

UNICAMP
BIBLIOTECA CENTRAL
SEÇÃO CIRCULANTE

Este exemplar corresponde à redação final da
Tese/Dissertação devidamente corrigida e defendida
por: Claudio Márcio Silveira
e aprovada pela Banca Examinadora.
Campinas, 30 de maio de 2001
M. Mendonça
COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Uma Facilidade de Gerenciamento de
Configuração para Aplicações CORBA

Cláudio Márcio da Silveira

Dissertação de Mestrado

Uma Facilidade de Gerenciamento de Configuração para Aplicações CORBA

Cláudio Márcio da Silveira

Fevereiro de 2001

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (IC/UNICAMP) (Orientador)
- Prof. Dr. Paulo Lício de Geos (IC/UNICAMP)
- Prof. Dr. Markus Endler (IME/USP)
- Prof. Dr. Célio Cardoso Guimarães (IC/UNICAMP) (suplente)

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Silveira, Cláudio Márcio

S139f Uma facilidade de gerenciamento de configuração para aplicações
CORBA / Cláudio Márcio Silveira -- Campinas, [S.P. :s.n.], 2000.

Orientador : Edmundo Roberto Mauro Madeira

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

1. Gerenciamento de configurações de Software. 2. Programação
orientada a objeto. 3. CORBA. I. Madeira, Edmundo Roberto Mauro. II.
Universidade Estadual de Campinas. Instituto de Computação. III.
Título.

Uma Facilidade de Gerenciamento de Configuração para Aplicações CORBA

Este exemplar corresponde à redação final da
Dissertação devidamente corrigida e defendida
por Cláudio Márcio da Silveira e aprovada pela
Banca Examinadora.

Campinas, 9 de Fevereiro de 2001.

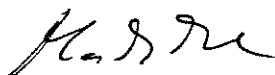


Prof. Dr. Edmundo Roberto Mauro Madeira
(IC/UNICAMP) (Orientador)

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 14 de dezembro de 2000, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Markus Endler
IME/USP



Prof. Dr. Paulo Lício de Geus
IC – UNICAMP



Prof. Dr. Edmundo Roberto Mauro Madeira
IC – UNICAMP

© Cláudio Márcio da Silveira, 2001.
Todos os direitos reservados.

*“Bem-aventurado o homem que acha sabedoria,
e o homem que adquire conhecimento.”*

(Pv 3.13)

*“Adquire a sabedoria, adquire a inteligência
e não te esqueças nem te apartes
das palavras da minha boca.”*

(Pv 4.5)

Ao meu Deus, minha herança e vida
de minha alma.

À minha querida e amada família.

Resumo

Este trabalho apresenta o projeto e implementação da Facilidade de Gerenciamento de Configuração que tem sido desenvolvida para compor a Arquitetura de Gerenciamento Integrado de Sistemas Distribuídos da plataforma Multiware. A Facilidade desenvolve, a partir da especificação XCMF, um modelo para o projeto de aplicações CORBA distribuídas cujas estrutura e interconexões entre os objetos distribuídos possam ser gerenciadas dinamicamente e interativamente a partir de gerentes externos. Os componentes da aplicação serão Instâncias Configuráveis capazes de receberem operações de gerenciamento de configuração.

Abstract

This work presents the design and implementation of the Configuration Management Facility which has been developed to compose the Integrated Management Architecture for Distributed Systems of the Multiware platform. From the XCMF specification, the Facility develops a model for the design of distributed CORBA applications whose structure and interconnections between distributed objects can be interactively managed at run-time by external managers. Components that make up the application are “XCMF Instances” that are able to receive configuration management operations.

Agradecimentos

*“Não te maravilhes de ter dito:
Necessário vos é nascer de novo.”
(Jo 3.7)*

Ao Senhor meu Deus por ter escolhido esta fase de minha vida para demolir todos os fundamentos da minha mente e da minha vida, e me dar outros tantos novos e preciosos! Por não permitir que eu desistisse deste trabalho. *“Clamou este pobre e o Senhor o ouviu; e o salvou de todas as suas angústias.”* (Sl 34.6)

Ao meu Orientador Edmundo R. M. Madeira, pela confiança, paciência, disposição e, principalmente (!), pela educação e serenidade demonstrados em todos os momentos de convivência. Espero que não tenha outro orientando que te dê tanto trabalho quanto eu (principalmente nas apresentações...).

Aos meus pais, José Luiz e Sebastiana de Lourdes, pelo apoio, confiança e pela compreensão quando minhas decisões me levam para cada vez mais longe de casa.

Aos meus amigos Dário e Silvio, pelo companheirismo, pela amizade, e principalmente pela compreensão nos momentos de “grandes mudanças”!

Aos amigos Arlindo e Guilherme pelos momentos de incentivo mútuo na batalha contra a “mardita”.

Ao camarada Cesar, simplesmente por tê-lo como amigo.

Ao Instituto de Computação, pelo apoio acadêmico, e aos seus funcionários pela colaboração.

A Fapesp e CNPq, pelo imprescindível apoio financeiro.

Conteúdo

Resumo	vii
Abstract	viii
Agradecimentos	ix
Lista de Figuras	xiii
1 Introdução	1
1.1 Motivação	3
1.2 Objetivo	3
1.3 Estrutura da Dissertação	4
2 CORBA	5
2.1 Visão Geral	5
2.2 A Arquitetura de Gerência de Objetos (OMA) do OMG	5
2.2.1 O Modelo de Referência	6
2.2.2 O Modelo de Objetos	7
2.3 A Especificação de CORBA	8
2.3.1 A Estrutura de CORBA	9
2.3.2 Implementações de CORBA	10
2.4 Considerações finais sobre CORBA	11
3 Gerenciamento de Sistemas Distribuídos	12
3.1 Aspectos Gerais do Problema de Gerenciamento	12
3.2 Requerimentos para o Gerenciamento de Sistemas Distribuídos	18
3.2.1 Requerimentos de Negócio	18
3.2.2 Requerimentos do Gerenciamento de Aplicações	19
3.2.3 Requerimentos para o Gerenciamento do Ambiente de Suporte	19
3.2.4 Requerimentos não Funcionais	20

3.3	CORBA como Ambiente de Gerenciamento	20
3.4	Especificação de um Serviço de Gerenciamento para CORBA . . .	21
3.5	O Projeto MAScOTTE	23
4	Gerenciamento de Configuração	25
4.1	Gerenciamento de Configuração	25
4.2	Definindo um Serviço de Configuração para Sistemas Distribuídos	26
4.3	A Especificação XCMF	28
5	O Projeto da Facilidade de Gerenciamento de Configuração	34
5.1	O Contexto do Projeto	34
5.2	A Proposta do Trabalho	37
5.3	A Modelagem da Facilidade	41
5.4	Trabalhos Relacionados	47
6	A Implementação da Facilidade	50
6.1	O Ambiente de Desenvolvimento	50
6.2	O ORB OrbixWeb	50
6.3	A implementação da Camada XCMF	52
6.3.1	A Facilidade de Gerenciamento de Conjuntos	55
6.3.2	A Facilidade de Gerenciamento de Instâncias	57
6.3.3	O Mecanismo de Criação de Instâncias	58
6.4	A implementação da Facilidade de Configuração	61
6.4.1	Organizando o Ambiente de Gerenciamento	65
6.4.2	Trabalhando com a Facilidade de Configuração	66
6.4.3	Exemplo de Implementação	68
7	Conclusões	73
7.1	Trabalhos Futuros	74
	Bibliografia	75
A	Descrições IDL da Facilidade de Configuração	79
A.1	Modulo ConfigFacility	79

Lista de Figuras

2.1	Arquitetura de Gerência de Objetos do OMG.	7
2.2	A Estrutura de CORBA	9
3.1	Modelo de Gerenciamento Gerente/Agente	13
3.2	Interações de Gerenciamento com base em ORB (Projeto MAS- cOTTE)	14
3.3	Integração da Tecnologia de Gerenciamento	15
3.4	Dimensões de Gerência	17
3.5	Invocação das Operações de Gerenciamento nos Objetos Gerenci- ados.	24
4.1	Interfaces definidas na especificação XCMF.	29
4.2	“Conjuntos Gerenciados”.	31
4.3	Papéis definidos no Serviço de Gerenciamento de Instâncias.	32
4.4	Relacionamento entre os objetos na Facilidade de Políticas.	33
5.1	Contexto do Projeto.	35
5.2	Ambiente de Gerenciamento.	40
5.3	Serviços de Configuração.	41
5.4	Modelagem da Facilidade.	42
5.5	Ilustração dos Conceitos.	47
6.1	Estrutura da Implementação do ORB OrbixWeb	52
6.2	Organização do Ambiente de Gerenciamento para XCMF	54
6.3	SetBrowser	56
6.4	Movendo ou copiando um LifecycleObject	59
6.5	Movendo ou copiando um ConfigInstance	60
6.6	ConfigBrowser	63
6.7	ConfigBrowser (ilustração)	64
6.8	Organização do Ambiente de Gerenciamento.	66
6.9	Modelagem do Sistema de Monitorização de Pacientes.	69

6.10	Relacionamentos Gerenciados no Sistema.	69
6.11	Configuração da Aplicação no Instante 1.	71
6.12	Configuração da Aplicação no Instante 2.	72

Capítulo 1

Introdução

Com o avanço da tecnologia de comunicação, a indústria a muito tempo substituiu o antigo modelo de um único computador central, servindo a todas as necessidades da organização, por sistemas de redes de computadores e de processamento distribuído.

As redes de computadores surgiram para possibilitar o compartilhamento de recursos computacionais e o acesso às informações geograficamente dispersas. A coexistência de redes heterogêneas levou ao desenvolvimento de um modelo aberto e público voltado para a interconexão dessas redes. Com esse objetivo, a ISO¹ desenvolveu o modelo RM-OSI² para a interconexão de sistemas abertos. No entanto, as aplicações de rede, desenvolvidas segundo o modelo cliente-servidor, ainda possuem uma estrutura monolítica, no que se refere a utilização de recursos.

Os atuais sistemas de comunicação são capazes de compartilhar seus recursos usando diferentes tipos de redes de computadores. Isto permitiu o desenvolvimento de aplicações distribuídas sobre um número de nós através da rede, com um esperado ganho de performance, atuando como um conjunto de entidades cooperativas provendo e usando serviços. Neste contexto, tais serviços podem oferecer mais funcionalidade do que aqueles oferecidos por um sistema operacional ou uma rede de computadores.

A fim de lidar com os problemas surgidos da heterogeneidade e da distribuição, a ISO e o ITU-T³ desenvolveram o Modelo de Referência ODP⁴ com o objetivo de padronizar os sistemas de processamento distribuído aberto [Ray95]. O modelo de referência trata dos conceitos de distribuição, de interoperabilidade, de transparências e de portabilidade de aplicações distribuídas, usando o paradigma

¹International Organization for Standardization

²Reference Model for Open Systems Interconnection (ISO 7498-1)

³International Telecommunication Union

⁴Open Distributed Processing

da orientação a objetos.

Diversos *middleware* para processamento distribuído tem sido desenvolvidos, como por exemplo, a DCE⁵ da Open Software Foundation, a ANSAware, e o Java RMI⁶ da Sun. Em especial, mencionamos o padrão CORBA⁷, componente central da arquitetura OMA⁸ desenvolvida pelo OMG⁹. O termo *middleware* designa serviços que situam-se 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 gerenciamento destes sistemas de comunicação surgiu com sucesso através dos protocolos de gerenciamento de redes, SNMP¹⁰ do modelo TCP/IP¹¹ e CMIP¹² do modelo OSI. No entanto, estes modelos de gerenciamento não atendem aos requisitos para a gerência dos sistemas distribuídos. Estes requisitos serão discutidos adiante neste trabalho.

De acordo com Orfali [OHE96], um sistema de gerenciamento é limitado pela capacidade de seu protocolo de gerência e pelos componentes usados para representar o ambiente gerenciado.

O uso confiável dos sistemas distribuídos requer um gerenciamento adequado de seus componentes, a saber: os recursos de rede e do sistema operacional, os serviços de suporte e *middleware*, e os componentes das aplicações distribuídas. A busca de novas tecnologias para o gerenciamento destes sistemas tornou-se um tema de especial interesse para pesquisadores e projetistas.

Nos últimos anos, a indústria da computação tem adotado a tecnologia de objetos distribuídos como a solução *middleware* para integração de aplicações e processamento distribuído. Isto tem feito de CORBA o padrão “*de facto*” como plataforma distribuída orientada a objetos.

CORBA também ganha espaço como o “protocolo” de gerência destes sistemas distribuídos. Uma idéia central presente nos trabalhos de gerenciamento de sistemas distribuídos [Que97] [SMTK93] [Slo95] é a de que os mesmos conceitos e técnicas utilizados para se projetar uma aplicação distribuída devem também ser usados para o seu gerenciamento. Assim, o gerenciamento de uma aplicação deve ser projetado em paralelo com sua funcionalidade normal. Os objetos em um ambiente distribuído devem ter, além de suas interfaces funcionais, uma ou

⁵Distributed Computing Environment

⁶Remote Method Invocation

⁷Common Object Request Broker Architecture

⁸Object Management Architecture

⁹Object Management Group

¹⁰Simple Network Management Protocol

¹¹Transmission Control Protocol/Internet Protocol

¹²Common Management Information Protocol

mais interfaces de gerenciamento [SMTK93] [Slo95].

1.1 Motivação

Na busca de novas tecnologias para gerenciamento destes sistemas, modelos e conceitos para arquiteturas de gerenciamento de sistemas distribuídos têm sido estudados e propostos nos últimos anos ([Que97], [Slo95], [MaS97] e [XOp97]).

A arquitetura OMA, em particular, adotou por alguns anos a especificação XCMF¹³ [XOp97] como padrão para a sua Facilidade de Gerenciamento. Atualmente, devido a um redirecionamento nos trabalhos de gerenciamento no OMG, esta especificação não vigora mais como padrão.

Os serviços da especificação XCMF formam um *framework* projetado para dar suporte às aplicações que gerenciam um grande número de objetos representando recursos. Usando os serviços XCMF, os projetistas das aplicações podem definir conjuntos de objetos (modelando *containers*) e associar políticas de inicialização ou validação para estes conjuntos. Estes objetos serão gerenciados pelo *tipo* (tipo IDL) e gerados dinamicamente.

Neste ambiente, os objetos são tratados e gerenciados individualmente e não existe o conceito de uma aplicação composta por um grupo destes objetos. Isto dificulta o tratamento de alguns problemas envolvidos no Gerenciamento de Configuração, a saber, os relacionados com o gerenciamento da estrutura das aplicações distribuídas.

Por Configuração de uma aplicação, queremos dizer, principalmente: a *estrutura* da aplicação, que é resultado dos *bindings* estabelecidos entre seus componentes, e o *estado* destes componentes, que será determinado pelo valor em um dado instante de um conjunto de atributos pré-definidos para cada componente.

A estrutura da aplicação é frequentemente identificada como sua Configuração. Ela deve estar ao alcance das aplicações de gerenciamento para uma possível reconfiguração, ou seja, mudança na estrutura da aplicação em tempo de execução.

1.2 Objetivo

Este trabalho tem como objetivo projetar e implementar uma Facilidade de Gerenciamento de Configuração para Aplicações CORBA que permita:

- Gerenciar um *grupo de objetos* definidos como uma Aplicação Distribuída;

¹³X/Open's Common Management Facilities

- Gerenciar a estrutura formada pelos *bindings* estabelecidos entre estes objetos;
- Gerenciar o *estado* destes componentes da Aplicação Distribuída;
- Suportar todas as operações de *Ciclo de Vida* de um objeto CORBA, mantendo a *consistência* do ambiente da Aplicação Distribuída por ocasião destas operações;
- Ter gerenciamento *Dinâmico e Interativo*.

