

Este exemplar é a cópia final da
Tese / Dissertação, já corrigida e
defendida por Patricia
Ropelatto
e aprovada pela Banca Examinadora.
Campina, 29 de Julho de 1999

COORDENADOR DE GRADUAÇÃO
E. INC

**Gerência de Monitorização de Sistemas
Distribuídos em um Ambiente CORBA
usando Agentes Móveis**

Patricia Ropelatto

Dissertação de Mestrado

Gerência de Monitorização de Sistemas Distribuídos em um Ambiente CORBA usando Agentes Móveis

Patricia Ropelatto¹

18 de junho de 1999

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. Célio Cardoso Guimarães (Suplente)
Instituto de Computação - UNICAMP

¹Este trabalho foi parcialmente financiado pelo CNPq e FAPESP, processo 96/09899-9.



UNIDADE	BC
N.º CHAMADA	
	38577
	229/99
PREÇO	R\$ 11,00
DATA	28/08/99
N.º CPD	

CM-00125848-4

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Ropelatto, Patricia

R681g Gerência de monitorização de sistemas distribuídos em um ambiente CORBA usando agentes móveis / Patricia Ropelatto – Campinas, [S.P. :s.n.], 1999.

Orientador : Edmundo Roberto Mauro Madeira

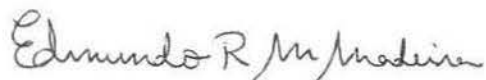
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação .

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

Gerência de Monitorização de Sistemas Distribuídos em um Ambiente CORBA usando Agentes Móveis

Este exemplar corresponde à redação final da
Dissertação devidamente corrigida e defendida
por Patricia Ropelatto e aprovada pela Banca
Examinadora.

Campinas, 18 de junho de 1999.



Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

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

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 18 de junho de 1999, pela Banca Examinadora composta pelos Professores Doutores:

Noemi de la Rocque Rodriguez

Prof. Dra. Noemi De La Rocque Rodriguez
PUC-Rio

Paulo Lício de Geus

Prof. Dr. Paulo Lício de Geus
IC- UNICAMP

Edmundo Roberto Mauro Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP

*Dedico este trabalho aos meus pais e às
minhas irmãs pelo amor e incentivo
dispensados ao longo de minha vida.*

Agradecimentos

Agradeço ao Papai do Céu que sempre esteve ao meu lado em todos os momentos da minha vida, dando-me força, luz e coragem para enfrentar as tantas dificuldades encontradas.

Aos meus pais Antonio e Irma, exemplos de vida que tenho para mim, que em meio a TANTAS dificuldades em suas vidas conseguiram de maneira sábia possibilitar estar onde estou. Tudo que aprendi, tudo que sou devo a eles. Muito obrigada pai e mãe, Eu Amo Vocês! Às minhas irmãs Cristiana e Franciane pelo carinho, pela força, pelas conversas, pelas brincadeiras, obrigada. Também Amo Muito Vocês!

À minha amada “nona” Giuseppina Ropelatto (*in memoriam*) que sempre torceu muito por mim, encorajou-me a nunca desistir dos meus sonhos e quem eu sei que é um anjo em minha vida.

Ao meu querido orientador Edmundo Roberto Mauro Madeira, que me guiou por este caminho do mestrado tão difícil com suas conversas, brincadeiras, paciência, compreensão, e com quem aprendi muito, não só sobre os assuntos acadêmicos, mas principalmente sobre a vida. Muito obrigada! Ao senhor minha eterna amizade!

Agradeço de todo coração a todos os meus amigos catarinenses que me incentivaram e me apoiaram muito e que me ajudaram a chegar até aqui, entre eles: Christiane Zappellini, Cristine, Darli, Simone e Sérgio, Tina (orientadora de língua portuguesa), Verônica, Vitor, Wagner, Beth e Marta (minhas eternas orientadoras). Aos amigos da minha querida cidade “Rio do Sul” por toda força e carinho, entre eles: Tânia (quem vai ter que me agüentar por bons meses), Marilene (Lene), Cátia, Naisa, Eneide, Juliana, Daniela, Taciane, Tatiane, Rita, Douglas, Bentinho, Luli e Facca.

Aos N’s da minha vida (e respectivas famílias), pela grande amizade, por tanto que aprendi, por tantos momentos maravilhosos que passei e principalmente por me fazerem conhecer uma outra Patricia alegre e capaz de muita coisa: Nalú, Nayane, Nayara e Norma.

À toda família Andrade, principalmente Lauro, Eneuza, Laurinho, Alcides e Ana, por todo carinho, força e amizade que sempre demonstraram comigo. Muito obrigada por tudo!

Agora, àqueles amigos que conquistei ao longo destes anos de mestrado. À Turma

*as lágrimas e as alegrias,
eu te admirei no rosto sereno do meu amigo.
Graças te dou, meu Deus,
porque te revelas
em gestos tão humanos
que posso experimentar-te sempre
na pessoa deste amigo que me ama!
Faze que ele seja muito feliz
e que eu te encontre sempre
na transparência de nossa amizade. Amém.*

Adriana Zuchetto

Resumo

Os modelos tradicionais de gerência de redes e sistemas distribuídos utilizam agentes estáticos. Este trabalho propõe uma nova abordagem explorando a mobilidade de agentes em um ambiente OMG/CORBA para a plataforma *Multiware* desenvolvida na Universidade Estadual de Campinas. Uma grande vantagem dos agentes móveis sobre os agentes estáticos é a de evitar a sucessão de requisições e respostas na rede tornando a gerência mais flexível e eficiente. Além disso, é dada aos mesmos uma certa autonomia para agirem em benefício dos seus gerentes promovendo uma descentralização parcial da gerência.

Nesta dissertação, a informação de gerência é obtida através do conceito de **Instrumentação** definido pelas *CORBAfacilities*. Com base neste conceito, foi utilizada uma estrutura de filtragem de dados associada aos objetos distribuídos CORBA. Um conjunto de agentes foi definido com a finalidade de coletar informações nas áreas de contabilização e desempenho em três níveis de detalhamento sucessivos: *host*, processo e objeto.

Abstract

Traditional management models of networks and distributed systems use static agents. This work proposes a new approach which explores the mobility of agents in an OMG/CORBA environment for the Multiware platform developed at University of Campinas. A strong advantage of the mobile agents against the static ones is to avoid successive requests and responses in the network what makes the management more flexible and efficient. Besides, a relative autonomy is given to the agents what allows them to act in benift of their managers and to promote the partial management decentralization.

In this dissertation, the management information is obtained through the concept of **Instrumentation** defined by the CORBA facilities. Based on this concept, a data filtering structure was used associated to the distributed CORBA objects. A pool of agents was defined to gather information with regards to accounting and performance in three levels of successive detailing: host, process and object.

Conteúdo

Agradecimentos	v
Resumo	viii
Abstract	ix
1 Introdução	1
1.1 Motivação	2
1.2 Objetivo	2
1.3 Estrutura da Dissertação	3
2 CORBA e Gerência de Sistemas Distribuídos	4
2.1 CORBA	4
2.1.1 O OMG e sua Arquitetura de Gerência de Objetos	4
2.1.2 IDL (<i>Interface Definition Language</i>)	6
2.1.3 A estrutura de um ORB	7
2.1.4 Implementações de ORB	9
2.1.5 Considerações finais de CORBA	9
2.2 Gerência de Sistemas Distribuídos	10
2.2.1 Modelo de Gerência OSI/CMIP	13
2.2.2 Modelo de Gerência Internet/SNMP	15
2.2.3 CORBA como um Modelo de Gerência	17
2.2.4 Comparação entre os Modelos de Gerência	21
2.3 Plataforma Multiware	23
2.4 Resumo	26
3 Conceitos de Agentes Móveis	27
3.1 Definições e Características de Agentes	27
3.2 Tecnologia de Agentes	29
3.3 Agentes Móveis	32

3.3.1	Aplicações de Agentes Móveis	33
3.3.2	Vantagens, Desafios e Desvantagens	34
3.3.3	Requerimentos de Infra-estrutura para Agentes Móveis	35
3.3.4	Sistemas de Agentes Móveis	36
3.4	Mobilidade e CORBA	40
3.5	Gerência utilizando Agentes Móveis	42
3.6	Resumo	43
4	Modelagem Proposta para a Gerência de Monitorização	44
4.1	Integrando CORBA e Agentes Móveis para a Gerência	44
4.2	Trabalhos Relacionados	45
4.3	Informações de Gerência	46
4.4	Instrumentação no Ambiente CORBA	50
4.5	Agentes e Filtros	54
4.6	Interface para Gerência	56
4.7	Persistência do Estado dos Filtros	58
4.8	Cenários de Pools de Agentes Móveis para a Gerência	58
4.9	Resumo	61
5	Aspectos de Implementação	62
5.1	Ambiente de desenvolvimento	62
5.2	O ORB OrbixWeb	63
5.2.1	A estrutura de Filtragem	65
5.3	O Sistema de Mobilidade utilizado	70
5.4	Infra-estrutura de Gerência	73
5.4.1	Implementação e Ativação/Desativação dos Filtros	73
5.4.2	Considerações sobre a Gerência	75
5.4.3	Persistência do Estado dos Filtros	76
5.4.4	Agentes Móveis para Gerência	76
5.5	Resumo	81
6	Conclusão	82
6.1	Contribuições da Modelagem	82
6.2	Contribuições da Implementação	83
6.3	Trabalhos Futuros	84
	Bibliografia	86

A	Descrição de Alguns Sistemas de Agentes Móveis	94
A.1	Telescript	94
A.2	AgentTcl	95
A.3	Aglets Workbench	96
A.4	Voyager	99
B	Estrutura completa de um filtro por processo	102
C	Código da Implementação	104

Lista de Tabelas

2.1	Comparação entre os modelos de gerência	24
4.1	Filtros modelados para a gerência	53
4.2	Classes de Agentes	54
5.1	Filtros e seus pontos de filtragem	74
5.2	Amostragem de dados coletados por um agente em largura	78
5.3	Tabela para níveis de decisão para ocupação de CPU, memória e throughput	80
5.4	Tabela do relatório retornado pelo agente agL1	80
5.5	Tabela do relatório retornado pelo agente agP1	81

Lista de Figuras

2.1	Arquitetura de Gerência de Objetos do OMG	5
2.2	A estrutura do ORB	7
2.3	Eixos ortogonais de gerência	11
2.4	CORBA como um modelo de gerência	22
2.5	Plataforma <i>Multiware</i>	25
3.1	Classificação de agentes segundo Nwana	31
3.2	RPCs para comunicação entre componentes de sistemas cliente/servidor . .	31
3.3	Migração de um agente móvel para utilização de recursos remotos	32
3.4	Migração de um serviço para o <i>host</i> cliente	32
4.1	Agentes móveis no modelo de gerência CORBA	45
4.2	Extensões de gerência de um objeto	47
4.3	Papéis de Cliente e Servidor	49
4.4	Pontos de obtenção das métricas para cliente e servidor	52
4.5	Agente de Ativação e Agente em Largura em dois <i>hosts</i>	59
4.6	Agente de Ativação/Desativação e Agente em Largura em três <i>hosts</i>	60
4.7	Agente de Ativação/Desativação e Agente em Profundidade em um <i>host</i> . .	60
5.1	O <i>daemon</i> do <i>OrbixWeb</i>	64
5.2	Estrutura de filtragem do <i>OrbixWeb</i>	66
5.3	Filtros por processo (a) e por objeto (b)	67
5.4	Modelagem do Sistema de Agentes Móveis da Plataforma <i>Multiware</i>	71
A.1	Estrutura de um <i>Aglet</i>	97

Acrônimos

ACSE	Association Control Service Element
ANSA/APM	Advance Networked Architecture/Architecture Projects Management
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BOA	Basic Object Adapter
CCITT	Consultative Committee for International Telegraph and Telephone
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
CORBA	Common Request Broker Architecture
CSCW	Computer Supported Cooperative Work
DCE/OSF	Distributed Computing Environment/Open Software Foundation
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DMI	Definition of Management Information
DSI	Dynamic Skeleton Interface
ECRC	European Computer-Industry Research Center
FDDI	Fiber Distributed Data Interface
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Inter-ORB Protocol
GMI	Generic Management Information
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIMC	ISO-Internet Management Coexistence
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ISO	International Organization for Standardization
ITU-T	International Telecommunications Union - Telecommunication Standard Sector

JDMK	Java Dynamic Management Kit
JIDM	Joint Inter-Domain Management
JVM	Java Virtual Machine
MAF	Mobile Agent Facilities
MASIF	Mobile Agent System Interoperability Facilities Specification
MIB	Management Information Base
MIM	Management Information Model
MIT	Management Information Tree
MPD	Message Processing and Dispatch
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
ORB	Object Request Broker
OSI	Open Systems Interconnection
PDU	Protocol Data Units
POA	Portable Object Adapter
RFI	Request For Information
RFP	Request For Proposal
RM-ODP	Reference Model for Open Distributed Processing
RMI	Remote Method Invocation
ROSE	Remote Operations Service Element
RPC	Remote Procedure Call
SMI	Structure of Management Information
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
USM	User-based Security Model
WWW	World Wide Web

Capítulo 1

Introdução

Os grandes avanços tecnológicos tem tornado os sistemas distribuídos cada vez maiores, fazendo com que a sua gerência seja um tema atual e complexo, ampliando assim a área de gerência de redes.

Na gerência de sistemas distribuídos é necessário analisar uma grande quantidade e variedade de informações, tanto do próprio ambiente quanto das aplicações que estão sendo executadas sobre ele. Entretanto, a manipulação (localizar, acessar, coletar, etc) dessas informações nestes ambientes é difícil. Os modelos de gerência tradicionais gerente/agente centralizados começam a não ser mais tão eficientes. A descentralização da gerência vem de encontro à descentralização das informações pelas redes e sistemas distribuídos que permite o compartilhamento de informações e recursos, aumentando a disponibilidade e a confiabilidade obtendo uma melhor relação entre custo e desempenho.

Visando resolver o problema da heterogeneidade e da distribuição, vários desenvolvedores passaram a oferecer serviços que operavam com protocolos e interfaces de programação padrões. Estes serviços foram chamados de serviços de *middleware* por se situarem no meio (*middle*), ou seja, acima da camada do sistema operacional e software de rede e abaixo das aplicações. O núcleo do *middleware* é o serviço de comunicação inter-processos. O OMG (*Object Management Group*) com sua arquitetura OMA (*Object Management Architecture*) orientada a objetos possui como base um ORB (*Object Request Broker*) que é especificado através da CORBA (*Common Request Broker Architecture*). O ORB atua como um barramento de objetos. Os objetos que se ligam ao ORB são classificados pela OMA de acordo com os serviços prestados por eles.

Segundo Orfali et al. [OHE96], um sistema de gerência é limitado pelas capacidades de seu protocolo de gerência e pelos objetos usados para representar o ambiente gerenciado. Os protocolos tradicionais SNMP (*Simple Network Management Protocol*) da Internet e CMIP (*Common Management Information Protocol*) do modelo OSI (*Open Systems Interconnection*), sendo SNMP o mais difundido hoje, têm a finalidade de gerência de

redes (componentes e protocolos) e a busca por novas tecnologias que abordem também a gerência de sistemas distribuídos começou a despertar interesses de muitos pesquisadores.

Desta forma, CORBA começou a ganhar espaço na área de gerência de sistemas distribuídos e redes como uma nova tecnologia de objetos distribuídos. O surgimento da linguagem Java também trouxe muitos avanços nesta área. Por ser uma linguagem orientada a objetos e com várias bibliotecas prontas e voltadas para o desenvolvimento de aplicações distribuídas, é muitas vezes apontada como uma forte concorrente de CORBA. Um dos projetos desenvolvidos pela Sun *Microsystems* é o Java RMI (*Remote Method Invocation*) que permite invocações remotas a objetos. Embora Java ofereça a independência de plataformas, não se trata de uma tecnologia integradora como CORBA.

1.1 Motivação

Com tamanho crescimento de heterogeneidade dos sistemas distribuídos, o trabalho de um operador/gerente numa rede tornou-se mais complexo. Muitas informações redundantes e até mesmo não tão importantes chegam a um gerente que acaba por não gerenciar devidamente a rede. Mecanismos de gerência devem sempre ser aprimorados e atender às necessidades dos usuários.

A tecnologia de objetos distribuídos também começa tomar lugar nos sistemas distribuídos e, conseqüentemente, CORBA é uma plataforma bastante adequada para gerência. O protocolo SNMP atendeu durante muito tempo, de forma eficaz, a gerência de redes. Mas, hoje, o paradigma tradicional gerente/agente fornecido pelos modelos tradicionais de gerência não traz muita eficiência e flexibilidade aos sistemas. Assim, a possibilidade de se migrar um agente para execução das tarefas de gerência parece ser um caminho de grandes progressos. Usando agentes móveis, a eficiência na gerência de sistemas distribuídos pode ser aumentada consideravelmente, coletando e filtrando informações relevantes para a gerência, alocando dinamicamente funcionalidades e informações de gerência.

A idéia de usar a tecnologia dos agentes móveis ao invés de agentes estáticos traz uma grande vantagem que é a de evitar a sucessão de requisições e respostas através da rede. A mobilidade do agente faz com que o código seja levado para mais perto dos dados. Isto implica que os agentes de gerência devem ser leves.

1.2 Objetivo

Este trabalho tem como objetivo monitorar as atividades dos objetos distribuídos CORBA nos *hosts*, no contexto da plataforma *Multiware*. Esta tarefa é desempenhada por agentes móveis que coletam informações de gerência. Entende-se por gerência de monitorização,

a gerência aplicada em duas áreas funcionais definidas pelo modelo OSI: contabilização e desempenho.

Os objetos distribuídos podem estar agrupados dentro de processos, notadamente quando estes objetos executam sobre sistemas operacionais orientados a processos. Desta maneira, três níveis de abordagens de informação são considerados: *host*, processo e objeto. Isto possibilita que os agentes móveis possam fazer *zooms* sucessivos de coleta de informações.

A tarefa do operador/gerente com esta abordagem pode então ser automatizada detectando possíveis *hosts* críticos. O tráfego na rede pode ser reduzido devido a uma quantidade menor de informações de gerência circulando pela rede.

Neste trabalho, CORBA é o modelo de gerência utilizado e o ambiente a ser gerenciado. Para que os agentes possam se mover pelo ambiente, um sistema de agentes móveis também é necessário.

1.3 Estrutura da Dissertação

O Capítulo 2 apresenta a arquitetura OMA com todos os seus serviços e partir desta a plataforma CORBA. Os dois modelos de gerência Internet/SNMP e OSI/CMIP são descritos brevemente, pois já são bastante conhecidos e discutidos na literatura. CORBA é apresentada e discutida como um modelo de gerência numa comparação ao modelo OSI. Em seguida é feita uma comparação entre os três modelos apresentados e mais o Java RMI. Por fim, é apresentada a plataforma *Multiware*.

O Capítulo 3 cobre os conceitos relacionados a agentes, mais especificamente, a classe de agentes móveis e apresenta alguns sistemas de agentes móveis. A forma como o OMG aborda a Facilidade de Agentes Móveis e a característica de Passagem de Objetos por Valor também é discutida. Este capítulo apresenta a nova abordagem da gerência utilizando agentes móveis.

No Capítulo 4, a proposta deste trabalho é detalhada descrevendo como é possível instrumentar um ambiente CORBA utilizando métricas e uma estrutura de filtragem, onde são modelados filtros. Após definida esta base para a gerência, alguns agentes móveis são sugeridos, baseados nos filtros, e uma interface de gerência é apresentada.

O Capítulo 5 discute como foi feita a implementação baseada na modelagem do Capítulo anterior. São mostrados alguns resultados obtidos da execução de um cenário de agentes móveis para a gerência. Finalmente, através do Capítulo 6 este trabalho é concluído apresentando contribuições e sugestões para trabalhos futuros.

Capítulo 2

CORBA e Gerência de Sistemas Distribuídos

Este capítulo descreve brevemente a arquitetura CORBA e os modelos de gerência de redes, aplicados também a sistemas distribuídos, mais conhecidos: SNMP e CMIP. Discute também CORBA como um modelo de gerência de sistemas distribuídos, o qual é utilizado neste trabalho, e apresenta uma comparação entre os modelos de gerência. Finalmente a plataforma *Multiware* é comentada.

2.1 CORBA

CORBA é uma plataforma especificada pelo OMG em resposta à necessidade de interoperabilidade, devido à rápida proliferação de produtos de *hardware* e *software* disponíveis hoje. A *Microsoft* criou seu próprio negociador de objetos chamado DCOM (*Distributed Component Object Model*), mas CORBA é considerada, em [OH98], como o projeto *middleware* mais importante e ambicioso no mundo da indústria. Ela é baseada no paradigma cliente/servidor e permite a comunicação entre objetos num ambiente distribuído heterogêneo, sem considerar onde estão localizados ou como foram projetados. Características da especificação CORBA como escalabilidade, heterogeneidade, interoperabilidade, flexibilidade, entre outras, serão apresentadas no decorrer das próximas seções.

2.1.1 O OMG e sua Arquitetura de Gerência de Objetos

Em 1989, o OMG foi criado com o propósito de promover a teoria e prática da tecnologia de objetos em sistemas computacionais distribuídos. Seus objetivos são: ajudar a reduzir a complexidade, diminuir os custos e acelerar a introdução de novas aplicações de *software* baseadas em objetos distribuídos. Este grupo é formado por mais de 800 membros,

entre eles indústrias, centros de pesquisas e universidades, e preocupa-se em desenvolver especificações — e não *software* — através de idéias e tecnologias de seus membros que respondem a RFIs (*Request For Information*) e RFPs (*Request For Proposal*). Dessa maneira, o OMG estabeleceu a OMA que fornece a infra-estrutura conceitual sobre a qual todas as especificações do OMG estão baseadas. Esta arquitetura consiste de cinco componentes que podem ser divididos em três segmentos (Figura 2.1): orientado ao sistema (ORB e Serviços de Objetos), orientado à aplicação (Objetos de Aplicação e Interfaces de Domínios) e orientado ao mercado vertical (Interfaces de Domínios e Facilidades Comuns).

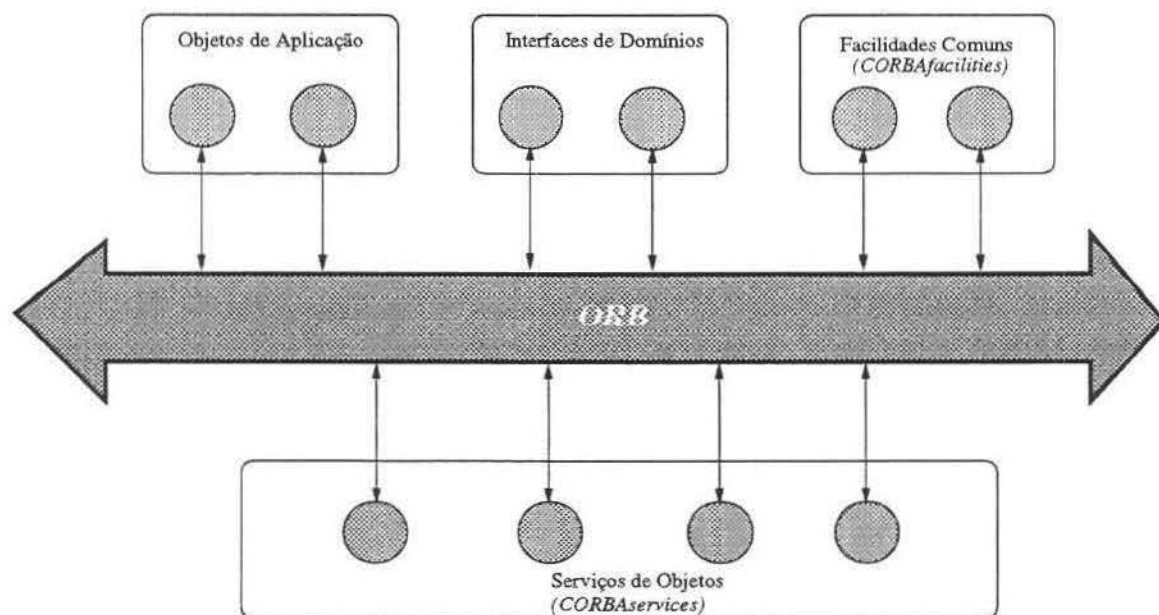


Figura 2.1: Arquitetura de Gerência de Objetos do OMG

Independente de qual segmento predomine, todas as comunicações entre os componentes são gerenciados pelo ORB, que é o fundamento da OMA. A tecnologia ORB é uma abordagem muito importante para soluções de computação heterogêneas, distribuídas e abertas.

Definições dos Componentes da OMA

- ORB: é o centro de comunicação do padrão CORBA, o qual fornece uma infra-estrutura permitindo que objetos conversem independentemente de plataformas específicas e técnicas usadas para implementá-los. O ORB garante portabilidade e interoperabilidade de objetos sobre uma rede de sistemas heterogêneos.
- Serviços de Objetos: chamados pelo OMG de *CORBAservices* [OMG97a], padronizam a gerência do ciclo de vida dos objetos definindo uma coleção de serviços

(interfaces e objetos) que oferecem funções básicas para uso e implementação de objetos. Fornecem consistência das aplicações e ajudam a aumentar a produtividade do programador. Existem quinze serviços de objetos publicados pelo OMG: nomes, eventos, ciclo de vida, persistência, relacionamento, externalização, transação, controle de concorrência, licença, consulta, propriedades, segurança, tempo, coleções e *trader*. Todos os serviços de objetos, com exceção da persistência, estão padronizados.

- **Facilidades Comuns:** publicadas pelo OMG como *CORBAfacilities* [OMG97b] fornecem um conjunto de funções de aplicações genéricas que podem ser configuradas para requisitos específicos de uma configuração particular e, por isso, sempre estão em expansão. Estas facilidades se aproximam mais do usuário. Exemplos de facilidades comuns são: banco de dados, comércio eletrônico, gerência de sistemas e agentes móveis.
- **Interfaces de Domínios:** são interfaces de aplicações específicas como telecomunicações, manufaturas e objetos de negócios. Podem combinar algumas facilidades comuns e serviços de objetos, mas são projetadas para fazer tarefas particulares para usuários dentro de um mercado ou indústria. Até 1997, esta área de padronização fazia parte das *CORBAfacilities* e era chamada de Facilidades Verticais.
- **Objetos de Aplicação:** são interfaces desenvolvidas para as aplicações específicas dos usuários e, por este motivo, não são padronizadas pelo OMG. Beneficiam grandemente o desenvolvimento de sistemas de objetos robustos.

2.1.2 IDL (*Interface Definition Language*)

IDL é uma linguagem puramente declarativa para definição de interfaces de objetos CORBA; logo, não provê detalhes de implementação. Das descrições de interfaces IDL, um produto ORB automaticamente gera código em uma linguagem suportada pela CORBA (atualmente C, C++, *Smalltalk*, *Ada*, COBOL e Java) para realizar a integração e distribuição — uma espécie de “cola” que conecta componentes e gerencia a comunicação entre eles. Dessa forma, objetos cliente e servidor escritos em linguagens diferentes podem interoperar através de redes de computadores e sistemas operacionais.

A sintaxe de IDL é extraída de C++, possuindo algumas palavras-chaves adicionais para dar suporte à distribuição. Não existem comandos de programação, tão somente seu propósito é definir interfaces. Para isto, as construções abaixo são suportadas:

- **Constantes:** participam da declaração de tipos;
- **Declarações de tipos de dados:** determinam os tipos de parâmetros;

- Atributos: permitem alterar e conseguir um valor de um tipo particular;
- Operações: podem ter parâmetros e valores de retorno;
- Interfaces: agrupam os tipos de dados, atributos e operações e permitem o uso de herança simples e múltipla;
- Módulos: definem escopos de nomeação.

Todas as declarações feitas em IDL podem estar disponíveis através do Repositório de Interfaces, que será apresentado durante a próxima seção.

2.1.3 A estrutura de um ORB

O ORB é o *middleware* que estabelece o relacionamento entre objetos e sua especificação é dada pela CORBA. Esta arquitetura é composta pelo núcleo do ORB, as interfaces de chamadas deste e pelos lados cliente e servidor (Figura 2.2). No mundo CORBA, um objeto é um componente de programa que possui uma interface definida em IDL. O cliente é a entidade que acessa e invoca operações sem saber onde os objetos residem, qual a linguagem de programação ou qualquer outro aspecto que não esteja refletido na interface do objeto. A implementação de objetos, conhecida como servidor ou ainda *servant* segundo Schmidt [Sch98] é o código e os dados que realmente implementam o objeto.

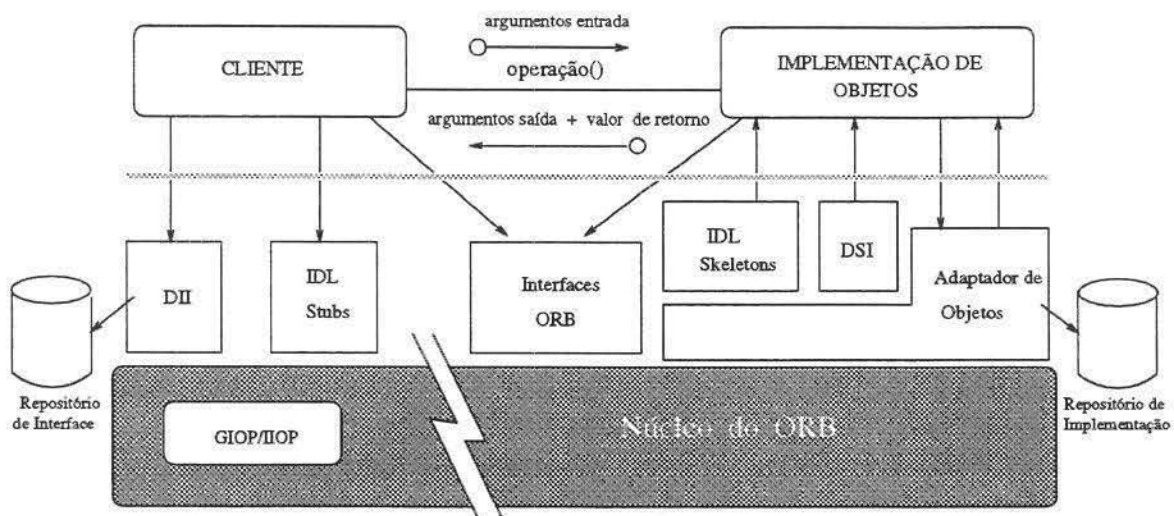


Figura 2.2: A estrutura do ORB

O núcleo do ORB é responsável por receber as requisições dos clientes através da interface estática (IDL *Stubs*) ou da Interface de Invocação Dinâmica (DII - *Dynamic Invocation Interface*) e localizar a implementação de objetos apropriada, transmitindo os parâmetros, invocando os métodos através de uma interface estática (IDL *Skeletons*) ou da Interface de Esqueleto Dinâmica (DSI - *Dynamic Skeleton Interface*) e retornando os resultados.

Quanto ao lado cliente, os *Stubs* são gerados pelo compilador IDL e incluem código para executar a serialização, que empacota (*marshal*) a operação e seus parâmetros para formatos de mensagens que poderão ser enviados ao servidor. A Interface de Invocação Dinâmica permite descobrir e empacotar os métodos que serão invocados em tempo de execução. Para isso, o cliente faz uso do Repositório de Interface que, como o próprio nome diz, armazena todas as interfaces dos objetos de uma maneira consistente.

Os *Skeletons* (*Stubs* do lado servidor) fornecem interfaces estáticas para cada serviço exportado pelo servidor e, assim como os *Stubs* no cliente, são criados pelo compilador IDL. Análogo à Interface de Invocação Dinâmica, existe a Interface de Esqueleto Dinâmica (DSI) no lado servidor que fornece um mecanismo de ligação para servidores que precisam manipular chamadas de métodos para componentes que não possuem esqueletos estáticos (*Stubs*). A Interface de Invocação Dinâmica faz o desempacotamento (*unmarshal*) da operação e seus parâmetros para descobrir a qual objeto pertence a requisição. Os esqueletos dinâmicos são muito úteis para implementação de pontes entre ORBs e também podem ser usados por interpretadores e linguagens *script* para gerar dinamicamente implementações de objetos.

A comunicação entre a implementação de objetos e o núcleo do ORB é realizada pelo Adaptador de Objetos. Ele manipula serviços como a geração e interpretação de referências de objetos, invocação de métodos, segurança de interações, instanciação dos objetos servidores, mapeamento das referências correspondendo às implementações de objetos e registro de implementações. Para executar estas tarefas, o Adaptador de Objetos conta com a disponibilidade do Repositório de Implementação que armazena as informações, em tempo real, sobre as classes que um servidor suporta, os objetos que estão instanciados e seus identificadores. CORBA especifica que cada ORB deve suportar um Adaptador de Objetos padrão chamado BOA (*Basic Object Adapter*) e que servidores podem suportar mais de um Adaptador de Objetos.

