Facilidade de Monitorização se mostrou suficientemente genérico tanto para a versão síncrona quanto para a assíncrona da Facilidade de Monitorização. Na primeira, estamos interessados em explorar o uso de um ORB como protocolo de gerência para o acesso a objetos gerenciados em um ambiente CORBA. Na variante assíncrona, investigamos o uso do

canal de eventos de forma a incorporar os módulos de coleta, processamento e armazenamento. Embora esta abordagem modular apresente módulos distintos de coleta, armazenamento e processamento, eles podem estar presentes em um único processo sem perda desta estruturação.

Na versão síncrona da Facilidade, o primeiro passo foi instrumentar os clientes e servidores CORBA com sensores, habilitando a sua monitorização. O protótipo implementa uma MIB para os componentes das aplicações distribuídas CORBA, definida em IDL, acessível por objetos distribuídos, com informações de gerência úteis para as áreas funcionais de desempenho e de contabilização, visualizadas em uma interface gráfica em Java. Os resultados observados num ambiente heterogêneo, como o da Multiware, levam-nos à direção da necessidade de se adotar o conceito pleno de gerência integrada de sistemas distribuídos.

A versão assíncrona da Facilidade descreve uma extensão do Serviço de Eventos CORBA, motivado pela necessidade de se ter um modelo de monitorização assíncrono flexível, genérico e configurável, apropriado para gerência, com mecanismos de assinatura e filtragem, de correlação de eventos e de armazenamento. Apresenta ainda os aspectos de sua implementação, usando o serviço da OrbixTalk.

A contribuição deste trabalho pode ser resumida pelos seguintes itens:

- ter demonstrado o uso do RM-ODP como um padrão de direito e como uma infra-estrutura para gerência através das técnicas e mecanismos existentes em CORBA (Queiroz, 1997b) ;
- ter proposto uma arquitetura de gerência integrada que oferece uma infra-estrutura para o desenvolvimento de aplicações distribuídas CORBA de gerência para os três domínios existentes: SNMP, CMIP e CORBA. Usa o suporte de um ORB, dos Serviços de Objetos, das Facilidades Comuns e dos adaptadores SNMP/CMIP;
- ter especificado a Facilidade de Monitorização em um ambiente de gerência CORBA, essencial para a integração da gerência de sistemas distribuídos heterogêneos ;

- ter definido em IDL uma MIB aberta para a gerência de objetos CORBA, acessível por gerentes CORBA ou por gerentes CMIP/SNMP através de adaptadores;
- ter traduzido métricas de desempenho para o ambiente da CORBA em uma abordagem "top-down", sendo usadas também para a área funcional de contabilização;
- ter demonstrado o uso dos filtros existentes na Orbix para a instrumentação de objetos gerenciados CORBA para a aquisição dessas métricas; e
- ter analisado a implementação parcial da extensão proposta do canal de eventos, usando o Serviço de Eventos OrbixTalk.

Espera-se que este trabalho seja o ponto de partida para vários outros. São propostos os seguintes trabalhos futuros, como extensão desta dissertação:

- analisar e adaptar a arquitetura de gerência proposta com a especificação da Facilidade Comum de Gerência de Sistemas, adotada recentemente pela OMG;
- analisar, especificar as APIs em IDL e implementar as Facilidades de Domínios; de Configuração, investigando o uso do Serviço de Ciclo de Vida; de Políticas; e de Controle;
- analisar, especificar e implementar a proposta de extensão do canal de eventos com o módulo de Correlação de Eventos; e
- investigar uma forma alternativa de instrumentar os objetos de uma aplicação distribuída CORBA, através do conceito de reflexão computacional, tanto estrutural quanto comportamental, por permitir o acesso ao estado de execução dos objetos.

O RM-ODP provê uma infra-estrutura para aplicações distribuídas, capaz de ser usada para a integração dos modelos atuais de gerência. Os esforços da padronização ODMA estão tentando suprir essa deficiência, tendo já incorporado CORBA em seu modelo. Os "clusters" de BEOs, agrupados em cápsulas, podem corresponder aos objetos CORBA, povoados de objetos auxiliares instanciados, que residem em implementações de objetos. Entretanto, os objetos CORBA não oferecem o conceito de múltiplas interfaces, previsto no RM-ODP. Particularmente

em gerência, esta restrição não permite que se tenham diferentes visões de um único objeto gerenciado, uma para cada área funcional, além de sua interface operacional.