A Facilidade desenvolve, a partir da especificação XCMF, um *Framework* para se projetar uma aplicação distribuída capaz de executar dinamicamente e interativamente operações de gerenciamento de configuração.

Este trabalho contribuiu para o projeto da Arquitetura de Gerenciamento da plataforma Multiware [LMM⁺94] com o desenvolvimento da Facilidade de Configuração e com o estudo das atividades e idéias presentes nos trabalhos de desenvolvimento de um serviço de gerenciamento para a arquitetura OMA.

1.3 Estrutura da Dissertação

O Capítulo 2 descreve brevemente a plataforma CORBA e os componentes da arquitetura OMA.

O Capítulo 3 apresenta os conceitos, problemas e requerimentos envolvidos na área de gerenciamento de sistemas distribuídos. CORBA é analisada como alternativa para o ambiente de gerenciamento e, ao final, é discutida a situação atual dos trabalhos de desenvolvimento de um serviço de gerenciamento para CORBA.

O Capítulo 4 discute o problema de Gerenciamento de Configuração para sistemas distribuídos e apresenta a especificação XCMF.

No Capítulo 5, a proposta deste trabalho é detalhada descrevendo os conceitos envolvidos e o projeto da Facilidade de Gerenciamento de Configuração. Trabalhos relacionados são apresentados ao final.

O Capítulo 6 discute como foi feita a implementação baseada na modelagem do capítulo anterior.

Finalmente, no Capítulo 7, concluímos o trabalho apresentado as conclusões e contribuições desta dissertação e as propostas para trabalhos futuros.

Capítulo 2

CORBA

Este capítulo descreve brevemente a arquitetura CORBA e os componentes da OMA. Ao final, a plataforma Multiware é comentada.

2.1 Visão Geral

CORBA é o componente central da arquitetura OMA e, segundo [Orfali], é o mais importante e ambicioso projeto de *middleware* desenvolvido pela indústria da computação. Ela é baseada no paradigma cliente/servidor e na tecnologia de objetos e propõe uma solução para os problemas atuais de heterogeneidade de *hardware* e *software*. CORBA especifica uma infra-estrutura visando a portabilidade e a interoperabilidade entre objetos distribuídos tornando transparente a localização dos objetos e como foram implementados. Além desta facilidade de comunicação, as demais especificações presentes na OMA fornecem ainda um extensivo conjunto de serviços para criação e deleção de objetos, obtenção de objetos pelo nome, persistência, externalização, entre outros.

2.2 A Arquitetura de Gerência de Objetos (OMA) do OMG

O OMG é um consórcio formado em 1989 e composto por mais de 800 membros, entre eles, centros de pesquisa, universidades e as principais indústrias da computação que desenvolvem seus trabalhos respondendo a RFPs e em encontros promovidos regularmente. A exceção notável no uso das especificações do OMG é a Microsoft que, apesar de membro, possui o seu próprio negociador de requi-

sições de objetos, o DCOM¹. O OMG foi formado com o objetivo de desenvolver *especificações* que venham ajudar a reduzir a complexidade, diminuir os custos e acelerar o desenvolvimento de novas aplicações de software composta por objetos distribuídos interoperáveis, cooperativos, portáteis e reusáveis.

Para alcançar estes objetivos, o OMG estabeleceu a OMA ([OMG97a]) através de um *Modelo de Objetos* e um *Modelo de Referência*, ou seja, uma infra-estrutura conceitual sobre a qual todas as especificações do OMG estão baseadas.

2.2.1 O Modelo de Referência

O Modelo de Referência identifica e caracteriza os componentes, interfaces e protocolos que compõem a OMA.

Os componentes identificados nesse modelo (Figura 2.1) são divididos nas seguintes categorias:

- **O ORB** - É o fundamento da OMA que habilita e gerencia a comunicação entre clientes e objetos em um ambiente distribuído sendo sua especificação dada pela CORBA. O ORB é baseado no paradigma cliente/servidor fornecendo mecanismos para que um cliente faça requisições em objetos servidores e receba respostas de forma transparente, independente de plataforma e de como estes objetos estejam implementados.
- **Serviços de Objetos** - Conhecidos como CORBAservices [OMG97c], são especificações de interfaces que fornecem um conjunto de serviços básicos, de infra-estrutura, que complementam a funcionalidade do ORB auxiliando a implementação e a portabilidade dos objetos. Suas interfaces são orientadas a objetos, descritas em IDL², com uma descrição precisa de sua semântica. Atualmente, o OMG especificou as interfaces para os serviços de Nomes, Eventos, Ciclo de Vida, Persistência, Relacionamento, Externalização, Transação, Controle de Concorrência, Licença, Consulta, Propriedades, Segurança, Tempo, Coleções e Trader.
- **Facilidades Comuns** - Conhecidas como CORBAfacilities [OMG97b], são especificações de interfaces fornecendo um conjunto de funções de uso geral, podendo ser especializações dos Serviços de Objetos. O OMG tem desenvolvido continuamente especificações de Facilidades Comuns para gerenciamento de sistemas, comércio eletrônico, agentes móveis, *workflow*, intercâmbio de dados, banco de dados, entre outras.

¹Distributed Component Object Model

²Interface Definition Language

- **Interfaces de Domínio** - São especificações de interfaces para aplicações específicas dentro de um determinado mercado ou setor como finanças, telecomunicações, manufaturas, entre outras. Esta área, anteriormente, era classificada como Facilidades Comuns Verticais.
- **Objetos de Aplicação** - São interfaces desenvolvidas para as aplicações específicas dos usuários e não são padronizadas pelo OMG.

A segunda parte da especificação do modelo de Referência inclui a noção de *Object Frameworks* que é uma coleção de objetos cooperantes, categorizados como aplicação, domínio, facilidades e serviços, que fornecem uma solução integrada dentro de um domínio de aplicação ou tecnologia. A especificação de um *Object Framework* define interfaces, estruturas, tipos, seqüência de operações, qualidade de serviços e objetos que deverão compor o *framework*. Sua utilização requer uma adaptação pelo usuário ou desenvolvedor.

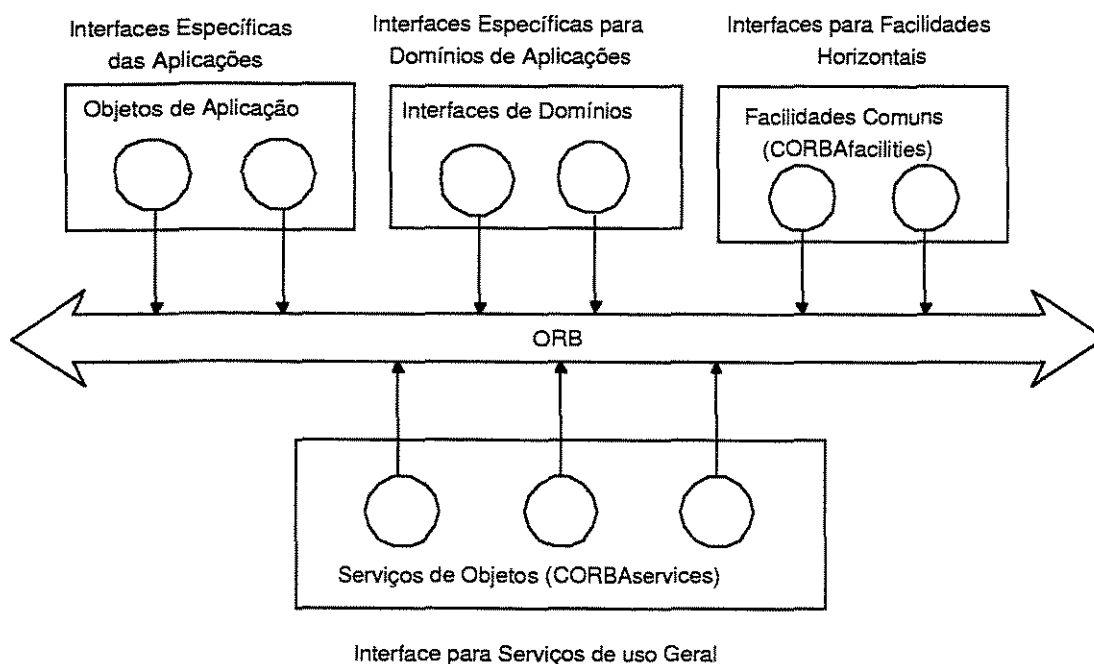


Figura 2.1: Arquitetura de Gerência de Objetos do OMG.

2.2.2 O Modelo de Objetos

Este modelo de objetos conceitual define uma semântica comum para especificar as características externas e visíveis dos objetos de uma forma padronizada e independente de implementação. Define, entre outros, um modelo formal de

tipos, requisições, operações e heranças (de interfaces). A idéia é a de se resolver os problemas de interoperabilidade entre diferentes modelos de objetos impostos pelas linguagens de programação, pelos sistemas operacionais, entre outros, através da padronização de um modelo de objetos integrador.

A maioria dos modelos integradores tem adotado a solução de usar uma linguagem de definição de interface para descrever tipos de objetos.

O OMG padronizou a IDL, puramente declarativa, como linguagem para a definição das interfaces dos objetos CORBA. As descrições em IDL funcionam como um contrato entre o implementador de um objeto e o cliente. Estas descrições passam por um compilador que as traduzem para adaptadores, chamados “stubs” e “skeletons”, em uma linguagem específica. Os “stubs” gerados podem ser ligados aos módulos clientes para compor a aplicação final. Os “skeletons”, de forma similar aos “stubs”, formam a base para a implementação dos objetos servidores. Atualmente, foram estabelecidos mapeamentos de IDL para as linguagens C, C++, Java, Cobol, Ada e Smalltalk.

2.3 A Especificação de CORBA

A especificação CORBA [OMG98] é composta por um núcleo, onde a estrutura do ORB é detalhada e especificada, pelas especificações de Interoperabilidade entre ORBs, pelas especificações de *Interworking* entre CORBA e DCOM e pelas especificações dos mapeamentos de IDL para C, C++, Smalltalk, Cobol, Ada e Java.

O requisito mínimo para uma implementação ser compatível com CORBA é aderir à especificação do núcleo e oferecer um mapeamento de IDL para uma das linguagens acima.

A especificação CORBA apresenta características importantes:

- Invocação estática e dinâmica de métodos. CORBA permite que a invocação do método seja definida estaticamente em tempo de compilação, ou dinamicamente, em tempo de execução;
- “Bindings” de linguagem em alto nível. CORBA separa os conceitos de interface e de implementação e permite a invocação de métodos em objetos servidores usando uma linguagem de programação de alto nível, não importando a linguagem em que o objeto servidor foi escrito;
- Sistema auto descritivo. CORBA fornece informações em tempo de execução para a descrição de todas as interfaces de servidores conhecidas pelo

sistema. O padrão define um Repositório de Interfaces que contém informações descrevendo as operações que um objeto servidor fornece e seus parâmetros.

- **Transparência de localização.** CORBA pode enviar e receber requisições para objetos em um único processo, em múltiplos processos executando em uma mesma máquina, ou múltiplos processos executando através de redes e sistemas operacionais distintos. Tudo ocorre de um modo transparente para os objetos.

2.3.1 A Estrutura de CORBA

A plataforma descrita pela CORBA é composta pelo núcleo do ORB, de suas interfaces e componentes do lado cliente e servidor (Figura 2.2).

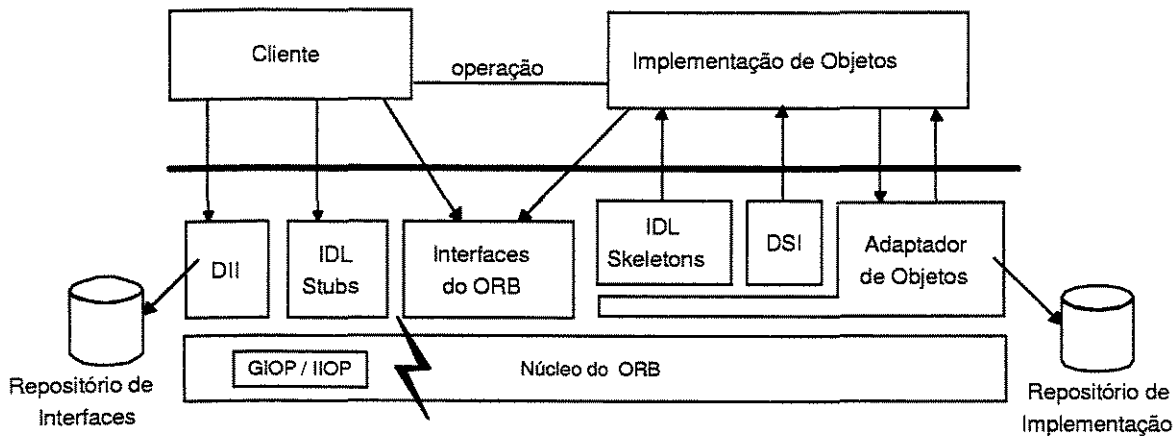


Figura 2.2: A Estrutura de CORBA

O núcleo do ORB é responsável pelos mecanismos de suporte a chamadas de métodos entre objetos distribuídos de forma transparente. Ele recebe as requisições dos clientes através das interfaces estáticas (*IDL Stubs*) ou dinâmicas (*DII*), localiza a implementação de objetos apropriada, transmite os parâmetros invocando o método correto através da interface estática (*IDL Skeletons*) ou da Interface de Esqueleto Dinâmica (*DSI*) e retorna os resultados.

Os elementos da arquitetura CORBA podem ser observados na Figura 2.2. São eles:

- **Stubs.** Permitem a invocação de um cliente a um objeto remoto, usando a sintaxe natural de sua linguagem de programação. São interfaces de invocação estática.

- Interface de Invocação Dinâmica (IID). Permite que uma invocação seja construída por um cliente em tempo de execução com o auxílio do Repositório de Interfaces, especificando o objeto CORBA, a operação a ser executada e a lista de parâmetros.
- Interface ORB. Permite que funções não oferecidas por outras interfaces sejam acessadas diretamente por clientes e implementações de objetos.
- Esqueletos Estáticos. Similares aos Stubs do lado servidor.
- Esqueleto Dinâmico. Permite o manuseio de invocações dinâmicas e o desenvolvimento de “gateways” entre sistemas CORBA e outros que não obedçam ao padrão. Este mecanismo foi introduzido na especificação CORBA 2.0 para permitir a entrega de chamadas de métodos a interfaces que não tem implementações de objetos registradas no Repositório de Implementações no momento.
- Adaptador de Objetos - Fornece um ambiente completo para a execução de um objeto servidor. Alguns serviços oferecidos por um adaptador de objetos são: geração e gerenciamento de referências de objetos; instanciação de objetos em tempo de execução; manipulação de invocações de clientes; registro de implementações no Repositório de Implementações.
- Repositório de Interfaces - Armazena informações IDL de interfaces de objetos, dispondo-as em tempo de execução. Assim, um objeto cliente pode utilizar, de forma dinâmica, objetos cujas interfaces não eram conhecidas em tempo de compilação.
- Repositório de Implementação - Provê informações que permitem o ORB localizar e ativar implementações de objetos.

2.3.2 Implementações de CORBA

Existem diversas implementações de CORBA disponíveis no mercado. Podemos citar: *Orbix* e *OrbixWeb* da *Iona Technologies* [ION], *VisiBroker* da *Visigenic* [Bor], *Java IDL* da *JavaSoft* [Jav], *ObjectBroker* e *Iceberg* da *BEA* [BEA], *Component Broker* da *IBM* [IBM] e implementações *free* como *MICO* [MIC] e o *Jacorb* [jac].