As interfaces ORB consistem de algumas APIs (*Application Programming Interface*) para serviços locais que podem ser de interesse para alguma aplicação, tanto para o cliente quanto para o servidor.

2.1.4 Implementações de ORB

Existem muitas implementações de ORB disponíveis atualmente, e estas variam em grau de proximidade com o modelo CORBA, qualidade de serviço, portabilidade e disponibilidade de características adicionais. Não existe ainda uma implementação de domínio público que seja completa à especificação CORBA.

Alguns exemplos de ORBs presentes no mercado são: *ObjectBroker* e *Iceberg* da BEA Systems [BEA], *PowerBroker* da ExperSoft [Exp], *Component Broker* da IBM [IBM], *Orbix* e *OrbixWeb* da Iona Technologies Ltd. [Ion], *VisiBroker* da Borland/Visigenic [Bor] e *Java IDL* da JavaSoft (incluído no pacote JDK1.2) [Jav]. Dessas, as que mais se destacam possuindo o mapeamento Java/IDL são o *OrbixWeb*, o *VisiBroker* e o *Java IDL*. O *OrbixWeb* é bastante sólido, com grande aceitação comercial e é o ORB utilizado no desenvolvimento deste trabalho, como será apresentado no Capítulo 5.

2.1.5 Considerações finais de CORBA

CORBA vem constantemente se expandindo, estimulando pesquisas em muitos domínios e conquistando uma grande faixa do mercado, por ser uma tecnologia de integração de ambientes heterogêneos. Esta integração é possível graças ao uso da IDL, que fornece uma flexibilidade máxima para os detalhes de implementação básicos. CORBA é uma plataforma orientada a objetos, porém IDL permite que módulos e aplicações não orientados a objetos façam parte do ambiente ORB, encapsulando-os e tornando-os objetos.

Com a versão 2.0 da CORBA, invocações dinâmicas no lado servidor (DSI) passaram a ser suportadas. Estas são mais difíceis de serem implementadas do que as invocações estáticas, porém mais úteis para ferramentas que precisam descobrir os serviços em tempo de execução. Interoperabilidade entre ORBs foi outra característica importante acrescentada a esta versão. O surgimento de ORBs de diferentes vendedores, como visto na seção anterior, fez com que o OMG especificasse um mecanismo de interligação¹ entre estes, como entre sistemas cliente/servidor que não seguem e os que seguem o modelo CORBA. Assim, foram definidos os protocolos GIOP (*General Inter-ORB Protocol*) para interação entre ORBs e o protocolo IIOP (*Internet Inter-ORB Protocol*) implementado usando o transporte TCP/IP (*Transmission Control Protocol/Internet Protocol*).

A CORBA 3.0, que será lançada brevemente, trará novas características importantes para a próxima geração de tecnologia ORB, passagem de objetos por valor, interfaces múltiplas para um objeto², mensagens assíncronas, procurador IIOP para suporte *firewall* e interoperabilidade CORBA/DCOM. O lado servidor de CORBA contará com o POA (*Portable Object Adapter*) que proporcionará escrever aplicações servidoras portáteis.

¹Do Inglês: *bridging mechanism*

²Até então, cada objeto CORBA só poderia ter uma interface

Além disso, o serviço de persistência estará mais eficiente, suportando persistência automática. Facilidade de agentes móveis, estrutura de objetos de negócios baseado em *JavaBeans* e facilidade de *Workflow* também serão acrescentadas nesta versão [OH98].

2.2 Gerência de Sistemas Distribuídos

Com os grandes e recentes avanços tecnológicos, grande quantidade e diversidade de informação trafega pelos sistemas nas redes de comunicação. Estes sistemas cada vez mostram-se mais distribuídos, possibilitando um compartilhamento de serviços e recursos através da rede de maneira transparente e robusta. Mas, com tamanho crescimento, os sistemas distribuídos tornam-se mais complexos e heterogêneos e problemas associados com as operações começam a surgir como falhas de dispositivos, ineficiências de desempenho, segurança e alocação imprópria de recursos [Gol93]. Metodologias e ferramentas de gerência são necessárias para ajudar os administradores a manipular estes problemas e fornecer o funcionamento dos serviços requeridos a eles.

Os termos gerência de redes e gerência de sistemas distribuídos são comumente utilizados na computação. Gerência de redes preocupa-se com os componentes físicos de uma rede como, por exemplo, um roteador, uma ponte, uma impressora, enquanto que a gerência de sistemas distribuídos abrange os recursos para o processamento de serviços de maneira distribuída. A gerência de redes é mais popularizada, pois tem sido estimulada pela crescente complexidade das redes e dependência das suas operações. A gerência de sistemas distribuídos emergiu muito nos últimos anos tornando-se cada vez mais importante e necessária. A maioria dos conceitos de gerência de sistemas tem sido desenvolvida para gerenciar componentes de rede. Muito trabalho tem sido feito nos últimos anos adotando estes conceitos para recursos de gerência de um sistema distribuído. Tais conceitos precisam ser estendidos para permitir planejamento, configuração, contabilização e avaliação de desempenho e da segurança dos sistemas distribuídos. Um sistema distribuído deve ser equipado com mecanismos e políticas que permitam uma gerência eficiente de seus recursos. Assim, para que os sistemas distribuídos e redes continuem seu funcionamento fornecendo os serviços requeridos a eles, a gerência é um mecanismo essencial.

De acordo com Sloman [Slo94], a grande variedade de recursos de computação que deve ser gerenciada em um ambiente distribuído leva a uma diversidade de métodos incompatíveis para aqueles recursos. A heterogeneidade de abordagens administrativas, ferramentas inconsistentes e facilidades inadequadas, faz da gerência de sistemas distribuídos uma tarefa difícil. Apesar de grandes progressos nessa área, ainda há muito que se fazer.

Em 1993, Hegering [HA93] mostrou que as aplicações e as tarefas de gerência podiam ser analisadas e agrupadas em três diferentes dimensões ortogonais, sendo elas as áreas

funcionais, os cenários de gerência e o tempo. Mas, em 1994, Hegering [HA94] ampliou de três para cinco dimensões ortogonais, ilustradas nos eixos da Figura 2.3. Uma dimensão abrange as áreas funcionais de gerência definidas pelo modelo de referência OSI: configuração, desempenho, falhas, segurança e contabilização. Em outra dimensão, seguindo no sentido horário da figura, são consideradas as diferentes fases do ciclo de vida das tarefas de gerência, compondo as fases de projeto, implementação e operação. No terceiro eixo, são apresentados os diferentes tipos de dados que precisam ser gerenciados. Em seguida, são mostrados os tipos de recursos gerenciados, desde os elementos de rede até as aplicações, considerando os requisitos de gerência de uma organização. No último eixo, são abordadas as diferentes tecnologias usadas em redes locais, metropolitanas, geográficas, corporativas e inteligentes. Segundo Hegering, 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.

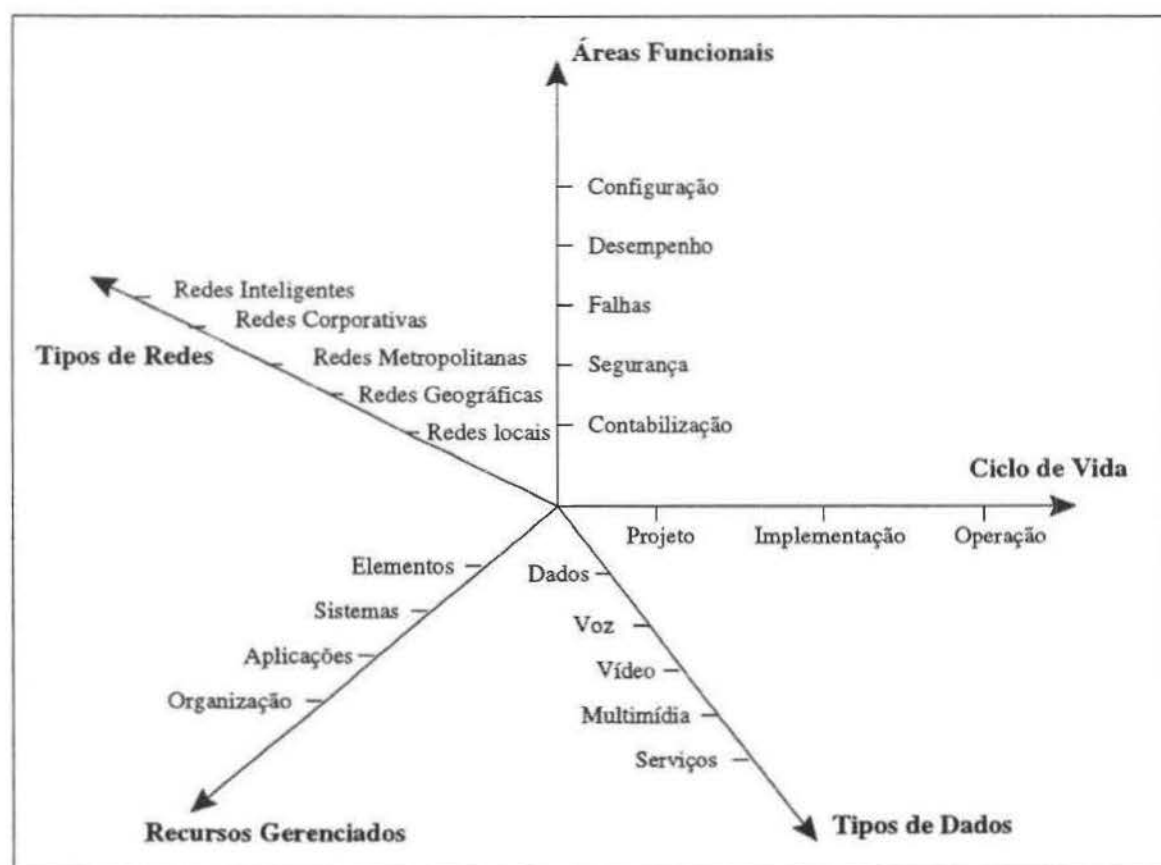


Figura 2.3: Eixos ortogonais de gerência

Vale ressaltar que, a gerência seguindo o modelo RM-ODP (*Reference Model for Open Distributed Processing*) define cinco diferentes áreas funcionais: configuração, contabili-

zação, qualidade de serviço, monitorização e definição de políticas de gerência. Não existe uma correspondência direta com as funções de gerência OSI. A gerência de qualidade de serviço é uma generalização de desempenho e falhas e as funções de definição de políticas de gerência deverão incluir funções de segurança [MP95].

Assim, em meio ao crescimento dos sistemas, aplicações e todos os elementos que compõem uma rede, sistemas de gerência foram padronizados. Os sistemas de gerência correntes seguem um paradigma centralizado gerente/agente, onde agentes monitoram o sistema e coletam dados, os quais podem ser alcançados pelos gerentes através de protocolos de gerência como SNMP ou CMIP [GY98]. Estes protocolos são mais adequados para gerência de redes, porém são usados também para gerenciar sistemas distribuídos. Mesmo o modelo OSI/CMIP de gerência sendo totalmente orientado a objetos e de grande potencialidade e, conseqüentemente, mais complexo, não tem sido grandemente implementado e desenvolvido. Assim, SNMP e protocolos proprietários são ainda soluções preferidas por uma grande faixa do mercado. Isto se deve principalmente, à dificuldade de implementação do modelo OSI, mas por ser tão completo é sempre referenciado na literatura. Entretanto, uma nova alternativa deve ser considerada: CORBA o qual está ganhando uma crescente importância na área de gerência de sistemas distribuídos (aplicações, serviços e objetos distribuídos) e redes.

Segundo Meyer et al. [MEBS95], o paradigma centralizado tem algumas limitações fundamentais e propõe que graus de descentralização seriam apropriados para as aplicações de gerência de rede e sistemas distribuídos. A classificação das aplicações de gerência apresentada pelo autor é: aplicações centralizadas, aplicações parcialmente descentralizadas e aplicações descentralizadas. Aplicações centralizadas são aquelas que não precisam de controle distribuído, não requerem *polling* freqüente ou alta freqüência de computação de MIB (*Management Information Base*) e não têm necessidade de conversações semanticamente ricas e freqüentes entre gerente e agente. Aplicações parcialmente descentralizadas requerem algum grau de controle administrativo centralizado, enquanto que aplicações descentralizadas necessitam de controle distribuído e assumem características opostas às aplicações centralizadas.

Muito progresso tem ocorrido para uma descentralização do controle de gerência. Exemplo disto é a gerência por delegação (MbD - *Management by Delegation*) [GY95, Gol96, GY98], no qual o gerente delega funções de gerência ao agente que, por sua vez, pode mudá-las em tempo de execução. Outra forma de descentralizar a gerência é utilizar agentes móveis autônomos, apresentada na Seção 3.5, objetivo de estudos e desenvolvimento desta dissertação. Contudo, aplicações centralizadas não devem ser extintas, mas, sim, coexistirem com as descentralizadas. Dependendo da escalabilidade da rede e das atividades de gerência, aplicações centralizadas podem ser mais eficientes; por exemplo: monitorar o estado de um único roteador envolve consultar e apresentar o valor de um

pequeno número de variáveis da MIB. Isto é apropriado à gerência centralizada, já que rapidamente pode-se estabelecer conexão entre gerente e agente.

A questão da interoperabilidade entre as tecnologias de gerência em domínios diferentes torna-se também muito interessante. Muito já se tem feito neste sentido, como, por exemplo, a interoperabilidade SNMP/CMIP pelo grupo IIMC (*ISO-Internet Management Coexistence*) do Fórum de Gerência de Redes e de CMIP/CORBA e SNMP/CORBA pelo JIDM (*Joint Inter-Domain Management*), estes descritos em [SH97].

As próximas seções apresentam estes três sistemas de gerência separadamente e, logo após, uma breve comparação é mostrada.

2.2.1 Modelo de Gerência OSI/CMIP

O modelo de gerência OSI foi desenvolvido pela ISO (*International Organization for Standardization*) em conjunto com o CCITT (*Consultative Committee for International Telegraph and Telephone*, hoje padronização ITU-T - *International Telecommunications Union - Telecommunication Standard Sector*) em 1989, com o objetivo de atender a enorme diversidade de elementos gerenciáveis existentes em uma rede [BRI97]. É considerado uma padronização de *jure* e é constituído de quatro modelos:

- Modelo Organizacional: estabelece a hierarquia entre sistemas de gerência, dividindo o ambiente gerenciado em domínios funcionais e administrativos;
- Modelo de Informação: apresenta a estrutura de informação gerenciada, obedecendo ao paradigma de orientação a objetos;
- Modelo Funcional: associado às várias atividades de gerência dentro das áreas funcionais de configuração, de falhas, de desempenho, de contabilização e de segurança;
- Modelo de Comunicação: composto pelo protocolo e suas primitivas.

O ambiente de gerência OSI inclui os conceitos de gerente, agente e objetos gerenciados. Um objeto gerenciado é a representação abstrata de um recurso físico ou lógico sujeito à gerência. Ele é definido em termos de seus atributos, de operações a que pode ser submetido, de notificações que pode emitir para informar ocorrência de eventos internos, e de suas relações com outros objetos gerenciados, caso exista. 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, podendo enviar notificações ao gerente, armazenando as informações pertinentes na MIB. A MIB é organizada na forma de árvore e armazena logicamente os dados que representam os objetos gerenciados de um sistema aberto.

O modelo funcional ISO/OSI apresenta quinze funções de gerência definidas nas normas ISO 10164-1 a ISO 10164-15 [BRI97], que dão suporte às cinco áreas funcionais (configuração, desempenho, falhas, segurança e contabilização). As áreas funcionais constituem processos de aplicação de gerência que utilizam os serviços oferecidos pela camada de aplicação do modelo OSI. Cada área funcional possui objetivos diferentes, porém relacionam-se no sentido de que informações geradas em uma área podem ser utilizadas como suporte para decisões em outras áreas. Assim, algumas funções de gerência são específicas a uma área funcional, enquanto que outras são genéricas e servem a um conjunto de áreas funcionais.

A troca de informações entre as entidades de aplicação de gerência é feita pelo protocolo CMIP e pelas primitivas de serviços do CMISE (*Common Management Information Service Element*). Também são utilizados outros elementos de serviço da camada de aplicação: o ACSE (*Association Control Service Element*) para gerenciar as associações gerente/agente e o ROSE (*Remote Operations Service Element*) para ações remotas. O CMIP é um protocolo orientado à conexão e faz uso de todas as facilidades de comunicação da pilha de protocolo OSI.

A informação de gerência diz respeito àquela usada para definir como gerenciar tipos particulares de recursos. Consiste em definições de objetos gerenciados juntamente com outras definições genéricas, a fim de manter a consistência entre diferentes tipos de objetos gerenciados que possuem a mesma função [Slo94]. Assim, a ISO definiu o padrão de Estrutura da Informação de Gerência (SMI - *Structure of Management Information*), que especifica o modelo de informação a ser adotado e é composto de quatro partes: o Modelo de Informação de Gerência (MIM - *Management Information Model*) que apresenta os conceitos fundamentais de objetos gerenciados; a Definição de Informação de Gerência (DMI - *Definition of Management Information*) mostrando todas as definições de informação de gerência necessárias para os padrões de gerência de sistemas; as Regras para a Definição de Objetos Gerenciados (GDMO - *Guidelines for the Definition of Managed Objects*) fornecendo as técnicas de especificação da informação de gerência e a Informação de Gerência Genérica (GMI - *Generic Management Information*) que especifica a informação genérica comum nas camadas OSI. A SMI baseia-se na abordagem orientada a objetos para a definição dos recursos gerenciados através de classes e introduz os conceitos de hierarquias de herança, de nomeação e de registros. A hierarquia de herança ou de classes está relacionada com as propriedades associadas aos objetos com seus atributos, comportamentos, pacotes condicionais, operações e notificações. Utiliza fortemente o conceito de herança múltipla. A hierarquia de nomeação descreve as relações entre instâncias de objetos com seus respectivos nomes. Por último, a hierarquia de registro é a que identifica os objetos de maneira única e universal, independentemente das outras hierarquias.

As classes de objetos gerenciados são definidas de acordo com um *template* contido no GDMO descrito em [ISO91] e utiliza a notação de sintaxe ASN.1 (*Abstract Syntax Notation One*) para estruturar as definições das informações de gerência desses objetos. O GDMO é quem descreve a estrutura de árvore de registros adotada no modelo OSI para alocação dos identificadores dos objetos.

A SMI define de forma abstrata, as operações de gerência que podem ser feitas sobre os atributos de um objeto, as operações sobre os objetos gerenciados como um todo, e a operação de notificação (*Event Report*) emitida pelo agente ao gerente quando da ocorrência de eventos. As operações sobre os atributos são:

- *Get*: usada na leitura de um atributo ou de uma lista de atributos;
- *Replace*: permite alterar valores de atributos por valores fornecidos;
- *Set*: substitui o valor de atributos por valores especificados quando da definição dos objetos;
- *Add*: é utilizada no caso de atributos cujos valores são definidos como conjunto, permitindo acrescentar novos elementos a este;
- *Remove*: usada na remoção de um elemento num conjunto de valores de um atributo.

As operações que podem ser feitas sobre os objetos gerenciados como um todo definidas no modelo de informação são:

- *Create*: permite criar um objeto e inicializar os valores dos atributos do mesmo;
- *Delete*: elimina um objeto em particular;
- *Action*: usada para determinar ao objeto que execute uma ação específica e apresente o seu resultado.

2.2.2 Modelo de Gerência Internet/SNMP

O SNMP surgiu no início de 1988, padronizado pelo IETF (*Internet Engineering Task Force*) como uma proposta rápida às necessidades do mercado e compõe a base do sistema de gerência de redes da arquitetura Internet TCP/IP. Em princípio o SNMP deveria ser apenas um protocolo provisório, mas devido à sua simplicidade de implementação tem estado presente na maioria dos produtos comerciais tornando-se padronizado de *facto*.

O modelo utiliza os conceitos de agentes e gerentes vistos anteriormente e usa o termo objeto para representar algum recurso que precisa ser gerenciado. O SNMP não é

orientado e nem baseado em objetos, pois não suporta os conceitos relacionados a esta abordagem.

O modelo apresenta três documentos que constituem a base para a estrutura de gerência padrão Internet. O primeiro documento intitulado Estrutura de Informação de Gerência (SMI) [RM90] identifica as estruturas de dados SNMP e especifica como os objetos são representados e nomeados em uma MIB. Esses objetos são definidos usando uma linguagem de definição de objeto padrão adotada do OSI e chamada ASN.1. A MIB é organizada em uma hierarquia, sendo que a estrutura de árvore básica é definida no documento do SMI. O segundo, corresponde à versão atual da MIB (MIB-II) [MR91], na qual os objetos são arranjados em dez grupos considerados essenciais para a gerência de redes de computadores. O terceiro documento trata do protocolo de gerência SNMP [CFD90], que é usado na comunicação entre agentes e gerentes para troca de informações de gerência.

Este modelo consiste em um esquema centralizado, no qual a tarefa de gerência da rede é feita pela estação (*host*) de gerência que atua como gerente. Esta estação nada mais é que um equipamento de propósito geral que executa as aplicações de gerência, comunicando-se com os agentes pela rede. Os agentes SNMP recebem comandos do gerente e, assim, encaminham informações ou alteram valores das variáveis que representam os objetos gerenciados. Porém, quando eventos não planejados acontecem, um agente pode avisar o gerente do ocorrido na forma de *traps*. Para aqueles equipamentos que não possuem a capacidade de ter um agente SNMP executando internamente é definido um *proxy agent*.

Dentro do universo dos tipos de dados da sintaxe ASN.1, o SNMP utiliza apenas um pequeno conjunto. Os tipos básicos utilizados são *INTEGER*, *OCTET STRING*, *BIT STRING*, *NULL* e *OBJECT IDENTIFIER*. Os tipos específicos de contexto são *IpAddress*, *NetworkAddress*, *Counter*, *Gauge*, *TimeTicks* e *Opaque*; e os tipos estruturados *SEQUENCE* e *SEQUENCE OF*. Os tipos *REAL*, *BOOLEAN* e estruturas mais complexas não são permitidas. Diferente do OSI, o SMI não utiliza o conceito de classes de objetos e seus respectivos atributos. São definidos tipos de objetos e cada objeto possui um identificador único e universal dado pelo *OBJECT IDENTIFIER*, compondo um nó na árvore de hierarquia de registros. Cada novo objeto faz parte de uma MIB. Essa estrutura é administrada de forma conjunta pela ISO e ITU-T.

O protocolo SNMP é o protocolo de aplicação de gerência na arquitetura Internet. Ele suporta dois mecanismos de comunicação entre gerente e agentes: *polling* e notificações (*traps*). Os dados são enviados usando a sintaxe de transferência ASN.1. O SNMP baseia-se no protocolo UDP (*User Datagram Protocol*), que não é orientado à conexão. A primeira versão do SNMP define cinco tipos de PDUs (*Protocol Data Units*) que podem ser enviadas:

- *GET-REQUEST*: usada para recuperar uma informação de gerência específica;

- *GET-NEXT-REQUEST*: retorna o próximo valor armazenado;
- *GET-RESPONSE*: responde a uma operação de *GET-REQUEST*, *GET-NEXT-REQUEST* ou *SET-REQUEST*, contendo os valores dos objetos ou uma mensagem de erro;
- *SET-REQUEST*: altera valores das variáveis dos objetos;
- *TRAP*: relata a ocorrência de algum evento.

Embora o SNMPv1 tenha sido bem aceito, ele possui restrições que se tornaram evidentes à medida que os sistemas a serem gerenciados foram se tornando mais complexos. As limitações da versão 1 motivaram o surgimento do SNMPv2 que traz melhorias principalmente na parte de segurança incluindo algumas facilidades como autenticação, privacidade e controle de acesso. Duas novas PDUs foram incluídas: *GET-BULK-REQUEST* que recupera múltiplos valores através de uma única troca de PDU, e *INFORM-REQUEST* que permite troca de informações de gerência entre gerentes. Através desta última PDU, o SNMPv2 oferece a estratégia de gerência distribuída, além da centralizada suportada na versão anterior, na qual as estações podem exercer o papel de gerente e de agente. Para isso, foram criadas três novas MIBs. A MIB "SNMPv2" contém informações relativas à utilização do próprio protocolo. A MIB "Gerente-Gerente" possui informações relativas aos alarmes e eventos resultantes das trocas de informações entre gerentes. Por fim, a MIB "Segurança" está relacionada às informações de segurança. A operação *GET* também sofreu modificação com a versão 2, deixando de ser uma operação atômica, que somente fornecia resultados parciais, para apresentar todas as informações da lista de objetos de um *GET-REQUEST* [BRI97].

Recentemente, foi lançado o SNMPv3 que, como o SNMPv2, possui a mesma arquitetura básica que o SNMPv1. A principal característica do SNMPv3 é que ele trata as duas grandes deficiências relacionadas a sua versão anterior: segurança e administração. O IETF descreve cinco documentos que definem o novo protocolo: Uma Arquitetura para Descrição das Estruturas de Gerência SNMP, Despacho e Processamento de Mensagem para o SNMP (MPD - *Message Processing and Dispatch*), Aplicações SNMPv3, O Modelo de Segurança baseado em Usuário para Versão 3 do SNMP (USM - *User-based Security Model*) e o Controle de Acesso baseado em Visão (VACM - *View-based Access Control*).

2.2.3 CORBA como um Modelo de Gerência

CORBA vem sendo incentivada pela comunidade a possuir interfaces padronizadas de serviços para a área de gerência, mesmo ainda não possuindo as facilidades de gerência de SNMP e CMIP, os quais são protocolos específicos para tal funcionalidade. O uso de

CORBA como uma arquitetura para gerência de sistemas distribuídos pode ser discutido com uma comparação aos quatro modelos da arquitetura de gerência OSI (organizacional, de comunicação, de informação e funcional) e também como difere deste e do modelo Internet [Kel96].

Quanto ao modelo organizacional, CORBA não tem noção de hierarquia, mas assume relacionamentos par a par simétricos entre objetos; em termos de gerência, isto resulta em uma extensão dos modelos tradicionais: permite não somente relacionamentos gerente/agente ou gerente/gerente, mas também relacionamentos agente/agente. Isto simplifica a distribuição da funcionalidade de gerência e suporta cooperação entre componentes diferentes da infra-estrutura de gerência.

Do modelo de comunicação, ao contrário dos modelos de gerência existentes, CORBA não necessita de um protocolo de gerência específico. A infra-estrutura de comunicação é o ORB, visto na Seção 2.1.3, e as operações de gerência são invocações de métodos em um objeto remoto, não sendo fixas e determinadas (por exemplo, *Get* e *Set*). CORBA possui transparência de localização, cujo pré-requisito é que ORBs executando em diferentes sistemas estejam aptos para interoperarem. Uma referência de objeto interoperável identifica unicamente um objeto CORBA. Um 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 invocar seus métodos através de interfaces estáticas ou 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, oferecendo flexibilidade e assim permitindo que um cliente construa e realize chamadas a objetos em tempo de execução através do Repositório de Interface. Para a gerência, esta flexibilidade possibilita o desenvolvimento de aplicações gerentes sob a forma de clientes CORBA, usufruindo das facilidades de verificação oferecidas pelo uso de uma IDL. Com este repositório, pode-se estender o modelo tradicional fazendo com que a aplicação gerente possa oferecer ao usuário serviços que não estão descritos em uma MIB, e ao agente o poder de comunicar à aplicação gerente alterações na sua funcionalidade.

Do ponto de vista de informação, a descrição dos objetos CORBA é estabelecida pela especificação de suas interfaces em IDL, que é comparável ao GDMO e superior ao SMI, permitindo a clara separação entre declaração de interface e sua implementação. 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 esquemas de codificação e decodificação. A sintaxe de IDL é basicamente uma extensão de C++, tendo sido especificada com o objetivo de alcançar uma integração simples entre as linguagens de implementação (C, C++, *Smalltalk*, *Ada*, *COBOL* e *Java*). CORBA oferece o paradigma da orientação a objeto da mesma forma que

o modelo OSI, mas com um maior grau de abstração. Cada classe de objeto gerenciado é definida por uma interface IDL descrevendo atributos e operações dos objetos gerenciados. Existem ferramentas de desenvolvimento que suportam a tradução de modelos de objetos OMT (*Object Modeling Technique*) [RBP⁺91] em IDL com compiladores que geram automaticamente os *stubs* e esqueletos dos objetos de aplicações.

O modelo funcional CORBA é baseado nos componentes da OMA, apresentada na Seção 2.1.1, e os divide em camadas. Devido a orientação a objetos natural do OMG, serviços contidos nas camadas superiores podem herdar as funcionalidades das camadas inferiores. A camada mais inferior é composta pelos *CORBA services* que agrupam coleções de componentes definidos com interfaces IDL mais complexas com um complemento às funcionalidades de *middleware* do ORB. De todos os *CORBA services*, existem aqueles particularmente úteis para os propósitos de gerência [Que97]:

- Serviço de Nomes: empregado para resolução de nomes em um contexto de um ambiente ORB;
- Serviço de Eventos: permite a comunicação entre objetos através de eventos;
- Serviço de Ciclo de Vida: usado para permitir as operações de criação, destruição, cópia e movimentação de objetos;
- Serviço de Tempo: desejável para se ter um tempo padrão acurado e preciso para sincronizar atividades distribuídas;
- Serviço de Transação: suporta múltiplos modelos de transações, inclusive do tipo aninhadas;
- Serviço de Controle de Concorrência: coordena 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: apoia a persistência de um objeto, independentemente do seu tempo de vida, tanto da aplicação que acessa o objeto quanto da implementação que executa os métodos dos objetos;

- Serviço de Segurança: controla o acesso às informações, autentica as partes envolvidas e cifra mensagens quando necessário.

A próxima camada na hierarquia são os *CORBA facilities* que oferecem serviços para serem utilizados diretamente pelas aplicações. São divididas em quatro conjuntos: Interface do Usuário, Gerência de Informações, Gerência de Tarefas e Gerência de Sistemas. Este último fornece a base para a introdução da funcionalidade de gerência dentro do OMG e é de especial interesse para o tema desta dissertação. Na gerência de sistemas, uma das facilidades de suporte oferecida para esta tarefa é a **Instrumentação**. Segundo Friedrich et al. [FSH⁺95], a instrumentação é definida como componentes de software especializados incorporados dentro de programas, com mecanismos para medição, cálculo e análise de medidas de desempenho. Esta facilidade é de grande importância para o desenvolvimento deste trabalho.

Interfaces de Domínios compõem a terceira camada e são serviços de alto nível para domínios de aplicação específicos. Como a gerência de sistemas é algo importante em todos os domínios de aplicação, esta camada não define qualquer funcionalidade de gerência. Finalmente, no topo da hierarquia estão os Objetos de Aplicação. Parte de suas funcionalidades podem ser herdadas das três camadas abaixo. Um exemplo de um objeto de aplicação é uma aplicação de gerência distribuída.

De uma forma geral e resumida, 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, o modelo de gerência mais popular e difundido hoje em dia, criado na 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. Assim, não é necessário ter-se uma MIB central na memória do sistema de gerência, evitando os problemas de inconsistência no caso de uma mudança em um agente;
- 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;

- no modelo de objetos distribuídos CORBA um gerente pode ter acesso diretamente ao objeto gerenciado e este invocar a entidade gerente quando houver necessidade de reportar um evento significativo (Figura 2.4 (a)), podendo utilizar a facilidade de invocação dinâmica. CORBA também permite o paradigma tradicional gerente/agente (Figura 2.4 (b)), como nos modelos OSI e Internet, no qual o agente invoca o objeto gerenciado pelo ORB (1 e 2) ou não (1a);
- 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;
- as implementações CORBA em Java [VD98] e o protocolo IIOP, já incluído na última versão do navegador *Netscape*, permitirão mais facilmente a integração das ferramentas de gerência com a tecnologia WWW (*World Wide Web*).

2.2.4 Comparação entre os Modelos de Gerência

Os modelos OSI/CMIP e Internet/SNMP, os mais tradicionais, e CORBA como uma tecnologia mais recente, apresentam suas características para a gerência de sistemas distribuídos e redes, descritas nas Seções anteriores. Porém, um pode estar a frente do outro, quando abordada alguma característica em particular.

O SNMP (versão 2) é ainda o protocolo de gerência mais predominante hoje, com limitações para os requisitos de gerência de sistemas total. O SNMPv2, CMIP e CORBA requerem mais memória e processadores potentes do que SNMPv1. SNMP e CMIP precisam de aplicações de gerência que interajam com um número grande de atributos no nível mais baixo através de *Get* e *Set*. SNMPv2 não permite que se registre operações nos objetos gerenciados, enquanto que CMIP faz isto de uma maneira grosseira [OHE96]. O SNMP resolveu muitos problemas reais quando a gerência era mais simples e escassa. Segundo Orfali et al. [OHE96], o futuro de gerência de sistemas pode estar no mundo de objetos da CORBA.