A experiência de acompanhar os temas de interesse, abordados através de listas de discussão e publicados através de RFPs e RFCs, disponíveis na página da OMG, foi muito interessante. Permitiu acompanhar os esforços de padronização e as diversas opiniões, muitas vezes incompatíveis. Outras vezes, a apresentação de uma única RFC não permitiu que um determinado assunto, como a Facilidade Comum de Gerência de Sistemas, não tenha sido suficientemente investigado no âmbito da OMG. Provavelmente isto tenha ocorrido pela existência de outros organismos de padronização do tópico abordado. Muitos deles têm adotado CORBA como uma solução de interoperabilidade.

Atualmente, existem dois paradigmas usados em gerência, um para redes de computadores e outro para telecomunicações. Apesar dos esforços despendidos para unificar estes padrões, há muitas dificuldades para apresentar soluções concretas. O modelo de objetos distribuídos CORBA tem-se mostrado como uma tecnologia promissora de interoperabilidade em gerência, como apresentado neste trabalho. A proposta imediata não é substituir as arquiteturas SNMP, OSI/CMIP e TMN por CORBA, mas investigar a melhor forma de CORBA ser usada em conjunto com a atual infra-estrutura investida em gerência. Entretanto, a nova era da informação exigirá soluções que garantam a adequada gerência dos seus elementos de comunicação, dos recursos de suporte e das aplicações e serviços.

Referências Bibliográficas

Autrata, M. e Strutt, C. (1994) DME Framework and Design. *Network and Distributed Systems Management (Ed. Morris Sloman)*, Cap. 23; Addison-Wesley , pp 605-627.

Ban, B. (1996) Towards an Object-Oriented Framework for Multi-Domain Management. *Proceedings of ECOOP'96 Workshop on System and Network Management*, julho

Bauer, M.A. et all. (1994) Reference architecture for distributed systems management. *IBM Systems Journal vol 33, no 3*, pp 426-444.

Bernstein, P. A. (1996) Middleware: A Model for Distributed System Services. *Communications of the ACM Magazine*, vol 39, no 2, pp 87-98, fevereiro.

Becker, K.; Raabe, U.; Sloman, M.; e Twidle, K. (1993) Domain and Policy Service Specification. *IDSM Deliverable D6, SysManDeliverable MA2V2.*

BRISA (1993) Gerência de Redes - Uma abordagem de Sistemas Abertos. Ed. Makkron Books.

Case J., Fedor M. e Davin J. (1990) A Simple Network Management Protocol (SNMP). *RFC 1157 Network Working Group*, maio.

Cerqueira, R. e Ierusalimshy, R. (1996) Uma avaliação das arquiteturas para interoperabilidade entre objetos. *Anais do X Simpósio Brasileiro de Engenharia de Software*, pp 371-386, 1996.

Chadha, R. e Wuu, S. (1996) Managing Distributed Systems using OSI Systems Management. *Proc. Of Second IEEE Workshop on System Management*, pp 117-126, junho.

Cohen, R. S. (1994) The Telecommunications Management Network (TMN) . *Network and Distributed Systems Management (Ed. Morris Sloman)*, Cap. 9; Addison-Wesley; pp 217-243.

Costa, F.M. e Madeira, E.R.M. (1996) An object group model and its implementation to support cooperative applications on CORBA. *Distributed Platforms (*Ed. A. Schill, C. Mistach, O. Spaniol and C. Popien), Chapman & Hall, pp 213-227.

Dittrich, A.; Rasmussen, S. e O'Sullivan, D. (1997) Co-existence of TMN and CORBA for Service Management. *Proc. IEEE Third International Symposium on Autonomous Decentralized*, pp 35-42, abril.

Farooqui, K. (1995). OSI Management in the ODP Architectural Framework. *Proc. of the Sixth IEEE/IFIP International Workshop on Distributed Systems: Operation and Management Workshop (DSOM'95)* - pp 1-13, outubro.

Farooqui, K. (1997). Reference Model for Open Distributed Processing (ODP) - A Guided Tour. Tutorial "E", *IFIP Joint International Conference on Open Distributed Processing & Distributed Platforms (ICODP/ICDP'97)*, maio.

Feldkhun, L; Marini, M; Borioni, S. (1997). Integrated Customer-Focused Network Management: Architectural Perspectives. *Integrated Network Management V*, Chapman & Hall, pp 16-30.

Flanagan, D. (1996). Java in a Nutshell - A Desktop Quick Reference for Java Programmers. O'Reilly & Associates, Inc.

Friedrich, R.; Martinika, J.; Sienknecht, T. e Saunders, S. (1995a) Integration of Performance and

Modelling for Open Distributed Processing. *Proc. of International Conference on Open Distributed Processing (ICODP'95)*, pp 341-352, fevereiro.