O padrão CORBA não trata de questões relativas a sua implementação. Na prática, as implementações podem vir na forma de bibliotecas *runtime*, processos *daemons*, uma máquina servidora ou componentes do sistema operacional.

Para a implementação deste trabalho foi escolhido o *OrbixWeb*, um ORB com mapeamento para Java, bastante sólido. Ele é implementada por meio de um par de bibliotecas para os lados cliente e servidor, além de um *daemon* de ativação, que deve estar presente em nós da rede rodando servidores. No capítulo 6 apresentaremos maiores detalhes.

2.4 Considerações finais sobre CORBA

CORBA vem continuamente se desenvolvendo. Em 1995, com o lançamento da versão 2.0, CORBA passou a apresentar características como:

- suporte a invocações dinâmicas do lado servidor (DSI);
- suporte para a interoperabilidade entre ORBs com a definição do protocolo IIOP³, uma implementação para TCP/IP do protocolo genérico de interoperabilidade GIOP⁴.

A nova versão 3.0 marcará a próxima geração de tecnologia ORB. CORBA 3.0 apresentará, entre outros:

- suporte a *firewall* com um *proxy* IIOP;
- suporte ao GIOP com conexões bidirecionais imprescindíveis para *callbacks* de notificações de eventos em ambientes tratados com *firewall*;
- Mensagens Assíncronas com Controle da Qualidade de Serviço;
- especificação de CORBA Mínimo, Tolerante a Falhas e de Tempo Real;
- definição de uma Arquitetura para componentes CORBA (CORBAcomponents);

³Internet Inter-ORB Protocol

⁴General Inter-ORB Protocol

Capítulo 3

Gerenciamento de Sistemas Distribuídos

Este capítulo discute os problemas e requisitos envolvidos na área de gerenciamento de sistemas distribuídos, analisando-se também CORBA como alternativa para a infra-estrutura de gerenciamento. Será discutida a situação atual dos trabalhos de desenvolvimento de um serviço de gerenciamento para CORBA. Finalmente, será apresentado o projeto MAScOTTE que teve grande influência nestes trabalhos dentro do OMG.

3.1 Aspectos Gerais do Problema de Gerenciamento

A tecnologia de objetos distribuídos tem se estabelecido como a solução *middle-ware* para integração de aplicações e processamento distribuído em ambientes heterogêneos. Isto tem feito da tecnologia CORBA o padrão “*de facto*” para a plataforma distribuída orientada a objetos.

CORBA também pode se estabelecer como a plataforma integradora para as atividades de gerenciamento, tanto em nível de gerenciamento de redes quanto de gerenciamento de sistemas distribuídos.

As atividades de gerenciamento de redes se baseiam no modelo gerente/agente (Figura 3.1). Os recursos (uma aplicação, um computador, um roteador, etc.), para serem gerenciados, são representados como um conjunto de atributos formando os *objetos gerenciados* que são organizados coletivamente na MIB. Um gerente, para exercer sua atividade de gerenciamento precisa ler e modificar estes atributos. Para tanto, ele se comunica com um agente, através de protocolos de gerenciamento como CMIP e SNMP, requisitando e transmitindo informações

de gerenciamento. Uma entidade gerenciada também pode notificar um gerente da ocorrência de um evento ou mudança de estado (como em caso de falha) através do agente, emitindo um evento de notificação de forma assíncrona. Neste modelo, um objeto gerenciado só pode ser acessado por um gerente através de um agente e a comunicação entre a entidade gerenciada real e o agente se faz de forma proprietária.

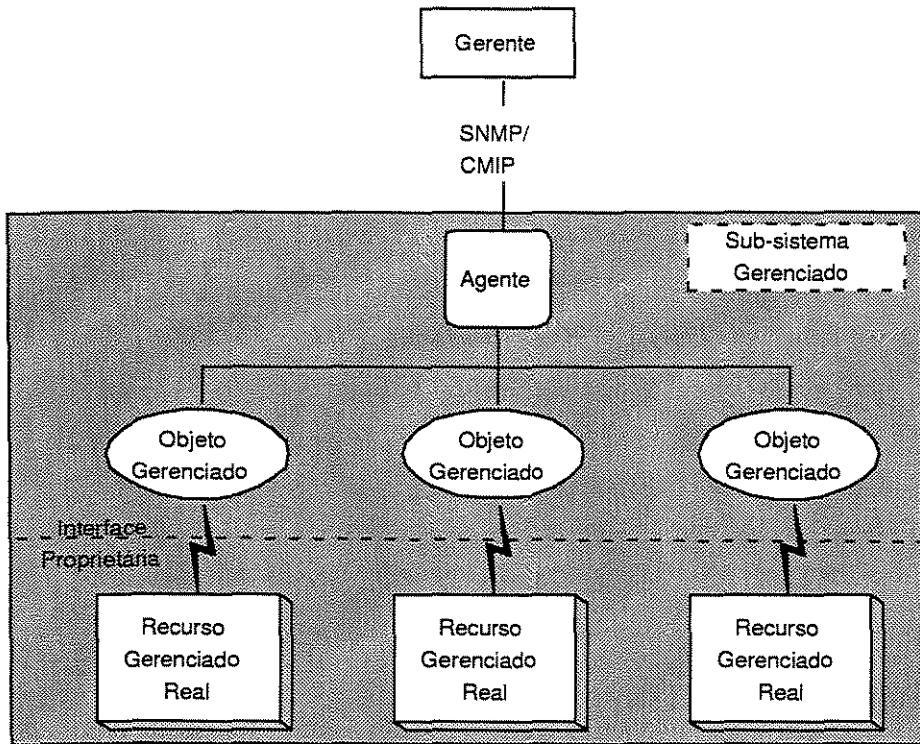


Figura 3.1: Modelo de Gerenciamento Gerente/Agente

O gerenciamento de rede foi desenvolvido através dos protocolos de gerenciamento OSI/CMIP e TCP-IP/SNMP. Cada protocolo define um conjunto limitado de mensagens que podem ser passadas entre gerentes e agentes para a condução das informações e operações de gerenciamento. Este modelo, utilizando CORBA como veículo de transporte das informações de gerenciamento, pode ser implementado como ilustrado na Figura 3.2, que ilustra os trabalhos sendo desenvolvidos no projeto *MAScOTTE* [MaS97].

Esta implementação usando CORBA, além de trazer os benefícios inerentes à plataforma como flexibilidade, facilidade de configuração, independência de transporte e suporte dos serviços de objetos, permite ainda a interoperabilidade com os demais padrões de gerenciamento através de *gateways*.

No entanto, modelos de gerenciamento como SNMP e CMIP não atendem

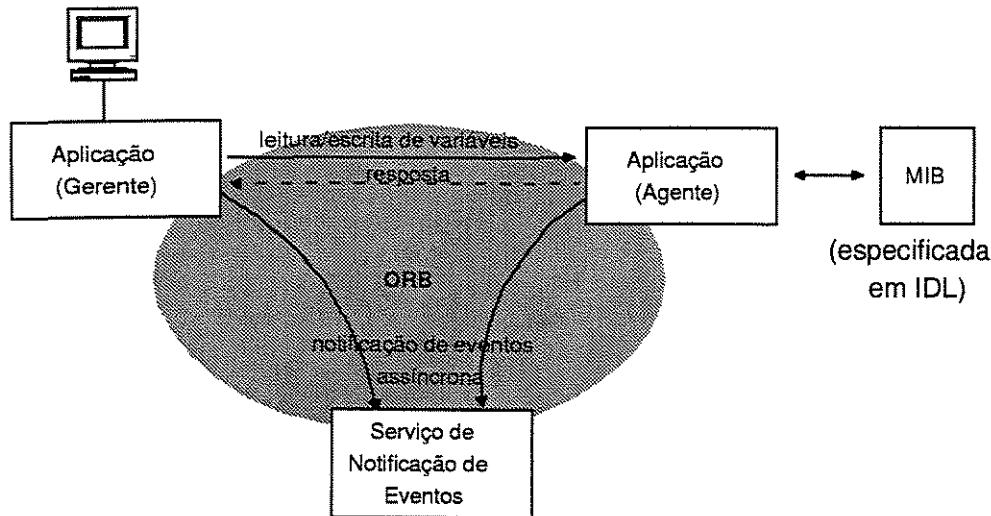


Figura 3.2: Interações de Gerenciamento com base em ORB (Projeto MAS-cOTTE)

aos requisitos para a gerência dos sistemas distribuídos [Slo95]. Eles enfatizam a separação entre gerenciamento de serviço e a funcionalidade normal fornecida por um serviço. Os mesmos conceitos e técnicas utilizados para se projetar uma aplicação distribuída devem também ser usados para o seu gerenciamento [Slo95]. Assim, o suporte para o gerenciamento da aplicação deve ser projetado ao mesmo tempo que sua funcionalidade normal. Isto, na plataforma CORBA, se traduziria em objetos que teriam parte de sua interface funcional dedicada ao gerenciamento, ou teriam um componente com sua interface dedicada ao seu gerenciamento.

Em CORBA, portanto, o caminho é adaptar os conceitos aplicados no gerenciamento de redes para o mundo dos recursos baseados em CORBA e desenvolver outros conceitos tais que estes cubram a necessidade de gerenciamento desse ambiente tanto em nível de suporte (o ORB, os serviços de objetos e facilidades comuns) quanto no nível das aplicações distribuídas.

Os recursos em um ambiente de computação distribuída podem ser classificados nas seguintes categorias:

- **Redes:** dispositivos que fornecem serviços de comunicação tais como comutadores, roteadores, pontes e hubs.
- **Sistema:** recursos do sistema operacional tais como espaço em disco e de *swap*, memória, CPU, usuários e impressoras.
- **'Middleware':** uma infra-estrutura de software construída sobre o sistema operacional e abaixo (servindo) das aplicações tal como um DBMS¹

¹Data Base Management System

que fornece serviços de armazenamento de dados, ou um ORB que fornece transparência de distribuição para as aplicações.

- **Aplicação:** a aplicação formada por todos os seus componentes.

As atividades de gerenciamento podem ser vistas a partir de cada uma destas camadas como é mostrado na Figura 3.3. Esta estrutura de camadas implicitamente define relacionamentos entre os serviços fornecidos por camadas distintas de forma que as atividades de gerenciamento não podem ser realizadas independentemente uma da outra. Por exemplo, uma falha em um componente de aplicação tentando se comunicar com um componente remoto pode ser relacionada com uma falha no Serviço de Nomes (camada *Middleware*) ou uma quebra de links entre nós.

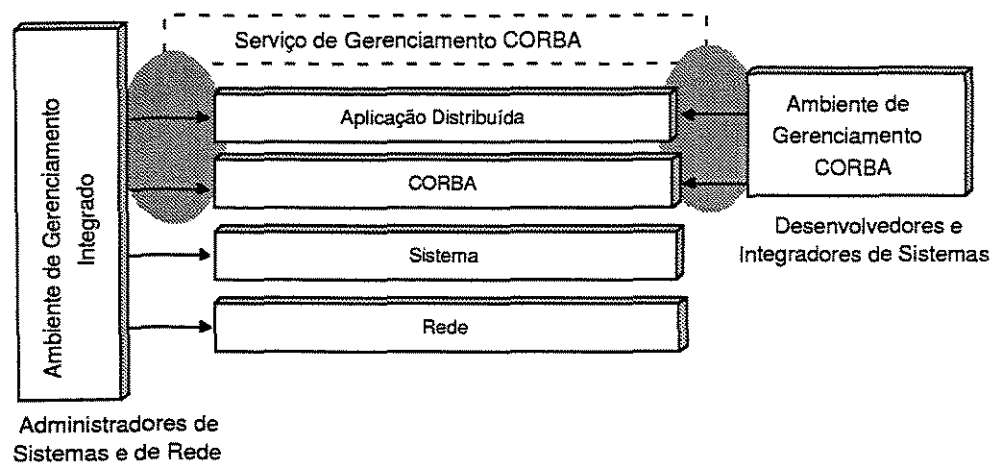


Figura 3.3: Integração da Tecnologia de Gerenciamento

O gerenciamento de aplicações, portanto, necessita de um ambiente de gerenciamento *Integrado*, possuindo uma visão sobre todos os recursos em um ambiente de computação distribuída e conhecendo os relacionamentos de dependência entre os serviços fornecidos por diferentes camadas (Figura 3.3).

Como mencionado, a maioria dos conceitos de gerência de sistemas foram desenvolvidos para o gerenciamento de redes. Muito trabalho tem sido feito nos últimos anos para adaptar e estender estes conceitos para o âmbito de gerenciamento de sistemas distribuídos.

Uma característica importante do modelo de gerenciamento de redes OSI é a sua classificação das atividades de gerenciamento em cinco áreas funcionais. Meyer ([MP95]), de acordo com esta classificação, propõe que o gerenciamento de aplicações possua:

Gerenciamento de Contabilização:

- administração do método de pagamento por uso de serviços;
- administração do método de cálculo de custos do uso de serviços;

Gerenciamento de Configuração:

- controle do estado de um componente da aplicação;
- armazenamento das propriedades dos componentes da aplicação;
- inicialização e término dos componentes da aplicação;
- modificação de configuração;
- administração da migração dos componentes da aplicação;

Gerenciamento de falhas:

- administração de *checkpoints*;
- uso de serviços tolerantes a falhas;
- coleta de mensagens de falhas;
- reconhecimento da causa de mensagens de falhas;

Gerenciamento de desempenho:

- monitorização das características de uma interface, por exemplo, número de operações, invocações, volume de dados transferidos e tempo de resposta;
- monitorização de características de performance de um componente, por exemplo, *throughput* e retardo total;
- determinação de gargalos de desempenho;
- evitar ou contornar gargalos;
- monitorar e controlar a reserva de recursos da rede e do sistema;
- balanceamento de carga;

Gerenciamento de Segurança:

- administração de autenticação de acessos a múltiplos servidores;

O modelo RM-ODP também define cinco áreas funcionais de gerenciamento, embora não exista uma correspondência direta com o modelo OSI: configuração, contabilização, qualidade de serviço, monitorização e definição de políticas de gerência. A gerência de qualidade de serviço pode ser vista como uma generalização de desempenho e falhas, e as funções de definição de políticas de gerência deverão incluir funções de segurança ([Rop99]).

Hegering, em [HA94], mostrou que as aplicações e atividades de gerência poderiam ser analisadas e agrupadas em cinco diferentes dimensões ortogonais, ilustradas nos eixos da Figura 3.4. 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.