A CORBA, como descrito na Seção 2.2.3, fornece um protocolo moderno e natural para a representação de entidades gerenciadas, definição dos seus serviços, especificação de dados e invocação de métodos via um ORB. Os Repositórios de Implementação e Interface podem ser usados para descobrir e dinamicamente invocar métodos nos objetos gerenciados em tempo de execução. Os objetos gerenciados podem diretamente chamar o gerente quando eles tiverem algo significativo para relatar (ao contrário do SNMP que usa geralmente *polling*), ou através de um objeto agente, mostrando assim sua flexibilidade ao modelo gerente/agente de SNMP e CMIP. O SNMP coloca uma quantidade mínima de lógica de emissão de eventos nos agentes e deixa toda a “inteligência” para o gerente.

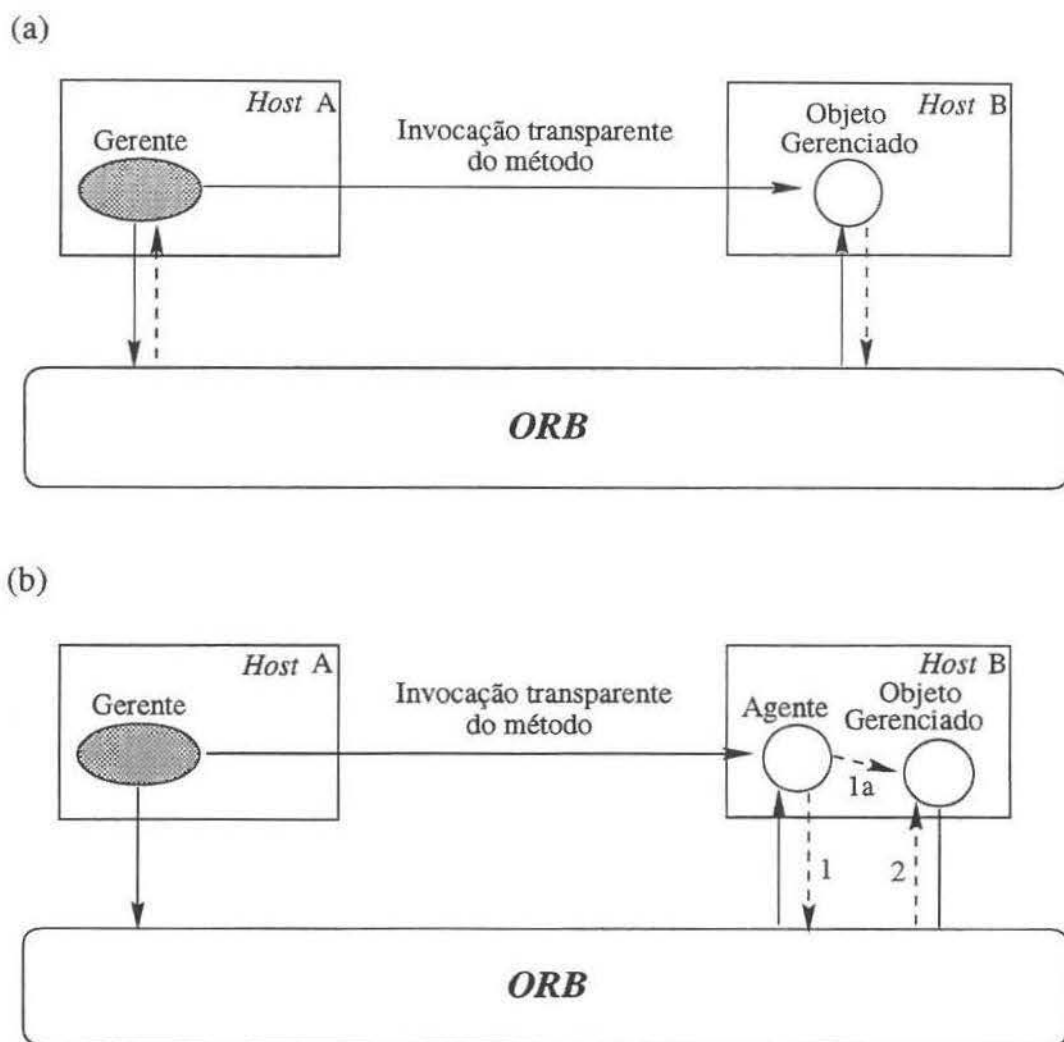


Figura 2.4: CORBA como um modelo de gerência

Assim, *traps* são raramente utilizados e o gerente deve consultar (*polling*) os agentes sobre alguma coisa que estiver acontecendo.

CMIP e CORBA são orientados a eventos, e isto significa que os agentes possuem mais "inteligência"; o gerente não precisa fazer tantos *pollings*. Desta forma, um gerente CORBA pode manipular um número muito maior de objetos gerenciados. Vale lembrar, que apesar do CMIP também ter esta característica, ele é implementado sobre a pilha completa de protocolos OSI de comunicação, exigindo um ambiente operacional, muitas vezes, não disponível em situações de falhas.

O SNMP possui uma simplicidade que torna a gerência de redes grandes mais fácil, porém algumas conseqüências precisam conviver juntas como, por exemplo, a segurança. A versão 2 já tratava este ponto mas, não totalmente, e agora a versão 3 do SNMP trata

isto com mais amplitude.

A Tabela 2.1, originalmente apresentada em [OHE96] e com algumas modificações, compara as características do SNMPv1, SNMPv2 (a recente versão 3 não está em discussão aqui), CMIP, CORBA e ainda aborda o RMI da plataforma Java. RMI foi incluído nesta comparação porque Java conquistou um espaço muito grande no mercado e está amplamente difundida trazendo muitas contribuições para a gerência. As 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-compilados para código nativo da plataforma, através de compiladores sob-demanda. O interpretador Java, conhecido como JVM (*Java Virtual Machine*), executa programas Java pré-compilado em *bytecodes*, permitindo múltiplas linhas de execução (*threads*) e oferecendo primitivas para sincronização entre elas. Estas características despertaram o interesse do OMG em propor o seu mapeamento para IDL e da comunidade no seu uso como tecnologia complementar na área de gerência. Desta forma, Java RMI oferece classes que permitem o desenvolvimento de aplicações distribuídas, de forma razoavelmente simples e transparente. Na verdade, RMI é um protocolo conceitualmente parecido com RPC (*Remote Procedure Call*), adaptado às características próprias de Java em que os objetos, no papel de servidores, ao receberem uma requisição são serializados (*Object Serialization*) e passados por valor pela rede. Esta facilidade tem sido bastante discutida pelo OMG e provocou uma adaptação a sua arquitetura especificando a passagem de objetos por valor (este assunto será melhor discutido na Seção 3.4).

Analisando a Tabela 2.1, é possível arriscar dizer que o SNMP talvez não venha a ser o modelo mais predominante, pois CORBA tem grandes chances de crescer muito no campo de gerência de sistemas distribuídos e redes.

2.3 Plataforma Multiware

A plataforma *Multiware* [LM95a, MLM⁺94], ilustrada na Figura 2.5, utiliza 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/Architecture Projects Management*) e DCE/OSFs (*Distributed Computing Environment/Open Software Foundation*). Ela é composta por três camadas: *Hardware/Software* básico, *Middleware* e *Groupware*. A primeira camada é formada pelos sistemas operacionais e protocolos de comunicação. A camada *Middleware* é

Característica	SNMPv1	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/Agente	Gerente/Agente	Gerente/Agente	Serialização de Objetos	Objetos Distribuídos
Visão dos Objetos Gerenciados	Variáveis simples organizadas em MIBs	Variáveis simples 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 e Agente	<i>Polling</i> , raros <i>Traps</i>	<i>Polling</i> , raros <i>Traps</i>	Eventos	Requisição e Resposta	Requisição, Resposta e Eventos
Invocação Dinâmica	Não	Não	Não	Não	Sim
Ação Explícita	Não	Não	Sim	Sim	Sim
Segurança	Não	Sim	Sim	Sim	Sim
Interação entre Gerentes	Não	Sim	Sim	Sim	Sim
Transferência em Bloco	Não	Sim	Sim	Sim	Sim
Criação e Destruição de Objetos Gerenciados	Não	Não	Sim	Sim	Sim
Modelo de Comunicação	Datagrama	Datagrama	Sessão	RPC	ORB
Órgão de Padronização	IETF	IETF	ISO	SunSoft	OMG

Tabela 2.1: Comparação entre os modelos de gerência

responsável por fornecer facilidades de processamento distribuído aberto para a camada *Groupware* e para as aplicações distribuídas. Possui ORBs e funcionalidades desenvolvidas sobre estes, como o *Trader* [LM95b], Suporte à Transação e a Grupo [CM96], Negociação de Qualidade de Serviço [LM97], Suporte à Mobilidade [VM98, Vas99], Gerência de Sistemas Distribuídos [Que97] e a utilização de Agentes Móveis para a Gerência de Sistemas Distribuídos [RSM98, RM98] sendo o trabalho desta dissertação. As três funcionalidades envoltas por uma linha pontilhada mostrada na Figura 2.5, na verdade compõem uma outra funcionalidade denominada *MomentA (Mobile Management Architecture)* [SMRao]. Esta, acrescenta a utilização de um balanceamento dos recursos computacionais com o auxílio de um serviço de disponibilidade. Atualmente, a *Middleware* está sendo desenvolvida sobre a *Orbix* e *OrbixWeb* da *Iona Technologies* [Ion]. Existe ainda uma subcamada de Processamento Multimídia que permite a troca de mensagens multimídia com uma qualidade de serviço especificada. A última camada é a *Groupware*, a qual suporta aplicações CSCW (*Computer Supported Cooperative Work*).

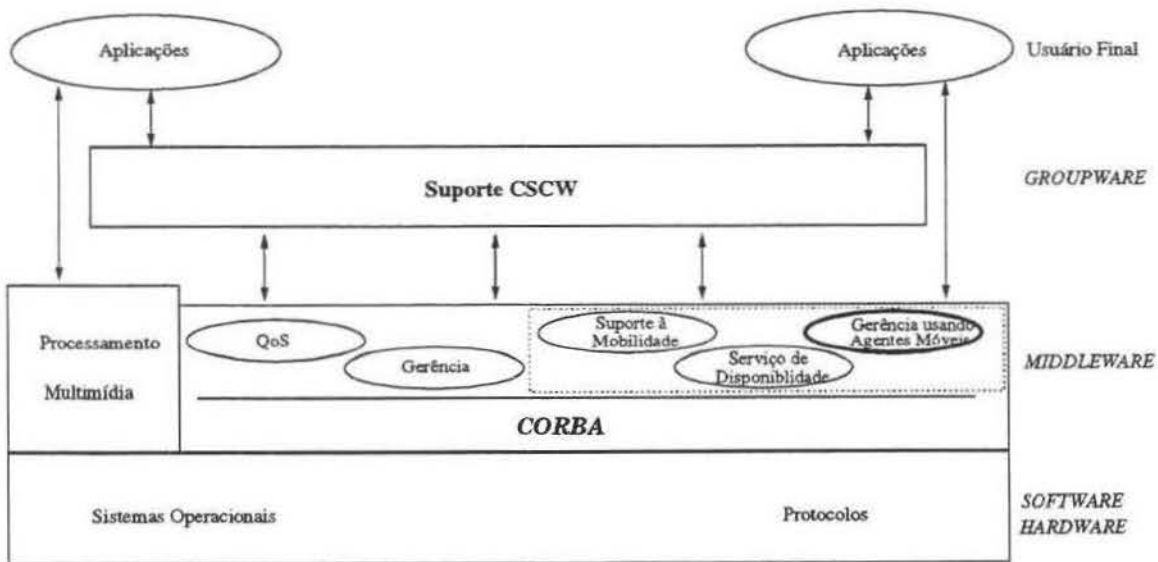


Figura 2.5: Plataforma Multiware

No projeto *Multiware*, os conceitos da arquitetura OMA/OMG são estendidos para prover novas funcionalidades de processamento distribuído aberto desenvolvidas sobre os ORBs. Este trabalho vem como uma nova função ODP na camada *Middleware* oferecendo uma nova abordagem de gerência de sistemas distribuídos e, neste caso, estendendo os conceitos das *CORBA facilities*.

O ambiente da plataforma *Multiware* consiste de um grande número de *hosts* de diferentes arquiteturas, dispersos em dois laboratórios da Universidade Estadual de Campinas (UNICAMP), um no Instituto de Computação e outro na Faculdade de Engenharia

Elétrica e de Computação, conectados por redes *Ethernet*, *Fast Ethernet* e FDDI (*Fiber Distributed Data Interface*). Em ambos laboratórios são utilizados os ORBs comerciais da *Iona Technologies Ltd.* [Ion]. 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 estender as suas interações e correlações. 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 [Que97].

2.4 Resumo

A gerência de redes e sistemas distribuídos, durante muito tempo, foi dominada por modelos tradicionais gerente/agente como SNMP e CMIP. O SNMP ainda é o protocolo predominante hoje no mercado, enquanto que o CMIP é o protocolo exemplo, por ser tão completo, citado em toda a literatura de gerência. Com o crescimento cada vez maior dos sistemas, a heterogeneidade de equipamentos e informações faz com que se busque novas tecnologias que possam acompanhar este crescimento. CORBA se mostra como uma plataforma de objetos distribuídos capaz de gerenciar grandes sistemas distribuídos, podendo competir futuramente com o SNMP.

Uma descentralização da gerência também é um fator a ser considerado nesta área. Os protocolos tradicionais, geralmente, utilizam uma maneira centralizada para gerenciar os recursos e isto muitas vezes pode não mais ser eficiente. O SNMPv2, através de seus gerentes distribuídos, consegue obter uma gerência mais descentralizada. Porém, utilizando os objetos distribuídos CORBA isto pode ser alcançado mais facilmente.

A plataforma *Multiware*, oferece um ambiente para execução de aplicações distribuídas em um *middleware*, com várias funcionalidades desenvolvidas sobre um ORB. Este trabalho vem no sentido de incluir mais uma funcionalidade nesta plataforma.

Capítulo 3

Conceitos de Agentes Móveis

Este capítulo apresenta os agentes de forma geral, para então se concentrar em uma classe específica de agentes denominada agentes móveis. Existem vários sistemas de agentes móveis disponíveis para uso, e um conjunto deles é citado e outro descrito com mais detalhes. Também é abordado neste capítulo a questão da mobilidade em CORBA e a gerência utilizando a nova tecnologia de agentes móveis.

3.1 Definições e Características de Agentes

A abordagem baseada em agentes está presente em muitas aplicações usadas hoje representando uma área de grande interesse, principalmente em inteligência artificial, sistemas distribuídos, comunicação de computadores e engenharia de software. Grandes estudos e avanços têm-se feito em direção aos sistemas baseados em agentes e, contudo, não há uma definição de consenso para o termo agente. Ao contrário, há uma grande diversidade no entendimento deste termo e quais são suas características fundamentais. Para ilustrar tal diversidade, são transcritas abaixo algumas definições retiradas da literatura:

- “um agente é um programa que auxilia um usuário na execução de uma tarefa (ou um conjunto de tarefas), possivelmente através da manutenção de um estado persistente e da comunicação com seu dono, outros agentes ou seu ambiente” [LD95]
- “agentes são programas de computador que simulam um relacionamento humano, fazendo alguma coisa que alguém poderia fazer por outra pessoa” [Sel94]
- “agente autônomo é um sistema situado num ambiente e parte deste ambiente, que percebe e age sobre o ambiente, no tempo, completa sua agenda de trabalho, de maneira a afetar o que ele percebe no futuro” [FG96]

- “agentes são elementos de software auto-contidos ¹ os quais são responsáveis por carregarem uma ou múltiplas tarefas programáticas” [KSM+97]
- “agentes de informação são programas que modelam um espaço de informação de usuário, gerenciando os recursos e tomando ações quando apropriadas” [CGWK94]
- “agentes inteligentes são entidades de software que executam um conjunto de operações em favor de um usuário ou de um outro programa, com algum grau de independência ou autonomia, e desta maneira, empregam algum conhecimento ou representação das metas ou desejos do usuário” [IBM96]

De acordo com [GHN+97], existe um senso comum nas definições de agente, o qual diz que um agente é uma entidade que age em benefício ou interesse de outra entidade.

Dentro de um contexto no qual o agente está inserido, este é modelado conforme os objetivos da aplicação e assim possuindo suas próprias características. As características dos agentes julgadas fundamentais que aparecem freqüentemente na literatura são:

- autonomia: são aptos a tomar iniciativa, exercer o controle sobre suas próprias ações e estado interno;
- delegação: usuários de outros programas podem delegar tarefas a agentes e revesti-los com autoridade para agir em seu favor;
- comunicabilidade: também chamado de habilidade social, é a capacidade que agentes têm de interagir com outros programas (agentes ou não), com ambientes que os abrigam e com seres humanos;
- flexibilidade: agentes têm interfaces e comportamento bem definidos e não assumem papéis fixos, eles podem agir como clientes, servidores, observadores, etc., dependendo de suas necessidades correntes;
- equidade: agentes comportam-se como pares, não há relações de hierarquia entre eles;
- auto-iniciação: diferente de programas padrões os quais são diretamente invocados pelo usuário, um agente pode perceber mudanças em seu ambiente e decidir quando agir;
- reatividade: capacidade de perceber e agir, no tempo, a mudanças no ambiente externo;

¹Do Inglês: *self-contained*

- orientação por metas: é a chamada pró-atividade. Um agente aceita requisições de alto nível indicando o que um cliente quer e é responsável por decidir como e onde satisfazer a requisição.

Outras características não fundamentais, mas que estão fortemente relacionadas a alguns tipos de agentes são:

- cooperação: capacidade de interação com outros agentes visando atingir metas comuns; para tanto, agentes devem possuir habilidade social;
- inteligência: habilidade de raciocínio e aprendizado apresentada por agentes à medida que eles reagem ao ambiente externo e/ou interagem com este ambiente externo promovendo uma mudança no seu comportamento (adaptabilidade);
- persistência: capacidade de manter um estado interno consistente no tempo;
- personalização: o usuário determina como o agente interage;
- mobilidade: agentes podem se mover numa rede de computadores heterogênea, visando progressivamente completar tarefas que lhes foram atribuídas.

Nesta dissertação, os agentes possuem mobilidade e são programas autônomos que recebem autoridade de seus donos (usuários ou outros programas) para agir em favor destes na execução de tarefas. Por serem autônomos, estes agentes possuem uma certa inteligência, porém nenhum método de inteligência artificial é utilizado. Para completar as tarefas que lhes foram atribuídas, os agentes podem se comunicar com outros programas, com o ambiente que os abriga ou com seres humanos.

3.2 Tecnologia de Agentes

A tecnologia de agentes possui uma abrangência muito vasta, assunto de muitas pesquisas, porém está longe de ter sido totalmente explorada. O número de aplicações que reivindicam o uso de agentes tem crescido ao longo dos anos, gerando tipos específicos de agentes. Vários esquemas de classificação de agentes são encontrados na literatura baseados em critérios. Uma delas, bastante abrangente, é descrita por Nwana [Nwa96] e aponta alguns critérios possíveis para a sua classificação:

- mobilidade: capacidade de se mover numa rede. Agentes podem ser estáticos ou móveis;

- arquitetura: agentes podem ser deliberativos ou reativos. Agentes deliberativos possuem um modelo interno simbólico de raciocínio e fazem uso de planejamento e negociação para realizar coordenação com outros agentes. Agentes reativos não contêm modelos simbólicos do seu ambiente e agem usando um comportamento do tipo estímulo/resposta para responder ao estado presente do ambiente no qual estão inseridos;
- características básicas: autonomia, inteligência e cooperação. Tais características permitem a obtenção de quatro tipos de agentes: agentes inteligentes, agentes colaborativos, agentes de interface e agentes colaborativos que aprendem. Agentes inteligentes possuem as três características básicas. Agentes colaborativos se baseiam nas características de cooperação e autonomia. Agentes de interface se caracterizam por autonomia e aprendizado. Agentes colaborativos que aprendem se baseiam nas características de cooperação e aprendizado;
- papéis desempenhados: agentes da Internet ou agentes de informação auxiliam na gerência de informação em redes de longa distância como a Internet.

O conjunto de possíveis critérios é, então, combinado para gerar uma lista de tipos de agentes (Figura 3.1) que engloba: agentes colaborativos, agentes de interface, agentes móveis, agentes de informação, agentes reativos, agentes inteligentes, agentes híbridos (que combinam dois ou mais critérios) e sistemas de agentes heterogêneos (conjunto integrado de dois ou mais agentes que pertencem a dois ou mais tipos diferentes de agentes). A maioria dos trabalhos de classificação existentes até o presente momento não se preocupa em gerar uma lista extensiva de tipos de agentes, mas, sim, em apresentar alguns tipos vistos como mais importantes sob a ótica dos seus autores [Oli97]. Desta maneira, a classificação que cabe aos agentes desta dissertação são aqueles pertencentes à classe de agentes móveis, descritos com mais detalhes na Seção 3.3.

Apesar da abrangência da tecnologia de agentes estar crescendo, o mecanismo de RPC ainda é bastante utilizado para a comunicação entre componentes de sistemas cliente/servidor (Figura 3.2). As mensagens transportadas pela rede são requisições ou respostas. Uma requisição inclui dados que são os argumentos do procedimento, enquanto que, uma resposta inclui dados que são seus resultados. RPC se baseia em conexões síncronas entre um cliente e um servidor, isto é, o processo cliente tem sua execução suspensa na chamada do procedimento remoto até que o retorno seja recebido. Assim, recursos permanecem alocados mesmo quando não são utilizados (ineficiência de utilização) e custos de comunicação se tornam maiores do que o necessário.

Uma nova alternativa para o RPC é a Programação Remota. Com este novo paradigma, em vez de um processo cliente enviar os dados para o processamento de um método

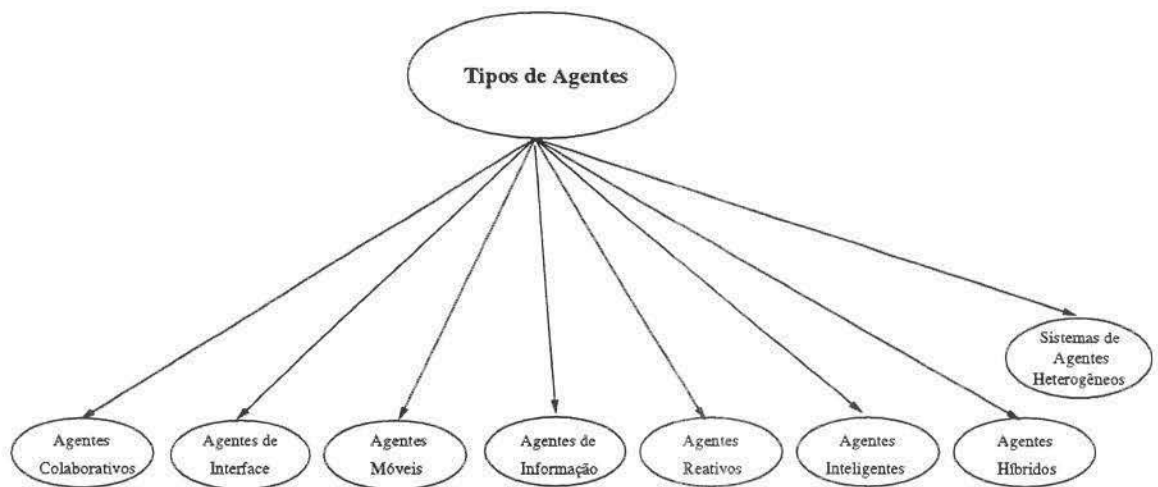


Figura 3.1: Classificação de agentes segundo Nwana

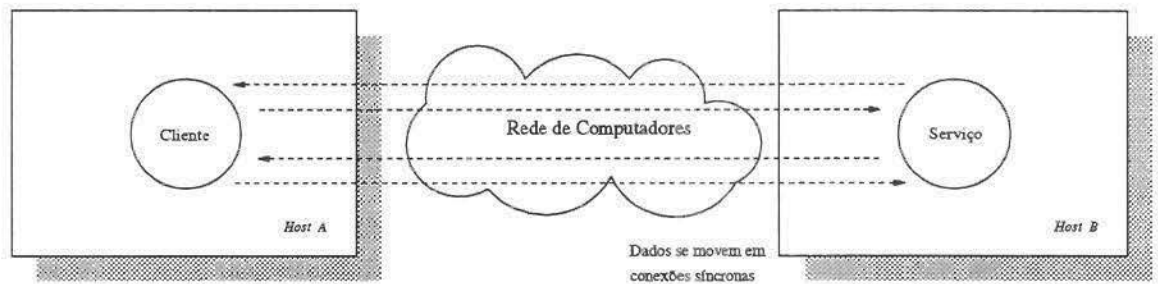


Figura 3.2: RPCs para comunicação entre componentes de sistemas cliente/servidor

implementado somente no servidor, envia-se não só os dados como também o próprio código. Cada mensagem que a rede transporta é composta de um procedimento a ser executado e dos dados que representam seus argumentos. Assim, chamadas de procedimentos passam a ser locais ao invés de remotas. Os agentes móveis são construídos sob o conceito de programação remota e podem trazer ganhos consideráveis de flexibilidade e eficiência ao paradigma cliente/servidor tradicional. Uma das grandes vantagens desta abordagem é evitar o sucessivo número de requisições e respostas através da rede. A Figura 3.3 ilustra esta nova abordagem. Mais detalhes sobre agentes móveis são descritos na próxima seção.

Outra solução que pode ser considerada para reduzir a comunicação na rede é apresentada em [BDM96] e consiste em mover o serviço para o computador cliente (Figura 3.4). Novamente a comunicação se torna local e não remota. Esta solução é menos dependente do servidor, tanto em termos de desempenho quanto de confiabilidade, mas exige que a potencialidade do cliente seja considerada.

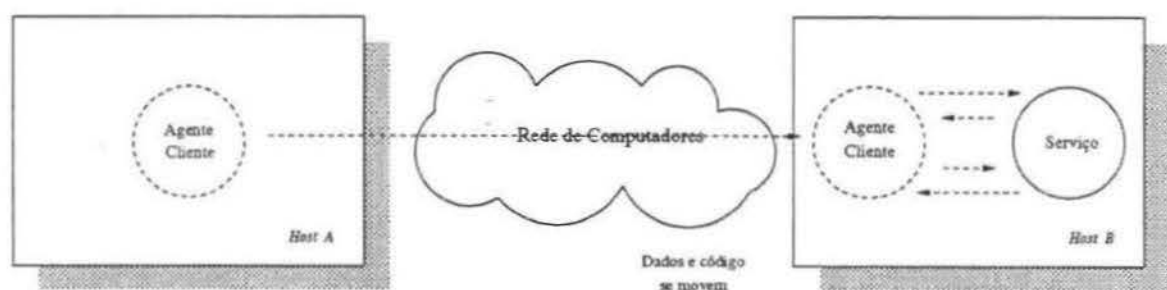


Figura 3.3: Migração de um agente móvel para utilização de recursos remotos

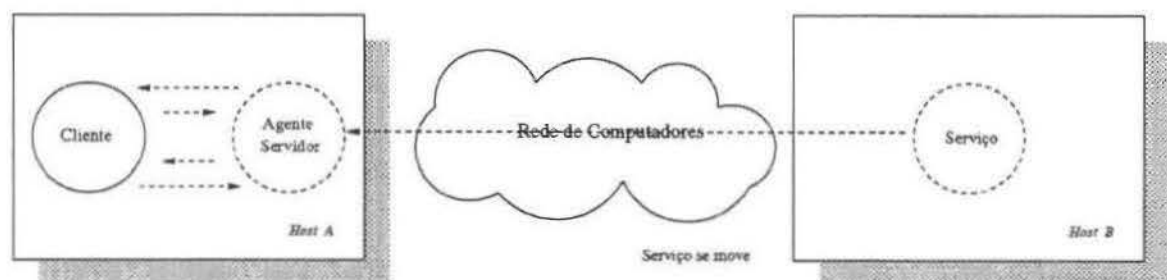


Figura 3.4: Migração de um serviço para o *host* cliente

O uso de agentes, naturalmente, não é novo. O relacionamento gerente/agente é intrínseco à maioria dos protocolos de gerência de rede padrão, incluindo SNMP e CMIP vistos no capítulo anterior. Contudo, o uso de agentes tem-se constituído numa alternativa para o desenvolvimento de sistemas distribuídos. O que é relativamente novo é o uso de agentes móveis para gerenciar atividades no nível de sistemas, foco de estudo deste trabalho.

3.3 Agentes Móveis

Ao contrário dos agentes estáticos, os quais permanecem em um único lugar durante seu tempo de vida, os agentes móveis, também chamados de itinerantes [CGH⁺95] ou transportáveis [KK96], viajam entre os *hosts* de uma rede para executarem suas tarefas. Agentes móveis são objetos que possuem código (comportamento), dados, estado de execução e itinerário. São autônomos e, portanto, podem escolher quando e para onde migrar, podem suspender sua execução em algum ponto arbitrário e podem mover para outra máquina e continuar (terminar) a execução em uma nova máquina. Para tal comportamento autônomo, os agentes móveis contêm algum nível de inteligência, o qual pode ser realizado desde simples mecanismos baseados em *scripts* até mecanismos complexos de inteligência artificial. Ainda com respeito à autonomia, a maioria dos agentes móveis se comunica com o usuário, recursos do sistema e/ou outros agentes para realizar as tarefas

que lhes foram designadas.

Agentes Móveis têm sido objeto de pesquisa desde meados dos anos 80, mas recentemente ganharam um grande impulso com o aparecimento da linguagem Java, devido às suas características de independência de plataforma, capacidade de carregamento dinâmico de código e orientação a objeto [End98].

Os agentes móveis necessitam de uma infra-estrutura que permita a sua mobilidade, na forma de ambientes locais que os ajudem a executar suas tarefas. Assim, sistemas de agentes móveis fornecem a infra-estrutura e oferecem um ambiente de desenvolvimento de agentes móveis. Alguns sistemas de agentes móveis são descritos na Seção 3.3.4.

3.3.1 Aplicações de Agentes Móveis

Os agentes móveis têm potencial de aplicação para qualquer tipo de sistema distribuído. Algumas aplicações mais frequentes são:

- Recuperação de informação
- Gerência de redes e sistemas distribuídos
- Comércio eletrônico
- Computação móvel
- Ambientes de apoio à cooperação

A recuperação de informação na rede pode ser suportada muito mais eficientemente se um agente representando uma indagação pode se mover para o lugar onde os dados estão realmente armazenados, ao invés de ter de mover todos os dados pela rede (subseqüentemente descartando grande parte da transmissão). Na área de gerência de redes e sistemas distribuídos, o uso de agentes móveis pode facilitar a tarefa do usuário (gerente da rede) no sentido de permitir uma coleta seletiva e direcionada de dados do sistema e componentes da rede, efetuando uma filtragem adequada da informação, e eventualmente executar algumas funções de inferência relacionando diferentes eventos a suas possíveis causas [End98]. Goldszmidt e Yemini [GY98] introduzem um sistema de gerência baseado em agentes delegados, cujas funcionalidades são associadas em tempo de execução do agente.

Com os negócios na Internet tornando-se uma realidade cada vez maior, o comércio eletrônico é um domínio que parece mais agradável para os agentes móveis. Eles podem ajudar a localizar ofertas de preços, negociar quantidades ou mesmo concluir transações de negócios a favor de seus donos.

Uma aplicação importante dos agentes móveis está na computação móvel. Os usuários simplesmente submetem agentes móveis para execução de alguma tarefa e se desconectam, retornando mais tarde para colher os resultados depositados pelos agentes móveis.