Friedrich, R.; Saunders, S.; Zaidenweber, G.; Bachman, D. and Blumson, S. (1995b) Standardized Performance Instrumentation and Interface Specification for Monitoring DCE Based Applications. *OSF DCE RFC33.0,* maio.

Gering, M. (1994).Comparison of SNMP and CMIP Management Architectures. *Network and Distributed Systems Management (Ed. Morris Sloman)*, Cap. 8; Addison-Wesley; pp 197-216.

Goulart, C.C.; Nogueira, J.M.S. (1995) Utilização do Modelo TMN no Gerência de Redes TMN. *XIII Simpósio Brasileiro de Redes de Computadores*, pp 389-408, maio.

Harrison, T. H.; Levine, D. L. e Schmidt, D. C. (1997) The Design and Performance of a Real Time CORBA Object Event Service. *Proc of OOPSLA'97.* , outubro.

Hegering, H.G. and Abeck, S. (1994) Integrated Network and System Management. *Data Communication and Network Series.* Addison-Wesley.

Hoffner, Yigal. Monitoring in Distributed Systems. *ANSA Phase III APM 1018.01*.outubro.

Iona Technologies Ltd. (1995) Orbix 2 programming and reference guide, release 2.0.

Iona Technologies Ltd. (1996) OrbixTalk programming guide.

Iona Tecnologies Ltd (1997) OrbixWeb programming and reference guide, release 2.0.

ITU-T Rec X901/2/3 | ISO/IEC 10746-1/2/3 (1995a) ODP Reference Model. Part 1. Overview and Guide to use; Part 2. Foundations; Part 3. Architecture.

ITU-T | ISO/IEC (1995b) Open Distributed Management Architecture, Working Draft 3, novembro.

Lima, F. H. S. e Madeira, E. R. M (1997) Qualidade de Serviço Multinível baseada em ATM para a Plataforma Multiware. *Anais do XV Simpósio Brasileiro de Redes de Computadores*, pp 366-382, maio.

Lima, L.A.P. e Madeira, E.R.M. (1995) A model for a Federative Trader. *Proc. of International Conference on Open Distributed Processing (ICODP'95)*, pp 155-166, fevereiro.

Loyolla, W.; Madeira, E.R.M.; Cardozo, E.; Magalhães, M.F. e Mendes, M.J. (1994) Multiware Platform: An open distributed environment for multimedia cooperative applications. *IEEE COMPSAC'94*, novembro.

Mansouri-Samani, M. e Sloman, M. (1997). An event service for open distributed systems. *Open Distributed Processing and Distributed Platforms. (Ed. J. Rolia, J. Slonin e J. Botsford).* Chapman & Hall., pp 183-194.

Mansouri-Samani, M. e Sloman, M. (1994). Monitoring Distributed Systems. *Network and Distributed Systems Management (Ed. Morris Sloman)*, Cap. 12; Addison-Wesley; pp 303-344.

Mansouri-Samani, M. e Sloman, M. (1993). Monitoring Distributed Systems. *IEEE Network magazine*, pp 20-30, 7(6), novembro.

Marriot, D; Mansouri-Mansani, M; e Sloman, M. (1994) Specification of Management Policies. *Proceedings of the Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'94).*

McCloghrie, K. e Rose, M. (1991) Management Information Base for Network Management of TCP/IP-based internets: MIB - II. *RFC 1213 Network Working Group*, março.

Meira, D. M. e Nogueira, J. M. S. (1997). Métodos e Algoritmos para Correlação de Alarmes em Redes de Telecomunicações. *XV Simpósio Brasileiro de Redes de Computadores*, pp 79-98, maio.

Negroponte, N. (1995) A Vida Digital, *Ed. Companhia das Letras.*

OMG (1997a) CORBAfacilities Specification - Full Book, OMG formal/ 97-06-15, junho.

OMG (1997b) CORBAservices Specification - Full Book, OMG formal/97-06-01, junho.

OMG (1997c) CORBA 2.0 IIOP Specification, OMG formal /97-02-25, fevereiro.

OMG (1997d) Notification Service RFP, OMG /telecom 97-01-03, janeiro.

OMG (1996a) TINA notification service description, OMG /telecom 96-07-02, julho.

OMG (1996b) Description of New OMA Reference Model, draft1, OMG /ab96-05-02, maio.

OMG (1995) OMG Common Facilities Architecture, Revision 4.0, OMG /TC95 95-01-02, janeiro.