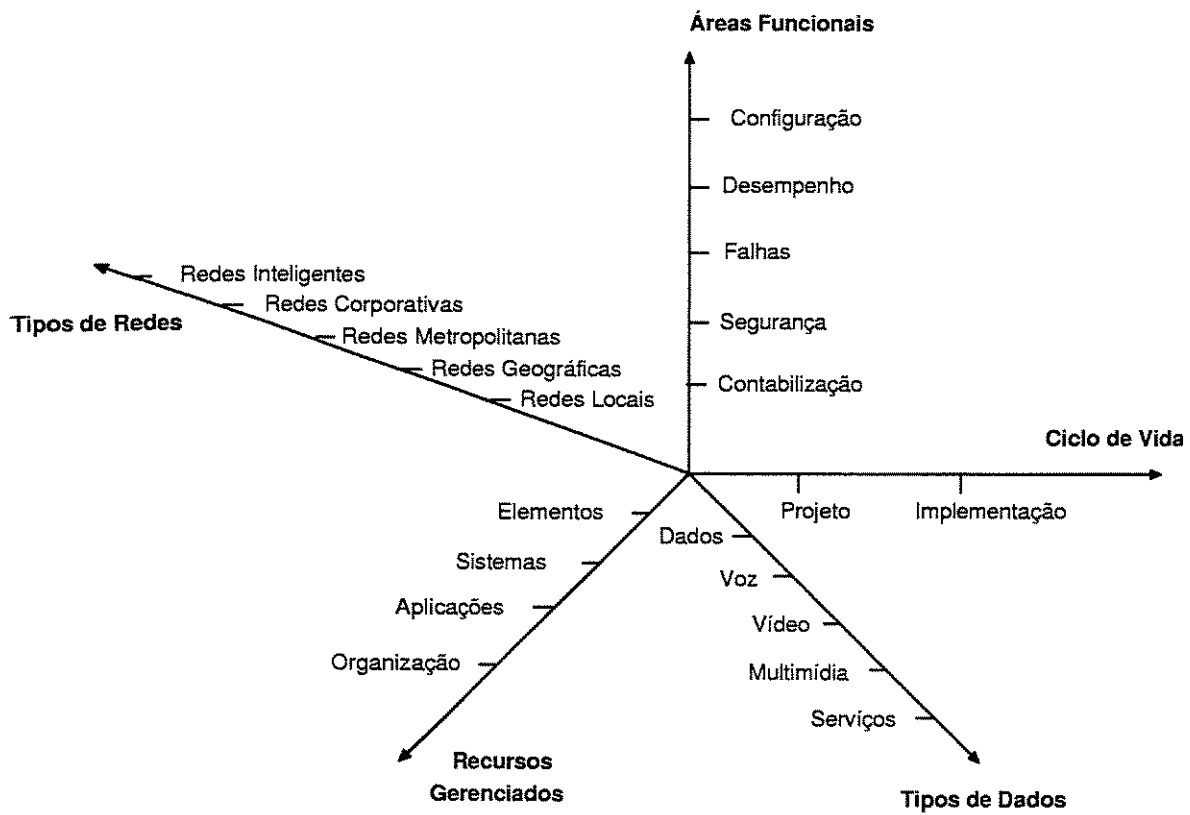


Figura 3.4: Dimensões de Gerência

3.2 Requerimentos para o Gerenciamento de Sistemas Distribuídos

No desenvolvimento de serviços de gerenciamento para um ambiente distribuído como CORBA é preciso conhecer os requerimentos determinados pelos diversos usuários presentes no ambiente.

Através do ponto de vista destes usuários podemos identificar atores e papéis que ora impactam, ora são impactados pelas atividades de gerenciamento.

Atores que impactam o gerenciamento do sistema:

- os *operadores* controlam e monitoram os recursos gerenciados executando aplicações de gerenciamento;
- os *desenvolvedores* de aplicações de gerenciamento e aplicações dos *usuários finais*;
- os *reguladores* que definem e aplicam políticas de gerenciamento.

Atores impactados pelo gerenciamento do sistema:

- os *usuários finais* das aplicações que estão sendo gerenciadas;
- os *fornecedores* que procuram gerenciar as aplicações e sua infra-estrutura para alcançar seus objetivos de negócio.

Combinando os requerimentos, eles podem ser agrupados em:

- Requerimentos de Negócio (requerimentos de *fornecedores* e *usuários finais*);
- Requerimentos do Gerenciamento de Aplicações (requerimentos de *desenvolvedores*, *operadores* e *reguladores*);
- Requerimentos para o Gerenciamento do Ambiente de Suporte (Ex.: CORBA);
- Requerimentos não Funcionais.

3.2.1 Requerimentos de Negócio

Os Requerimentos de Negócio procuram determinar a qualidade dos serviços prestados a um *usuário final*, visto que o gerenciamento tem um impacto limitado na funcionalidade dos serviços prestados.

Com relação ao *fornecedor*, seus principais benefícios seriam a maximização dos ganhos e a satisfação dos *usuários finais*.

Com relação aos *usuários finais*, seus requerimentos estão relacionados com a disponibilidade dos serviços e não com a forma como esta disponibilidade é alcançada. Eles desejam executar as suas atividades com tempos de resposta adequados, com o mínimo custo e o mínimo de tempo de espera em caso de falhas.

3.2.2 Requerimentos do Gerenciamento de Aplicações

Estes requerimentos são determinados pelos *operadores*, *reguladores* e pelos *desenvolvedores* das aplicações.

O *regulador* é responsável por definir e aplicar políticas de gerenciamento. Para isso ele precisa de utilitários que o permitam agrupar recursos e atividades de gerenciamento em regiões (ou domínios) associadas a políticas de gerenciamento e coletar dados estatísticos a fim de definir políticas de gerenciamento.

O *operador* controla e monitora os recursos gerenciados a fim de garantir a entrega dos serviços aos *usuários finais*. O *operador* precisa receber informações e reagir rapidamente. Ele necessita, por exemplo, de utilitários para detecção de falhas e mudança de configuração dos recursos gerenciados.

O *desenvolvedor* implementa as aplicações distribuídas. Ele é responsável por integrar as informações de gerenciamento na aplicação durante o processo de desenvolvimento. Este *overhead* requerido pelo processo de integração das informações de gerenciamento deve ser o menor possível. Para tanto, ele precisará de utilitários e blocos de construção disponíveis para tornar sua aplicação uma aplicação gerenciável.

3.2.3 Requerimentos para o Gerenciamento do Ambiente de Suporte

Os requerimentos até aqui mencionados são identificados por um processo orientado ao usuário. No entanto, os Requerimentos para o Gerenciamento do Ambiente de Suporte são determinados especificamente para a tecnologia da plataforma distribuída em questão.

O projeto MAScOTTE ([MaS97]), discutido na seção 3.5, identificou requerimentos para a plataforma CORBA. Ele identificou na arquitetura da plataforma CORBA as seguintes entidades sujeitas ao gerenciamento:

- os clientes;
- os servidores;
- as implementação de objetos;

- núcleo do ORB;
- os adaptadores de objetos;
- conexões.

Os requerimentos aqui devem ser analisados e agrupados segundo o ponto de vista funcional cobrindo as áreas funcionais definidas no modelo de gerenciamento OSI.

3.2.4 Requerimentos não Funcionais

Os Requerimentos não Funcionais estão relacionados com aspectos de qualidade e restrições de projeto do sistema de gerenciamento sendo especificado. Eles estão relacionados com “*como*” o sistema fornece sua funcionalidade ao invés de “*qual*” funcionalidade é suportada. Eles se preocupam, por exemplo, com o número máximo de recursos do ambiente sujeitos ao gerenciamento ou com o custo máximo de performance que a adição da funcionalidade de gerenciamento irá impor à aplicação sendo gerenciada.

3.3 CORBA como Ambiente de Gerenciamento

Como já apresentado na seção 3.1, podemos aplicar CORBA como plataforma integradora para as atividades de gerenciamento, tanto em nível de gerenciamento de redes quanto de gerenciamento de sistemas distribuídos.

Também podemos avaliar CORBA como plataforma de gerenciamento comparando seu comportamento e atributos com os característicos dos modelos de gerenciamento de redes:

- As iterações e relacionamentos entre objetos CORBA são diretos, par a par. Isto estende a estrutura organizacional presente nos modelos de gerenciamento de redes permitindo não somente os relacionamentos gerente/agente e gerente/gerente, mas também, relacionamentos agente/agente e gerente/objeto-gerenciado. Isto permite novos níveis de cooperação e de distribuição da funcionalidade de gerência.
- CORBA não necessita de um protocolo de gerência específico pois sua estrutura de comunicação é o próprio ORB. O ORB se encarrega, de forma transparente, da localização, ativação, invocação da operação no objeto remoto e entrega das respostas.

- CORBA possui a sua própria linguagem de descrição de interfaces, a IDL, aplicável ao gerenciamento. IDL, assim como a GDMO do modelo OSI, enfatiza a separação entre declaração de interface e a implementação, com a vantagem de ser uma solução muito mais simples e dominada por qualquer desenvolvedor CORBA.
- Além do ORB, toda a infra-estrutura e funcionalidade da OMA estará disponível para o desenvolvimento das atividades de gerenciamento.

Podemos ainda enumerar, resumidamente, motivações para o uso de CORBA como plataforma de gerenciamento:

- a necessidade de se substituir o protocolo SNMP, criado em uma época em que as especificações de gerenciamento eram restritas;
- CORBA permite que os mesmos conceitos usados para especificar, projetar e implementar a aplicação sejam também utilizados para o gerenciamento;
- permite que os programadores não mudem o seu estilo original de programação, usando a sua linguagem favorita e conceitos familiares;
- objetos gerenciados podem chamar diretamente a estação de gerenciamento, quando necessitam reportar um evento significativo;
- a implementação de uma MIB, usando CORBA, torna-a aberta, permitindo que seja, potencialmente, acessada por qualquer cliente CORBA. Neste caso, porém, deve-se tratar do problema de segurança.

[HS98] e [Pav99] apresentam uma análise interessante das vantagens do uso de CORBA em gerenciamento.

3.4 Especificação de um Serviço de Gerenciamento para CORBA

A demanda por uma capacidade de gerenciamento, trouxe para os ORBs comerciais interfaces de gerenciamento proprietárias geralmente dedicadas à configuração do ORB e dos Serviços de Objetos, sendo sua funcionalidade dependente do produto. Poucos oferecem a funcionalidade completa necessária aos *Integradores de Sistemas* e aos *Desenvolvedores* que requerem capacidades de instrumentação para o suporte da configuração do sistema, análise de performance, detecção de

falhas, etc; e aos *Administradores de Rede e Sistemas*, que requerem soluções de gerenciamento integradas para um suporte completo e uniforme (Figura 3.3).

Uma implementação CORBA poderia incorporar um agente proprietário CMIP e um SNMP para prover as capacidades de gerenciamento através dos produtos já utilizados em uma empresa. Embora esta integração seja desejável, tal caminho requer esforço adicional para desenvolvimento e manutenção (protocolos adicionais e interfaces) e não suporta *Desenvolvedores e Integradores de Sistema*.

Os trabalhos para desenvolvimento e padronização dos serviços de gerenciamento no OMG têm sido reacuecidos e tomado novos rumos.

Atualmente, com o fim dos trabalhos do “*Common Facilities Task Force*”, os problemas de gerenciamento passaram a ser novamente discutidos no “*ORB and Object Services Task Force*”(ORBOS TF) . Este novo grupo de trabalho passou a dar uma nova abordagem para os problemas de gerenciamento e o principal resultado disto foi a retirada da especificação XCMF, em meados de 1999, como especificação padrão para a Facilidade Comum de Gerenciamento de Sistemas.

O ORBOS TF identificou a necessidade das plataformas CORBA disponibilizarem informações padronizadas para gerenciamento, depuração e controle destes ambientes. Sua nova abordagem almeja um gerenciamento uniforme de CORBA e das aplicações CORBA distribuídas, sendo que, nesta visão, *CORBA Components* será o *framework* para as futuras aplicações CORBA. Para isto, foi planejado um desenvolvimento das facilidades de gerenciamento em fases. A primeira fase lidará com os aspectos de gerenciamento relacionados com o ORB e os Serviços de Objetos (*Instrumentação e Controle*). A segunda fase irá focar o Gerenciamento de Aplicações, podendo ser implementada de forma independente. Fases adicionais ainda poderão ser definidas.

A primeira fase dos trabalhos editou o RFP intitulado “*Distributed Instrumentation and Control For ORBs and Services*” [OMG00], tratando o problema geral de Instrumentação dos ORBs e Serviços CORBA, considerando o uso desta instrumentação para gerenciamento (configuração e controle), depuração e perfilamento de implementações CORBA heterogêneas. O RFP não endereça ainda o problema de gerenciamento dos ORBs e Serviços CORBA. Ele é focado no fornecimento das informações que irão permitir este gerenciamento. O RFP espera pela definição das diversas interfaces de objetos e serviços que irão ser invocadas pelos utilitários de gerenciamento, depuração e perfilamento. Nesta fase dos trabalhos, o projeto MAScOTTE, comentado na próxima seção, tem surgido como referência importante para os trabalhos do grupo.

Com relação a retirada da especificação XCMF, analisada no próximo Capítulo, é importante mencionar que ela foi derivada de um padrão X/Open [XOp97] desenvolvido num tempo (1997) onde existia somente CORBA e os Serviços de

Objetos originais (Nomes, Ciclo de Vida, Eventos). XCMF foi projetado para dar suporte aos requerimentos básicos de ciclo de vida dos objetos CORBA, de forma que os desenvolvedores de aplicações poderiam estender objetos XCMF básicos com sua lógica de negócio e ter uma aplicação gerenciável CORBA. De certo modo, XCMF estava ligeiramente à frente de seu tempo. Por não haver muitos serviços de infra-estrutura definidos pelo OMG em seu tempo, XCMF “inventou” muitos conceitos que agora têm sido mais completamente formalizados em especificações de Serviços.

Em retrospecto, a principal idéia por trás de XCMF é muito semelhante àquelas por trás dos objetivos de padrões atuais como os *CORBA Components*. Na verdade, XCMF tentou empregar soluções um tanto cruas diante dos padrões atuais.

Com a retirada da XCMF, os problemas relativos ao gerenciamento de aplicações deverão ser novamente discutidos na segunda fase prevista dos trabalhos de gerenciamento. No entanto, a forte tendência para o gerenciamento de *building blocs* (Java beans, Corba Components, DCOM) deverá ser mantida. As idéias da XCMF deverão contribuir para a construção de uma estrutura de gerenciamento de aplicações com base nos *CORBA Components*.

3.5 O Projeto MAScOTTE

As discussões no OMG têm mencionado, frequentemente, o projeto MAScOTTE [MaS97] como abordagem e solução a ser analisada para os problemas de gerenciamento. Este projeto propõe uma solução de gerenciamento básica para os recursos da plataforma CORBA.

A solução implementada no projeto consiste de um conjunto de Facilidades de Gerenciamento CORBA, cobrindo 4 das 5 áreas funcionais do modelo de gerenciamento OSI (a área funcional de segurança não foi abordada). Estas Facilidades são usadas tanto por aplicações de gerenciamento CORBA quanto por utilitários de gerenciamento externos baseados em outras arquiteturas como a OSI/CMIP. As Facilidades coletam dados dos recursos OMA sendo gerenciados (por meio de extensões de gerenciamento que instrumentam estes recursos), elaboram estes dados e disponibilizam informações de gerenciamento. No caso de aplicações de gerenciamento externas, estas informações são acessadas através de um *gateway* CMIP-to-CORBA.

As principais Facilidades identificadas e implementadas no projeto são:

- **Facilidade de Agente CORBA:** dedicada ao gerenciamento da infra-estrutura de CORBA;

- **Facilidade de Notificação:** suporte para atividades de monitorização com capacidade de filtragem e *logging*;
- **Facilidade de Definição de Aplicação:** dedicada a oferecer uma visão gerencial de uma aplicação completa.