3.3.2 Vantagens, Desafios e Desvantagens

O uso de agentes móveis pode ter vantagens sobre outras implementações de agentes. Isto não implica que outras tecnologias (como objetos remotos) não possam ser usadas, pois qualquer tarefa que pode ser feita com agentes móveis pode também ser feita com objetos estacionários. As soluções tradicionais podem ser menos eficientes, difíceis de serem empregadas, ou complicadas [BPW98]. Algumas vantagens do uso de agentes móveis para determinadas aplicações são:

- **Eficiência:** em certos casos podem consumir menos recurso de rede, migrando para onde os dados se encontram. Também podem migrar para onde existem melhores recursos computacionais;
- **Economia de espaço:** o consumo de recurso é limitado, porque um agente móvel reside somente em um *host* por vez, não precisando ter sua funcionalidade duplicada, como acontece com os múltiplos objetos estáticos;
- **Provável redução do tráfego na rede:** a maioria dos protocolos de comunicação envolve várias interações entre as entidades comunicantes remotas. Com o uso dos agentes móveis estas interações passam a ser locais;
- **Interação autônoma assíncrona:** os agentes móveis podem ser delegados para fazer determinadas tarefas mesmo se a entidade que delegou não está ativa;
- **Suporte de ambientes heterogêneos:** os agentes móveis são separados dos *hosts* pela infra-estrutura de mobilidade. Se existe uma infra-estrutura num *host* então os agentes podem se mover até ele, aumentando a interoperabilidade entre estes ambientes;
- **Paradigma de desenvolvimento conveniente:** a parte mais difícil da criação de sistemas distribuídos baseados em agentes móveis é a infra-estrutura de mobilidade. Quando tal infra-estrutura está presente em um *host*, o desenvolvimento de aplicações é facilitado;
- **Robustez e tolerância a falhas:** a habilidade de reagirem dinamicamente em situações adversas tornam os agentes mais propícios a um comportamento tolerante a falhas;

- Extensibilidade de serviços: os agentes móveis podem ser usados para estender capacidades de aplicações, como por exemplo, fornecendo serviços. Isto permite a construção de sistemas que são extremamente flexíveis;
- Interação com sistemas em tempo real: o problema de latência, causado pelo congestionamento da rede, é algo intolerável no controle de aplicações de tempo real. O uso de agentes móveis pode evitar este problema.

Com relação aos desafios é unânime que segurança é o ponto chave para a consolidação da abordagem de sistemas baseados em agentes móveis. Isto porque o controle e monitoramento de agentes móveis é muito difícil e se faz necessária a proteção de ambientes contra ações nocivas executadas por agentes neles recebidos.

De acordo com Harrison et al. [HCK95] existem algumas desvantagens no uso de agentes móveis e estas estão diretamente relacionadas ao desafio na área de segurança. Segundo os autores, isto requer um estudo aprofundado e grande inovação técnica:

- necessidade de ambientes de execução com grande segurança;
- limitações funcionais e de desempenho como consequência do nível de segurança necessário;
- necessidade de mecanismos de controle e rastreamento de vírus de computador.

3.3.3 Requerimentos de Infra-estrutura para Agentes Móveis

As características de autonomia, comunicabilidade, mobilidade, auto-iniciação e orientação por metas relacionadas aos agentes, já explicadas na Seção 3.1, determinam a infra-estrutura para os agentes móveis. Assim, para os agentes se moverem em um sistema heterogêneo é necessária uma infra-estrutura sólida na forma de ambientes locais que os receba e os ajude a executar suas tarefas. Esses ambientes são chamados de agências [BDJ96]. As tarefas básicas das agências são:

- Execução: provê facilidades para agentes executarem como alocar recursos se necessário e possibilitar o comportamento autônomo. É necessário que o agente seja registrado, para permitir sua localização e interação com outros agentes;
- Comunicação: suporta a comunicação com outros agentes e possivelmente com seu proprietário (usuário/gerente), ambos locais ou remotos. Para isso, mecanismos como RPC ou invocação de objetos precisam ser suportados;

- **Transporte:** provê facilidades para o transporte dos agentes em um ambiente heterogêneo. Mover um objeto envolve mover seu código e seu estado (inclusive dados). Enquanto que, para um objeto móvel é o sistema quem determina quando e para onde será feito o transporte, para um agente móvel, por ser autônomo, é este quem decide quando se mover. A decisão para onde se mover poderá ser simplesmente tomada através de uma rota pré-estabelecida ou ser tomada de forma dinâmica, baseada nos serviços e recursos necessários ou na interação com o ambiente. Quando um agente decidir se mover, diferentes partes da agência precisam ser informadas para que as ações apropriadas sejam tomadas, como o preparo do agente para transporte, a retirada do respectivo registro da agência e a liberação dos recursos que estavam sendo utilizados;
- **Segurança:** provê segurança dos agentes de suas interações. Desde a recepção do agente é importante verificar se existem problemas relacionados à origem do agente ou às condições do mesmo ao ser recebido (código suspeito ou confuso). O ambiente de execução de agentes poderá estabelecer um apropriado controle de acesso a recursos e, por outro lado, prover mecanismos para evitar erros de interpretação ou possíveis modificações no código do agente.

3.3.4 Sistemas de Agentes Móveis

Apesar de agentes móveis serem uma tecnologia recente, já existem vários sistemas de agentes móveis disponíveis para uso. As principais características comuns à maioria dos sistemas são [End98]:

- Disponibilidade de uma linguagem de programação de agentes. Teoricamente, qualquer linguagem pode ser usada. Porém, são preferíveis as linguagens interpretadas como Java ou Tcl, às compiladas. As linguagens interpretadas possuem adaptação para todas as arquiteturas nas quais existe um interpretador para a linguagem e são facilmente extensíveis, permitindo assim a incorporação de comandos específicos para o transporte e comunicação entre agentes;
- Fornecimento de uma agência que precisa executar em cada *host* da rede potencialmente visitado por um agente, com as características citadas na Seção anterior;
- Capacidade de serialização do estado de agentes. A linguagem Java já provê tal facilidade, enquanto que em alguns ambientes esta facilidade precisa ser incorporada na forma de primitivas da linguagem de programação dos agentes;

- A maioria dos sistemas tem a comunicação baseada em protocolos do nível de transporte ou superiores bem estabelecidos como o TCP, HTTP (*Hypertext Transfer Protocol*) ou SMTP (*Simple Mail Transfer Protocol*) para realizar a comunicação entre as agências;
- Muitos sistemas já permitem iniciar e interagir com um programa baseado em agentes móveis através de *browsers* WWW;
- A maioria dos sistemas permitem uma definição dinâmica de itinerários, isto é, os itinerários podem ser definidos pelas interações prévias com outros agentes e lugares visitados pelo agente.

Linguagens como Obliq, Scheme, Python, Perl, Tcl muito usadas para o desenvolvimento de agentes móveis, deram lugar à linguagem Java. Hoje, a grande maioria dos sistemas são desenvolvidos utilizando esta linguagem. Alguns sistemas possuem origem acadêmica, outros na indústria. Os mais conhecidos e utilizados são descritos brevemente a seguir. Maiores detalhes dos quatro primeiros sistemas são encontrados no Apêndice A e, do SMM (Suporte à Mobilidade da plataforma *Multiware*), utilizado para o desenvolvimento deste trabalho, são encontrados no Capítulo 5. Porém, detalhes de projeto e desenvolvimento do SMM são melhores descritos em [VM98, Vas99].

Tabriz e Odyssey

O *Tabriz* [Whi97], mais conhecido pela linguagem utilizada para os agentes, *Telescript*, foi apresentado pela *General Magic* no ano de 1994 e se tornou a primeira implementação comercial disponível do gênero. Este sistema é robusto e abrangente. Entretanto, é uma tecnologia proprietária, custosa e que requer recursos computacionais significativos. Assim sendo, a *General Magic* partiu para o desenvolvimento de uma alternativa baseada na linguagem Java, chamada de *Odyssey* [Gen]. O *Odyssey* é baseado na arquitetura *Telescript* e é independente de transporte. Enquanto um agente viaja de um lugar para outro, o *Odyssey* utiliza uma API de transporte projetada pela *General Magic* para tal propósito. Desta forma, ele possui suporte aos seguintes mecanismos de transporte: RMI da linguagem Java, DCOM da *Microsoft* e IIOP do OMG.

AgentTcl

O *AgentTcl* [Dar] é um sistema de agentes móveis desenvolvido no *Dartmouth College* que permite a programação de agentes móveis em Tcl/Tk, Java e Scheme. Ela foi desenvolvida sobre Tcl/Tk, uma linguagem de *scripts* e interpretada e, por isso, *AgentTcl* é independente de plataforma. Esta infra-estrutura provê uma série de facilidades para a

movimentação, a comunicação e a depuração de agentes, além de um elaborado sistema de segurança. Sua principal contribuição é o modelo de estaleiro (*docking system*), permitindo que os agentes móveis aguardem a conexão de computadores móveis à rede para poderem se transferir para os mesmos.

Aglets Workbench

Aglets Workbench (AWB) [LO97, Ven97] é um ambiente visual para construção de aplicações de agentes móveis em Java. Ele foi desenvolvido pelo laboratório de pesquisa da IBM em Tóquio e disponibilizado comercialmente a partir de meados de 1996. O AWB introduz o conceito de *aglet*. Um *aglet* (= *applet* + agente) é um objeto Java móvel que visita os *hosts* de uma rede de computadores estendendo o modelo de código carregável dinamicamente introduzido por Java *applets*. Este sistema progrediu muito nos últimos dois anos, atingindo um público cada vez maior. Tal fato é consequência de uma série de fatores como: foi o primeiro a utilizar a linguagem de programação Java, foi submetido na forma de RFP à facilidade de agentes móveis do OMG e possui um modelo abrangente e bem projetado, facilidade de operação, boa documentação e grande exposição a contextos externos ao seu ambiente de desenvolvimento.

Voyager

Voyager é um ambiente de programação de agentes móveis comercial baseado na tecnologia Java, cujo principal componente é um ORB que provê suporte a objetos e agentes móveis. Além do ORB, serviços de persistência, comunicação em grupo e diretório de nomes fazem parte deste sistema. *Voyager* foi desenvolvido pela empresa *ObjectSpace* e lançado comercialmente em setembro de 1997. *Voyager* possui uma boa documentação. A sua utilização é sem custos para uso comercial com algumas restrições. Trata-se de um sistema robusto, bem projetado e com funcionalidades amplas. Versões beta teste foram colocadas à disposição para uso externo desde abril de 1997. A versão atual é a 3.0 beta.

SMM - Suporte à Mobilidade da plataforma Multiware

O sistema de suporte à mobilidade SMM [VM98, Vas99] foi desenvolvido no Instituto de Computação da UNICAMP em meados de 1997 como parte da plataforma *Multiware* apresentada na Seção 2.3. O SMM permite o desenvolvimento de agentes móveis na linguagem Java. As principais características relacionadas a este suporte são:

- os agentes são objetos CORBA, ou seja, possuem sua interface definida em IDL de modo a serem acessíveis por qualquer objeto cliente e acessarem qualquer outro objeto CORBA, independente de linguagem de programação;

- os agentes são identificados por sua referência de objeto e assim também localizados transparentemente pelo ORB;
- suporta qualquer número de agentes, de qualquer tipo, em *threads* independentes e com igual prioridade;
- permite a captura do estado do agente, representado pelas variáveis globais;
- para a mobilidade é utilizado o protocolo IIOP;
- permite a auto-iniciação e a autonomia de execução de agentes, bem como a possibilidade de se terminar agentes através de comandos externos aos mesmos;
- fornece transparência de localização, permitindo a comunicação com os agentes independente de sua localização, pois a referência de objeto é atualizada quando o agente se move.

Os sistemas de agentes móveis conhecidos desenvolvidos em Java utilizam o *Object Serialization*; alguns exemplos, além dos descritos acima, são: *Mole* [Unic] da *University of Stuttgart*, *MOA (Mobile Objects and Agents)* [Ope] da *Open Group*, *MuBot (Mobile unstructured Business object technology)* [Cry] da *Crystaliz*, *Concordia* [Mit] da *Mitsubishi*, *AMETAS (Asynchronous MESSage Transfer Agent System)* [Dis] da *Universitat at Frankfurt* e *Java-To-Go* [Unib] da *University of California at Berkeley*. Alguns sistemas de agentes móveis que não usam Java são: *ARA (Agents for Remote Actions)* [Unia] da *Universität Kaiserlautern*, *Tacoma* [Cor] da *Cornell* e *Tromso's University*, *Coast* [Pur] da *Purdue University* e *MSAs (Mobile Service Agents)* [Eur] do *ECRC (European Computer-Industry Research Center)*.

Mais recentemente, no final de 1998, surgiu o *Grasshopper* [IKV98], cujo percussor foi o *MAGNA* [KSM⁺97]. É um ambiente de agentes móveis inteligentes que segue o padrão *OMG/MASIF (Object Management Group/Mobile Agent System Interoperability Facilities Specification)*. Permite a criação de aplicações distribuídas com agentes se beneficiando de comunicação e acesso a dados localmente. Ele implementa uma facilidade de agentes móveis que poderia ser usada como a estrutura para o desenvolvimento de um sistema de gerência baseado no paradigma de agentes móveis. Um problema é que ele é dependente do mecanismo de geração automática de *stubs* não padronizado e somente disponível no *VisiBroker* para Java. O *Grasshopper* é o único sistema de agentes que procura realmente se adequar à migração dos agentes em um ORB.

3.4 Mobilidade e CORBA

A especificação das *CORBAfacilities* [OMG97b] oferece serviços qualificados como Gerência de Sistemas, Gerência de Tarefas e, como parte deste, a Facilidade de Agentes Móveis (MAF - *Mobile Agent Facilities*). Para especificar este serviço, o OMG emitiu em novembro de 1995 um RFP[OMG95] . Neste documento são descritos os seguintes requisitos básicos que um modelo deverá suportar:

- Empacotamento (*marshalling*) e desempacotamento (*unmarshalling*) de agentes. A Facilidade deverá estabelecer uma maneira de transmitir os pacotes codificados;
- Codificação e decodificação de pacotes de agentes para transporte;
- Mover agentes de uma agência para outra;
- Registro e invocação de agências;
- Consulta, em tempo de execução, pelos agentes, sobre serviços que podem ser fornecidos;
- Segurança de acesso e execução de agentes.

Opcionalmente, a Facilidade de Agentes Móveis pode prover:

- Identificação e localização de agentes;
- Inicialização, parada e suspensão da execução de agentes;
- Descoberta e monitorização, por outras agências ou outros agentes, de agentes pelo nome, por suas características ou por suas capacidades.

A Facilidade de Agentes Móveis está relacionada com outras *CORBAfacilities*, e utiliza alguns *CORBA services* como serviço de nomes, ciclo de vida e *trader*.

Com as diferentes abordagens apresentadas, várias re-submissões foram feitas em busca de consenso de modo que a Facilidade de Agentes Móveis se transformou, em 1997, na Facilidade de Interoperabilidade de Sistemas de Agentes Móveis (MASIF) [GMD97]. Este documento passa a não tratar os agentes como objeto CORBA e não definir mecanismos de transporte dos agentes por um ORB. Também diz que, a interoperabilidade é vista somente para sistemas de agentes móveis escritos na mesma linguagem, pois além da interoperabilidade de linguagem ser tecnicamente difícil de se alcançar, o suporte para linguagens diferentes pode ser replicado em cada *host*. O fato de não prover mecanismos de transporte de agentes foi pela independência de linguagens de programação.

Considerando sistemas de agentes móveis escritos na mesma linguagem, a especificação da MASIF define três tipos de interoperabilidade: gerência, rastreamento e transporte de agentes. A gerência de agentes permite que um agente controle agentes de um outro sistema de agentes. A sua implementação é relativamente direta para a maioria dos sistemas de agentes móveis. O rastreamento de agentes permite que agentes de diferentes sistemas, registrados com serviços de nomes, possam ter sua execução rastreada. Esta interoperabilidade também é relativamente direta para a maioria dos sistemas. Por fim, a interoperabilidade do transporte de agentes é complexa e requer uma certa cooperação entre desenvolvedores de diferentes sistemas de agentes móveis.

Uma nova característica e de grande importância está surgindo com a especificação da CORBA3.0: **passagem de objetos por valor**. O OMG já havia percebido sua necessidade quando emitiu um RFP intitulado *Object by Value* [OMG96] em 1996. O que ainda se tem hoje, até que CORBA3.0 seja disponibilizada comercialmente, é o uso da referência de objeto que faz com que uma requisição se mova pela rede até o objeto, e em certas situações, isto pode não ser muito eficiente. Na submissão conjunta adotada pelo OMG [OMG98], a passagem de objetos por valor entre aplicações CORBA é suportada por uma nova estrutura adicionada à IDL do OMG, denominada *valuetype*. O *valuetype* possui características de *struct* e *interface* pois suporta tanto dados quanto operações e, ainda pode derivar de outros *valuetypes*. Quando um *valuetype* é passado como argumento para uma operação remota, é criada uma cópia no espaço de endereçamento do receptor. A identidade desta cópia é totalmente separada da original, e operações em uma não afetam a outra. Assim, todas as operações invocadas são sempre locais ao processo onde o *valuetype* existe, não envolvendo a transmissão de requisições e respostas na rede [Vin98].

Porém, a especificação *Objects By Value* do OMG não trata a passagem de objetos CORBA por valor, fugindo assim do objetivo inicial estabelecido pelo RFP. Embora esta especificação permita que *valuetypes* tornem-se objetos CORBA por herança de interfaces IDL, somente as referências de objetos poderão ser passadas como argumentos neste caso. Passar objetos CORBA por valor é um problema difícil devido às interações com adaptadores de objetos, às questões relacionadas a localização de objetos, identidade dos objetos e quando estes podem ser copiados e movidos, e às questões relacionadas à migração de objetos entre ORBs de diferentes fabricantes. Com esta especificação, segundo Vinoski [Vin98], o OMG quebrou a tradição de se basear em tecnologias bem conhecidas e comprovadas, mas está se empenhando para evitar situações similares no futuro.

3.5 Gerência utilizando Agentes Móveis

Os modelos de gerência mais conhecidos OSI e Internet, apresentados no Capítulo 2, possuem uma mesma arquitetura central gerente/agente: os agentes fornecem uma visão de gerência sobre os recursos gerenciados que, através de protocolos de gerência, respectivamente CMIP ou SNMP, reportam as informações de gerência ao gerente da rede. Destes dois, o SNMP é o protocolo mais tradicional e utilizado comercialmente. Enquanto isso, CORBA vem ganhando confiança em muitas áreas e gerência é uma delas. De formas diferentes, estes três modelos seguem o paradigma cliente/servidor, o qual o gerente faz o papel de cliente e o agente de servidor. A base para este paradigma é o mecanismo de RPC visto na Seção 3.2, onde a rede é um elemento sensível.

Nos últimos tempos, tem havido a necessidade de buscas por novos meios de gerenciar redes e sistemas distribuídos de maneira mais eficiente, facilitando a tarefa do operador da rede. Na Seção 2.2 foi visto que a descentralização do sistema de gerência é algo a considerar. O crescimento rápido das redes e sistemas distribuídos afeta diretamente a escalabilidade dos sistemas de gerência e isto envolve os seguintes fatores: o número de recursos gerenciados, o número de parâmetros dos recursos gerenciados e a banda larga do canal de comunicação entre o sistema de gerência e o recurso gerenciado. Para sistemas de gerência centralizados como SNMP e CMIP, nos quais a escalabilidade é um fator preocupante, se qualquer um destes fatores aumentar ou o último diminuir, pode haver um congestionamento no sistema de gerência e no canal de comunicação e a rede se tornar não gerenciável [Kel96]. Vale ressaltar que, o SNMPv2 apresenta em sua arquitetura uma maneira de descentralizar a gerência, utilizando gerentes distribuídos.

Entretanto, ao invés de otimizar o desempenho e melhorar a eficiência de sistemas de gerência já bem estabelecidos, uma nova abordagem pode ser adotada: agentes móveis. Estes surgem como uma solução alternativa para a gerência de sistemas distribuídos, com todas as suas características e vantagens descritas na Seção 3.3. Os agentes dos modelos de gerência tradicionais são estáticos e estão próximos do recurso gerenciado, fazendo com que os dados e/ou comandos se movam pela rede. A mobilidade dada aos agentes geralmente reduz a latência na rede e aumenta a velocidade de processamento, já que o código é movido em direção aos dados – *caching* inverso [Gol96]. Isto implica que o código dos agentes de gerência deve ser leve.

Como dito em [Obj97], a troca de mensagens locais é de 10.000 a 1.000.000 de vezes mais rápida que as remotas. Isto mostra que apesar do agente (estático) estar próximo do recurso gerenciado, onde a troca de mensagens é local, a conversação entre gerente e agente remotos põe em circulação muitos dados, possivelmente desnecessários, na rede.

O uso de agentes móveis na gerência vem facilitar o trabalho do operador que, com a crescente diversidade de sistemas e redes, tem se tornado difícil e cada vez mais com-

plexo. A quantidade de dados operacionais que devem ser monitorados e processados é maior. Como mais dados são coletados, informações não importantes e redundantes são repassadas ao operador. Num paradigma tradicional gerente/agente (cliente/servidor), no qual as partes são estáticas, um agente meramente coleta e reporta dados monitorados e todo o processamento e inteligência estão no gerente. Este paradigma clássico pode então ser mudado para uma gerência baseada em agentes móveis autônomos que podem automatizar e descentralizar as atividades de gerência.

O assincronismo dos agentes móveis necessita menor organização de mensagens e sincronismo entre processamentos. Eles mudam a maneira de manipular os dados de gerência e a forma de acessar os recursos gerenciados [Sus97]. Como eles se deslocam até os recursos gerenciados e conseqüentemente aonde os dados estão, todo o processamento de informações ocorre localmente e apenas aquelas informações relevantes são levadas para o operador/gerente da rede.

Este novo enfoque na gerência é algo ainda recente e bastante inovador, por isso a literatura é um tanto restrita. Mas, nos últimos anos esta área tem despertado a curiosidade e o interesse de muitos pesquisadores, possibilitando discussões sobre novas abordagens na gerência, como a utilização dos agentes móveis.

3.6 Resumo

Uma classe de agentes tem despertado grande interesse de pesquisadores nos últimos anos: agentes móveis. Eles têm a característica de migrar de um *host* para outro da rede, carregando seu código e estado, executando tarefas em benefício de alguém. A sua aplicabilidade pode ser vista pela quantidade de sistemas de agentes móveis disponíveis hoje, alguns deles vistos na Seção 3.3.4.

O OMG sentiu o crescimento deste paradigma e acrescentou na especificação CORBA a facilidade de agentes móveis, antes MAF e hoje denominada MASIF.

A área de gerência de redes e sistemas distribuídos está sofrendo bastante influência do paradigma de agentes móveis, trazendo uma nova abordagem e muitas vantagens. Uma delas e talvez a mais importante é que, ao contrário da gerência tradicional gerente/agente, os agentes móveis fazem o processamento localmente evitando sucessão de requisições e respostas pela rede. O próximo Capítulo descreve uma maneira de aplicar os agentes móveis na gerência de sistemas distribuídos num ambiente CORBA.

Capítulo 4

Modelagem Proposta para a Gerência de Monitorização

A proposta deste trabalho é monitorar os componentes de aplicação CORBA com a utilização dos agentes móveis. Este capítulo apresenta a idéia desenvolvida para alcançar este objetivo.

4.1 Integrando CORBA e Agentes Móveis para a Gerência

CORBA e agentes móveis são duas tecnologias inovadoras que juntas podem trazer muitos benefícios à gerência de sistemas distribuídos. Isto não significa que esta abordagem deva ser competitiva aos outros modelos e paradigmas que existem, mas sim uma forma de acrescentar soluções no mundo crescente e complexo de sistemas heterogêneos, assim como soluções baseadas em tecnologia de código não móvel podem sempre ser propostas. Desta forma, a utilização de agentes móveis adiciona mais um cenário ao modelo de gerência CORBA, em complemento àqueles ilustrados pelas Figuras 2.4 (a) e (b) do Capítulo 2. A Figura 4.1 mostra este novo cenário, onde os agentes estáticos são substituídos pelos agentes móveis. Nesta Figura, pode-se ver que o agente móvel possui o itinerário B e C (passos 1 e 2), retornando para a origem A (passo 3). A Figura também ilustra que os agentes podem não se moverem pelo ORB (passos a, b e c).

A arquitetura CORBA pode ser vista como um modelo de gerência que adota o paradigma centralizado ou com graus de descentralização, dependendo de como a arquitetura é abordada para tal funcionalidade. Neste trabalho, os agentes móveis se distribuem pela rede monitorando os objetos distribuídos CORBA e possuem uma certa autonomia se reportando a um ou mais gerentes (aplicações e/ou humanos) da rede, caracterizando uma

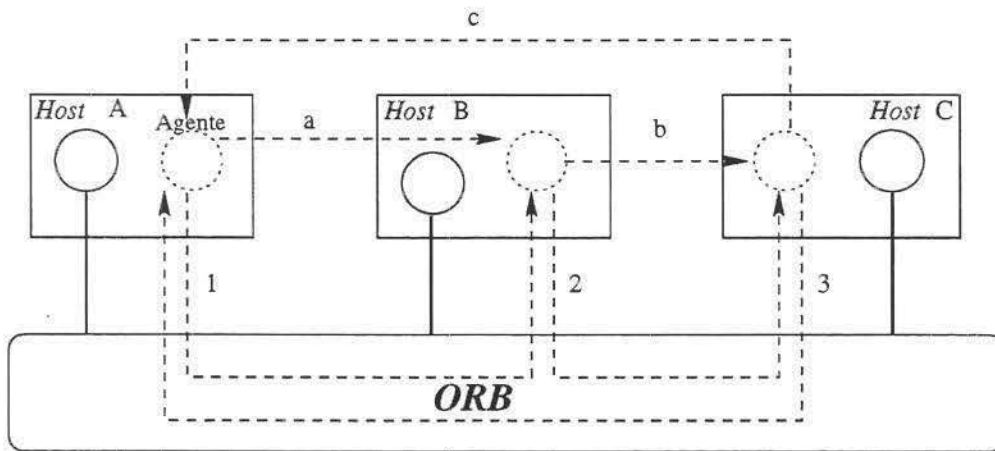


Figura 4.1: Agentes móveis no modelo de gerência CORBA

descentralização parcial da gerência. Esta descentralização não é total porque, como será visto, os agentes móveis não são completamente autônomos a ponto de mudarem suas funcionalidades dinamicamente para agirem pelos operadores/gerentes, e os gerentes distribuídos ainda não compartilham informações. CORBA é o modelo de gerência utilizado e o ambiente a ser gerenciado pelos agentes móveis, dentro do contexto da plataforma *Multiware*.

4.2 Trabalhos Relacionados

A arquitetura JDMK (*Java Dynamic Management Kit*) [Sun99], desenvolvida pela *Sun*, consiste de um conjunto de agentes estáticos que recebem *beans* dinamicamente distribuídos pela rede, sendo eventual e momentaneamente incorporados no agente, a fim de adicionar, modificar ou suprimir serviços. Possui uma biblioteca com um núcleo de serviços de gerência implementados como componentes *JavaBeans* e permite que novos serviços de gerência possam ser criados. Os agentes JDMK podem colaborar diretamente na resolução de problemas, ao invés de tradicionalmente repassá-los a um gerente num nível mais elevado. O JDMK possibilita a integração de infra-estruturas de rede existentes, incluindo os produtos de gerência baseados em SNMP. A inteligência integrada nos dispositivos pretende: reduzir o tráfego de gerência na rede, permitir que problemas de baixo nível sejam tratados localmente sem geração de alarmes, permitir uma resposta mais rápida aos eventos e reduzir os custos de administração. Nesta arquitetura, os *beans* são a parte móvel do código visto que os agentes são estáticos, trazendo benefícios como: integração rápida, compatibilidade, reuso de código, economia de tempo e escalabilidade dinâmica.

Trabalhos desenvolvidos na *Carleton University* pertencentes ao projeto *Perpetuum* tratam da gerência de redes utilizando agentes móveis. A infra-estrutura desenvolvida é toda em Java, e fornece o suporte à mobilidade, o acesso a recursos gerenciados e a comunicação entre agentes [Sus97]. Um relatório técnico [Cam97] publicado em abril de 1997, descreve como os agentes móveis gerenciam falhas em redes, sendo enviados a dispositivos particulares, onde o protocolo utilizado é o SNMP. Para fazer os testes, foi desenvolvido um simulador de rede baseado em programas de software Java. O projeto *Perpetuum* mostra a favorável viabilidade dos agentes móveis na gerência, porém não os trata num ambiente CORBA.

Um trabalho desenvolvido no mestrado do Instituto de Computação da UNICAMP fez a instrumentação de componentes de aplicação distribuídos CORBA nas áreas de desempenho e contabilização, sem utilizar código móvel. Os dados são gerados através de sensores incorporados aos processos clientes e servidores para a aquisição de métricas de interesse para a gerência [Que97]. Da mesma forma, em [BU96] é descrito um projeto para a monitorização de aplicações baseadas em CORBA, o qual esboça uma MIB CORBA para a gerência.

Porém, destes trabalhos citados, nenhum trata a gerência de aplicações CORBA utilizando agentes móveis. A maioria dos poucos trabalhos que existem nesta área, abordam os agentes móveis para a gerência de rede em outros ambientes que não a CORBA. Esta dissertação tomou como partida o trabalho descrito em [Que97], citado anteriormente, incluindo os agentes móveis na gerência de sistemas distribuídos.

4.3 Informações de Gerência

Para que um sistema possa ser gerenciado, os recursos que compõem este sistema precisam estar aptos a fornecerem informações de gerência. No mundo de gerência de redes e sistemas distribuídos, estes recursos são comumente chamados de objetos gerenciados, como visto nos modelos Internet, OSI e no ambiente de objetos distribuídos CORBA (Capítulo 2). Um objeto gerenciado fornece uma visão de gerência sobre um recurso. Esta visão de gerência consiste de uma funcionalidade adicional que não é necessária para a operação “normal” do recurso (Figura 4.2).

Esta extensão funcional é apenas para propósito de gerência e consiste de:

- atributos que representam propriedades dos componentes como número da versão e estado do componente;
- operações de gerência (métodos/ações) sobre os objetos gerenciados;

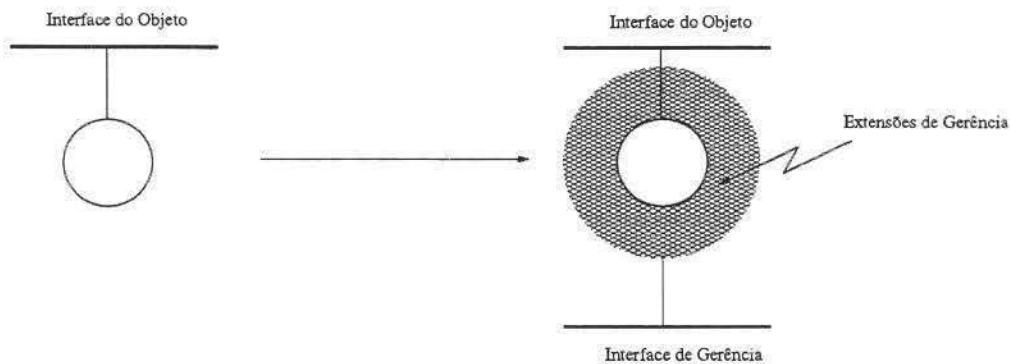


Figura 4.2: Extensões de gerência de um objeto

- notificações que representam eventos assíncronos a serem reportados pelos objetos gerenciados.

Gerenciar um sistema baseado em CORBA envolve primeiramente conhecer quais são os componentes que podem estar sujeitos à gerência, para então obter informações de gerência acrescentando funcionalidades a eles para tal propósito. Segundo Brunne e Uslander [BU96], estes componentes são:

Componentes de aplicação CORBA:

- implementação cliente
- implementação de objeto, ou objeto servidor (incluindo objetos de aplicação, *CORBAfacilities* ou *CORBAservices*)

Componentes centrais da CORBA:

- Adaptador de Objetos
- núcleo do ORB

Componentes de Repositório:

- Repositório de Implementação
- Repositório de Interface

Além destes componentes deve-se considerar que existe uma lacuna entre os componentes da arquitetura abstrata descritos acima e os componentes que participam num ambiente CORBA real. Por exemplo, o núcleo do ORB frequentemente não é visível como uma entidade única, mas como reside numa biblioteca a qual é ligada aos programas

de aplicação no caso de muitas implementações CORBA, o núcleo do ORB está presente em tempo de execução. Assim, considerando os ambientes ORBs que suportam o BOA, outro componente importante torna-se sujeito à gerência:

- servidor de objetos: composto por uma ou mais implementações de objetos.

O servidor de objetos, o qual contém objetos que fornecem serviços, é o componente registrado num ORB. Isto permite que ele seja ativado automaticamente (caso não esteja) quando um cliente fizer uma requisição a algum objeto pertencente a este servidor, através da referência de objeto. Em um sistema operacional orientado a processos, as implementações cliente e os servidores de objetos, quando ativos, são processos executando num *host*. A CORBA define quatro modos de ativação para um servidor [OH98]: compartilhado, vários objetos residem no mesmo processo servidor num *host*, é o modo mais utilizado; não compartilhado, cada objeto reside em um processo servidor diferente; chamada por método, para cada requisição feita um processo servidor é instanciado; e servidor persistente que é um caso especial do modo compartilhado, sendo que a diferença é que ele não é manipulado pelo ORB.