Orfali, R.; Harkey, D.; Edward, J. (1996). The Essential Distributed Objects - Survival Guide. John Wiley & Sons, Inc.

Queiroz, J. A. G. e Madeira, E .R. M (1997a). Facilidade de Monitorização Assíncrona em um ambiente de gerência CORBA. *Anais do 2º Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos/2 ème Seminaire Franco-Bresilien sur les Architecture des Systemes Distribues*, pp 135-146, novembro.

Queiroz, J. A. G. e Madeira, E .R. M (1997b) Management of CORBA objects monitoring for the Multiware platform. *Open Distributed Processing and Distributed Platforms. (Ed. J. Rolia, J. Slonin e J. Botsford).* Chapman & Hall. pp 122-133.

Queiroz, J. A. G. e Madeira, E. R. M (1997c) Gerência de Objetos CORBA para a plataforma Multiware. *Anais do XV Simpósio Brasileiro de Redes de Computadores*, pp 432-444, maio.

Rodriguez, N.; Cerqueira, R.; Ierusalimschy, R. e Moura, A. L. (1997) Aplicações de gerência com comportamento dinâmico. *Anais do 2º Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos/2 ème Seminaire Franco-Bresilien sur les Architecture des Systemes Distribues,* pp 476-487, novembro.

Rolia, J.A. (1993) Distributed Application Performance, Metrics and Management. *Open Distributed Processing II (Ed. J. de Meers, B. Mahr e S. Storp)*, Elsevier Science B. V. (North-Holland), pp 235-246.

Rolia, J.A. et all (1995) Distributed Application Management - The MANDAS Project. *Proc. of the Sixth IEEE/IFIP International Workshop on Distributed Systems: Operation and Management Workshop (DSOM'95)*, pp 1-6.

Rose, M. e McCloghrie, K. (1990) Structure and Identification of Management Information for TCP/IP-based Internets. *RFC 1155 Network Working Group*, maio.

Rumbaugh, J.; Blaha, Michael; Premerlani, W.; Eddy, F.; Lorensen, W.; (1991) Object-Oriented a Modelling and Design, Prentice-Hall.

Schade, A. (1997) An Event Framework for CORBA-Based Monitoring and Management Systems. *Open Distributed Processing and Distributed Platforms. (Ed. J. Rolia, J. Slonin e J. Botsford).* Chapman & Hall. pp 210-223.

Schade, A.; Trommler, P. and Kaiserswerth, M. (1996) Object Instrumentation for Distributed Applications Management. *Distributed Platforms (*Ed. A. Schill, C. Mistach, O. Spaniol and C.

Popien), Chapman & Hall, pp 173-185.

Sloman, M.; Magee, J.; Twidle, K. and Krammer, J. (1993) An Architecture for Managing Distributed Systems. *Proc. 4th IEEE Workshop on Future Trends of Distributed Computing Systems,* pp 40-46.

Sloman, M. (1995) Management Issues for Distributed Services. *Proc. IEEE SDNE'95*, pp 52-59, junho.

Soukoti, N. e Hollberg, Ulf (1997) Joint Inter Domain Management: CORBA, CMIP and SNMP. *Integrated Network Management V*, Chapman & Hall, pp 153-163.

Stallings, W. (1996) SNMP, SNMPv2, and RMON: Practical Network Management - Second Edition. Ed. Addison Wesley.

Stallings, W. (1993) SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards. Ed. Addison Wesley.

Sun Microsystems Computer Corporation, (1996) Java Remote Method Invocation Specification, dezembro.

Tanenbaun, A. (1996) Computer Networks - Third Edition. *Prentice Hall PTR.*

Tanenbaun, A. (1995) Distributed Operating Systems. *Prentice Hall PTR.*

TINA-C (1995) Overall Concepts and Principles of TINA, version 1.0, *http://www.tinac.com,* fevereiro.

Üslander, T. and Brunne, H. (1996) Management View upon CORBA Clients and Servers. *Proc. of ICDP'96,* Industrial and Poster Session, pp 165-169.

Vinoski, S. (1997) CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, vol 14, n. 2, fevereiro.

Vogel, A. e Duddy, K. (1997) Java Programming with CORBA. John Wiley & Sons, Inc.

X/Open Company (1995) System Management: Common Management Facilities, Volume 1, Version 2 - OMG 95-12-05, dezembro.

Yemini, Y. (1993) The OSI Network Management Model. *IEEE Communications Magazine*, pp 20-29, 31(5), maio.

Waldbusser, S. (1991) Remote Monitoring Management Information Base, *RFC 1271 Internet Workin Group*, julho.