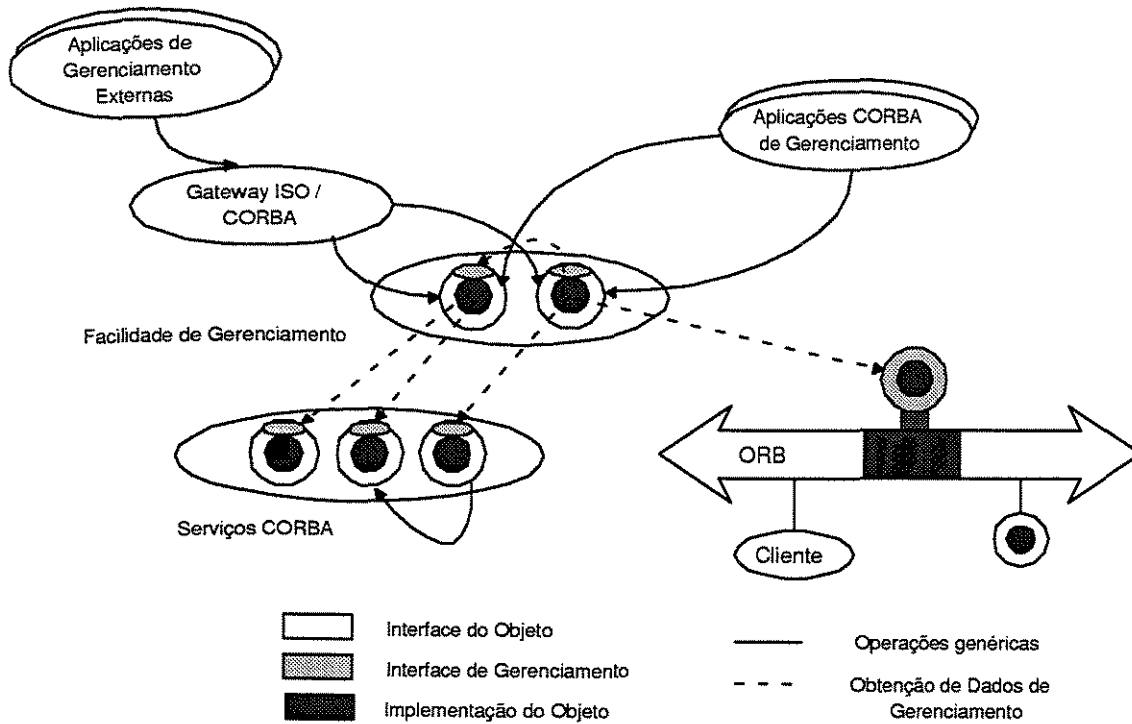


Figura 3.5: Invocação das Operações de Gerenciamento nos Objetos Gerenciados.

As Facilidades de Agente e Notificação fornecem uma interface de baixo nível para o gerenciamento da infra-estrutura CORBA e para objetos gerenciados genéricos. É pretendido que sejam usadas por Facilidades de alto nível ou diretamente por aplicações tais como *browsers* para a MIB. A Facilidade de Agente CORBA fornece acesso uniforme para os objetos gerenciados CORBA e adiciona informações estruturais de uma forma independente do produto CORBA subjacente.

Em um nível mais alto, a Facilidade de Definição de Aplicação permite a definição e configuração de objetos como um conjunto significativo.

A Figura 3.2 apresenta o modelo estrutural adotado no projeto MAScOTTE, e a Figura 3.5 o mecanismo desenvolvido para disponibilização das informações de gerenciamento.

Capítulo 4

Gerenciamento de Configuração

Este capítulo discute o problema de Gerenciamento de Configuração para sistemas distribuídos, finalizando com uma apresentação da especificação XCMF, adotada neste trabalho como infra-estrutura de suporte no desenvolvimento da Facilidade de Gerenciamento de Configuração.

4.1 Gerenciamento de Configuração

Gerenciamento de Configuração, na visão do modelo OSI de gerenciamento, está associado com a manipulação do estado dos dispositivos na rede como, por exemplo, na atribuição de tamanhos de *buffers* em pontes ou o estabelecimento dos dispositivos “on-line”. O Gerenciamento de Configuração pode ser aplicado tanto para entidades de software quanto de hardware. Configurar tais entidades geralmente significa determinar e atribuir parâmetros e valores ao estado de um objeto que afetam seu comportamento. Esta visão de Gerenciamento de Configuração foi desenvolvida para o contexto de gerenciamento de redes onde se pretendia suportar *hardwares* heterogêneos e objetos de *softwares* relativamente simples.

Como observado por [Fos97], tais dispositivos e componentes são assumidos de serem parte do sistema por um longo período de tempo. O gerenciamento de configuração é aplicável após a entidade gerenciável ter sido instalada na rede, como por exemplo, a conexão de uma nova impressora ou a instalação de um novo servidor de nomes. O Gerenciamento de Configuração, nesse sentido, considera estas atividades individualmente e não se preocupa com o problema da criação (ou inclusão) e destruição (ou remoção) de serviços de software.

O Gerenciamento de Configuração no contexto de aplicações distribuídas, no entanto, possui aspectos distintos. Neste caso, a preocupação fundamental

é com as operações de Ciclo de Vida dos componentes da aplicação. Torna-se necessária a definição do processo pelo qual objetos de software são criados, conectados ao sistema e gerenciados. Tais operações devem permitir que as aplicações sejam gerenciadas dinamicamente e alteradas incrementalmente, devido, muitas vezes, à impossibilidade de tais aplicações serem interrompidas para sua necessária manutenção (por exemplo, sistemas bancários).

Aplicações distribuídas possuem uma estrutura que é resultado dos relacionamentos entre os componentes da aplicação. Esta estrutura é frequentemente identificada com a Configuração da aplicação. Ela deve estar ao alcance das aplicações de gerenciamento de configuração para uma possível reconfiguração, ou seja, mudança na estrutura da aplicação em tempo de execução.

4.2 Definindo um Serviço de Configuração para Sistemas Distribuídos

Um serviço de Configuração deve prover mecanismos para o estabelecimento da configuração inicial (configuração *Estática*) de um sistema distribuído e para o acompanhamento das mudanças, executando modificações quando necessárias. Tais modificações são necessárias para acomodar mudanças *Operacionais* e *Evolucionárias*.

Mudanças *Operacionais* se preocupam com a reorganização do sistema como resultado de um redimensionamento, balanço de cargas ou para a recuperação de falhas. Os requerimentos para estas mudanças são conhecidos em tempo de análise e projeto e podem ser programados no sistema. Mudanças *Evolucionárias* ocorrem em função de mudanças nos requerimentos do sistema ou na tecnologia usada para implementá-lo. Tais mudanças alteram a funcionalidade do sistemas e são requeridas em tempo de operação do sistema. ([Mag94]).

O problema é que, muitas vezes, uma aplicação distribuída não pode ser paralisada totalmente para a realização destas mudanças em partes da mesma. Conseqüentemente, tais mudanças devem ser efetuadas dinamicamente com a aplicação em execução (*Reconfiguração Dinâmica*).

Estes conceitos estão relacionados com os dois tipos de reconfiguração, identificados por [Endl92], necessários para a maioria das aplicações:

- reconfigurações “ad-hoc”: são reconfigurações projetadas (mudanças evolucionárias) com o sistema em operação e são usualmente realizadas interativamente por meio de um gerenciador de reconfiguração;

- reconfigurações “programadas”: são reconfigurações identificadas e programadas no sistema em tempo de análise e projeto.

Como identificado por [Endl92], problemas surgem quando os dois tipos de reconfiguração são suportados juntos. A execução concorrente de reconfigurações *programadas* e *ad-hoc* pode levar os sistemas a um estado inconsistente. A execução destes tipos de mudanças devem ser coordenadas garantindo que mudanças programadas sejam executadas atomicamente e somente quando nenhuma mudança *ad-hoc* esteja sendo realizada.

Mecanismos e ferramentas devem ser fornecidos ao projetista da aplicação ou sistema distribuído para que o mesmo possa antever e estabelecer os pontos configuráveis do sistema. Deste modo, a tarefa de Gerenciamento de Configuração de uma aplicação distribuída se inicia em tempo de projeto e desenvolvimento da aplicação.

Além dos mecanismos e ferramentas oferecidos ao projetista, recursos também devem estar à disposição do gerente de uma aplicação distribuída para o estabelecimento das mudanças *Evolucionárias*. Ao gerente devem estar disponíveis recursos para substituição, remoção, adição e migração de componentes de um sistema distribuído.

Ao mesmo tempo, devemos garantir que tais operações mantêm a consistência do sistema modificado. Por exemplo, quando um servidor é migrado de um nó para outro, as possíveis referências a este servidor existentes no ambiente devem continuar se mantendo válidas. Portanto, mecanismos para o estabelecimento, restabelecimento e remoção de associações (“*bindings*”) entre os componentes da aplicação distribuída também devem ser providos.

No entanto, é improvável que esta consistência possa ser mantida sem um suporte desenvolvido na própria aplicação. Como abordado por [Mag94], a capacidade de evoluir deve ser implantada dentro do sistema. Ela não pode simplesmente ser adicionada como uma facilidade de gerenciamento. Como exemplo, considere a substituição da implementação de um servidor de impressão. Não seria desejável fazer a substituição no meio de uma impressão. Seria mais satisfatório que o servidor indicasse para o sistema de gerenciamento que ele não está imprimindo nada e que pode ser reconfigurado seguramente, sem perda de dados.

Isto nos sugere uma outra funcionalidade provida pelo Serviço de Configuração: A capacidade de acompanhar o estado dos componentes do sistema. Isto poderia nos levar à definição de atributos que seriam incorporados à definição dos componentes do sistema.

4.3 A Especificação XCMF

Esta seção apresenta uma breve descrição dos conceitos envolvidos na especificação XCMF. Detalhes completos podem ser encontrados no documento [XOp97].

XCMF é baseada no Modelo de Referência para Gerenciamento de Sistemas da X/Open. Ela foi criada como infra-estrutura para desenvolvimento de aplicações distribuídas gerenciáveis.

Os serviços da especificação XCMF formam um *framework* projetado para suportar aplicações que gerenciam um grande número de objetos representando recursos. Usando os serviços XCMF, os projetistas das aplicações podem definir conjuntos de objetos (modelando *containers*) e associar políticas de inicialização ou validação para estes conjuntos. Estes objetos são gerenciados pelo tipo e gerados dinamicamente. As políticas associadas a um determinado tipo de instâncias podem ser modificadas dinamicamente pelo gerente.

As Facilidades de Gerenciamento que compõem a especificação são:

- A Facilidade de Gerenciamento de Conjuntos (“Managed Sets”);
- A Facilidade de Gerenciamento de Instâncias (“Instance Manager”);
- A Facilidade de Políticas.

O modelo de gerenciamento de ciclo de vida XCMF foi concebido em concordância com o modelo e as interfaces definidas na especificação OMG do Serviço de Ciclo de Vida [OMG97c].

A Figura 4.1 apresenta a estrutura hierárquica de todas as interfaces definidas na especificação XCMF. As interfaces destacadas em negrito pertencem ao Serviço de Ciclo de Vida.

O Serviço de Ciclo de Vida do OMG

O Serviço de Ciclo de Vida [OMG97c] define interfaces para criação, remoção, cópia e movimentação de objetos. O modelo de criação de objetos é definido em termos de objetos “Factory” onde um “Factory” é um objeto que cria outro objeto.

São três, as principais interfaces definidas neste serviço:

- **FactoryFinder**, que determina um mecanismo para a localização de objetos “Factory” no ambiente. Uma referência para um “*FactoryFinder*” é também passado como parâmetro para as operações “*copy*” e “*move*” de “*LifeCycleObject*”, onde é utilizada para a localização de um “*Factory*”

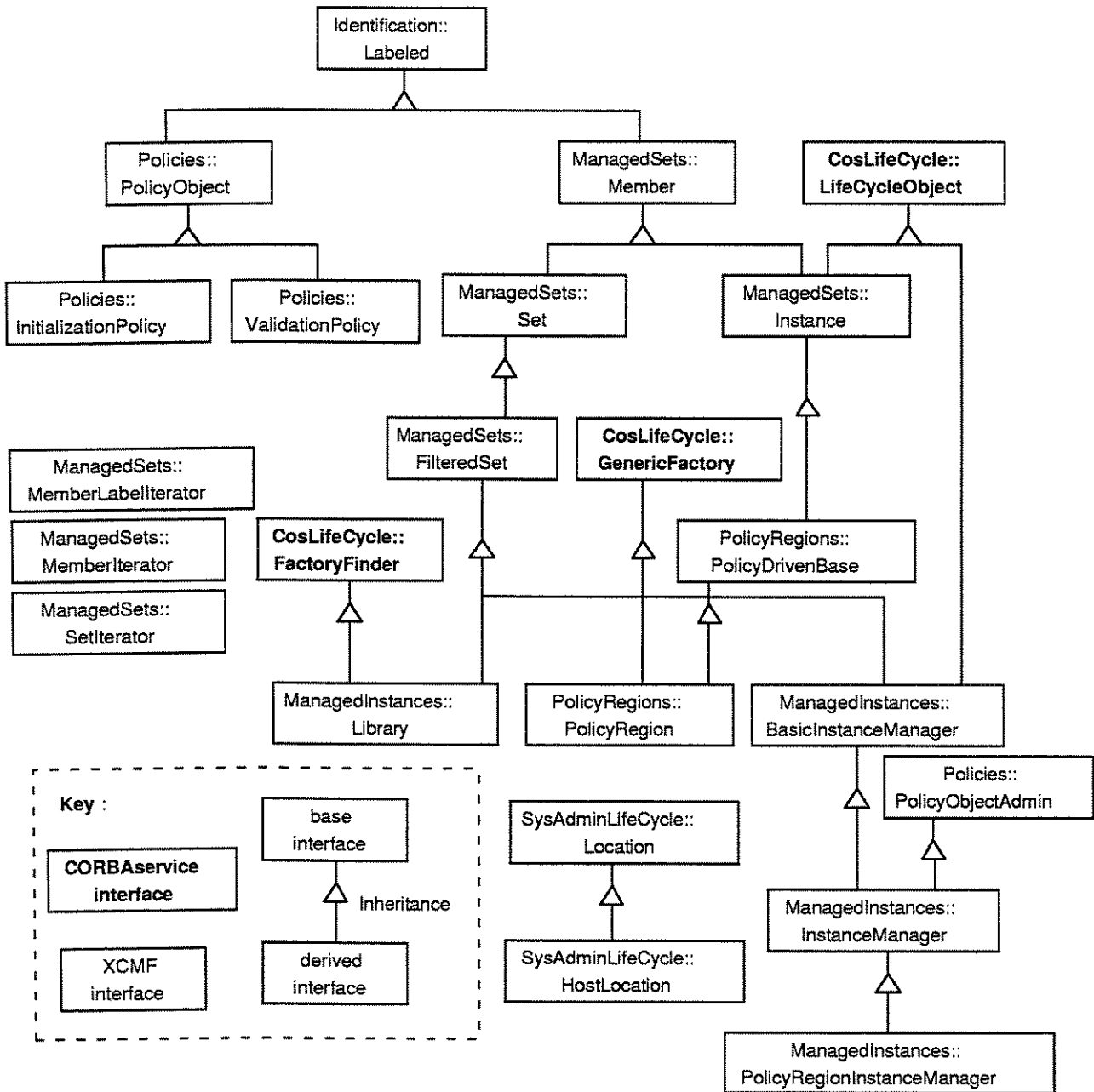


Figura 4.1: Interfaces definidas na especificação XCMF.

apropriado. Isto significa que a cópia ou a movimentação será feita para o domínio deste “*FactoryFinder*”.

- **GenericFactory** que representa um serviço genérico de criação. Enquanto não existe uma padronização para a interface do “*Factory*” (dependente de implementação) uma interface para um “*Factory*” genérico é definida.
- **LifeCycleObject** que representa um objeto capaz de sofrer operações de ciclo de vida. É importante notar que o próprio objeto implementa o procedimento para sua cópia, movimentação e remoção.

O Serviço de Ciclo de Vida pode ser melhor considerado como um serviço abstrato, com muitas possibilidades de implementação compatíveis mas com comportamentos completamente diferentes.

A Facilidade de Gerenciamento de Conjuntos

Esta Facilidade equivale ao conceito da Facilidade de Domínios presente na arquitetura de gerenciamento da plataforma Multiware descrita no próximo Capítulo. Este serviço define como principais interfaces: **Member**, **Set** e **FilteredSet**.