A implementação cliente, por ser apenas uma aplicação cliente (requisitante), não possui interface IDL e logo não recebe requisições de outros objetos. Isto implica que ela não precisa ser um componente registrado no ORB. Porém nada impede que um cliente seja preparado para o contrário.

Como o propósito deste trabalho é a gerência orientada basicamente na monitorização da atividade dos objetos servidores (nível de aplicação) num *middleware* ORB, as implementações de objetos e conseqüentemente os servidores de objetos, tornam-se os componentes de interesse para a gerência daqui para frente. Porém, em aplicações CORBA reais, os papéis de cliente e servidor são freqüentemente combinados, ou seja, uma implementação de objeto pode usar outros objetos CORBA [BU96]. A Figura 4.3 ilustra esta situação. O objeto servidor (Objeto B) exerce a função de cliente por algum momento quando este precisar consultar outro objeto servidor (Objeto C) para completar a execução de uma requisição vinda do cliente (Objeto A). Os Objetos B e C representam as implementações de objetos e pertencem a servidores de objetos diferentes (processos). Apesar do cliente (Objeto A) também ser um componente de aplicação importante a ser gerenciado, este é temporário num ambiente, e gerenciá-lo está fora do escopo deste trabalho. Porém, todas as requisições e respostas estão sendo monitorados no lado servidor. Também estão fora do alcance de gerência deste trabalho, os componentes centrais da CORBA e os componentes de repositório.

Meyer e Popiens [MP95] sugerem que, de acordo com as áreas funcionais OSI, a gerência de aplicações inclua: contabilização, configuração, falhas, desempenho e segurança. Entretanto, a abrangência dessas funções de gerência é muito grande, permitindo

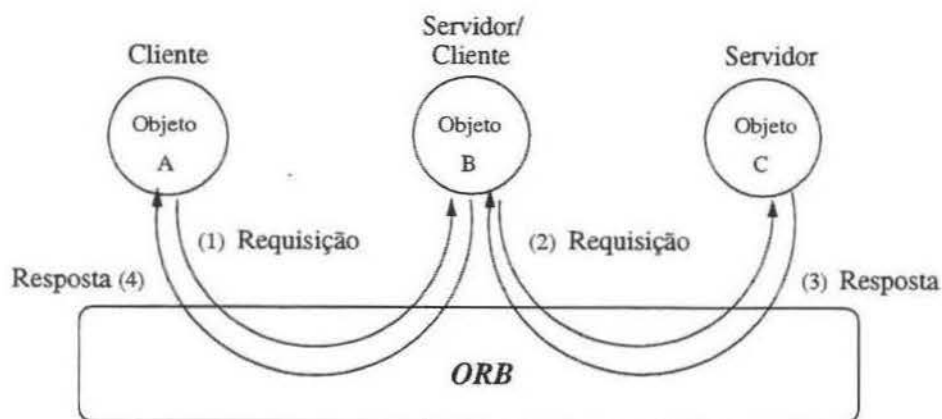


Figura 4.3: Papéis de Cliente e Servidor

que um subconjunto delas seja analisado. Assim, as informações de gerência que fazem parte do enfoque deste trabalho se concentram em duas áreas funcionais: desempenho e contabilização. A gerência de desempenho permite a avaliação do comportamento dos recursos gerenciados, e a gerência de contabilização permite a análise da utilização dos recursos e conseqüentes custos.

Devido à complexidade de gerenciar sistemas distribuídos, as informações de gerência foram classificadas em dois tipos:

- visão global do sistema: fornece pouca informação num domínio mais amplo;
- visão detalhada do sistema: fornece mais informação num domínio menor.

Para abranger estes dois tipos de informação, foi proposto um *pool* de agentes: **agentes em largura (agL)** e **agentes em profundidade (agP)**. Agentes em largura fornecem uma visão geral, geralmente visitando vários *hosts*. Eles coletam pouca informação tornando-os mais simples. Ao contrário, os agentes em profundidade permitem uma visão mais detalhada coletando muita informação num número menor de *hosts*. Isto implica que o agente é mais sofisticado, porém ainda leve.

Considerando os componentes a serem gerenciados neste trabalho e que as implementações de objetos pertencem a processos (servidores de objetos) num *host*, o *pool* de agentes possibilita *zooms* sucessivos de informações em três níveis diferentes: *host*, processo e objeto. No nível por *host*, todos os processos servidores de objetos do ORB que estão ativos naquele *host* são analisados, enquanto que no nível por processo apenas alguns destes processos ativos são analisados. No nível por objeto a análise atinge implementações de objetos em particular, pertencentes a um servidor de objetos. Geralmente, os agentes em largura compreendem os níveis de *host* e processo e, os agentes em profundidade os níveis de processo e objeto.

Para que as informações de gerência possam ser coletadas dos componentes alvos de gerência num ORB, estes precisam ser instrumentados. A próxima Seção descreve o mecanismo de instrumentação.

4.4 Instrumentação no Ambiente CORBA

Um dos conjuntos das *CORBA facilities*, apresentados na Seção 2.2.3, é o de Gerência de Sistemas. Ele fornece facilidades que são utilizadas para gerenciar componentes CORBA. Uma das facilidades de suporte à tarefa de monitorização dos componentes de aplicação CORBA é a **Instrumentação**. Esta facilidade permite obter dados oriundos dos recursos gerenciáveis através da incorporação de software especializados dentro de programas, com mecanismos para medição e cálculo dos dados. Baseado neste conceito, pode-se criar um ambiente para obtenção de informações de gerência. Para isso, definições de métricas de desempenho são necessárias para que informações possam ser medidas. A maioria das métricas obtidas para a gerência de desempenho é também usada para a contabilização [HA94]. Elas definem quantidades de medidas que fornecem dados para avaliar o desempenho do sistema, como por exemplo, identificar as possíveis causas de um excessivo tempo de residência de um servidor e os servidores responsáveis pelo consumo de recursos em um ambiente heterogêneo como o da *Multiware*. As definições de métricas para o desenvolvimento deste trabalho foram baseadas na especificação apresentada por Friedrich et al. [FSH⁺95] para monitorização de aplicações em um ambiente DCE, que podem também ser aplicadas ao ambiente CORBA. As métricas básicas definidas foram:

- intervalo de tempo em que as observações foram feitas: *T*
- número de requisições
- número de respostas
- tempo de empacotamento (*marshalling*) dos parâmetros pelos *stubs*
- tempo de desempacotamento (*unmarshalling*) dos parâmetros pelos *stubs*
- tempo de residência de processamento (tempo que o servidor leva para processar a requisição)
- tempo de resposta (tempo que o cliente espera pela resposta)
- *throughput* (relação do número de requisições e respostas pelo tempo *T*)

Essas métricas, de um modo geral, podem ser definidas sob o ponto de vista de um cliente ou de um servidor. A Figura 4.4 mostra os pontos de obtenção destas métricas para as duas partes que compõem a computação distribuída num ORB. Assim, derivando-as para a perspectiva do servidor, as métricas são:

- número de requisições recebidas: S1
- tempo de desempacotamento dos parâmetros: S2
- tempo de residência total de processamento: S3
- número de respostas enviadas: S4
- tempo de empacotamento dos parâmetros: S5
- *throughput*: S6
relação do (S1) e do (S4) pelo tempo de observação T

Da mesma forma são derivadas métricas para o ponto de vista do cliente:

- número de requisições enviadas: C1
- tempo de empacotamento dos parâmetros: C2
- número de respostas recebidas: C3
- tempo de desempacotamento dos parâmetros: C4
- tempo de resposta (se houver alguma resposta): C5
tempo desde o envio da requisição (C1) até o recebimento de uma resposta (C3).
Isto implicitamente inclui os tempos C2, S2, S3, S5, C4 e o tempo de transmissão no ORB
- *throughput*: C6
relação do C1 e C3 pelo tempo de observação T

As instanciações lógicas da instrumentação são chamadas de sensores e estes estão baseados em métricas [FSH⁺95]. Cada sensor corresponde a um ou mais pontos de obtenção de dados, como aqueles mostrado na Figura 4.4, para cálculo de uma métrica em particular. Usando o conceito da orientação a objetos, os sensores são objetos especializados para a geração de dados de gerência, estando localizados no próprio recurso gerenciado. A partir disso, um passo em direção à implementação deste trabalho foi investigar uma

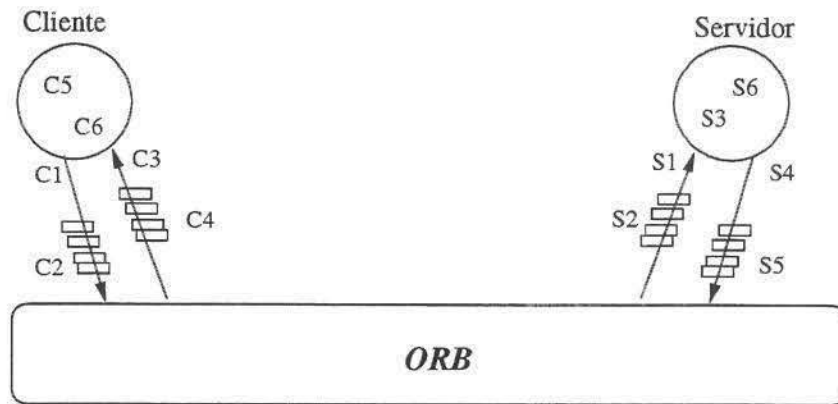


Figura 4.4: Pontos de obtenção das métricas para cliente e servidor

maneira de se instrumentar os objetos CORBA. Para tanto, a utilização de uma estrutura de filtragem capaz de adotar a idéia dos sensores foi um fato muito importante para o desenvolvimento do presente trabalho. O OMG ainda não especificou a estrutura de filtragem a ser usada pelos ORBs, porém emitiu RFPs a respeito deste assunto. Mas principais implementações de ORB possuem a estrutura e portanto são proprietárias. No *VisiBroker* [Bor] por exemplo, ela é chamada de interceptador e no *OrbixWeb* [Ion97] de filtro. A implementação deste trabalho segue o mecanismo de filtro do *OrbixWeb*, descrito no próximo Capítulo.

A estrutura de filtragem tem por finalidade obter informações de gerência refinadas através da inserção de sensores em pontos de interesse, onde ocorram transições de eventos significativos. Para o contexto desta dissertação, um filtro (apenas uma coincidência com o nome da estrutura do *OrbixWeb*) é o conjunto de um ou mais sensores. Assim, tendo-se como base as métricas da perspectiva do cliente e servidor e os tipos de informações definidas anteriormente, foram modelados alguns filtros como mostra a Tabela 4.1.

Estes filtros ajudam na coleta das informações de gerência pelos agentes móveis num *middleware* ORB. Vale lembrar que a métrica básica de tempo de observação está extremamente ligada na coleta destas informações. Os sensores de métricas com natureza parecidas foram agrupados num mesmo filtro. É o que acontece com os filtros F0, F1, F2 e F3. O filtro F0 preocupa-se com o número de requisições e respostas, e através destes valores calcula o *throughput* total das requisições enviadas/recebidas e das respostas enviadas/recebidas no tempo de observação. O filtro F1 coleta tempos que exprimem o desempenho das aplicações CORBA. O filtro F2 mede a quantidade de dados em bytes enviados/recebidos pelas requisições/respostas. Existe um tempo associado ao empacotamento/desempacotamento das requisições/respostas num ORB, e este é considerado no filtro F3. Os filtros F4 e F5 não estão relacionados com as métricas básicas definidas mas, com métricas um pouco mais formuladas importantes para as informações de gerência.

Filtros	Métricas
F0	número de requisições recebidas/enviadas número de respostas enviadas/recebidas <i>throughput</i> total
F1	tempo de residência tempo de resposta
F2	taxa de dados enviada por requisição/resposta taxa de dados recebida por requisição/resposta
F3	tempo de empacotamento (<i>marshalling</i>) tempo de desempacotamento (<i>unmarshalling</i>)
F4	método (operação) mais requisitado(a)
F5	ocupação de memória e CPU por requisição

Tabela 4.1: Filtros modelados para a gerência

O filtro F4 verifica qual foi a operação mais requisitada durante o tempo de observação, enquanto que o filtro F5 verifica a ocupação de memória e CPU no *host* em cada evento ocorrido. Este último filtro tem um processamento custoso, já que faz uso de recursos do sistema operacional e, portanto, deve ser utilizado em casos estritamente necessários.

Da mesma forma que os filtros podem ser aplicados a um processo servidor/cliente como um todo (não importando qual implementação de objeto está sendo requisitada), eles também podem atuar em uma implementação de objeto em particular. Por exemplo, o filtro F4 pode calcular o método mais requisitado de um processo ou de uma implementação de objeto. Para os filtros aplicados a implementações de objetos é possível detectar ainda dois pontos distintos de inserção dos mesmos: antes ou depois da execução de um método, podendo capturar informações mais precisas daquele objeto. Porém, métricas para um *host* também são necessárias. As métricas definidas para o nível de *host* neste trabalho foram: nome dos servidores de objetos ativos, nome do servidor de objetos mais utilizado, identificador do processo (pid) servidor/cliente e nome da implementação de objeto mais requisitada. Estas informações não fazem parte de um filtro, pois são independentes do momento da ocorrência de uma requisição ou resposta. Elas são obtidas pelos agentes em largura, que podem ter uma visão total das atividades das aplicações CORBA no *host*, processando os dados fornecidos pelos filtros. Assim, pode-se perceber que os três níveis de detalhamento de informações de gerência (processo, objeto e *host*), considerados para este trabalho e definidos na Seção 4.3, podem facilmente ser abordados.

4.5 Agentes e Filtros

Para a coleta de informações de gerência num ambiente ORB foram propostos alguns agentes móveis em largura e em profundidade, os quais estão associados aos filtros modelados na Seção anterior, e também agentes especiais necessários para o desenvolvimento deste trabalho. Estes agentes são apresentados na Tabela 4.2 e explicados posteriormente. Pode-se perceber que em um ORB é possível o cascadeamento de filtros instalados nos componentes gerenciados.

Agentes em Largura	Tarefa	Filtros
agL1	requisições e <i>throughput</i>	F0
agL2	tempos de residência e resposta	F1
agL3	requisições e tempos de residência/resposta	F0 + F1
Agentes em Profundidade		
agP1	requisições, <i>throughput</i> e dados	F0 + F2
agP2	tempos	F1 + F3
agP3	objetos	F0 + F2 + F4
agP4	CPU e memória	F1 + F5
agP5	geral	F0 + F1 + F2 + F3
Agentes Especiais		
agE1	ativação/desativação de filtros	nenhum
agE2	ocupação de CPU e memória	nenhum

Tabela 4.2: Classes de Agentes

Agentes em Largura

Estes agentes coletam pouca informação dos processos em cada *host* e possuem um ou dois filtros no máximo associados. O agL1 preocupa-se com as requisições e o *throughput* dos processos e por isso faz uso do filtro F0. Este agente pode fornecer, por exemplo, o servidor de objetos mais requisitado. O agL2 coleta tempo de residência do processamento nos processos como servidores e tempo de resposta dos processos como cliente utilizando o filtro F1. Através do agL2 é possível saber se algum processo servidor/cliente está provocando uma espera muito grande. O agL3 associa a quantidade de requisições aos tempos de residência e resposta, e para isso utiliza os filtros F0 e F1.

Agentes em Profundidade

Estes agentes coletam mais informações dos processos e/ou implementações de objetos em cada *host*, e conseqüentemente, pelo nível de detalhamento, possuem mais filtros associados. O agP1 coleta o número de requisições, o *throughput* e a quantidade de dados (bytes) enviados e recebidos e para isso utiliza os filtros F0 e F2. O agP2 preocupa-se com os tempos e relaciona a espera de uma resposta com o tempo para empacotar e desempacotar os dados, através dos filtros F1 e F3. Para a análise de uma implementação de objeto em particular é utilizado o agP3 que verifica requisições/respostas, quantidade de dados e o método mais requisitado, utilizando os filtros F0, F2 e F4. O agP4 verifica a quantidade de requisições e respostas e a média da ocupação de CPU e memória do *host* em cada um desses eventos ocorridos. Este agente tem associado os filtros F1 e F5, e devido ao último é muito custoso. O agP5 coleta dados de requisições, respostas, tempos, e quantidade de dados possibilitando uma visão completa de um processo ou uma implementação de objeto. Utiliza os filtros F0, F1, F2 e F3.

Agentes Especiais

Estes agentes não possuem filtros associados, pois apenas axiliam os agentes em largura e em profundidade a executarem suas tarefas configuradas pelos gerentes. Para que os agentes móveis em largura e profundidade possam coletar as informações de gerência, os filtros precisam estar ativos (instalados) nos processos servidores/clientes e/ou implementações de objetos. Devido ao custo de processamento associado à instrumentação, devendo este ser o menor possível pois pode até mesmo afetar o desempenho funcional do recurso gerenciado, existe a facilidade de se ativar e desativar a instrumentação de acordo com as necessidades de monitorização. Isto significa que o agente pode ativar e desativar o(s) filtro(s) possibilitando que ele colete as informações desejadas. Esta tarefa é desempenhada pelo agente especial (agE1). Outro agente especial (agE2) coleta informações de ocupação de memória e CPU no *host* utilizando comandos do sistema operacional. Este agente tem a particularidade de ficar no *host*, que está sendo gerenciado, o tempo que filtro(s) relacionado(s) a agente(s) em largura ou profundidade estiver(em) ativado(s). Isto é necessário pois um ambiente gerenciado CORBA não é formado apenas por processos CORBA executando num *host* e, assim, com o auxílio deste agente, presente em quase todas as análises, o gerente pode obter uma visão mais real do ambiente e verificar a influência das aplicações CORBA no sistema. É importante ressaltar que o agE2 não utiliza o filtro F5, já que o agente coleta a ocupação de CPU e memória constantemente, e não somente na ocorrência de uma requisição/resposta.

Pelo que foi descrito acima, pode-se concluir que para cada agente móvel em largura ou em profundidade há um outro agente correspondente de ativação/desativação do(s)

filtro(s). Em suma, há sempre duas etapas a serem seguidas: um agente para ativar o(s) filtro(s), e os agentes que coletam e processam as informações. Entre estas duas etapas existe o tempo de observação, no qual os filtros estão gerando dados. Normalmente, existe também uma outra etapa que é a desativação dos filtros, ocorrendo logo antes da coleta das informações pelos agentes.

Um agente móvel visita os *hosts* do itinerário previamente estabelecido pelo gerente e coleta as informações geradas pelo(s) filtro(s); processa-as e através destes resultados, que envolvem valores mínimos, médios e máximos, pode tomar alguma decisão. *Thresholds* podem ser determinados pelo gerente com a finalidade de reduzir a quantidade de dados transmitida com o agente pela rede e ajudar este a tomar decisões. Estas decisões, que provavelmente estão baseadas em históricos anteriores, incluem mudanças de itinerário, envio de mensagens ao gerente ou a comunicação com outro agente. Toda informação coletada é repassada a um gerente que decide sobre etapas adicionais, isto é, emitir outros agentes (agL ou agP) ou intervir diretamente na realocação de processos. O envio de outros agentes caracteriza o *pool* de agentes, onde o objetivo é fazer *zooms* sucessivos a fim de detalhar as informações nos níveis *host*, processo e objeto, se necessário, até que o gerente possa ter uma possível conclusão sobre o estado do ambiente gerenciado.

Numa gerência centralizada, o gerente tem uma visão global do sistema, o que traz uma certa vantagem, porém sobrecarrega o gerente com muitas informações. Neste trabalho, os agentes móveis e gerentes (estáticos) se distribuem pela rede, interagindo um com o outro para a troca de informações. Desta forma, não existe um gerente com uma visão total do sistema mas, gerentes com uma visão de um segmento da rede permitindo uma gerência mais automatizada. Porém, nada impede que um gerente possa ter seu domínio no mesmo segmento de rede que outro gerente ou ainda que haja intersecção entre domínios dos gerentes, desde que tenham aspectos de informação de gerência diferentes.

4.6 Interface para Gerência

Todas as informações de gerência estão especificadas em uma interface IDL comum para os agentes e os recursos gerenciados. A modelagem foi definida para ter duas interfaces, uma para os filtros aplicados aos processos e outra para os filtros aplicados às implementações de objetos (ou simplesmente objetos). Isto torna a interface mais simples, pois para a gerência proposta em três níveis, a cada vez ou existirá um conjunto de filtros em processos ou um conjunto de filtros em objetos particulares. Mas nada impede que eles convivam juntos. Para os filtros de processos, a interface é chamada de *EncapsFilterProcess*, enquanto que para os de implementações de objetos é *EncapsFilterObj*. Ambas, possuem a mesma estrutura *dataFilter* que encapsula tanto as informações geradas pelos filtros como aquelas obtidas diretamente do ambiente (*server_name*, *activation_mode*,

startup_time e final_time). A interface EncapsFilterProcess é mostrada abaixo.

```
interface EncapsFilterProcess {

    struct dataFilter{
        long number_of_request_received;
        long number_of_request_sent;
        long number_of_response_received;
        long number_of_response_sent;
        double residence_time;
        double response_time;
        double rate_of_data_received;
        double rate_of_data_sent;
        double marshallig_time;
        double unmarshallig_time;
        long memory_occupancy;
        long cpu_occupancy;
        double throughput;
        char operation_name;
        char server_name;
        short activation_mode;
        double startup_time;
        double final_time;
    };
};

void Activate_Deactivate_Proc(in seqbool filtrosproc);
dataFilter AcessData();
```

Algumas informações contidas na estrutura *dataFilter* precisam ser explicadas (as outras são as informações dos filtros, portanto já explicadas): *operation_name* armazena o nome da operação mais requisitada; *server_name* é o nome do servidor o qual ao filtro pertence; *activation_mode* indica qual o modo de ativação do servidor, descrito na Seção 4.3; *startup_time* armazena o tempo inicial registrado pelo sistema e *final_time* o tempo final. Estas duas últimas informações são utilizadas para calcular o tempo total de observação da instrumentação.

A interface também possui dois métodos utilizados pelos agentes móveis: *Activate_Deactivate_Proc* que ativa ou desativa os filtros tendo como parâmetro quais são estes filtros, e o *AcessData* que obtém os dados da estrutura *dataFilter*. No caso dos filtros para objetos, o método é denominado *Activate_Deactivate_Obj* para ativar e desativar

os filtros e possui dois parâmetros: um para indicar quais os filtros aplicados antes da execução do método e outro para aqueles aplicados depois da execução do método. Maiores detalhes são descritos no próximo Capítulo.

Esta interface IDL de gerência é conceitualmente equivalente a uma MIB SNMP/CMIP, gozando de todas as vantagens que CORBA oferece para gerência.

4.7 Persistência do Estado dos Filtros

Existem objetos que por questões particulares precisam ser persistentes. Dois aspectos estão relacionados com o ciclo de vida de um objeto: a ativação e a desativação. Tipicamente um objeto é desativado por razões de gerência de memória. A desativação libera a memória associada ao objeto e seu estado precisa ser armazenado, por exemplo, numa base de dados, para que quando ele for ativado o seu estado possa ser recuperado.

No presente trabalho, a persistência de objetos é abordada de forma parcial, ou seja, somente para as informações de gerência. Assim, os estados dos filtros que estão instalados nos processos servidores/clientes e/ou implementações de objetos precisam ser armazenados, já que são eles que geram as informações de gerência. Se isto não for feito quando os objetos forem desativados, tudo será perdido e não será possível para os agentes móveis monitorarem os componentes de aplicação CORBA. A solução para esta persistência será descrita na Seção 5.4.3 do próximo Capítulo.

4.8 Cenários de Pools de Agentes Móveis para a Gerência

Esta seção ilustra dois cenários do uso do *pool* de agentes móveis para a gerência. A legenda da Figura 4.5 é utilizada também pelas Figuras do Cenário 2.

Cenário 1

Neste cenário, ilustrado pela Figura 4.5, um agente de ativação/desativação de filtros (agE1) é enviado pelo gerente do *host* X, aos *hosts* A e B (1 e 2). Este agente ativa um filtro por processo em cada servidor de objetos executando no *host*, caracterizando uma abordagem no nível de *host*. Após algum tempo, o gerente envia um agente em largura (agL) para coletar as informações nos processos em cada *host* (3 e 4), processá-las localmente e então retorna à origem (5) trazendo as informações relevantes de gerência para o gerente.

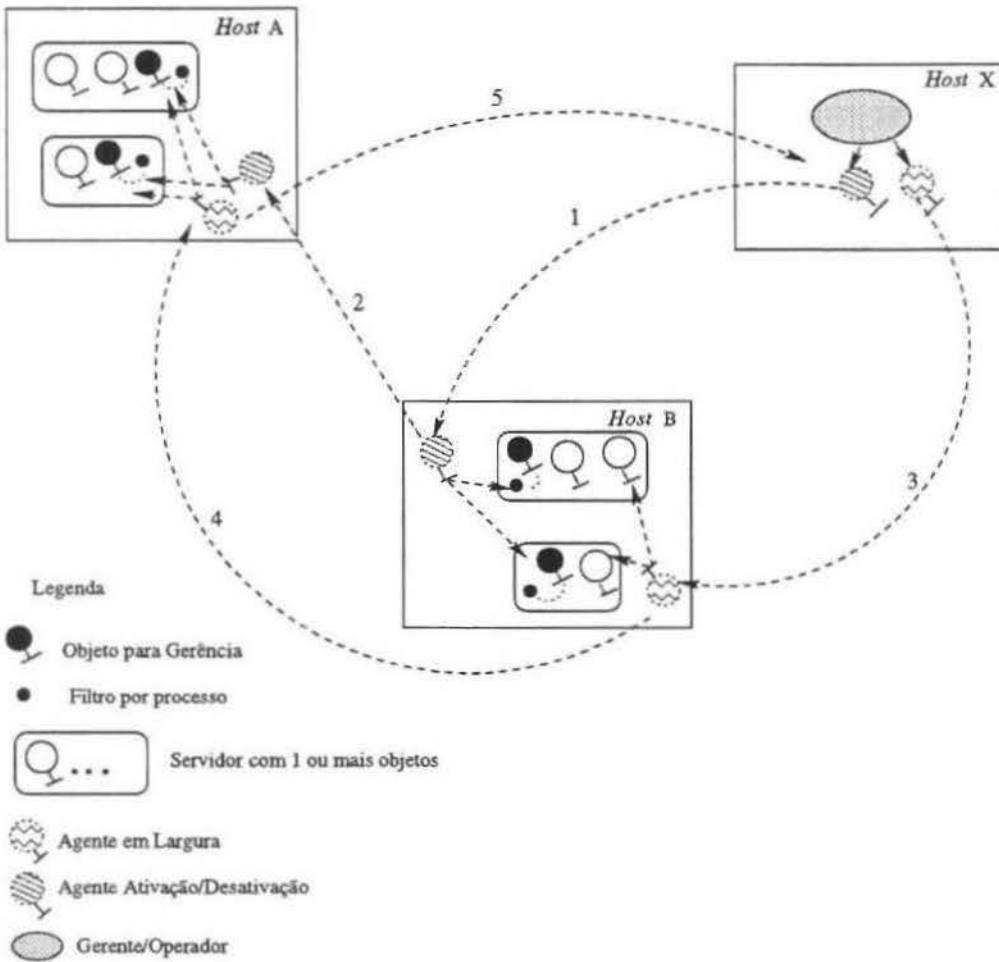


Figura 4.5: Agente de Ativação e Agente em Largura em dois *hosts*

Cenário 2

A Figura 4.6 mostra o envio de um agente de ativação/desativação (agE1) aos *hosts* B, A e X (1, 2 e 3) e depois de um agente em largura (agL) (4, 5 e 6). Até aqui, a filosofia foi a mesma aplicada no cenário anterior. Porém, o gerente analisa as informações trazidas pelo agL e verifica que o *host* B pode estar com problema, mais especificamente o servidor com dois objetos.

Assim, como mostra a Figura 4.7, o gerente envia um agE1 para os *hosts* B, A e X (1, 2 e 3) para ativar mais um filtro no processo suspeito de problema no *host* B e desativar os filtros nos outros processos dos *hosts* A e X. Decorrido um tempo, um agente em profundidade (agP) é enviado ao *host* B (4) numa abordagem de nível por processo, retornando à origem X (5) trazendo informações ao gerente que pode decidir enviar um outro agP numa abordagem por objeto, por exemplo.

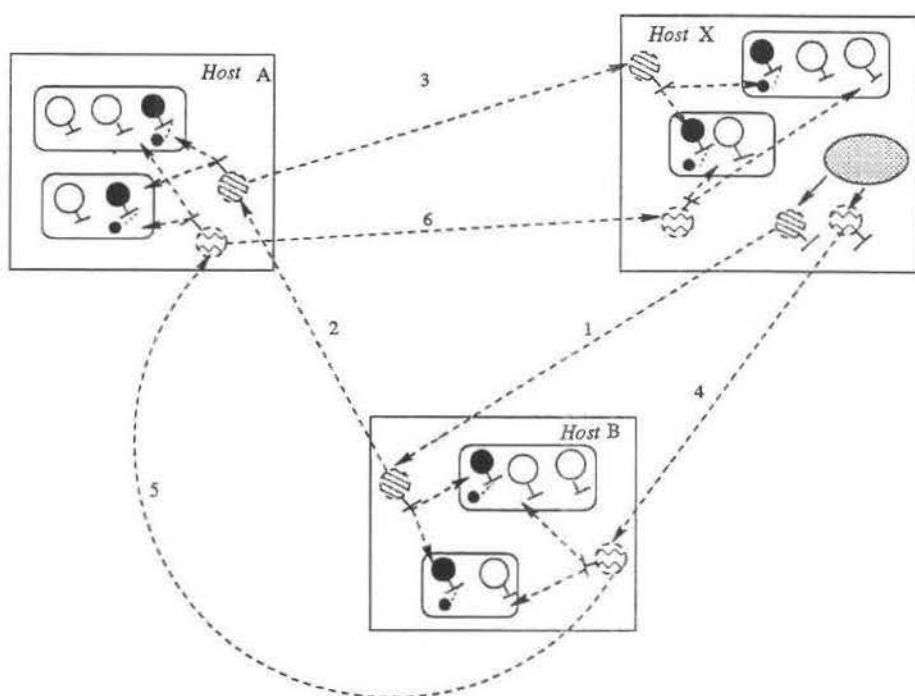


Figura 4.6: Agente de Ativação/Desativação e Agente em Largura em três *hosts*

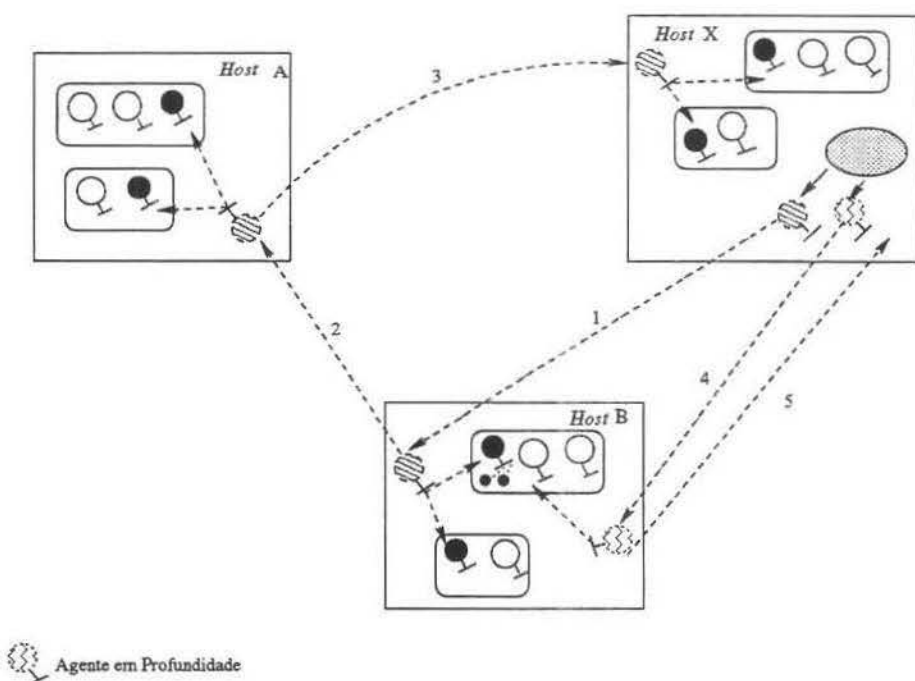


Figura 4.7: Agente de Ativação/Desativação e Agente em Profundidade em um *host*

4.9 Resumo

Os componentes de interesse para a gerência deste trabalho são as aplicações CORBA. Estes envolvem os objetos CORBA registrados num ORB que, conseqüentemente, pertencem a um servidor de objetos. Os servidores de objetos são processos executando num *host* e possuem uma ou mais implementações de objetos.

Um *pool* de agentes móveis em largura e profundidade é utilizado para fazer *zooms* sucessivos em três níveis de detalhamento: *host*, processo e objeto. As informações de gerência coletadas por eles são possíveis devido ao mecanismo de instrumentação. Para esta instrumentação são definidas métricas usadas por uma estrutura de filtragem, onde então são modelados alguns filtros e estes são utilizados pelos agentes móveis para auxiliá-los na execução de suas tarefas.

É definida uma interface em IDL para a gerência comum para os agentes e os objetos CORBA. Através desta interface, os agentes podem ativar e desativar os filtros em um processo servidor e/ou implementação de objetos e coletar os dados gerados por estes. Uma questão que precisa ser considerada é a persistência de objetos. O próximo Capítulo descreve como a modelagem foi implementada.

Capítulo 5

Aspectos de Implementação

A implementação segue o modelo proposto no Capítulo 4. Neste Capítulo são descritos os caminhos seguidos para alcançar o que foi proposto, e então fornecer funcionalidades e flexibilidade para a gerência no ambiente CORBA.

5.1 Ambiente de desenvolvimento

A plataforma *Multiware* está sendo desenvolvida em estações IBM RS/6000 e Sun Sparc, e PCs, com os sistemas operacionais AIX, Solaris e Windows, conectados por redes Ethernet, Fast Ethernet e FDDI. Este ambiente está disperso em dois laboratórios da Universidade Estadual de Campinas, sendo um no Instituto de Computação e outro na Faculdade de Engenharia Elétrica e de Computação, interconectados por um enlace de 10 Mbps. Esta plataforma atualmente utiliza o Orbix 2.1 [Ion95] (com mapeamento IDL para C++) e *OrbixWeb* 2.0.1 [Ion97] (com mapeamento IDL para Java), como implementações de um ORB. A nova versão do ORB com mapeamento Java, o *OrbixWeb* 3, está instalada no Instituto de Computação e se encontra em avaliação para um futuro *upgrade*. Portanto, a gerência utilizando agentes móveis deve ser portátil e interoperável para tais sistemas heterogêneos.

Java é a linguagem adotada para o desenvolvimento deste trabalho por sua portabilidade de código e principalmente, dentre as linguagem comumente usadas para agentes móveis (ver Seção 3.3.4), é a única que já possui o mapeamento para IDL especificado pela CORBA. Algumas ORBs comerciais já trazem compiladores IDL-Java.

Os componentes básicos que fazem parte do desenvolvimento da gerência de sistemas distribuídos proposta aqui são: um **ORB**, um **suporte à mobilidade de agentes** e um **conjunto de agentes móveis de gerência**. O ORB utilizado é o *OrbixWeb* com mapeamento IDL-Java. Para a mobilidade dos agentes, o sistema empregado é o SMM, visto no Capítulo 2. Este suporte foi totalmente escrito em Java e desenvolvido no Instituto

de Computação da UNICAMP, o qual considera os agentes como objetos CORBA. Os agentes móveis de gerência são desenvolvidos sobre o sistema SMM, conseqüentemente implementados em Java, e seguem a modelagem descrita na Tabela 4.2 do Capítulo anterior. Além destes componentes existe uma estrutura de filtragem obtida de um ORB, chamada de filtro no *OrbixWeb*, que auxilia os agentes na monitorização dos objetos CORBA.

É importante ressaltar que apesar dos agentes serem implementados e registrados no *OrbixWeb*, isto não impede que estes sejam lançados por ou se comuniquem com objetos implementados em outras linguagens como os objetos CORBA do *Orbix 2.1* (mapeamento para C++) ou de outro ORB futuramente instalado, visto que é usado o protocolo IIOP [Vas99].

O *OrbixWeb* com sua estrutura de filtragem é descrito na Seção 5.2, enquanto que o SMM é apresentado com maiores detalhes na Seção 5.3. O restante do Capítulo descreve a implementação da infra-estrutura de gerência baseada nos agentes móveis.

5.2 O ORB OrbixWeb

O *OrbixWeb* faz parte da família *Orbix* da *Iona Technologies Ltd.*. Ele possui um compilador IDL que é usado para produzir classes Java correspondentes a cada interface IDL. O compilador verifica a sintaxe e mapeia todas as definições IDL para tipos Java equivalentes. Os métodos correspondentes a cada um dos atributos e operações especificados na interface IDL são implementados na linguagem Java (embora para o cliente a linguagem de programação é transparente). Instâncias das classes estão disponíveis em qualquer *host* onde o *OrbixWeb* esteja configurado. Esses objetos residem em processos servidores do *Orbix* em um *host*. Um servidor de objetos (na *Iona* simplesmente chamado de servidor) pode ter coletivamente diferentes objetos CORBA com interfaces IDL distintas e, ainda, pode ter objetos auxiliares que não tenham interface IDL, sendo invisíveis aos clientes remotos.

O compilador IDL gera classes para as duas abordagens de implementação de um objeto CORBA: *BOAImpl* e *TIE*. Ambas afetam o desenvolvimento dos componentes das aplicações. O *BOAImpl* usa o mecanismo de herança para as classes de implementação, impondo limites de flexibilidade nestas classes e eliminando a possibilidade de reuso de implementações já existentes quando utilizando interfaces derivadas. A abordagem *TIE* não possui estas restrições e por este motivo é mais recomendada. Cada objeto faz uso de apenas uma das duas abordagens porém, nada impede que elas sejam misturadas no mesmo servidor de objetos.

Além do compilador, o *OrbixWeb* consiste de um conjunto de bibliotecas em Java que são incluídas nos componentes de aplicações e de processos permanentemente ativos que auxiliam a conexão remota destes componentes. Tais processos são os *daemons*, chamados

no *OrbixWeb* de *orbixd*. Eles são responsáveis por localizar e ativar, se necessário, os processos servidores. Residem obrigatoriamente nos *hosts* onde se queira que os componentes servidores das aplicações sejam ativados (Figura 5.1).

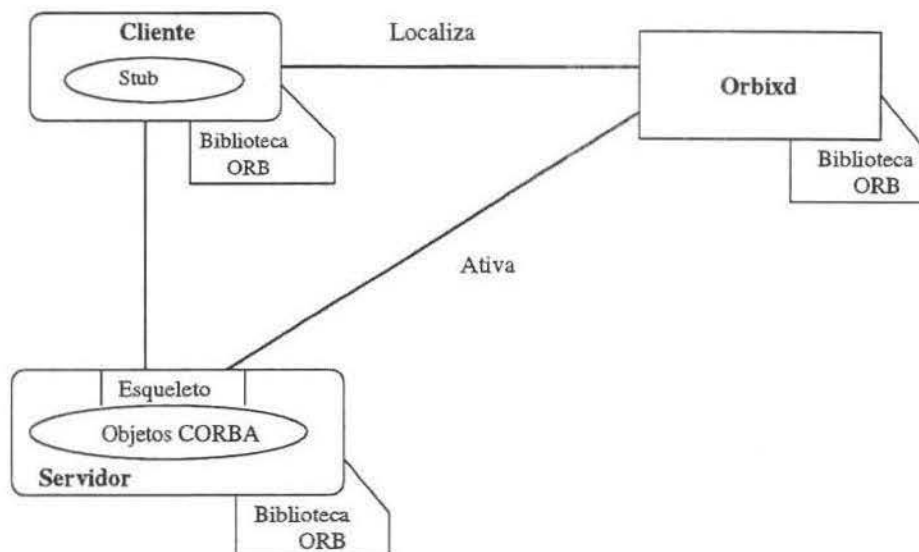


Figura 5.1: O *daemon* do *OrbixWeb*

A classe servidora possui poucas linhas de código, apenas instancia o(s) objeto(s) CORBA e após isso faz uma chamada à operação `impl_is_ready()` indicando a sua disponibilidade na rede, dentro do escopo do método `main()`. O Repositório de Implementação do *OrbixWeb* permite o registro dos servidores de objetos através do utilitário `putit`. O nome do servidor é único num *host*, porém o mesmo nome pode ocorrer em *hosts* diferentes. Quanto à ativação de um servidor, este pode ser registrado usando um dos modos de ativação da CORBA, já explicados na Seção 4.3: compartilhado, não compartilhado e chamada por método. Utilizando o modo compartilhado, considerado como o *default*, um servidor pode ser lançado manualmente caracterizando assim um servidor no modo persistente. Quando ativado pelo *orbixd*, um servidor é desativado caso não receba nenhuma outra invocação durante um período de tempo pré-determinado.

Um cliente se conecta a um objeto CORBA através de sua referência de objeto obtida pelo método proprietário `bind()`, e assim faz invocações de métodos remotos usando a sintaxe normal de chamada da linguagem. Uma referência de objeto identifica um objeto em um espaço de endereçamento diferente do cliente, seja ele local ou remoto. No *OrbixWeb* o seu formato é composto de:

- o identificador único do objeto no servidor, o qual é conhecido como *marker*;
- o nome da implementação de objetos; e

- o nome da máquina onde está sendo executada a respectiva implementação de objetos.

O *OrbixWeb* suporta dois protocolos para comunicação entre componentes de aplicação: o protocolo *Orbix* e o IIOP que é o padrão CORBA para interoperabilidade entre implementações ORBs diferentes. Cada um dos dois protocolos requer um formato de referência de objeto diferente: o protocolo *Orbix* requer o formato *OrbixWeb* descrito acima e o IIOP requer o formato IOR (*Interoperable Object Reference*) da CORBA. Uma IOR inclui uma referência de objeto interna do ORB mais o endereço Internet do *host* e o número da porta.

Como especificado na CORBA 2.0, existem duas outras maneiras de invocar uma operação IDL numa aplicação *OrbixWeb* sem que o cliente fique bloqueado (modo síncrono): modo *oneway*, no qual a operação é declarada em IDL como *oneway* tal que o cliente poderá continuar em paralelo com o servidor, porém não terá valor de retorno; e modo *deferred synchronous* que permite o cliente mais tarde verificar se uma resposta está disponível e então obtê-la. Este último modo é suportado pela Interface de Invocação Dinâmica do *OrbixWeb*.

Maiores detalhes da descrição do funcionamento e implementação do *OrbixWeb* podem ser encontrados em [Ion95].

5.2.1 A estrutura de Filtragem

O *OrbixWeb* e toda a família *Orbix* fornecem uma estrutura, denominada de **filtros**. Eles permitem que um código adicional seja executado antes ou depois do código normal de uma operação de requisição ou resposta. Também podem ser usados como mecanismos de auxílio para autenticação, para criação de *threads* e para monitorização [OHE96]. São capazes de monitorar 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 resposta. Desta forma, os filtros atendem a funcionalidade de uma estrutura de filtragem que auxilia os agentes móveis na monitorização dos objetos distribuídos CORBA, descrita na Seção 4.4, no qual um filtro é composto por um ou mais sensores.

Existem dois tipos de filtros definidos pelo *OrbixWeb*: **por processo** e **por objeto**. Filtros por processo monitoram todas as requisições de atributos e operações que chegam e saem de um espaço de endereçamento. Filtros por objeto monitoram um objeto em particular, ao invés de todos os objetos num espaço de endereçamento como nos filtros por processo. Cada processo pode ter uma cadeia de filtros por processo, assim como

cada objeto pode ter sua cadeia de filtros por objeto. Um filtro de uma cadeia executa suas próprias ações. A Figura 5.2 ilustra os dois filtros: por processo e por objeto.

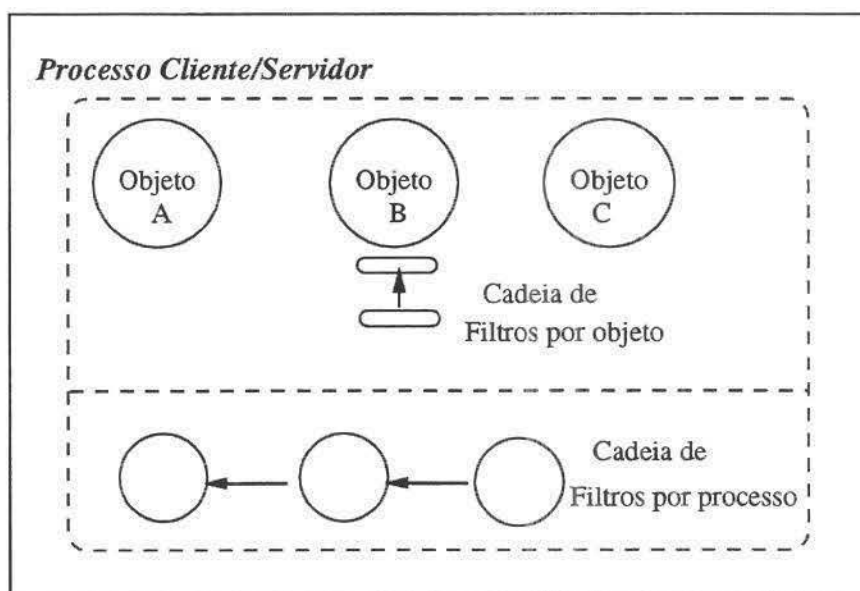


Figura 5.2: Estrutura de filtragem do *OrbixWeb*

Os filtros por processo possuem dez pontos diferentes de monitorização de acordo com requisição/resposta e empacotamento (*marshalling*)/desempacotamento (*unmarshalling*) dos dados, sendo que dois destes pontos são respostas a falhas (Figura 5.3 (a)) :

- *out request pre marshal* (1): ação disponível nas requisições enviadas antes do empacotamento de dados;
- *in request pre marshal* (2): ação disponível nas requisições recebidas antes do desempacotamento de dados;
- *out reply pre marshal* (3): ação disponível nas respostas enviadas antes do empacotamento de dados;
- *in reply pre marshal* (4): ação disponível nas respostas recebidas antes do desempacotamento de dados;
- *out request post marshal* (5): ação disponível nas requisições enviadas depois do empacotamento de dados;
- *in request post marshal* (6): ação disponível nas requisições recebidas depois do desempacotamento de dados;

- *out reply post marshal* (7): ação disponível nas respostas enviadas depois do empacotamento de dados;
- *in reply post marshal* (8): ação disponível nas respostas recebidas depois do desempacotamento de dados;
- *out reply failure* (9): ação disponível para quando uma exceção for gerada. Isto pode acontecer de três formas: por um objeto destino, por qualquer um dos pontos de filtros precedentes (*in request* ou *out reply*) ou pelo valor de retorno de um desses dois pontos indicando que a chamada não deve prosseguir;
- *in reply failure* (10): ação disponível para quando uma exceção for gerada. Isto também pode acontecer de três formas: por um objeto destino, por qualquer um dos pontos de filtros (*out request*, *in request*, *out reply* ou *in reply*) ou pelo valor de retorno de um desses quatro pontos indicando que a chamada não deve prosseguir.

Os filtros por objeto, que monitoram um objeto em particular, possuem apenas dois pontos (Figura 5.3 (b)):

- *per-object pre* (1): ação aplicada antes que a invocação da operação seja passada para o objeto;
- *per-object post* (2): ação aplicada depois que a invocação da operação foi processada pelo objeto.

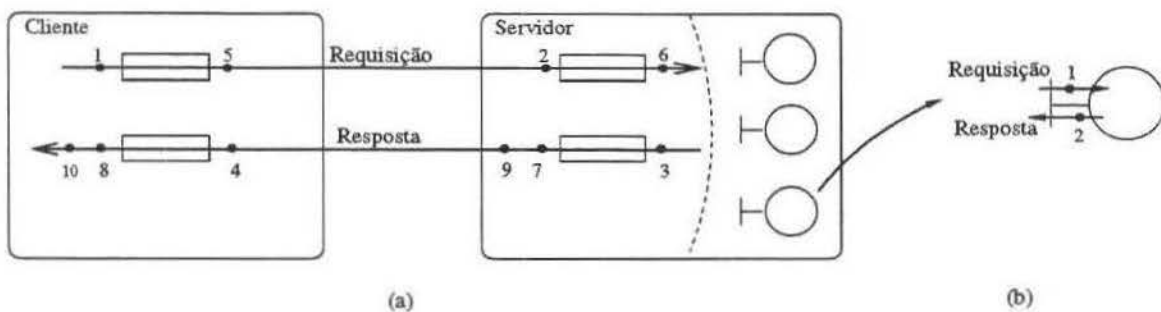


Figura 5.3: Filtros por processo (a) e por objeto (b)

É importante ressaltar que, um filtro por objeto ou é um pré-filtro ou um pós-filtro, ao contrário de um filtro por processo que pode ter ações para quaisquer ou todos os seus dez pontos.

Pode-se perceber que, os pontos de monitorização (Figura 5.3) fornecidos pelos filtros do *OrbixWeb* atendem perfeitamente à filtragem associada às métricas definidas para este trabalho no Capítulo anterior (Figura 4.4).

Um novo filtro por processo pode ser adicionado à cadeia de filtros apenas definindo uma classe que herda da classe `CORBA.Filter` e a instanciando. Os métodos desta classe referentes aos dez pontos de monitorização são redefinidos para executar as operações desejadas. Assim, através destes métodos, os filtros modelados na Tabela 4.1 podem facilmente ser implementados, como será visto na Seção 5.4.1. Num filtro, normalmente, não são usados todos os dez pontos, mas é muito comum um filtro manipular os quatro primeiros pontos descritos anteriormente. O Apêndice B mostra um esqueleto completo de um filtro por processo. Cada um dos métodos tem como parâmetro um objeto da classe `CORBA.Request`, permitindo que os detalhes de uma requisição sejam monitorados como o nome do objeto CORBA alvo da invocação através do `Request.target()`, e o nome do método chamado através do `Request.operation()`. No escopo deste trabalho, os filtros por processo são instalados (instanciados) no espaço de endereçamento de cada servidor.

No caso dos filtros por objeto, a implementação e instalação não são tão simples como nos filtros por processo. Em princípio, a documentação do *OrbixWeb* oferece uma maneira para isto, de certa forma automática, através dos métodos `_attachPre()` e `_attachPost()`. Um exemplo será mostrado para que este procedimento fique claro. Considere a seguinte interface IDL:

```
interface Inc{
    unsigned long increment (in unsigned long vin);
}
```

Após o compilador IDL ter gerado as classes correspondentes, esta interface pode ser implementada assim:

```
public class IncImplementation implements _IncOperations {
    public int increment (int vin) {
        return (vin + 1); }
}
```

Um objeto é criado, utilizando a abordagem *TIE*, dentro do código do servidor de objetos:

```
Inc.Ref  object;
objectr = new _tie_Inc (new IncImplementation());
```

Assim, para se fazer uma pré filtragem, uma classe, por exemplo `FilterPre`, deve ser definida tal que ela tenha os mesmos métodos e parâmetros da classe de implementação definidos na interface IDL do objeto a ser gerenciado, e herde da classe gerada com o mesmo nome do tipo IDL como mostrado abaixo:

```
public class FilterPre extends Inc {  
    public int increment (int vin) {  
        // pre código a ser executado  
        return 0; }  
}
```

Similarmente, o mesmo acontece para a pós filtragem, apenas trocando o nome da classe, por exemplo `FilterPost`, e colocando o código a ser executado após a execução do método. Para que o pré-filtro e o pós-filtro sejam aplicados a um objeto são criadas instâncias destes filtros e em seguida são afixados no objeto apontado por `object` (dentro do código do servidor de objetos o qual pertence este objeto):

```
Inc.Ref ObjPre, ObjPost;  
ObjPre = new _tie_Inc (new FilterPre());  
ObjPost = new _tie_Inc (new FilterPost());  
object._attachPre (ObjPre);  
object._attachPost(ObjPost);
```

Nem sempre é necessário afixar ambos pré-filtro e pós-filtro em um objeto.

Mas toda esta estrutura de instalação de filtros por objeto não é ainda suportada pelo *OrbixWeb 2.x*. Uma maneira de contornar isto será mostrada na Seção 5.4.1. Porém, não há tanta necessidade destes filtros, pois os filtros por processo abordam dois dos três níveis de detalhamento propostos: *host* e processo, e através do objeto `Request` pode-se obter o nome do objeto e o nome do método invocados em um processo.

De um modo geral, o uso da Interface de Invocação Dinâmica não interfere no mecanismo dos filtros. Chamadas feitas usando esta interface resultarão no uso de todos os pontos de filtragem apropriados, com exceção do ponto *in reply post marshal*. Este ponto de filtro deve ser manipulado explicitamente pelo programador da Interface de Invocação Dinâmica [Ion97]. Nos testes realizados neste trabalho, não foi utilizada a Interface de Invocação Dinâmica. Porém clientes podem fazer invocações usando esta interface.

5.3 O Sistema de Mobilidade utilizado

Antes do SMM [Vas99] ter sido liberado para uso e então ter sido escolhido como o sistema para o desenvolvimento dos agentes móveis de gerência neste trabalho, outros caminhos alternativos foram analisados. Inicialmente o sistema de agentes móveis *Aglets Workbench* (AWB), apresentado no Capítulo anterior, foi estudado e alguns agentes móveis experimentais foram implementados. Porém, um problema operacional importante foi constatado: requisições de agentes para programas externos via *OrbitWeb* sempre eram rejeitados por motivo de violação de segurança. Tal problema teria que ser contornado se o sistema AWB fosse executado com o modo de segurança desabilitado. Outro fato foi que o uso da estrutura de filtragem pelos agentes móveis no AWB aumentou a complexidade do código gerando problemas. Assim, houve um incentivo na busca de outro sistema fácil de manipular e que não causasse problemas na integração com um ORB, a fim de alcançar os objetivos deste trabalho. O Odyssey e até o RMI do Java foram cogitados embora não explorados como o AWB.

Desta forma, o SMM surgiu como um suporte simples à mobilidade, com várias funcionalidades e que, principalmente, por tratar o agente como um objeto CORBA possibilitou muitos avanços no desenvolvimento deste trabalho. Teve sua origem acadêmica incentivada pela Facilidade de Agentes Móveis (MAF) do OMG, mas logo depois divergiu do objetivo do OMG quando a facilidade se transformou na Facilidade de Interoperabilidade de Sistemas de Agentes Móveis (MASIF). A Figura 5.4 apresenta o modelo de objetos do SMM.

A chegada de um agente a um *host* é gerenciada pela agência e isto torna este sistema mais simples quando apenas recebe agentes do que quando instancia um agente atendendo a uma aplicação. Para isso, existem duas classes distintas: MAF1 e MAF2. MAF1 permite a criação de agentes e controla seu histórico e MAF2 recebe agentes já criados em outro *host*.

O SMM define inicialmente três participantes do sistema: o suporte propriamente dito, o desenvolvedor de agentes e o usuário do suporte que é a aplicação. Cada agente possui uma interface IDL definida e sua classe é especializada da classe *BaseAgent* devendo ter o método *run* implementado. Este método traz o código a ser executado em cada *host* para cada tipo específico de agente. O agente viaja pelos *hosts* definidos no atributo *itinerary*, sendo que este pode ser alterado pelo método *set_itinerary*. O agente também utiliza o atributo *memory* para armazenar qualquer tipo e quantidade de dados usados ou coletados por ele. A classe *BaseAgent* ainda possui os atributos *Home* que armazena a referência de objeto do MAF1 que controla o agente e *msg* que permite qualquer tipo de dado ser passado como conteúdo de uma mensagem, e os métodos *set_memory* e *get_memory* para atuarem sobre o *memory*, o método *save* para capturar o estado do agente quando este se mover e o *send_msg* para o envio de mensagens ao cliente.

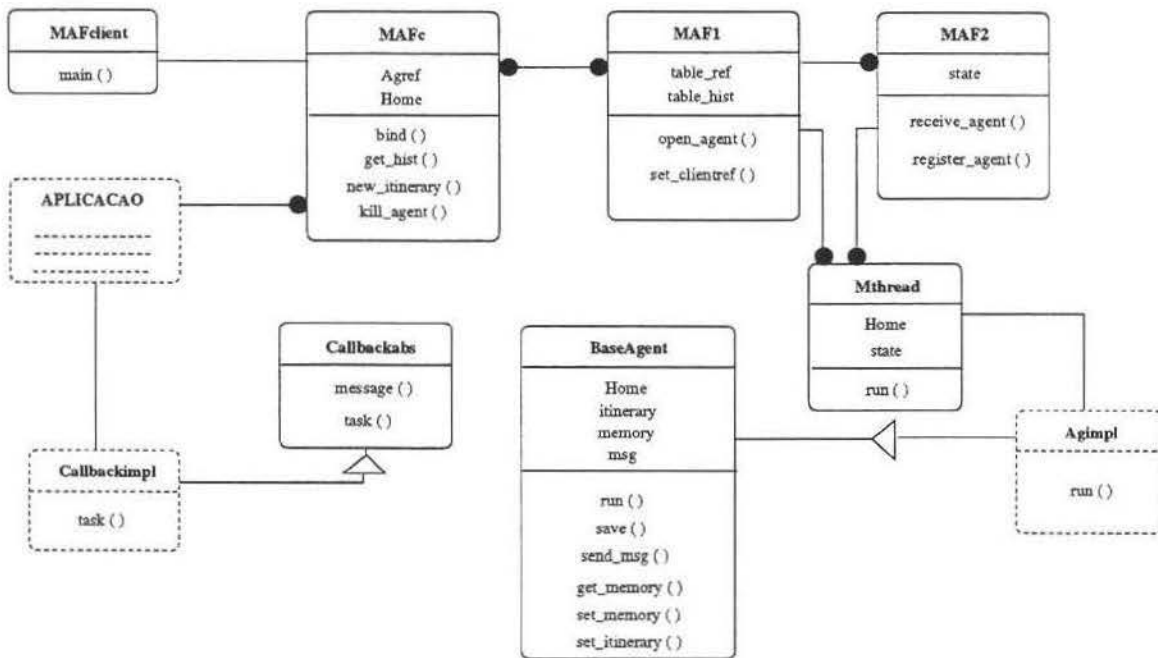


Figura 5.4: Modelagem do Sistema de Agentes Móveis da Plataforma *Multiware*

Um agente, já registrado no ORB, pode ser lançado por uma aplicação ou simplesmente pela linha de comando no caso do agente não retornar informações e não precisar estar ligado a uma aplicação. A classe `MAFclient` permite o lançamento de agentes via linha de comando da seguinte maneira:

```
MAFclient <agente> <host1> <host2> <host3> . . .
```

Para cada agente lançado uma classe, denominada `MAFc`, é instanciada. Esta classe oferece métodos para a obtenção do histórico do agente (`get_hist`), para modificar o itinerário (`new_itinerary`) e para finalizar agentes quando necessário (`kill_agent`).

A classe `MAF1` controla o ciclo de vida de um agente e é instanciada somente no primeiro *host* do itinerário do agente. O agente é criado no primeiro *host* do seu itinerário através do método `open_agent`, chamado pela instância de `MAFc`. Existirá somente uma instância de `MAF1` por *host* e esta poderá criar quantos agentes forem solicitados. Esta classe é de especial importância pois é o elo de ligação permanente dos agentes originados naquele *host* com a aplicação. O histórico de informações dos agentes e o controle das referências de objeto é feito pela classe `MAF1` através, respectivamente, das tabelas: `table_hist` e `table_ref`. A tabela `table_ref` é frequentemente atualizada, pois o agente muda constantemente de referência de objeto. O método `set_clientref` é usado para estabelecer a referência do cliente caso venha a ser necessário o envio de mensagens para

este. Cada objeto *MAFc* possui a referência do agente criado (*Agref*) e a do *MAF1* que o controla (*Home*).

Nos *hosts* subseqüentes do itinerário do agente, este é recebido e instanciado por um objeto da classe *MAF2*. É necessário somente uma instância desta classe por *host* e ela trata o recebimento de vários agentes. Se caso vários agentes criados em um *host* possuem diferentes itinerários, um mesmo *MAF1* se liga a diferentes *MAF2*. A principal diferença entre as classes *MAF1* e *MAF2*, se encontra no fato de que a segunda é bem mais simples já que nada armazena sobre os agentes. Ela apenas possui o atributo *state* que armazena o estado de execução do agente que será passado para *Mthread*, e os métodos para receber o agente e instanciá-lo (*receive_agent*) e para, se necessário, registrá-lo no ORB (*register_agent*).

A execução do código do agente é controlada por um *thread* instanciado por um objeto *MAF1* ou *MAF2* denominado *Mthread*. Quando determinado, internamente pelo próprio agente ou externamente por uma chamada no método *save*, o *Mthread* suspenderá a execução do agente e capturará seu estado de execução que será armazenado no atributo *state*. Esta execução em *threads* independentes, além de permitir a instanciação simultânea de vários agentes em um mesmo *host*, propicia também a igualdade do escalonamento destes agentes. Como cada agente estará associado a um *Mthread*, além do *state*, ele deverá possuir também o atributo *Home* que trata da referência de objeto do *MAF1* que o controla e o método *run*, típico de *threads*.

Para que ocorra a comunicação entre agentes ou entre agentes e outros objetos CORBA, o agente deve levar consigo a referência de objeto relacionada ao outro objeto CORBA e então a invocação via ORB ocorrerá normalmente. A única mudança para dois agentes de comunicarem é que cada agente, além da referência inicial, deverá armazenar também a referência do *MAF1* do outro agente. O agente pode enviar mensagens diretamente para a aplicação, desde que esta implemente uma classe especializada de *Callbackabs* definida pelo suporte. Esta classe trata o recebimento das mensagens de agentes pelo método *message* e oferece o método abstrato *task* para ser implementado na especialização da classe.

Independentemente da aplicação, o agente irá morrer quando terminar seu itinerário. O *MAF1* que criou o agente ficará ativo enquanto pelo menos um agente criado por esta classe estiver vivo. Se houver necessidade do agente ser morto antes do término do seu itinerário, o método *kill_agent* pode ser usado.

Quanto à segurança, o SMM não tratou deste aspecto na sua primeira versão. Seu enfoque primordial foi a migração de objetos CORBA, porém ele não inviabiliza o uso de mecanismos de segurança existentes atualmente como o da especificação CORBA e do MASIF.

Outro sistema de agentes móveis poderia ter sido utilizado neste trabalho, aplicando

toda a modelagem descrita no Capítulo anterior. Porém, além do SMM tratar os agentes como objetos CORBA, ele é um sistema que foi desenvolvido no ambiente da plataforma *Multiware* do Instituto de Computação da UNICAMP, e por isto possibilita qualquer alteração no código do mesmo, se necessário.

5.4 Infra-estrutura de Gerência

A infra-estrutura de gerência deste trabalho é composta basicamente de agentes móveis que monitoram a atividade dos objetos distribuídos CORBA. A monitorização é orientada a eventos e se concentra nas áreas de contabilização e desempenho. Os eventos estão baseados na transmissão e recepção de operações sobre os objetos. A abordagem é detectar problemas potenciais em *hosts*, processos ou ainda implementações de objetos através do *pool* de agentes móveis de tipos (em largura ou profundidade) e funcionalidades diferentes até um detalhamento sucessivo do problema. Para isso os agentes contam com a ajuda dos filtros definidos na Tabela 4.1 implementados pela estrutura de filtragem do *OrbixWeb*.

5.4.1 Implementação e Ativação/Desativação dos Filtros

A implementação dos filtros segue o que foi descrito na Seção 5.2.1. Porém, a implementação dos filtros por objeto tornou-se muito complicada, já que o *OrbixWeb* não os suporta. Para tentar contornar este problema, as instanciações dos pré-filtros e pós-filtros foram colocadas dentro da classe de implementação do objeto, permitindo que os métodos dos filtros pudessem ser executados antes ou depois do código que implementa os métodos correspondentes no objeto. Provavelmente seria isso que os métodos `_attachPre()` e `_attachPost()` fariam automaticamente, evitando a alteração do código da classe de implementação do objeto. Mas, isto não se mostrou muito viável e eficiente em se tratando de gerência, pois mais código é acrescentado ao recurso gerenciado. Sendo assim, este trabalho se concentra apenas nos filtros por processo. Apesar disso, como já foi dito anteriormente, os filtros por processo fornecem uma coleta de informações em dois dos três níveis abordados: *host* e processo, e isto já é adequado o suficiente para a gerência. Mesmo porque, através deles é possível saber qual objeto do servidor está sendo requisitado (método `target()`) e assim, obter o objeto mais requisitado de um servidor.

A Tabela 5.1 mostra quais os pontos de filtragem fornecidos pelo *OrbixWeb* são necessários para implementar os filtros modelados.