O serviço provê o conceito de conjuntos de objetos gerenciados, com a finalidade de organizar os objetos em grupos. As interfaces **Set** e **Member** fornecem a funcionalidade básica de conjuntos. A interface **Set** define as operações requeridas para um objeto manter referências para os objetos que o constituem. A interface **Member** define as operações que permitem um objeto manter referências para os conjuntos à que pertence. Este relacionamento é do tipo muitos-para-muitos.

Esta facilidade será usada para modelar o ambiente de gerenciamento em uma estrutura hierárquica de conjuntos (domínios) com a ilustrada na Figura 4.2.

A Facilidade de Gerenciamento de Instâncias

Fornece a infra-estrutura requerida para o gerenciamento de múltiplas instâncias de um determinado tipo. A Facilidade de Gerenciamento de Instâncias fornece as capacidades básicas para a criação e gerenciamento de todos os tipos de objetos gerenciados (Instâncias).

Em um ambiente OMG não há suporte explícito para a criação e gerenciamento de objetos. No entanto, para os projetistas de aplicações de gerenciamento é conveniente o uso de conceitos tradicionais orientados a objetos para gerenciar e criar objetos gerenciados.

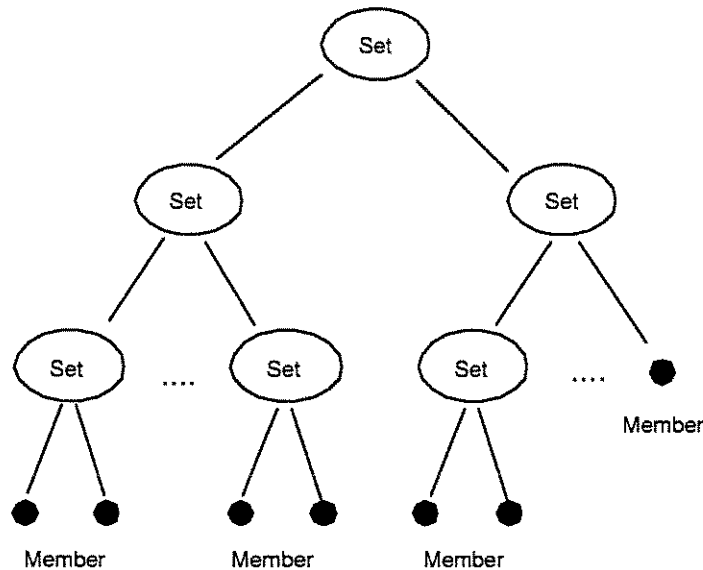


Figura 4.2: “Conjuntos Gerenciados”.

Esta Facilidade apresenta uma especificação sintática e semântica para um serviço de criação baseado na especificação do Serviço de Ciclo de Vida, além de um mecanismo para o gerenciamento destes objetos criados (Instâncias). Três papéis fundamentais são definidos na facilidade:

- *Instâncias*;
- *Gerenciador de Instâncias*, que é um *factory* para um tipo específico de *Instâncias* e um conjunto para estas *Instâncias*;
- *Biblioteca*, que é um *factory* de *Gerenciadores de Instâncias* e um conjunto para estes Gerenciadores.

Uma *Instância* é um tipo particular de um objeto gerenciado que é representado e gerenciado por um único *Gerenciador de Instâncias*. Uma *Instância* implementa a interface **Instance**. Toda *Instância* conhece o seu *Gerenciador de Instâncias*, que implementa a interface **BasicInstanceManager** ou **InstanceManager** (mais especializada).

Um *Gerenciador de Instâncias* atua como um *factory* para *Instâncias*, encapsulando o tipo e a informação específica da implementação necessária para a criação do objeto gerenciado. Podem existir diversos *Gerenciadores de Instâncias* para um determinado tipo de *Instâncias* em uma instalação. A interface **BasicInstanceManager** define as operações necessárias para criar *Instâncias* e

agrupá-las em conjuntos. A interface **InstanceManager** fornece todas as capacidades de **BasicInstanceManager** e a capacidade adicional para suportar a especificação de políticas para serem associadas com as *Instâncias*.

Uma *Biblioteca* atua como um *factory* para a criação de *Gerenciadores de Instâncias* e tem a habilidade de manter um conjunto destes objetos. Uma *Biblioteca* suporta a interface **Library** e também atua como **FactoryFinder**, permitindo que *Gerenciadores de Instâncias* sejam escolhidos com base em características tais como a interface da *Instância* que eles suportam e o tipo de objetos de políticas que estão registrados nos gerenciadores. A Figura 4.3 ilustra estes papéis apresentados.

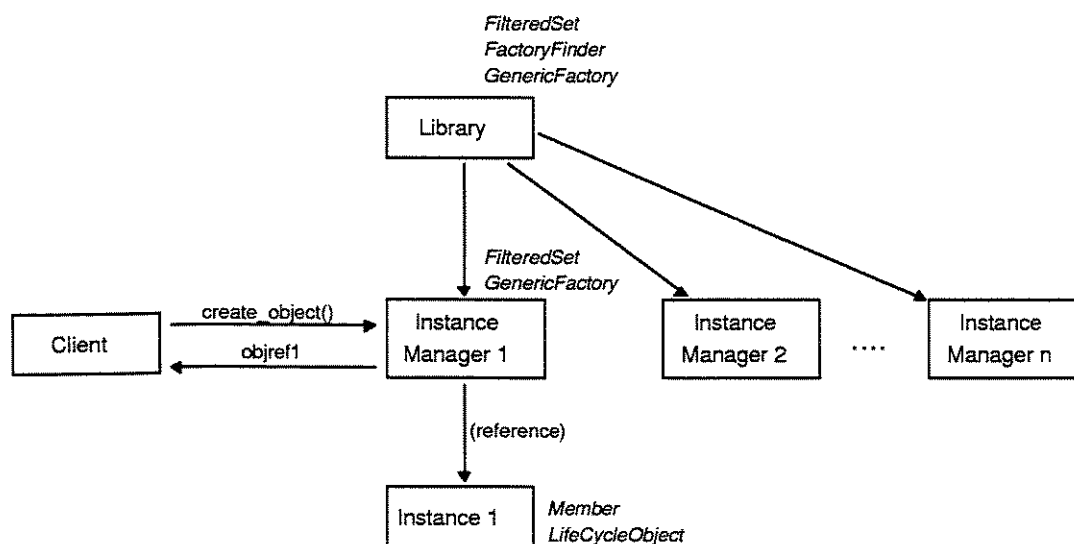


Figura 4.3: Papéis definidos no Serviço de Gerenciamento de Instâncias.

A Facilidade de Políticas

Esta Facilidade permite administradores adaptarem aplicações para suas necessidades específicas através da aplicação de políticas de gerenciamento. Uma política é uma regra que um administrador coloca no sistema. Interfaces para Políticas de Inicialização (**InitializationPolicy**) e de Validação (**ValidationPolicy**) são definidas, além da interface para **PolicyRegion**. Um objeto região de política associa políticas específicas com instâncias de objetos. É um tipo especial de conjunto para objetos sujeitos a políticas (objetos que suportam a interface **PolicyDrivenBase**) fazendo o gerenciamento necessário para estas políticas.

A interface **PolicyRegion** suporta operações que permitem a associação de gerenciadores de instâncias (**InstanceManager**) específicos com uma dada Poli-

cyRegion. Para cada **InstanceManager** associado com um **PolicyRegion**, no máximo uma de suas políticas de validação e no máximo uma de suas políticas de inicialização podem ser também associadas a este **PolicyRegion**. Estas relações são exibidas na Figura 4.4. Note que enquanto um **InstanceManager** pode manter uma lista de múltiplos objetos de políticas de ambos os tipos, é realmente o **PolicyRegion** que contém o objeto gerenciado que determina a política específica que será aplicada.

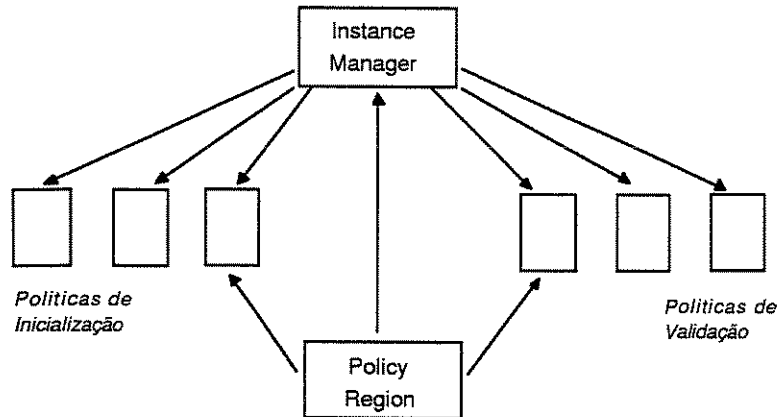


Figura 4.4: Relacionamento entre os objetos na Facilidade de Políticas.

Capítulo 5

O Projeto da Facilidade de Gerenciamento de Configuração

A proposta deste trabalho é criar um serviço que permite controlar a configuração de aplicações CORBA. Este capítulo apresenta o projeto desenvolvido para alcançar este objetivo.

5.1 O Contexto do Projeto

O projeto deste trabalho foi concebido para compor a Arquitetura de Gerenciamento da plataforma *Multiware*. Este contexto de projeto pode ser visto na Figura 5.1.

A plataforma *Multiware* [LMM⁺94], desenvolvida na UNICAMP, adota o padrão “*de jure*” do Modelo de Referência ODP e incorpora os conceitos do padrão “*de facto*” da OMA, assimilando idéias de outros padrões existentes como o DCE. A plataforma se constitui em um ambiente distribuído, heterogêneo, aberto e não proprietário sendo organizada em três camadas de suporte a aplicações distribuídas: uma de Hardware/Software básico, a *Middleware* e a *Groupware*.

A camada *Middleware* é responsável por fornecer facilidades de processamento distribuído aberto para a camada de *Groupware* e para as aplicações distribuídas. É composta por ORBs (*Orbix* e *OrbixWeb*) e de alguns serviços ODP, como o Trader [LM95], Gerenciamento de Sistemas Distribuídos [Que97], suporte a Transação e de Grupo [CM96], suporte a negociação de Qualidade de Serviço [LM97], suporte a Mobilidade [Vas99] e suporte ao gerenciamento de serviços usando agentes móveis [Rop99]. Em paralelo, a subcamada de Processamento Multimídia permite a troca de mensagens multimídia com qualidade de serviço específica. A última camada, a *Groupware*, é responsável pelo suporte às apli-

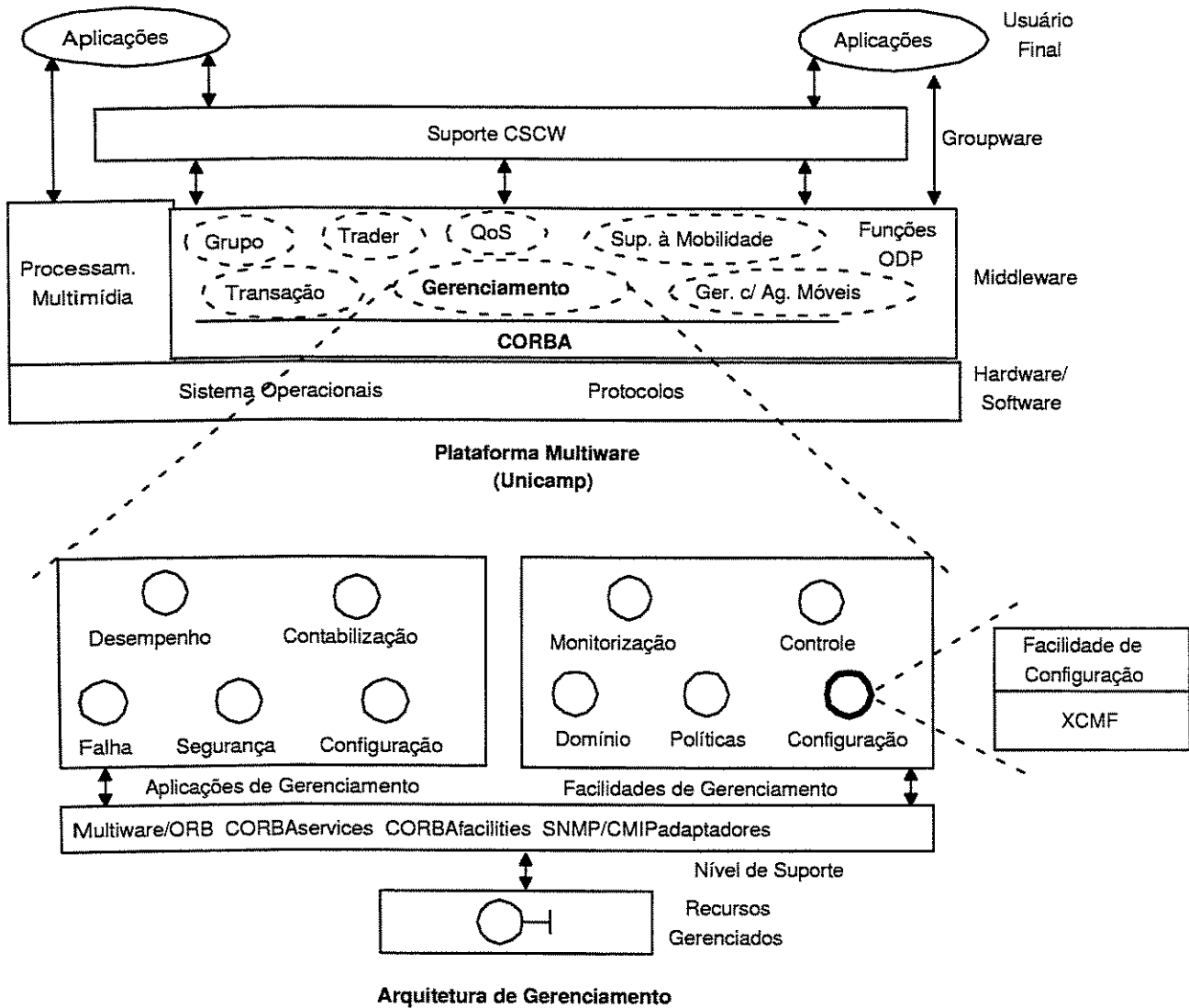


Figura 5.1: Contexto do Projeto.

cações CSCW¹.

O ambiente da plataforma *Multiware* consiste de um grande número de *hosts* de diferentes arquiteturas, dispersos em dois laboratórios da UNICAMP, conectados por redes *Ethernet*, *Fast Ethernet* e FDDI. Em ambos os laboratórios são utilizados ORBs comerciais da *Iona Technologies*.

O modelo para a Arquitetura de Gerenciamento foi proposto por [Que97] e tem por objetivo oferecer, de forma integrada, um conjunto de serviços úteis para o gerenciamento de aplicações distribuídas. O modelo prevê aplicações de gerenciamento, cobrindo as diversas áreas funcionais do modelo OSI, além de um conjunto de Facilidades de Gerenciamento Integradas, oferecendo serviços para o desenvolvimento destas aplicações.

Atualmente os esforços para o desenvolvimento da Arquitetura se concentram no desenvolvimento das Facilidades de Gerenciamento, como é o caso deste trabalho.

A Facilidade de **Monitorização** provê, através de requisições e notificações, a aquisição de dados estáticos e dinâmicos dos componentes dos sistemas distribuídos, permitindo a observação de suas atividades.

A Facilidade de **Controle** provê mecanismos para o controle do comportamento dos componentes do sistema, executando ações reativas e preventivas para evitar a degradação do sistema.

Políticas de gerenciamento definem quais operações de gerenciamento um gerente pode executar através de permissões, obrigações e proibições. A Facilidade de **Políticas** deverá oferecer mecanismos para a criação e destruição de políticas, para ler e escrever seus atributos, para verificar políticas associadas a um domínio e determinar os domínios sujeitos a uma determinada política.

A divisão de um sistema distribuído em domínios, refletindo a conectividade física da rede ou um organograma administrativo, permite a divisão de responsabilidade e autorização entre diferentes gerentes. A Facilidade de **Domínios** provê mecanismos para a criação e destruição de domínios, para a escrita e leitura dos atributos de um domínio, para listar seus componentes, inserir ou remover membros. A Facilidade de **Configuração** é o tema deste trabalho e será discutida nas próximas seções.

No entanto, cabe aqui uma justificativa quanto ao uso de XCMF como infraestrutura de suporte ao desenvolvimento da facilidade: seu uso se deve ao fato de que, no início das atividades deste trabalho (meados de 1997), XCMF era a especificação padronizada pelo OMG para a Facilidade de Gerenciamento. Atualmente, devido ao redirecionamento dos trabalhos de gerenciamento no OMG,

¹Computer Supported Cooperative Work

esta especificação não vigora mais como padrão. Sua retirada ocorreu em meados de 1999.

5.2 A Proposta do Trabalho

Como visto anteriormente, este trabalho teve como objetivo projetar e implementar uma Facilidade de Gerenciamento de Configuração para Aplicações CORBA que permita:

- Gerenciar um *grupo de objetos* definidos como uma Aplicação Distribuída;
- Gerenciar a estrutura formada pelos *bindings* estabelecidos entre estes objetos;
- Gerenciar o *estado* destes componentes da Aplicação Distribuída;
- Suportar todas as operações de *Ciclo de Vida* de um objeto CORBA, mantendo a *consistência* do ambiente da Aplicação Distribuída por ocasião destas operações;
- Ter gerenciamento *Dinâmico e Interativo*.

O caminho adotado para o desenvolvimento visa incorporar ao *framework* desenvolvido na especificação XCMF a capacidade de estabelecer, controlar e acompanhar a configuração de um conjunto destes objetos. Foi desenvolvido um modelo para o projeto de aplicações CORBA cujas estrutura e interconexões entre os objetos distribuídos possam ser gerenciadas dinamicamente e interativamente a partir de gerentes externos.

O projeto é baseado em dois serviços:

- a Facilidade de Gerenciamento de Conjuntos (XCMF) - Esta facilidade será usada para modelar e construir o *ambiente de gerenciamento* em uma estrutura hierárquica de conjuntos (domínios) com a ilustrada na Figura 4.2. Todos os recursos gerenciados estarão presentes neste ambiente, em algum ponto da *“árvore” de conjuntos (domínios)*.
- a Facilidade de Gerenciamento de Instâncias (XCMF)

Podemos perceber que a especificação XCMF, sem a implementação da Facilidade de Políticas, se resume praticamente a um serviço de criação de Instâncias. Com isto, o Projeto será um complemento à funcionalidade do *framework* XCMF,

contribuindo com novos conceitos e funcionalidades, visando atender aos requisitos de gerenciamento de configuração.

Novos Conceitos:

- Aplicação - Objeto responsável pelo gerenciamento de um *grupo de instâncias* que serão definidos como uma aplicação distribuída.
- Container de Aplicações - Objeto responsável pela criação e gerenciamento dos objetos *Aplicação*.
- Instância Configurável - São as instâncias componentes da aplicação. Estas instâncias serão especializações do Tipo *Instance* definido na especificação XCMF, agregando a capacidade de participar do mecanismo de gerenciamento de configuração do objeto *Aplicação*.
- Gerenciador de Referências - Uma funcionalidade presente em cada componente da aplicação para exercer o gerenciamento local das referências (*bindings*) de forma a garantir a consistência do mecanismo de gerenciamento de referências do objeto *Aplicação*.
- *Listener* de Configuração - São objetos capazes de receber do objeto da *Aplicação* notificações indicando mudança na configuração da aplicação.

Alguns Termos importantes neste projeto, são:

- Configuração de uma Aplicação - este termo indicará a *estrutura* da aplicação, que é resultado dos *bindings* estabelecidos entre seus componentes, e o *estado* destes *bindings* e Componentes, sendo este último determinado pelo valor em um dado instante de um conjunto de atributos pré-definidos para cada componente.
- Relacionamentos - este termo se referirá sempre a uma referência ou *binding* existente em um Componente da Aplicação. No entanto estes relacionamentos agregam informações sobre sua categoria (ou classe) e seu estado em um dado instante.

Novas Funcionalidades também surgem com o projeto:

- Gerenciamento do estado das instâncias;
- Gerenciamento dos relacionamentos entre os componentes da aplicação;
- Suporte completo às operações de Ciclo de Vida de uma instância;

- Suporte ao Gerenciamento Dinâmico e Interativo;
- Uso de notificações de eventos para manter gerentes externos atualizados com a configuração da aplicação;

O objeto da Aplicação foi modelado como um Conjunto e se realiza na interface **ConfigApplication**. Desta forma, todas as aplicações estarão presentes na modelagem do ambiente de gerenciamento (Figura 5.2). Este conjunto especial teria como membros os objetos instanciados da aplicação, os quais, teriam a capacidade de receberem operações de configuração. **ConfigApplication** é a interface central no processo de prover e controlar os serviços de configuração.

Estas operações de configuração, se resumem em que:

- Os objetos tenham a capacidade de sofrer operações de ciclo de vida;
- Os objetos tenham a capacidade de fornecer informações a respeito de seu estado e de seus relacionamentos, permitindo ainda que tais relacionamentos sejam administrados;
- Os objetos tenham a capacidade de sofrer operações específicas de configuração tais como a substituição de implementação e o “*rebind*” de referências.

O objeto Aplicação oferece serviços para:

- registrar todos os *Tipos* dos componentes da aplicação (**ConfigInstance**);
- enviar eventos notificando mudanças na estrutura da aplicação ou no estado dos componentes da aplicação;
- registrar *Listeners* de Configuração;
- localizar os demais componentes da aplicação baseando-se em critérios de pesquisa (nome, localização, identificação de interface, identificação de implementação);
- criar Instâncias para a aplicação baseando-se em critérios de criação, inclusive interativamente;
- copiar, mover e destruir Instâncias componentes da aplicação;
- executar o *rebind* de referências;

ConfigApplication tem ainda a capacidade de conhecer todos os tipos Instâncias Configuráveis pertencentes à aplicação, permitindo que estes objetos sejam criados interativamente, movidos e copiados. O conceito de “**Instância Configurável**” se realiza na interface **ConfigInstance** descrita na próxima Seção.

A Figura 5.2 fornece uma pequena ilustração destes conceitos. Ela ilustra a árvore de domínios (conjuntos) de um ambiente de gerenciamento acomodando uma aplicação distribuída desenvolvida e instalada segundo os moldes da Facilidade de Configuração. Ilustra também aplicações de gerenciamento que, localizando o objeto representante da aplicação, passam a gerenciá-la.

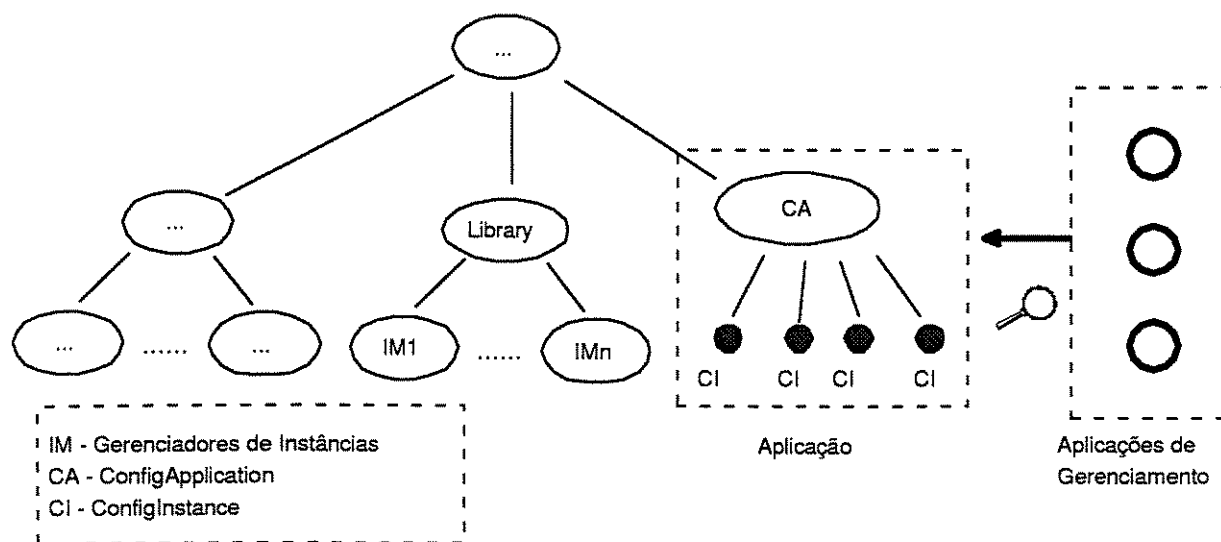


Figura 5.2: Ambiente de Gerenciamento.

O projeto procurou tornar os detalhes envolvidos nos mecanismos de gerenciamento da especificação XCMF (mais precisamente, o gerenciamento de Instâncias) transparentes tanto para os gerentes externos quanto para o projetista dos objetos gerenciados. Os serviços da Facilidade são oferecidos tanto para as aplicações de gerenciamento quanto para a própria aplicação gerenciada (Figura 5.3) [SM99a] e [SM99b].

Cabe aqui mencionarmos que nem todo componente de uma aplicação distribuída é um objeto servidor CORBA. Temos também os componentes que funcionam apenas como processos clientes destes objetos. Como estes processos participam da aplicação e armazenam referências de objetos, também devem participar do mecanismo de gerenciamento de configuração. Neste caso, o cliente precisa instanciar um objeto componente do mecanismo de gerenciamento para ser gerenciado.

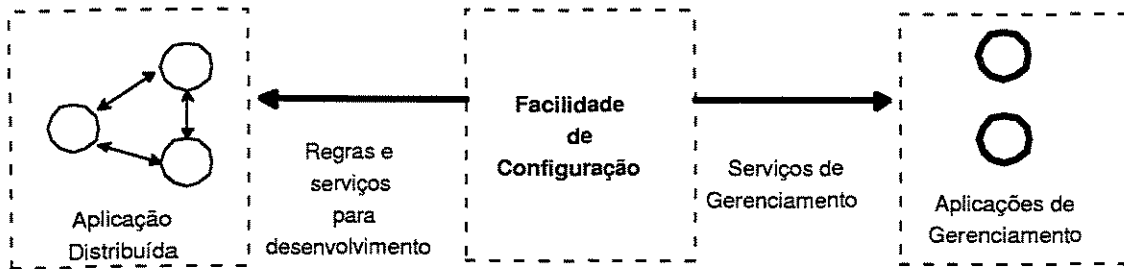


Figura 5.3: Serviços de Configuração.

De acordo com o apresentado na Seção 3.4, este projeto se enquadra na Segunda Fase (futura) dos trabalhos de gerenciamento no OMG onde pretende-se estudar os problemas relacionados com o Gerenciamento das Aplicações Distribuídas.

Um dos principais objetivos iniciais do projeto era alcançar a integração com as especificações já padronizadas pelo OMG, tendo as soluções presentes nestas especificações como ponto de partida para o nosso trabalho. Portanto, embora XCMF não esteja mais vigorando, no início de nossos trabalhos esta era a especificação padrão para interfaces de gerenciamento na OMA. Este é o motivo de adotarmos o framework de trabalho XCMF como fundamento para o nosso desenvolvimento.

Como comentado na Seção 3.4, XCMF foi projetado para dar suporte aos requerimentos básicos de criação de instâncias (objetos CORBA), de forma que os desenvolvedores de aplicações pudessem estender objetos XCMF básicos para se produzir uma aplicação gerenciável. Este tipo de visão para a estruturação e desenvolvimento do processo de gerenciamento, forma a base das idéias por traz de padrões atuais como os *CORBA Components*.

Portanto, as idéias desenvolvidas aqui poderiam ser aplicadas no desenvolvimento de uma Facilidade de Gerenciamento no contexto dos *CORBA Components* agregando a funcionalidade de gerenciamento de configuração de aplicações compostas de componentes CORBA. Isto poderá ser desenvolvido futuramente com a finalização dos trabalhos de desenvolvimento da especificação dos *CORBA Components*.

No Capítulo 6 será apresentado um exemplo de uma aplicação desenvolvida e gerenciada segundo os moldes da Facilidade de Configuração.

5.3 A Modelagem da Facilidade

A modelagem da Facilidade de Configuração pode ser vista na Figura 5.4. Na Figura, as caixas com as bordas mais largas representam as interfaces definidas

para a Facilidade enquanto que as demais representam interfaces definidas na especificação XCMF e no serviço de Ciclo de Vida.

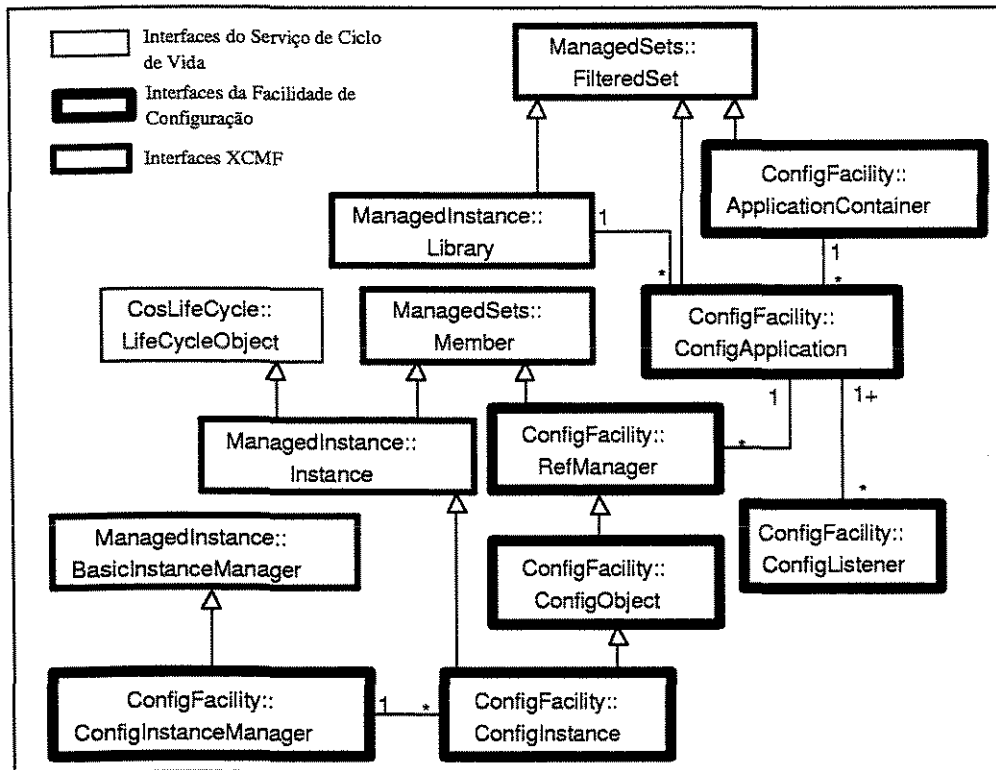


Figura 5.4: Modelagem da Facilidade.