De acordo com a Tabela acima, para alguns filtros, o fato de usar pontos de filtragem antes ou depois do empacotamento/desempacotamento dos dados (*pre* e *post*) pode influenciar no cálculo das métricas. Para os filtros F0, F4 e F5 isto é irrelevante, pois o importante é a ocorrência de uma requisição ou resposta. Para os filtros F1, F2 e F3 isto

Filtros	Métricas	Pontos de filtragem
F0	número de requisições recebidas número de requisições enviadas número de respostas enviadas número de respostas recebidas <i>throughput</i> total	<i>in request</i> <i>out request</i> <i>out reply</i> <i>in reply</i> baseados nos pontos anteriores
F1	tempo de residência tempo de resposta	<i>in request post marshal</i> + <i>out reply pre marshal</i> <i>out request pre marshal</i> + <i>in reply post marshal</i>
F2	taxa de dados enviada por requisição taxa de dados enviada por resposta taxa de dados recebida por requisição taxa de dados recebida por resposta	<i>out request pre marshal</i> <i>out reply pre marshal</i> <i>in request post marshal</i> <i>in reply post marshal</i>
F3	tempo de empacotamento tempo de desempacotamento	<i>out request</i> ou <i>out reply</i> (<i>pre</i> + <i>post marshal</i>) <i>in request</i> ou <i>in reply</i> (<i>pre</i> + <i>post marshal</i>)
F4	método (operação) mais requisitado(a)	<i>in request</i>
F5	ocupação de memória e CPU por requisição	<i>in request</i>

Tabela 5.1: Filtros e seus pontos de filtragem

não é suficiente. O filtro F1 mede o tempo de residência, que não deve contar o tempo de empacotamento e desempacotamento dos dados (tarefa do F3), ao contrário do tempo de resposta que deve incluí-lo (ver Figura 4.4 do Capítulo anterior). Para medir a taxa de dados em bytes, o filtro F2 utiliza pontos antes do empacotamento (requisição/resposta enviada) e depois do desempacotamento (requisição/resposta recebida). Por fim, o filtro F3 calcula o tempo de empacotamento do envio de uma requisição (pontos *out request pre marshal* e *out request post marshal*) e/ou de uma resposta (pontos *out reply pre marshal* e *out reply post marshal*), e o tempo de desempacotamento da chegada de uma requisição (pontos *in request pre marshal* e *in request post marshal*) e/ou de uma resposta (pontos *in reply pre marshal* e *in reply post marshal*).

Os filtros implementados foram o F0, F1, F4 e F5 para testes com os agentes móveis. Para que os filtros possam ser ativados ou desativados é necessária uma interface IDL comum entre o agente e os objetos CORBA. Apenas a interface de gerência (*EncapsFilterProcess*), apresentada no Capítulo anterior (Seção 4.6) é utilizada para a implementação (a

interface `EncapsFilterObj` não é utilizada, pois os filtros por objeto não estão sendo considerados neste trabalho). Ela possui um método chamado `Activate_Deactivate_Proc` cujo parâmetro é uma sequência de *booleanos* (*sequence of boolean*). Quando mapeada para Java, a *sequence* é uma classe com duas variáveis e um método, e sua manipulação é simples como a de um *array*. A vantagem do uso do tipo *sequence* sobre o tipo *array* no *OrbisWeb* é que não precisa ser definido um tamanho em IDL. Isto possibilita que novos filtros possam ser modelados e adicionados à presente infra-estrutura de gerência, tornando-a flexível. O tamanho do *array*, em tempo de execução, é igual ao número de filtros, onde cada índice deste *array* corresponde a um filtro: índice 0 ao filtro F0, índice 1 ao filtro F1 e assim por diante. Este *array* é preenchido pelo agente especial (agE1) com valores booleanos a fim de mudar o estado de todos ou de alguns filtros. A ativação (valor *True*) significa instanciar a classe que implementa o filtro, caso ele ainda não esteja instanciado, pois não é permitida mais de uma instância do mesmo filtro num processo. Também implica na inicialização de todos os atributos que não dependem de uma requisição/resposta e, principalmente, o tempo inicial da observação (*startup_time*). A desativação (valor *False*) do filtro (logicamente se ele estiver ativo) implica na chamada do método `_delete()` da classe `Filter` que o retira da cadeia de filtros e no preenchimento que determina o tempo final da observação (*final_time*).

5.4.2 Considerações sobre a Gerência

Em qualquer sistema onde os recursos computacionais são gerenciados existe sempre um custo adicional, devido à gerência. No caso deste trabalho, o custo adicional é o acréscimo de uma interface de gerência IDL a cada componente de aplicação CORBA. Esta interface de gerência, descrita na Seção 4.6, é uma classe em Java que implementa os seus dois métodos (`Activate_Deactivate_Proc` e `AcessData`) necessários para a gerência e invocados pelos agentes móveis. Esta classe, denominada `EncapsFilterProcessImpl`, é instanciada dentro do código do servidor, compondo assim um objeto adicional de gerência entre os outros objetos do processo servidor. Cada processo servidor possui uma única estrutura (chamada `dataFilter`) instanciada correspondente ao(s) filtro(s) que armazenará(ão) os dados gerados por este(s).

O modo de ativação do servidor considerado para os testes feitos nesta dissertação é o compartilhado, por ser o mais utilizado e simplificar a gerência dos objetos CORBA. Porém, se o servidor for ativado como não compartilhado ou persistente, a modelagem descrita no Capítulo anterior é totalmente válida. O modo de ativação chamada por método é infreqüentemente utilizada e para a implementação da monitorização é necessária a existência de comunicação entre processos. A arquitetura de *multithreading* não é abordada, pois é um mecanismo bastante complexo, especialmente para programação

de servidores, e muitos ORBs ainda não possuem um bom suporte para *threads* [Sch98]. Assim, cada processo é considerado ter um *thread* de execução, isto é, uma única requisição é atendida a cada vez. Além disso, todas as requisições consideradas aos objetos possuem resposta, caracterizando o modo de comunicação síncrono. No modo assíncrono não haveria problema mas, algumas métricas não fariam mais sentido como tempo de resposta, número de respostas enviadas e tempo de empacotamento/desempacotamento da resposta. Isto porque as métricas definidas refletem uma instrumentação geral e ampla.

Os agentes móveis de gerência desenvolvidos aqui são lançados por uma ou mais aplicação do sistema SMM e não simplesmente pela linha de comando, já que cada aplicação funciona como um gerente, configurado por um operador/gerente humano. Isto permite que a aplicação receba informações dos agentes para continuar seu processamento. Nada impede que um agente móvel de gerência seja enviado pela linha de comando e a aplicação gerente não espera por nenhum resultado como, por exemplo, um agente de ativação/desativação.

É importante ressaltar que, para um tempo determinado de monitorização dos objetos CORBA toda a atividade dos mesmos é observada, não importando se o servidor ficou um intervalo de tempo desativado ou não. Isto é controlado pelo objeto de gerência `EncapsFilterProcessImpl` que implementa a persistência do estado dos filtros, descrita na próxima Seção.

5.4.3 Persistência do Estado dos Filtros

Existe um cuidado que deve ser tomado quanto à persistência das informações de gerência, visto na Seção 4.7, mais especificamente ao estado dos filtros. Este estado envolve quais filtros estão ativos ou não e os dados armazenados em sua estrutura (`dataFilter`). Como não existe um serviço de persistência especificado pelo OMG e nenhum tipo de persistência implementada no *OrbisWeb* 2.0, uma maneira teve de ser encontrada neste trabalho para tornar viável o seu desenvolvimento. A maneira encontrada e implementada foi a seguinte: antes do processo servidor ser desativado pelo *daemon*, todo o estado do(s) filtro(s) é armazenado em um arquivo temporário através da execução do método `finalize` do Java, implementado pela classe `EncapsFilterProcessImpl`, e restaurado quando o processo for instanciado novamente. Esta recomposição do estado é feita pelo construtor da mesma classe e pelo construtor de cada objeto filtro que estava ativo anteriormente.

5.4.4 Agentes Móveis para Gerência

A implementação dos agentes móveis segue o modelo descrito no sistema SMM e se baseia nos agentes móveis sugeridos na Tabela 4.2.

Antes de qualquer agente móvel de gerência, em largura ou profundidade, visitar os *hosts* para coletar informações, um outro agente móvel de ativação/desativação (age6) deve ter visitado estes *hosts* a fim de ativar os filtros necessários. Se, por exemplo, um agente em largura quiser ter uma visão geral do *host*, então o(s) filtro(s) serão ativados em todos os processos servidores ativos naquele *host*. Isto é possível através da interface `IT_daemon::listActiveServers` do *daemon* do *OrbixWeb*, que retorna todos os servidores ativos naquele momento no *host*. Se caso, o agente quiser analisar somente alguns processos ou um único processo de um *host* então o agente age6 terá de conhecer o nome do processo servidor registrado no ORB para poder ativar o(s) filtro(s).

Um exemplo de código do agente age6 é mostrado logo abaixo. Neste caso, o agente ativa o filtro F0 em todos os processos servidores ativos nos *hosts* pertencentes ao seu itinerário (*itinerary*) configurado na aplicação que o lançou. Pode-se perceber que o código deste agente móvel é bastante simples e leve.

```
class Agent1 extends BaseAgent {
  Agent1( ) {
    super ( );
  }
  public run ( ) {
    IT_daemonRef daemon;
    serverDetailsSeq servers;
    _EncapsFilterProcessRef ref1;
    seqbool fprocess = new seqbool;
    fprocess.ensureCapacity(1);
    fprocess.buffer[0] = true;
    fprocess.buffer[1] = false;
    fprocess.buffer[2] = false;
    fprocess.buffer[3] = false;
    fprocess.buffer[4] = false;
    fprocess.buffer[5] = false;
    String host = itinerary.buffer[0];
    daemon = IT_daemon._bind(" ", host);
    daemon.listActiveServes(servers);
    for(int i = 0; i<servers.length;i++)
    {
      ref1 = EncapsFilterProcess._bind(servers.server, host);
      ref1.Activate_Deactivate_Proc(fprocess);
    }
  }
}
```


A ativação dos filtros em todos os processos ativos num *host* impõe uma exceção: os processos que envolvem o sistema de mobilidade, neste trabalho o SMM, que são o MAF1 e MAF2, assim como o próprio agente móvel e sua aplicação (se esta estiver no *host* que faz parte do itinerário) não são monitorados.

Após feita a ativação do(s) filtro(s) nos respectivos servidores, os filtros geram dados e os armazenam na estrutura *dataFilter* de acordo com a ocorrência dos eventos nos objetos. Decorrido um certo tempo, determinado pelo operador/gerente, um agente móvel é enviado para a coleta das informações. Para isso, o agente tem acesso à estrutura *dataFilter* através do método *AcessData()*. Este agente processará os dados localmente e carregará consigo, na variável *memory*, aqueles dados importantes a serem levados para o operador/gerente.

A seguir são mostrados dois exemplos que ilustram o *pool* de agentes móveis para a gerência das aplicações CORBA. O domínio dos testes envolveu *hosts* SunOS 5.5 do Instituto de Computação da UNICAMP.

Exemplo 1

Este exemplo é simples e relata um dos primeiros resultados obtidos neste trabalho [RM98]. Ele está relacionado com o cenário 1 (Figura 4.5) do Capítulo anterior. A idéia foi coletar informações gerais de gerência numa abordagem por *host* em dois *hosts* (pinheiros e jaguari). Para isso, um agente *agE1* com o código idêntico ao descrito anteriormente foi enviado a estes *hosts* e o filtro *F0* foi ativado nos dois processos ativos: *gridSrv* e *smartProxySrv*. Decorrido um tempo de onze minutos, o agente em largura *agL1* foi enviado àqueles *hosts* onde coletou e processou os dados levando-os para a aplicação (gerente). A Tabela 5.4 apresenta os dados coletados (o *throughput* é medido em número de requisições pelo tempo).

Nome do servidor	Nome do Host	Número de requisições recebidas	Número de requisições enviadas	Tempo de observação (ms)	Throughput
<i>gridSrv</i>	pinheiros	38	38	700000	0.000054
<i>smartProxySrv</i>	jaguari	22	22	400000	0.000036
<i>gridSrv</i>	jaguari	40	40	500000	0.00008
<i>smartProxySrv</i>	pinheiros	22	22	550000	0.00004

Tabela 5.2: Amostragem de dados coletados por um agente em largura

Neste exemplo, o tempo de observação se refere ao tempo que os processos servidores

ficaram ativos continuamente no *host*. O objetivo foi mostrar uma coleta típica de dados (informações) de gerência num ambiente CORBA.

Exemplo 2

A idéia deste exemplo é coletar informações dos *hosts* como um todo, ou seja, além de analisar as aplicações CORBA é verificado a ocupação de CPU e memória em cada *host* [SMRao]. Este exemplo está baseado no cenário 2 (Figura 4.6 e 4.7) do Capítulo anterior). Para simular um ambiente para gerência, foi utilizado um objeto CORBA com a função de consumir CPU e memória (servidor denominado *loadSrv*), com intuito de gerar *host(s)* crítico(s). O itinerário do agente móvel de gerência é composto por três *hosts*: gaivota, tuiuiu e bemtevi, e espera-se que os agentes móveis detectem os problemas simulados.

Para coleta de dados de CPU e memória é enviado um agente para cada *host* que permanecerá lá durante o todo o tempo de observação. Também será ativado o filtro F0 em todos os processos servidores CORBA ativos em cada *host*. De acordo com a modelagem descrita neste trabalho, seria enviado um agente agE2 de CPU e memória para cada *host* e um agente agE1 para ativação do filtro com o itinerário: gaivota, tuiuiu e bemtevi. Com o objetivo de melhorar a eficiência, estes dois agentes podem ser unidos em um só, ou seja, um agente móvel, digamos agE3, é enviado para cada *host*, chegando lá ativa o filtro nos processos servidores e após isso fica "hospedado" no *host* pelo tempo de observação coletando dados de CPU e memória. Para coleta de dados de CPU o agente utiliza o comando *sar* do Unix (com os parâmetros 1 e 10), e para memória o utilitário *top* com o parâmetro 0.

Foi esquematizada uma relação entre ocupação de CPU, ocupação de memória e *throughput* mostrada na Tabela 5.3. Estes itens estão baseados em *thresholds* determinados previamente pelo gerente (humano), por exemplo, se os três itens de relação da Tabela estiverem com valores acima do *threshold* então o *host* analisado pode estar com algum problema e é interessante enviar um agente em profundidade para coletar dados mais detalhados. Outras tabelas, relações ou regras poderiam ser definidas pelo gerente (humano) partindo do seu conhecimento da rede.

Neste exemplo, após decorrido quatro minutos a aplicação lança um agente em largura agL1 com o itinerário contendo os três *hosts*. Este agente implementa a Tabela de decisão apresentada anteriormente para que, baseado nos dados coletados dos processos servidores e os de CPU e memória obtidos através da comunicação com o agente agE3, ele possa processar e armazenar em seu *memory* os valores mínimos e máximos encontrados durante seu ciclo de vida. Os processos servidores encontrados em cada *host* foram o *gridSrv* e o *loadSrv*. A Tabela 5.4 apresenta o relatório retornado pelo agL1 e o Apêndice C descreve o código do agente em largura que implementa a coleta de dados gerados pelo filtro F0 e

Ocupação de CPU	Ocupação de Memória	Throughput	Enviar Agente agP (?)
acima	acima	acima	sim
acima	acima	abaixo	sim
acima	abaixo	acima	não
acima	abaixo	abaixo	sim
abaixo	acima	acima	não
abaixo	acima	abaixo	sim
abaixo	abaixo	acima	não
abaixo	abaixo	abaixo	não

Tabela 5.3: Tabela para níveis de decisão para ocupação de CPU, memória e throughput

o código de implementação do filtro F0.

Agente: agL1	Mínimo	Máximo	tuiuiu	gaivota	bemtevi
ocupação de CPU (%)	40	50	40	42	50
ocupação de memória (%)	80	94	80	92	94
throughput	0.0132	0.0198	0.0137	0.0198	0.0132
requisições	2160	2160	1000+ 1000+160	1000+ 1000+160	1000+ 1000+160
enviar outro agente			não	não	sim

Tabela 5.4: Tabela do relatório retornado pelo agente agL1

Após ter completado uma vez seu itinerário, este agente agL1 gera um conjunto de possíveis *hosts* críticos que provavelmente formará o itinerário de um outro agente em largura ou mesmo um agente em profundidade (deve-se notar que, este agente ainda poderia executar seu itinerário mais de uma vez, em tempos espaçados, para fazer ajustes dos *thresholds* melhorando mais ainda as conclusões de *hosts* críticos). Este novo itinerário, como pode ser visto pela Tabela anterior, é formado pelo *host* bemtevi. Porém, a aplicação (gerente) pode decidir enviar um outro agente em largura agL1 para o *host* tuiuiu. A Tabela 5.5 mostra os resultados trazidos por este segundo agente. O relatório deste agente não sendo conclusivo resulta no envio, por exemplo, de um agP2 e/ou agP3, e assim por diante.

Agente: agP1	Host 1	loadSrv	gridSrv
ocupação de CPU (%)	54		
ocupação de memória (%)	83		
throughput		0.0003833	0.0132
requisições		2000	80
enviar outro agente	sim/não		

Tabela 5.5: Tabela do relatório retornado pelo agente agP1

5.5 Resumo

No contexto da plataforma *Multiware*, o ORB utilizado foi o *OrbixWeb* com mapeamento Java-IDL devido ao fato dos agentes móveis usarem Java para a sua mobilidade. O sistema de agentes móveis SMM oferece muitas funcionalidades para a gerência e pelos agentes móveis serem objetos CORBA possibilitou muito progresso neste trabalho. Porém, nada impede que outros sistemas de agentes móveis sejam utilizados baseando-se na modelagem proposta.

O *OrbixWeb* oferece uma estrutura de filtragem, chamada filtros. Existem dois tipos de filtros: por processo e por objeto. Eles são capazes de interceptar as requisições e respostas nos processos cliente/servidor ou em implementações de objetos, respectivamente. Os filtros são objetos estáticos que são dinamicamente colocados ou retirados dos servidores ou implementações de objetos pelos agentes móveis à medida que níveis de informações de gerência diferentes são requeridos. Apesar dos filtros por objeto não serem suportados pelo *OrbixWeb*, os filtros por processo conseguem fornecer informações suficientes para uma gerência.

Capítulo 6

Conclusão

CORBA é uma plataforma rica em funcionalidades para suportar sistemas distribuídos. Uma das áreas na qual ela está avançando é a gerência de redes e sistemas. Isto vem ocasionando pesquisas e grandes avanços por parte dos pesquisadores que apontam CORBA como uma alternativa de integração entre diferentes modelos de gerência em uso hoje. Assim, modelos tradicionais de gerência como SNMP começam a ser questionados quanto a sua aplicabilidade e flexibilidade.

A utilização de agentes na gerência não é nada novo, porém a mobilidade dada a eles é algo muito recente. Dentre as várias classes de agentes, a de agentes móveis é uma das mais estudadas atualmente. Em sistemas distribuídos a possibilidade de migrar a computação para outro ponto da rede pode propiciar o desenvolvimento de soluções mais eficientes como, por exemplo, na gerência, no comércio eletrônico e outros.

Este trabalho se beneficia das duas tecnologias inovadoras, agentes móveis e CORBA, como uma nova alternativa para a gerência de monitorização de sistemas distribuídos.

6.1 Contribuições da Modelagem

A gerência de uma rede ou de um sistema distribuído implica na coleta de informações para este propósito. É interessante que estas informações sejam filtradas e processadas pelos agentes antes de serem repassadas à aplicação gerente. O uso de agentes móveis na gerência reduz consideravelmente o número de mensagens que trafegam pela rede, já que o código é movido em direção aos dados e o processamento torna-se local. Os agentes móveis desenvolvidos neste trabalho possuem uma certa autonomia contribuindo para uma gerência mais descentralizada.

A preocupação de gerência neste trabalho são as áreas de contabilização e desempenho envolvendo os componentes de aplicação do ambiente CORBA: clientes e servidores. Estes componentes quando ativos são processos executando num *host*. Para obter informações

de gerência destes componentes é utilizado um mecanismo das *CORBA facilities* chamado de instrumentação. A instrumentação conta com a definição de várias métricas tanto para cliente quanto para servidor.

Este trabalho contribui para a classificação dos agentes móveis de gerência, de acordo com a granularidade de informação, em dois tipos: largura e profundidade. Eles compõem um *pool* de agentes que possibilita *zooms* (como se fosse uma máquina fotográfica aproximando o foco para mais perto do objeto alvo) em três níveis diferentes: *host*, processo e objeto.

A utilização de uma estrutura de filtragem que instrumenta as aplicações CORBA possibilita a coleta de informações pelos agentes móveis. Os filtros são compostos por métricas e os agentes móveis estão associados a estes filtros. Um conjunto de filtros e agentes são sugeridos para a gerência de monitorização.

A gerência de monitorização proposta nesta dissertação permite que a instrumentação seja ativada ou desativada nos componentes de aplicação pelos agentes móveis conforme as necessidades de gerência do operador/gerente.

A utilização do código móvel ao invés do estático permite que os agentes saltem de nó em nó e através de um histórico possam avaliar as informações coletadas através de valores mínimos, máximos e médios. Isto permite uma conclusão mais precisa a respeito de um problema em algum *host*.

Neste trabalho, os agentes móveis e gerentes (estáticos) se distribuem pela rede, interagindo um com o outro para a troca de informações. Desta forma, não existe um gerente com uma visão total do sistema mas, gerentes com uma visão de um segmento da rede contribuindo para uma gerência mais automatizada. Porém, nada impede que um gerente possa ter seu domínio no mesmo segmento de rede que outro gerente ou ainda que haja intersecção entre domínios dos gerentes, desde que tenham aspectos de informação de gerência diferentes.

A flexibilidade deste trabalho é fornecida no sentido de que vários outros filtros e agentes móveis podem ser definidos para a gerência a qualquer momento, por exemplo, um filtro que obtenha quais clientes estão requisitando o servidor. Isto permite que se defina um agente móvel (em largura ou profundidade) que colete todos os clientes de um *host* indicando possivelmente ao gerente qual está consumindo mais recurso.

6.2 Contribuições da Implementação

O sistema de agentes móveis SMM utilizado traz a contribuição de tratar os agentes móveis como objetos CORBA, e devido ao fato do ambiente gerenciado ser CORBA eles podem se comunicar com e receber requisições de outros objetos CORBA. É totalmente desenvolvido em Java, sendo que é a única linguagem de agentes móveis que possui o

mapeamento para IDL especificado.

O uso dos filtros do *OrbixWeb* propiciou a implementação da estrutura de filtragem modelada, também chamada de filtros. Os filtros do *OrbixWeb* são classificados em dois tipos: filtros por processo e filtros por objeto, e são objetos estáticos que são dinamicamente colocados ou retirados dos processos servidores ou de objetos particulares pelos agentes móveis. A instalação dos filtros por processo não requer acrescentar muito código aos objetos distribuídos, enquanto que para os filtros por objeto mais código é acrescentado, já que a *OrbixWeb* ainda não os suporta. Apesar disso, os filtros por processo fornecem uma coleta de informações em dois dos três níveis abordados: *host* e processo e isto já é adequado o suficiente para a gerência.

Com a instrumentação dos componentes de aplicação CORBA através da estrutura de filtragem, os agentes móveis são relativamente leves fornecendo uma eficiência maior para gerência. Certamente é acrescentado um custo aos objetos gerenciados mas, isto é realidade para qualquer sistema de gerência.

Os agentes móveis podem se comunicar um com o outro para a realização de suas tarefas, desde que eles pertençam a mesma aplicação, pois só assim é possível saber a referência do outro agente para comunicação. Uma aplicação pode lançar quantos agentes forem necessários e ela é dita ser o aplicação gerente, que atende a configuração de algum operador humano.

6.3 Trabalhos Futuros

Alguns pontos não foram tratados neste trabalho e são deixados para trabalhos futuros:

- as aplicações clientes são monitoradas indiretamente pelos servidores, porém é importante instrumentá-las também;
- instrumentar as aplicações referentes a infra-estrutura de mobilidade, com a finalidade de avaliar o impacto da estrutura de gerência no sistema gerenciado;
- testar as aplicações em um sistema real de gerência com gerentes e/ou agentes processando as informações;
- fornecer maior autonomia aos agentes móveis, delegando tarefas a fim de diminuir a interação com o gerente possibilitando uma descentralização total da gerência;
- disponibilizar a comunicação entre as aplicações gerentes;
- modelar e implementar outros filtros e agentes móveis para a gerência;

- abordar a gerência dos sistemas distribuídos pelos agentes móveis em outras áreas funcionais: configuração, segurança e falhas;
- migrar o código do *OrbizWeb 2.0.1* para o *OrbizWeb 3*.

Bibliografia

- [BDJ96] Yolande Berbers, Bart De Decker, e Wouter Joosen. Infrastructure for Mobile Agents, 1996. Department of Computer Science - KULeuven <http://www.cs.kuleuven.ac.be/cwis/research/hermix/index-english.html>.
- [BDM96] Dominique Benech, Thierry Desprats, e Jean-Jacques Moreau. A Conceptual Approach to the Integration of Agent Technology in System Management. Em *DSOM*, Outubro 1996.
- [BEA] BEA System. *ObjectBroker and Iceberg*. URL:<http://www.beasys.com>.
- [Bor] Borland/Visigenic. *VisiBroker*. URL:<http://www.visigenic.com/prod>.
- [BPW98] Andrzej Bieszczad, Bernard Pagurek, e Tony White. *Mobile Agents for Network Management*. IEEE Communications Surveys, 1998.
- [BRI97] BRISA. *Arquitetura de Redes de Computadores - OSI e TCP/IP*. Makron Books, 2nd. edition, 1997.
- [BU96] Hajo Brunne e Thomas Uslander. Design of a Monitoring System for CORBA-based Applications. Em *International Workshop TreDS - Trends in Distributed Systems*, pp. 52-56, Aachen - Germany, 1996.
- [Cam97] André Campeau. *Managing Networks with Mobile Code - Fourth Year Project*. Technical report, Carleton University, April 1997. <http://www.sce.carleton.ca/netmanage/perpetuum.shtml>.
- [CFD90] J. Case, M. Fedor, e J. Davin. *A Simple Network Management Protocol (SNMP)*. Technical Report RFC 1157, Network Working Group, Maio 1990.
- [CGH⁺95] D. Chess, B. Grosz, C. Harrison, D. Levine, C. Parris, e G. Tsudik. *Itinerant Agents for Mobile Computing*. Technical Report RC 20010, IBM, Março 1995.

- [CGWK94] George Cybenko, Robert Gray, Yunxin Wu, e Alexy Khrabrov. *Information Architecture and Agents*. Thayer School of Engineering, Dartmouth College, Hanover, 1994.
- [CM96] F. M. Costa e E. R. M. Madeira. *An object group model and its implementation to support cooperative applications on CORBA. Distributed Platforms*. Chapman & Hall, 1996.
- [Cor] Cornell e Tromso's University. *Tacoma*. <http://www.tacoma.cs.uit.no>.
- [CPE97] P. E. Clements, T. Papaioannou, e J. M. Edwards. *Aglets: Enabling the Virtual Enterprise*. MSI Research Institute - Loughborough University, 1997.
- [Cry] Crystaliz, Inc. *MuBot*. <http://www.crystaliz.com/papers/mubot.html>.
- [Dar] Dartmouth College. *AgentTcl*. URL:<http://www.cs.dartmouth.edu/agent>.
- [Dis] Distributed Systems and Operating Systems Research Group (VSB) - J. W. Goethe - Universitat at Frankfurt. *AMETAS*. <http://www.vsb.informatik.uni-frankfurt.de/ametas/index.html>.
- [End98] Markus Endler. *Novos Paradigmas de Interação usando Agentes Móveis*. 16o. SBRC - Simpósio Brasileiro de Redes de Computadores, Maio 1998.
- [Eur] European Computer-Industry Research Centre - ECRC. *MSAs*. <http://www.ecrc.de/research/de/msa>.
- [Exp] Expersoft. *PowerBroker*. URL:<http://www.espersoft.com/>.
- [FG96] S. Franklin e A. Graesser. It is an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Em *3rd International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.
- [FSH+95] R. Friedrich, S. Saunders, G. Zaidnweber (HP), D. Bachmann (IBM), e S. Blumson. *Standardized Performance Instrumentation and Interface Specification for Monitoring DCE-based Application*, 1995. OSF DCE RFC 33.0.
- [Gen] General Magic. *Odyssey*. URL:<http://www.genmagic.com/>.
- [GHN+97] S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, e R. Evans. *Software Agents: A review*. Trinity College Dublin, Maio 1997.

- [GMD97] GMD Fokus, IBM, Crystaliz, General Magic e The Open Group. *Joint Submission – Mobile Agent System Interoperability Facilities Specification/97-10-05*, Outubro 1997.
- [Gol93] Germán S. Goldszmidt. On Distributed System Management. Em *The Third IBM/CAS Conference*, Toronto - Canadá, Outubro 1993.
- [Gol96] Germán S. Goldszmidt. *Distributed Management by Delegation*. PhD thesis, Columbia University, 1996.
- [GY95] Germán Goldszmidt e Yechiam Yemini. Distributed Management by Delegation. Em *15th International Conference on Distributed Computing Systems*, Junho 1995.
- [GY98] Germán S. Goldszmidt e Yechiam Yemini. Delegated agents for network management. *IEEE Communications Magazine*, Março 1998.
- [HA93] H.G. Hegering e S. Abeck. *Integrated Network and System Management*. Addison-Wesley Publishing Company, 1993.
- [HA94] H.G. Hegering e S. Abeck. *Integrated Network and System Management. Data Communication and Network Series*. Addison-Wesley Publishing Company, 1994.
- [HCK95] C. G. Harrison, D. M. Chess, e A. Kershenbaum. *Mobile Agents: Are they a good idea?* Technical Report RC 19887, IBM Research Division, 1995.
- [IBM] IBM. *Component Broker*. URL:<http://www.software.ibm.com/ad/cb>.
- [IBM96] IBM Corporation – IBM Intelligent Agent Center of Competency. *The Role of Intelligent Agents in the Information Infrastructure*, 1996. <http://www.raleigh.ibm.com/iag/iagptc2.html>.
- [IKV98] IKV. *Grasshopper*, 1998. <http://www.ikv.de/products/grasshopper/grasshopper.html>.
- [Ion] Iona Technologies. *Orbix and OrbixWeb*. URL:<http://www.iona.ie/>.
- [Ion95] Iona Technologies Ltd. *Orbix Programming and Reference Guide, release 2.0*, 1995.
- [Ion97] Iona Technologies Ltd. *OrbixWeb Programming and Reference Guide, release 2.0*, 1997.

- [ISO91] ISO/IEC - International Organization for Standardization/International Electrotechnical Commission. *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects*, Março 1991.
- [Jav] JavaSoft. *Java IDL*. URL:<http://www.javasoft.com>.
- [Kel96] Alexander Keller. Service-based Systems Management: Using CORBA as a Middleware for Intelligent Agents. Em *7th IFIP/IEEE International Workshop Distributed System Operations and Management*, L'Aquila - Italy, Outubro 1996.
- [KK96] K. Kotay e D. Kotz. *Transportable Agents*. Department of Computer Science - Dartmouth College, 1996.
- [KSM+97] Sven Krause, Flávio Morais de Assis Silva, Thomas Magedanz, Radu Popescu-Zeletin, Orandi Mina Falsarella, e Carlos Raul Arias Méndez. MAGNA - A DPE-based Platform for Mobile Agents in Electronic Service Markets. Em *IEEE Third International Decentralized Systems - ISADS*, pp. 93-102, 1997.
- [LD95] Anselm Lingnau e Oswald Drobnik. *An Infrastructure for Mobile Agents: Requirements and Architecture*. Fachbereich Informatik (Telematik), Johann Wolfgang Goethe-Universität - Frankfurt - Germany, <http://www.tm.informatik.uni-frankfurt.de/agents/>, 1995.
- [LM95a] L. O. B. Lento e E. R. M. Madeira. Um esquema para acessar objetos em ambientes distribuídos. Em *13o SBRC - Simpósio Brasileiro de Redes de Computadores*, 1995.
- [LM95b] L. A. P. Lima e E. R. M. Madeira. A Model for a Federative Trader. Em *ICODP - International Conference on Open Distributed Processing*, pp. 155-166, Fevereiro 1995.
- [LM97] F. H. S. Lima e E. R. M. Madeira. Qualidade de Serviço Multinível baseada em ATM para a Plataforma Multiware. Em *15o SBRC - Simpósio Brasileiro de Redes de Computadores*, pp. 366-382, Maio 1997.
- [LO97] Danny B. Lange e Mitsuru Oshima. *Programming Mobile Agents in Java (TM) - With the Java Aglet API*. URL:<http://www.trl.ibm.co.jp/aglets/aglet-book/index.html>, 1997.

- [MEBS95] Kraig Meyer, Mike Erlinger, Joe Betser, e Carl Sunshine. Decentralizing Control and Intelligence in Management Network. Em *4th International Symposium on Integrated Network Management*, Santa Barbara – CA, Maio 1995.
- [Mit] Mitsubishi. *Concordia*. <http://www.meitca.com/HSL/Projects/Concordia>.
- [MLM⁺94] E. R. M. Madeira, W. P. de Loyolla, M. J. Mendes, E. Cardozo, e M. F. Magalhães. Multiware Plataform: an Open Distributed Enviromment for Multimedia Cooperative Applications. Em *COMPSAC - IEEE Computer Software & Application Conference*, Teipei - Taiwan, 1994.
- [MP95] B. Meyer e C. Popien. Flexible Management of ANSAware Applications. Em *ICODP - International Conference on Open Distributed Processing*, Fevereiro 1995.
- [MR91] K. McCloghrie e M. Rose. *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*. Technical Report RFC 1213, Network Working Group, Março 1991.
- [Nwa96] H. S. Nwana. Sotware Agents: An Overview. *Knowledge Engineering Review*, 11(3):1–40, Setembro 1996.
- [Obj97] ObjectSpace, Inc. *Voyager and Agent Platforms Comparison*, Dezembro 1997. <http://www.objectspace.com/developers/voyager/white/index.html>.
- [OH98] Robert Orfali e Dan Harkey. *Client/Server Programming with Java and CORBA*. Wiley Computer Publishing, 2nd. edition, 1998.
- [OHE96] Robert Orfali, Dan Harkey, e Jeri Edwards. *The Essential Distributed Objects – Survival Guide*. John Wiley & Sons, Inc, 1996.
- [Oli97] Paulo César de Oliveira. *Sistemas Baseados em Agentes Móveis: uma Abordagem Alternativa para o Desenvolvimento de Sistemas Distribuídos*. Dissertação de Mestrado, Faculdade de Engenharia Elétrica e Computação - Universidade Estadual de Campinas – UNICAMP, Campinas - SP, Dezembro 1997.
- [OMG95] OMG Object Management Group. *Common Facilities RFP 3 - Mobile Agents Facility/95-11-03*, Novembro 1995.
- [OMG96] OMG Object Management Group. *Objects By Value RFP/96-06-14*, Junho 1996.

- [OMG97a] OMG - Object Management Group. *CORBA services Specification* OMG formal/97-06-01, Junho 1997.
- [OMG97b] OMG Object Management Group. *CORBA facilities Specification* OMG formal/97-06-15, Junho 1997.
- [OMG98] OMG Object Management Group. *Objects By Value* orbos/98-01-18, Janeiro 1998.
- [Ope] Open Group. *MOA*. <http://www.opengroup.org/RI/java/moa/index.html>.
- [Pur] Purdue University. *Coast*. <http://www.cs.purdue/homes/mcrosbie/research>.
- [Que97] Joao Augusto Gomes de Queiroz. *Gerência de Sistemas Distribuídos Heterogêneos: Facilidade de Monitorização em um Ambiente CORBA*. Dissertação de Mestrado, Instituto de Computação - Universidade Estadual de Campinas - UNICAMP, Campinas - SP, Dezembro 1997.
- [RBP+91] James Rumbaugh, Michael Blaha, Willian Premerlani, Frederick Eddy, e Wilian Lorensen. *Object-Oriented a Modelling and Design*. Prentice-Hall, 1991.
- [RM90] M. Rose e K. McCloghrie. *Structure and Identification of Management Information for TCP/IP-based Internets*. Technical Report RFC 1155, Network Working Group, Maio 1990.
- [RM98] Patricia Ropelatto e Edmundo R. M. Madeira. Gerenciamento de Sistemas Distribuídos em Ambiente CORBA usando Agentes Móveis. Em *WoSiD - II Workshop em Sistemas Distribuídos*, pp. 55-62, Curitiba - Pr, Junho 1998.
- [RSM98] Patricia Ropelatto, Bruno Schulze, e Edmundo R. M. Madeira. Distributed Objects Management in CORBA Environment using Mobile Agents. Em *NCS - International Conference on Networks and Communication Systems*, pp. 13-16, Pittsburgh - Pennsylvania - USA, Maio 1998.
- [Sch98] Douglas C. Schmidt. Evaluating architectures for multithreaded object request brokers. *Communications of the ACM*, 41(10):54-60, Outubro 1998.
- [Sel94] T. Selker. Coach: A Teaching Agent that Learns. *Communications of the ACM*, 37(7):92-99, 1994.
- [SH97] N. Soukouti e U. Hollberg. Joint Inter Domain Management: CORBA, CMIP and SNMP. Em *IFIP - Integrated Network Management V (IM)*, pp. 153-164, San Diego - USA, Maio 1997.

- [Slo94] Morris Sloman. *Network and Distributed Systems Management*. Addison-Wesley Publishing Company, 1994.
- [SMRao] Bruno Schulze, Edmundo R. M. Madeira, e Patricia Ropelatto. MomentA: Gerenciamento de serviços usando agentes móveis em ambiente corba. Em *17o. SBRC – Simpósio Brasileiro de Redes de Computadores*, Salvador - BA, Maio 1999 (aceito para publicação).
- [Sun99] Sun Microsystems. *Java Dynamic Management Kit (JDMK)*, 1999. <http://www.sun.com/software/java-dynamic/ds-jdmk.html>.
- [Sus97] Gatot Susilo. *Infrastructure for Advanced Network Management based on Mobile Code*. Technical Report SCE-97-10, Systems and Computer Engineering, Carleton University, Junho 1997.
- [Unia] Universitat Kaiserslautern. *ARA*. http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html.
- [Unib] University of California at Berkeley. *Java-To-Go*. <http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go>.
- [Unic] University of Stuttgart. *Mole*. Distributed Systems Group – Institute of Parallel and High-Performance Systems (IPVR) – <ftp://suntrec.informatik.uni-stuttgart.de/pub/MOLE/documentation.html>.
- [Vas99] Franciso J. S. Vasconcellos. *Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA*. Dissertação de Mestrado, Instituto de Computação - Universidade Estadual de Campinas – UNICAMP, Campinas - SP, Abril 1999.
- [VD98] Andreas Vogel e Keith Duddy. *JAVA Programming with CORBA*. Wiley Computer Publishing, 2nd edition, 1998.
- [Ven97] Bill Venners. Under the Hood: The Architecture of Aglets. URL:<http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html>, Abril 1997.
- [Vin98] Steve Vinoski. New features for corba 3.0. *Communications of the ACM*, 41(10):44–52, Outubro 1998.
- [VM98] Francisco J. S. Vasconcellos e Edmundo R. M. Madeira. Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA. Em *16o.*

SBRC – Simpósio Brasileiro de Redes de Computadores, pp. 501–520, Rio de Janeiro - RJ, Maio 1998.

[Whi97] Jim White. *Mobile Agents White Paper*. General Magic, URL:<http://www.genmagic.com/agents/>, 1997.

Apêndice A

Descrição de Alguns Sistemas de Agentes Móveis

A.1 Telescript

A arquitetura do *Telescript* é composta por uma linguagem de programação, de mesmo nome, através da qual é possível programar agentes e lugares, um interpretador para a linguagem e protocolos de comunicação para transporte de agentes.

A linguagem de programação *Telescript* é orientada a objeto, podendo ser combinada com código em C e C++, e define comandos para a movimentação de agentes e a interação entre agentes e lugares. Um lugar representa uma possível localização dos agentes numa rede de computadores. Aplicações inteiras podem ser escritas em *Telescript*, porém geralmente as linguagens C e C++ são usadas para implementar as partes não-móveis do código que tratam da interface com o usuário e do acesso a recursos locais como, por exemplo, um banco de dados. Como em outras linguagens orientadas a objeto, a unidade básica de programação é a classe. *Telescript* fornece uma série de classes pré-definidas, como *Object*, *Agent*, *Place*, *Meeting Place*, etc, que podem ser usadas pelo programador para derivar classes mais específicas. No entanto, herança múltipla só é permitida de forma limitada. Primitivas da linguagem como *go*, *meet*, *connect* são responsáveis pela comunicação e locomoção de agentes. Não há mecanismo de seleção de destinos.

O interpretador implementa as primitivas da linguagem *Telescript* através da manutenção e execução do código de um ou mais lugares e agentes localizados nestes lugares. Também implementa o controle e o compartilhamento dos recursos físicos da máquina em que está executando por parte de agentes e lugares. Para isto, o próprio interpretador tem acesso aos recursos locais através das APIs: uma API de armazenamento dá acesso à memória não volátil, uma API de transporte dá acesso aos protocolos de rede e transporte que permitem transferir o estado de agentes e uma API de aplicação permite a interação

com código de aplicação escrito em C/C++.

Os protocolos permitem que dois interpretadores se comuniquem de forma que agentes possam ser transportados na ocorrência da execução da primitiva `go`. O conjunto de protocolos executa sobre uma grande variedade de protocolos de rede, incluindo TCP/IP, X.25 e correio eletrônico. Os protocolos *Telescript* operam em dois níveis. O nível mais baixo cuida do transporte de agentes enquanto que o nível mais alto da sua codificação e decodificação. Comparado com as sete camadas do modelo OSI, o protocolo do nível mais alto do *Telescript* corresponde às camadas de apresentação e aplicação.

Um agente, no sistema *Telescript*, é implementado como um objeto e possui persistência para o ciclo de vida, cujo estado de execução do agente móvel é mantido quando o agente viaja de um local para outro. Também por questões de tolerância a falhas, a informação que os agentes carregam consigo são armazenadas em memória não volátil onde quer que eles estejam. Quanto à comunicação, os agentes podem interagir com outros agentes localizados no mesmo lugar através de chamadas de procedimentos ou em lugares diferentes de forma remota.

Quanto ao aspecto de segurança, o *Telescript* utiliza os conceitos de autoridades, regiões, identidades e permissões. Autoridades identificam pessoas ou organizações do mundo real que os agentes representam. Uma região é um conjunto de lugares que são operados pela mesma autoridade. Identidades distinguem agentes e locais com mesma autoridade. As permissões são usadas para proteger autoridades da execução errônea ou maliciosa de agentes e lugares e limitar o uso de recursos de um determinado lugar. Existem três tipos de permissões: do usuário, do lugar e da região. Um agente só pode executar uma operação se todas as três permissões assim o permitirem.

A.2 AgentTcl

A arquitetura do sistema *AgentTcl* consiste em quatro níveis. O nível mais baixo é uma interface de programação para mecanismos de transporte, que pode ser TCP/IP ou correio eletrônico. O segundo nível é um servidor que deve ser executado em cada *host* de uma rede e é responsável por controlar os agentes executando em seu *host*, receber e autenticar agentes que chegam no *host*, reiniciar agentes em um ambiente de execução adequado e prover mecanismos básicos para a comunicação entre agentes e sua migração. No nível acima do servidor estão os ambientes de execução para as linguagens de programação de agentes: os interpretadores Tcl e Scheme e a máquina virtual Java. O último nível da arquitetura é composto pelos próprios agentes móveis, sendo que alguns deles implementam serviços do sistema.

No sistema *AgentTcl* a persistência para ciclo de vida é implementada e assim o estado de execução do agente móvel é capturado e transportado quando o agente viaja. Isto

acontece quando a primitiva para a movimentação de agentes, `agent_jump`, é executada. O servidor no *host* destino é responsável por receber o agente e repassá-lo ao interpretador adequado.

A comunicação entre agentes é implementada através de mensagens (para comunicação assíncrona) e de encontros (para transferência síncrona de dados). As facilidades para construção de interfaces gráficas providas pela linguagem Tcl/Tk podem ser utilizadas para a comunicação com seres humanos. Além disso, *AgentTcl* provê RPC como mecanismo de comunicação de alto nível entre agentes, que podem estar localizados num mesmo *host* ou em *hosts* distintos da rede. No ambiente com RPC, as interfaces de comunicação de um agente são especificados em IDL, neste caso, chamada de *Agent IDL* (AIDL) a partir da qual um compilador gera *stubs* usados pelos agentes cliente e servidor para realizarem a comunicação.

AgentTcl tem sido usado para aplicações em busca de informações distribuídas como, por exemplo, agentes coletores de informações em relatórios técnicos. Nesta aplicação, os agentes móveis interagem com agentes dedicados à manutenção dos relatórios técnicos de diferentes instituições e destes coletam as identificações dos relatórios a partir da busca de palavras chave em seu texto. Além disso, agentes são replicados para efetuarem a busca em paralelo e resultados parciais destes clones são coletados por um agente mestre, que se encarrega de transportar os resultados de volta para a estação origem e apresentá-los ao usuário [End98].

A.3 Aglets Workbench

A arquitetura AWB consiste dos seguintes componentes: estrutura do *aglets*, servidor de *aglets*, padrões de *aglets* e para a mobilidade o ATP ¹ e o J-ATCI ². O ATP e a estrutura do *Aglets* foram a base tecnológica para a submissão da IBM à MASIF do OMG, antiga MAF.

Um *aglet* é autônomo, pois executa sua própria *thread* após chegar no *host*, e reativo, por causa da habilidade de responder às mensagens que chegam. A estrutura de um *aglet* consiste de quatro partes (Figura A.1) [CPE97]: o núcleo, o *proxy*, um identificador e um itinerário (se necessário). O núcleo é o coração do *aglet* e contém todas as variáveis internas e métodos, fornecendo interfaces através do qual o *aglet* pode se comunicar com seu ambiente. Este é encapsulado pelo *proxy* que age como uma proteção a qualquer tentativa de acesso direto a algum dos métodos e variáveis privados. Também pode esconder a localização real do *aglet* de *aglets* maliciosos. O identificador de um *aglet* é único e imutável através do seu tempo de vida. O itinerário é o plano de viagem do *aglet*

¹ *Agent Transfer Protocol*

² *Java Agent Transfer and Communication Interface*

e pode ser descrito de forma complexa com vários destinos e com tratamento automático de falhas.

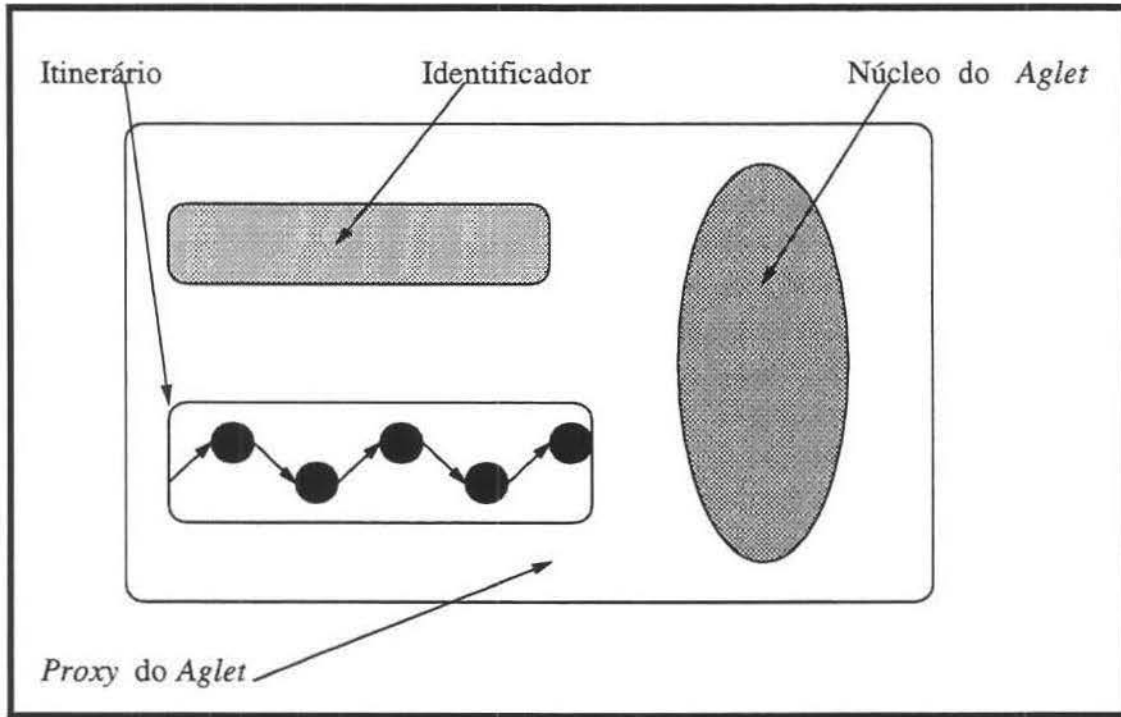


Figura A.1: Estrutura de um *Aglet*

O modelo de objetos *aglets* disponibiliza as classes *Aglet*, *AgletIdentifier*, *AgletProxy*, *Itinerary* e *Message* e as interfaces *AgletContext*, *FutureReply* e *MessageManager* como componentes da J-AAPI ³. Estas classes e interfaces abstratas servem de base para a programação de uma aplicação baseada em *aglets*. Os eventos suportados pelo modelo de objetos *aglets* são:

- criação: ocorre dentro de um contexto. Ao novo *aglet* é associado um identificador único, seu estado é inicializado e sua *thread* principal começa a execução;
- clonagem: uma cópia quase idêntica de um *aglet* é criada no mesmo contexto. As diferenças estão nos identificadores e no fato da execução ser reiniciada;
- despacho: um *aglet* é transportado de um contexto para outro, onde sua execução é reiniciada. As *threads* de execução não são migradas;

³ Java Aglet Application Programming Interface

- recolhimento: um *aglet* é trazido de um contexto para outro onde a requisição de recolhimento foi feita. É inverso ao evento de despacho;
- desativação: um *aglet* pode ser removido temporariamente do seu contexto corrente e armazenado em memória não volátil;
- ativação: recuperação de um *aglet* armazenado em memória não volátil para um contexto;
- término: a execução de um *aglet* é terminada e o agente é removido do seu contexto.

Um lugar de execução de agentes no AWB é chamado de contexto. Um contexto provê uma interface padrão com serviços disponíveis para os *aglets*, tais como: criação de outros *aglets*, transferência para outro contexto, clonagem, etc. Para que os *aglets* executem em seu contexto, uma aplicação chamada servidor de *aglets* (agência) precisa estar executando em todos os *hosts* do sistema distribuído. Este servidor instancia o gerente de segurança de Java para restringir e controlar as ações de *aglets* vindos de outros servidores. A transferência destes *aglets* é feita através de um carregador dinâmico de classes que transfere os arquivos com a classe e o estado de um *aglet* de uma máquina remota para a máquina local. O servidor de *aglets* fornece um gerente gráfico de agentes chamado *Tahiti* e o integrador do AWB com a tecnologia WWW, de nome *Fiji*. *Tahiti* permite que algumas funcionalidades de agentes móveis *aglets* possam ser executadas via interface gráfica como criação, término e migração. *Fiji* permite que *applets* Java também sejam capazes de manipular agentes *aglets*.

Outra maneira na qual a infra-estrutura AWB suporta o desenvolvedor de agente é através da introdução do uso de padrões de *aglets*. O projeto e desenvolvimento baseado em padrões tem fornecido muito sucesso em várias áreas. Estes padrões descrevem relacionamentos comuns entre agentes como Mestre-Escravo, Mensageiro-Receptor e Notificador-Notificação.

A inicialização de um *aglet*, assim como todas as demais operações sobre os *aglets*, geralmente requer a participação do próprio *aglet* a ser manipulado. Isto é realizado através do modelo *callback*. Para cada evento *E* na vida de um *aglet*, um método *callback onE* é invocado para permitir que o *aglet* possa reagir ou se preparar a este evento. Por exemplo, antes de um *aglet* ser enviado para um novo contexto através do método *dispatch()*, o método *onDispatch()* é invocado. Isto indica ao *aglet* sobre o evento que ocorrerá e este deve completar qualquer tarefa não terminada e preparar seu estado para a serialização. Os métodos *callback* podem ser reescritos para representarem o comportamento desejado.

O ATP é o protocolo do nível de aplicação para a Internet usado pelos servidores de *aglets* para transferir o código e estado de *aglets* de um *host* para outro. Os *aglets* são transformados numa sequência de bytes quando são migrados. O protocolo ATP usa os

URLs ⁴ como forma de endereçar recursos em sistemas de agentes. Embora este protocolo tenha sido implementado em Java como um pacote auto-contido do AWB, seu domínio de uso não é exclusivo de *aglets*, podendo ser usado em outros ambientes de programação de agentes móveis. Este pacote consiste de um conjunto de classes que define uma API para a criação de *daemons* ATP, conexão entre estes, processamento de requisições e geração de respostas no padrão ATP.

Reforçando o ATP em um nível mais alto da comunicação está o J-ATCI, um protocolo de agentes independente responsável pelo envio e recebimento de agentes, bem como pelo estabelecimento de comunicação entre eles. Ele possui uma interface de programação flexível e simples que possibilita aos programadores desenvolver agentes independentes de plataforma.

No AWB, a comunicação entre agentes é realizada através do mecanismo de troca de mensagens via contexto. Este processo requer a troca de um objeto de mensagem entre dois *aglets* e permite a passagem de mensagens síncrona e assíncrona. Já a comunicação entre seres humanos e agentes pode ser feita através da construção de interfaces gráficas oferecidas pelo AWT ⁵ da linguagem Java.

Quanto à segurança, o AWB permite duas categorias de *aglets*: confiáveis e não confiáveis. Isto está estritamente relacionado ao controle de acesso de recursos num dado *host*. O AWB faz uso das características suportadas pela linguagem Java como, por exemplo, a checagem de consistência realizada pelo verificador de *bytecodes*. A API de segurança Java também é utilizada e possibilita os desenvolvedores de agente incluírem funcionalidades de segurança em seus agentes como criptografia e autenticação. Em relação ao mecanismo de segurança específico do AWB, o *Tahiti* e o *Fiji* implementam um gerente de segurança configurável que fornece um alto grau de segurança aos *hosts* do sistema e seus proprietários.

Quanto ao aspecto de tolerância a falhas, o AWB não o trata em nenhum nível. Entretanto, as capacidades de desativação e ativação podem ser utilizadas pelo desenvolvedor para prover persistência e recuperação.

A.4 Voyager

Um conceito fundamental introduzido por este sistema é o de tornar uma classe habilitada para uso remoto. Isto ocorre quando instâncias de uma classe podem ser criadas fora do espaço de endereçamento local de um programa e, caso tais instâncias sejam capazes de receber mensagens, como se fossem locais. A implementação deste conceito se dá através de classes virtuais que se constituem como estruturas centrais do *Voyager*. O utilitário

⁴ *Universal Resource Locators*

⁵ *Abstract Windowing Toolkit*

VCC ⁶ é responsável pela geração de classes virtuais a partir de qualquer código fonte ou interpretável Java.

Instâncias de classes virtuais (objetos virtuais) podem ser criadas remotamente. Quando uma instância é criada, ela recebe um identificador global único. Sua identificação é registrada no serviço de diretórios do sistema. Se o código da classe original não estiver disponível remotamente, ele é carregado automaticamente pelo carregador de classes em rede. O acesso a objetos virtuais se dá através de referências virtuais.

Para que os objetos remotos possam ser encontrados sem conhecimento prévio de sua localização, o sistema *Voyager* oferece o serviço de diretórios de nomes. Ele se baseia num *alias* dado a objetos. Diretórios em rede também podem ser criados e conectados para formar um serviço de nomes federativo em grande escala. Cada objeto criado recebe um identificador global único e é registrado neste serviço de diretórios.

O suporte à comunicação em grupo é realizado através do conceito de espaço. Neste conceito, grupos locais de objetos são agregados em subespaços que, por sua vez, constituem um grupo lógico de maior escala chamado de espaço. Quando uma mensagem é recebida num subespaço, cópias idênticas são feitas e enviadas para os subespaços mais próximos antes de que a mensagem seja entregue a cada objeto no subespaço local.

A comunicação entre objetos e, conseqüentemente, entre agentes é realizada através do mecanismo de troca de mensagens que podem ser síncronas ou assíncronas. Mensagens são enviadas para referências virtuais de objetos que, por sua vez, as encaminham para os objetos remotos associados.

Devido à característica de serialização da linguagem Java, qualquer objeto virtual seriado pode migrar de um programa para outro no *Voyager*. A migração é ativada pelo recebimento da mensagem *moveTo*. Antes de um objeto migrar, ele deixa um objeto especial local ⁷ para repassar mensagens e lidar com requisições futuras de conexões.

Um agente móvel é um tipo especial de objeto neste sistema, com todas as características de objetos descritas anteriormente. Enviando a mensagem *moveTo* para si mesmo, o agente pode migrar autonomamente e também interagir eficientemente com um objeto remoto, movendo-se para aquele objeto e, então, enviar mensagens Java locais. O estado de execução não é migrado juntamente com um agente, apenas seu código e dados internos. A ação a ser executada no destino é especificada antes da migração. Para o transporte, é utilizado o protocolo TCP/IP.

Quanto aos aspectos de segurança, *Voyager* implementa um gerente de segurança seguindo o modelo da linguagem Java. A sua instalação é opcional e sua funcionalidade básica engloba os aspectos de autenticação, autorização e controle de acesso.

Tolerância a falhas é suportada pela infra-estrutura *Voyager* através de persistência.

⁶ *Virtual Code Compiler*

⁷ *forwarder*

Um objeto persistente contém uma cópia de segurança em uma base de dados. *Voyager* inclui um sistema de armazenamento de objetos de alto desempenho chamado *VoyagerDb*. Outras bases de dados, sejam relacionais ou orientadas a objetos, podem ser usadas pelos desenvolvedores.

Apêndice B

Estrutura completa de um filtro por processo

```
public class FilterProc extends Filter {

    public boolean outRequestPreMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }

    public boolean outRequestPostMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }

    public boolean inReplyPreMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }

    public boolean inReplyPostMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }

    public boolean inRequestPreMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
```



```
        return true;
    }
    public boolean inRequestPostMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }
    public boolean outReplyPreMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }
    public boolean outReplyPostMarshal (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }
    public boolean outReplyFailure (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }
    public boolean inReplyPreFailure (Request request)
        throws SystemException {
        // codigo a ser executado
        return true;
    }
}
```

Apêndice C

Código da Implementação

```
/ *****  
Implementacao de um Agente em Largura que coleta dados gerados  
pelo filtro F0.  
***** /
```

```
package Agentes;
```

```
import MAF.*;  
import Agentes._EncapsFilterProcessRef;  
import Agentes.EncapsFilterProcess;  
import Agentes._EncapsFilterProcess.dataFilter;  
import IE.Iona.Orbix2._CORBA;  
import IE.Iona.Orbix2.IT_daemonRef;  
import IE.Iona.Orbix2.IT_daemon.*;  
import IE.Iona.Orbix2.IT_daemon;  
import Agentes.CollectCM;  
import IE.Iona.Orbix2._CORBA;  
import IE.Iona.Orbix2.CORBA.SystemException;  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.Vector;
```

```
import Agentes._CollectCM.dataCM;
```

```
public class AgLargura extends BaseAgent {
```

```
    int counter;  
    String hostname;
```

```

Vector criticalHosts = new Vector();
Vector criticalHostsSecond = new Vector();
dataCM dtCM = new dataCM();
dataFilter datastruct = new dataFilter();
boolean secondlap;
EventProcessor e;

public AgLargura() {
    super();
    counter = 0;
    hostname = null;
    secondlap = false;
}

public void run() {

    Integer maxCPU = new Integer(0);
    Integer minCPU = new Integer(0);
    Integer maxMem = new Integer(0);
    Integer minMem = new Integer(0);
    Double maxthroughput = new Double(0);
    Double minthroughput = new Double(0);
    double throughputTotal;
    throughputTotal = 0;
    e = new EventProcessor();
    e.start();

    String host = itinerary.buffer[0];

    // Parte que vai ler os dados coletados pelo filtro nos processos servidores
    String strMaf1 = new String("MAF1");
    String strMaf2 = new String("MAF2");
    String strAg = new String("AgLargSrv");
    String strAgCPM = new String("AgAtCMSrv");
    _EncapsFilterProcessRef refl;
    _IT_daemonRef daemon;
    serverDetailsSeq servers = new serverDetailsSeq();
    servers.ensureCapacity(0);

    try {
        daemon = IT_daemon._bind("", host);

```

```

        daemon.listActiveServers(servers);
    } catch (SystemException e) {
        // Tratar a excecao
        return;
    }

    String srv = new String();
    int value1, value2, value3, value4;
    for(int i=0;i<servers.length;i++) {
        try {
            srv = servers.buffer[i].server;
            System.out.println(srv);
            value1 = srv.compareTo(strMaf1);
            System.out.println(value1);
            value2 = srv.compareTo(strMaf2);
            System.out.println(value2);
            value3 = srv.compareTo(strAg);
            System.out.println(value3);
            value4 = srv.compareTo(strAgCPM);
            System.out.println(value4);
            if ( (value1 == 0) || (value2 == 0) || (value3 == 0) || (value4 == 0) ) {
                System.out.println("O servidor eh: " + srv + ",nao faz nada");
            } else {
                System.out.println("Acessa dados no filtro por processo no
servidor: " + srv);
                refl = EncapsFilterProcess._bind(":" + servers.buffer[i].server, host);
                // Preenche os dados correspondentes a interface de gerencia definida em IDI
                datastruct = refl.AcessData();
                throughputTotal = throughputTotal + datastruct.throughput;
            }
        } catch (SystemException sel) {
            // Tratar a excecao
            return;
        }
    }

}

}

```

```

/*****
                        Implementação do Filtro F0
*****/

```

```

package gridFilter;
import IE.Iona.Orbix2.CORBA.Filter;
import IE.Iona.Orbix2.CORBA.Request;
import IE.Iona.Orbix2.CORBA.SystemException;

import gridFilter._EncapsFilterProcess.dataFilter;

public class ProcessFilter0 extends Filter {

    // objeto que representa as informações coletadas por este filtro
    structF0 dt = new structF0();

    public ProcessFilter0 (dataFilter da) {
        dt.num_req_received = da.number_of_request_received;
        dt.num_req_sent = da.number_of_request_sent;
        dt.num_resp_received = da.number_of_response_received;
        dt.num_resp_sent = da.number_of_response_sent;
        dt.throughput_total = da.throughput;
        dt.time_init = da.startup_time;
    }

    public boolean inRequestPreMarshal (Request r) {

        String s = "";
        String o = "";
        int countertotal = 0;
        double temp_obs, end_time;

        try {
            dt.num_req_received++;
            end_time = (double)System.currentTimeMillis();
            temp_obs = end_time - dt.time_init;
            countertotal = dt.num_req_received + dt.num_req_sent + dt.num_resp_received +
dt.num_resp_sent;
            if (countertotal != 1) {
                dt.throughput_total = (double)countertotal/(double)temp_obs; }

```

```

        s = (r.target())._object_to_string();
        o = r.operation();
    } catch (SystemException se) {
        // Trata a exceção
    }

    return true;
}

```

```

public boolean outRequestPreMarshal (Request r) {

    String s = "";
    String o = "";
    int countertotal = 0;
    double temp_obs, end_time;

    try {
        dt.num_req_sent++;
        end_time = (double)System.currentTimeMillis();
        temp_obs = end_time - dt.time_init;
        countertotal = dt.num_req_received + dt.num_req_sent + dt.num_resp_received +
dt.num_resp_sent;
        if (countertotal != 1) {
            dt.throughput_total = (double)countertotal/(double)temp_obs; }
        s = (r.target())._object_to_string();
        o = r.operation();
    } catch (SystemException se) {
        // Trata a exceção
    }

    return true;
}

```

```

public boolean InReplyPreMarshal (Request r) {

    String s = "";
    String o = "";
    int countertotal = 0;
    double temp_obs, end_time;

    try {

```



```

    dt.num_resp_received++;
    end_time = (double)System.currentTimeMillis();
    temp_obs = end_time - dt.time_init;
    countertotal = dt.num_req_received + dt.num_req_sent + dt.num_resp_received +
dt.num_resp_sent;
    if (countertotal != 1) {
        dt.throughput_total = (double)countertotal/(double)temp_obs; }
    s = (r.target())._object_to_string();
    o = r.operation();
} catch (SystemException se) {
    // Trata a exceção
}
return true;
}

```

```

public boolean outReplyPreMarshal (Request r) {

```

```

    String s = "";
    String o = "";
    int countertotal = 0;
    double temp_obs, end_time;

    try {
        dt.num_resp_sent++;
        end_time = (double)System.currentTimeMillis();
        temp_obs = end_time - dt.time_init;
        countertotal = dt.num_req_received + dt.num_req_sent + dt.num_resp_received +
dt.num_resp_sent;
        if (countertotal != 1) {
            dt.throughput_total = (double)countertotal/(double)temp_obs; }
        s = (r.target())._object_to_string();
        o = r.operation();
    } catch (SystemException se) {
        // Trata a exceção
    }
    return true;
}
}

```

```
class structF0 {  
  
    int num_req_received;  
    int num_req_sent;  
    int num_resp_received;  
    int num_resp_sent;  
    double throughput_total;  
    double time_init;  
  
}
```