ApplicationContainer

Esta interface representa uma *factory* para **ConfigApplication**. Nosso interesse é colocar este serviço de criação disponível no ambiente de gerenciamento juntamente com a *Library*. Este serviço torna direto a localização do objeto que representa a aplicação (**ConfigApplication**) por parte de seus componentes.

ConfigApplication

Esta é a interface central da Facilidade. Ela representa, no ambiente de gerenciamento, a própria aplicação distribuída fornecendo uma interface para o seu gerenciamento. **ConfigApplication** centraliza o controle da configuração da aplicação e oferece serviços tanto para os componentes da aplicação quanto para as aplicações de gerenciamento. Esta interface oferece, principalmente, serviços

para:

- registrar todos os tipos dos componentes da aplicação (**ConfigInstance**);
- enviar eventos notificando mudanças na estrutura da aplicação ou no estado dos componentes da aplicação;
- registrar “*listeners*” (**ConfigListener**) para eventos de notificação de atualização da configuração;
- localizar os demais componentes da aplicação baseando-se em critérios de pesquisa (nome, localização, identificação de interface, identificação de implementação);
- criar instâncias (**ConfigInstance**) para a aplicação baseando-se em critérios de criação, inclusive interativamente;
- copiar, mover e destruir instâncias componentes da aplicação;
- executar o *rebind* de referências;

ConfigApplication conhece toda a estrutura da aplicação, ou seja, todos os objetos componentes da aplicação e todas as referências mantidas por estes componentes. **ConfigApplication** mantém ainda uma “Lista de Dependências” para cada objeto da aplicação com informações dos demais **RefManager** que guardam uma referência registrada para aquele objeto (ver adiante).

Todas estas operações podem ser efetuadas dinamicamente, em tempo de execução. Após o processo de instalação, isto é, após a criação do objeto **ConfigApplication** e registro dos tipos dos componentes, a aplicação está pronta para ser gerenciada. **ConfigApplication** será responsável pelo preparo do ambiente de gerenciamento, principalmente dos recursos XCMF necessários (os Gerenciadores de Instâncias).

Isto significa que **ConfigApplication** usará a “XCMF Library” para criar um Gerenciador de Instâncias (**ConfigInstanceManager**) para cada tipo de componente (**ConfigInstance**) registrado.

No Anexo A, é apresentada a descrição IDL da interface **ConfigApplication**.

RefManager

Esta interface é responsável pelo gerenciamento das referências em um componente da aplicação. Para ser gerenciada ou para compor a estrutura da aplicação,

toda referência deverá ser registrada ao ser criada na aplicação. **RefManager** sempre estará associado a um único objeto **ConfigApplication**.

A interface **RefManager** fará o gerenciamento das referências em especializações das interfaces **ConfigObject** e **ConfigInstance** ou em “processos clientes” *multi-thread* que instanciarão um objeto implementando esta interface para exercer o gerenciamento de suas referências. Neste aspecto de sua atividade, **RefManager** oferece serviços para:

- registrar (e remover) referências do gerenciamento;
- obter de **ConfigApplication**, com base em sua identificação, a referência desejada atualizada;
- verificar se uma referência registrada está disponível para uso. A referência poderia estar indisponível no caso do objeto referenciado estar sofrendo uma operação de ciclo de vida como a movimentação.

Ao se registrar uma referência, uma classificação para a mesma é adicionada, permitindo um controle adicional sobre sua manipulação no ambiente de gerenciamento. Esta classificação também é oportuna para auxiliar uma possível construção gráfica da estrutura da aplicação enriquecendo as informações topológicas. Esta classificação para as referências se apresenta como:

- *ManageableRef* → Determina referências sujeitas a atualizações e operações de gerenciamento;
- *FixedRef* → Determina que a referência está sujeita apenas às atualizações, provenientes, principalmente, de operações de ciclo de vida dos objetos referenciados. Não permite operações de gerenciamento como o *rebind*;
- *OpenRef* → São referências a objetos externos à aplicação distribuída. Estas referências não sofrem qualquer tipo de atualização ou operação de gerenciamento.
- *CompositeRef* → São referências para objetos criados diretamente pelo componente associado ao **RefManager**. Como veremos adiante, estes objetos criados e instanciados diretamente por um componente da aplicação deverão ser, obrigatoriamente, do tipo **ConfigObject**. Estas referências não sofrem operações de gerenciamento.

As referências ainda possuem um estado associado. Este estado pode ser:

- *Normal* → A referência está desbloqueada e pronta para uso;
- *Suspended* → Indica que o objeto referenciado está sofrendo uma operação de ciclo de vida e, portanto, está indisponível;
- *Removed* → Indica que o objeto referenciado foi removido do ambiente (destruído).

No Anexo A, a especificação IDL desta interface é apresentada.

ConfigObject

Esta interface deverá ser suportada por todo objeto componente da aplicação distribuída. O objeto será então capaz de sofrer as operações básicas de configuração, operações estas relacionadas principalmente com o gerenciamento de referências.

Um objeto **ConfigObject**, não sendo uma Instância XCMF, não poderá ser criado interativamente por **ConfigApplication** ou sofrer operações de ciclo de vida. Estas operações são definidas na interface **ConfigInstance**.

Esta capacidade de **ConfigObject** de gerenciar suas referências deve-se a sua especialização da interface **RefManager**.

Um objeto **ConfigObject**, se existir, terá sua existência e seu gerenciamento de ciclo de vida a cargo de um outro componente da aplicação, provavelmente um objeto **ConfigInstance**. Esta interface é requerida para situações onde um objeto componente da aplicação atua como *factory* de outros objetos que também deverão compor a aplicação e participar de seu gerenciamento. Estes objetos terão a capacidade de gerenciar suas referências mas serão dependentes de seu *factory* no tocante ao seu processo de ciclo vida.

Quando um destes objetos *factory* instancia um **ConfigObject** ele deverá registrar uma referência para o mesmo do tipo *CompositeRef*. Estas referências não sofrem operações de gerenciamento, mas servem para indicar a situação.

ConfigObject

Esta interface deverá ser suportada por todo objeto componente da aplicação distribuída. O objeto será então capaz de sofrer as operações básicas de configuração, operações estas relacionadas principalmente com o gerenciamento de referências.

Um objeto **ConfigObject**, não sendo uma Instância XCMF, não poderá ser criado interativamente por **ConfigApplication** ou sofrer operações de ciclo de

vida. Estas operações são definidas na interface **ConfigInstance**. Isto se deve ao fato de que somente uma Instância XCMF participa do mecanismo de criação dinâmico desenvolvido na Facilidade de Gerenciamento de Instâncias.

Esta capacidade de **ConfigObject** de gerenciar suas referências deve-se a sua especialização da interface **RefManager**.

Um objeto **ConfigObject**, se existir, terá sua existência e seu gerenciamento de ciclo de vida a cargo de um outro componente da aplicação, provavelmente um objeto **ConfigInstance**. Esta interface é requerida para situações onde um objeto componente da aplicação atua como *factory* de outros objetos que também compõem a aplicação e participam de seu gerenciamento. Estes objetos terão a capacidade de gerenciar suas referências mas são dependentes de seu *factory* no tocante ao seu processo de ciclo vida.

Quando um destes objetos *factory* instancia um **ConfigObject** ele deverá registrar uma referência para o mesmo do tipo *CompositeRef*. Estas referências não sofrem operações de gerenciamento, mas servem para indicar este relacionamento de “pai para filho”.

ConfigInstance

Esta interface representa a realização do conceito de “Instância Configurável”. Isto significa que um objeto **ConfigInstance** pode ser criado interativamente além de ser copiado, movido e removido do ambiente da aplicação distribuída. Esta interface estende a funcionalidade de **ConfigObject**.

ConfigInstance tem dois modos de criação (*Normal* e *Requesting*). No modo *Normal*, os objetos são responsáveis pela obtenção de suas referências, estabelecendo seus relacionamentos. No modo *Requesting*, um objeto **ConfigInstance** irá requerer um conjunto de referências de **ConfigApplication**. Este mecanismo favorece a criação interativa dos componentes da aplicação. Deste modo, um gerente externo é capaz de participar, através da interface de **ConfigApplication**, do estabelecimento dos relacionamentos destes objetos.

Um objeto **ConfigInstance** sempre terá um **ConfigInstanceManager** associado. **ConfigInstanceManager** é um tipo especial de gerenciador de instâncias que define novas operações específicas para o gerenciamento de objetos **ConfigInstance**.

ConfigListener

Esta interface deverá ser implementada por qualquer objeto externo à aplicação (gerentes externos) que desejar receber notificações sobre a ocorrência de

mudanças na configuração da aplicação distribuída.

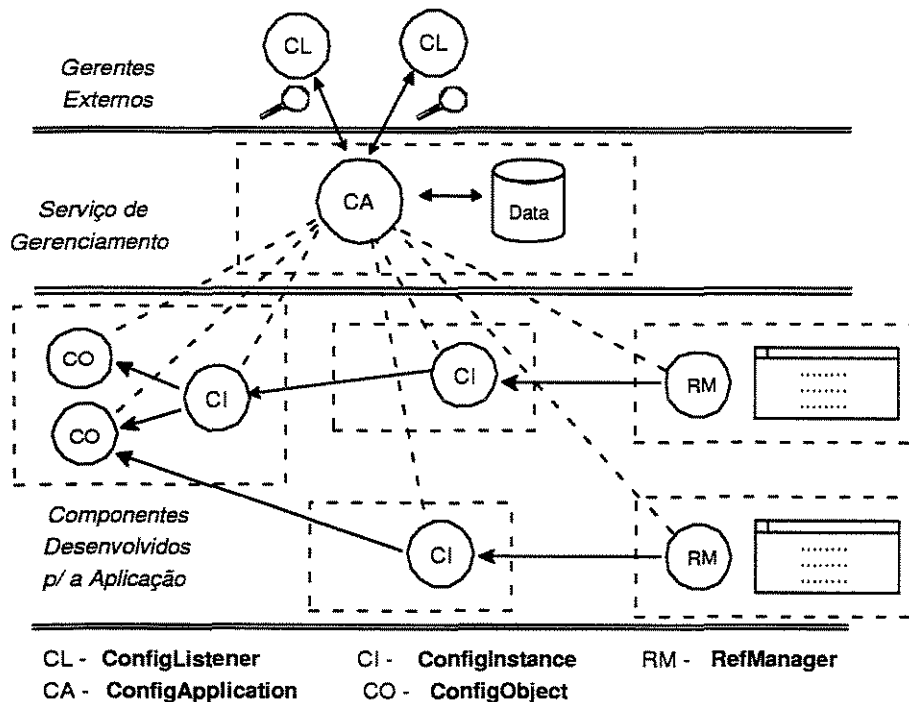


Figura 5.5: Ilustração dos Conceitos.

A Figura 5.5 ilustra os conceitos apresentados nesta seção. Ela apresenta uma aplicação composta por três objetos **ConfigInstance**, sendo que um deles atua como *factory* de dois objetos **ConfigObject** componentes da aplicação. Ela também apresenta dois processos clientes que instanciam, cada um deles, um objeto **RefManager** para o gerenciamento de suas referências. O controle da estrutura formada por todos estes componentes e a representação da aplicação no ambiente de gerenciamento fica por conta de um objeto **ConfigApplication**. Este ainda transmite notificações para dois gerentes externos que implementam a interface **ConfigListener**.

5.4 Trabalhos Relacionados

Outros trabalhos envolvendo gerenciamento de configuração como o Gerel [Endl92], o LuaSpace [Bat00], e o ICON [Fos97] merecem ser mencionados. Estes trabalhos são baseados em Linguagens de Configuração que permitem especificar a estrutura do sistema e definir as interconexões e interações entre seus componentes. Em geral, sistemas baseados em linguagem de configuração usam componentes cuja interface descrevem os serviços oferecidos e requisitados.

[Endl92] apresenta um trabalho baseado na linguagem de reconfiguração *Gerel* desenvolvida para lidar, de forma segura, com os problemas que surgem no suporte aos dois tipos de reconfiguração: *programadas* e *ad-hoc*. *Gerel* permite a definição de reconfigurações programadas genéricas cujo modelo de execução garante sua execução atômica. Estas definições são robustas o suficiente para lidar com as ocorrências de mudanças evolucionárias. *Gerel* é baseada em um mecanismo de linguagem capaz de selecionar componentes dinamicamente, de acordo com suas propriedades estruturais. Com este mecanismo de seleção de objetos, o projeto de reconfigurações pode ser feito de forma menos dependente de configurações concretas, sendo, desta forma, aplicáveis a uma série de possíveis configurações do sistema. *Gerel* ainda provê meios para definições de pré-condições para reconfigurações programadas, onde propriedades estruturais da configuração corrente são especificadas como requerimentos para a ocorrência uma determinada mudança.

[Bat00] apresenta o *LuaSpace*, que é um ambiente de desenvolvimento de aplicações distribuídas baseadas em componentes que segue o modelo de programação orientado à configuração caracterizando-se por separar os aspectos estruturais da aplicação da implementação dos componentes. O objetivo é oferecer suporte para configuração e reconfiguração dinâmica de aplicações. O ambiente é composto pela linguagem de configuração *Lua* e por ferramentas baseadas nesta linguagem. Uma destas ferramentas é o *LuaOrb*, um *binding* entre *Lua* e CORBA baseado na Interface de Invocação Dinâmica (DII), que permite o acesso dinâmico a objetos CORBA, e na Interface de Esqueleto Dinâmico, que permite a instalação dinâmica de novos objetos em um servidor em execução via *Lua*. *Lua* é uma linguagem de configuração interpretada cujo modelo de programação procedural é usado para descrever uma aplicação. A ligação entre os componentes é implícita, uma vez que não há comandos explícitos como *bind* e *link*. Como *Lua* é dinamicamente tipada, não é necessária a declaração prévia das instâncias dos componentes que serão usados no programa. Este aspecto permite a inserção dinâmica de novos componentes na configuração.

[Fos97] apresenta o ambiente *ICON* para gerenciamento de configuração interativo, integrado com a linguagem de configuração Darwin. Darwin é usada para descrever a composição e as interconexões dos componentes do sistema que é associado com um ambiente de gerenciamento gráfico. Seu ambiente de gerenciamento apresenta o serviço de gerenciamento de configuração estreitamente integrado ao serviço de domínios. O banco de dados de configuração é implementado através de Domínios de Configuração. A descrição de configuração de um componente é mapeada para um domínio quando o mesmo é instanciado. Os membros destes domínio são elementos representando as interfaces *requeridas*

e *oferecidas* pelo componente juntamente com informações sobre os *bindings* e *estados* destas interfaces. Se o componente for composto, seu domínio de configuração ainda conterà um sub-domínio para cada um dos componentes a partir do qual ele é referenciado. Desta forma, a partir da instanciação do sistema, o gerente pode, através de um *browser* gráfico de domínios, navegar pela hierarquia de domínios de configuração de um sistema, visualizar a estrutura atual do componente e efetuar operações de configuração. Implementações para este ambiente de configuração foram desenvolvidas para plataformas distintas como ANSAware e CORBA.

[BC95] apresenta o ambiente DisCo com sua abordagem para gerenciamento de configuração utilizando a linguagem de configuração CL, um Gerenciador de Configuração que recebe comandos de configuração já validados pelo ambiente de compilação da linguagem CL, um Servidor de Nomes que mantém informações sobre os módulos da aplicação e Gerenciadores Auxiliares cujo papel é executar comandos de configuração nas máquinas onde residem. O ambiente de comunicação adotado é baseado nos protocolos TCP/IP.

Outro trabalho relacionado ao nosso é o projeto MAScOTTE [MaS97] já discutido na seção 3.5. No entanto, nota-se neste projeto uma preocupação maior com o gerenciamento dos serviços e da própria ORB ao passo que o nosso trabalho visa o gerenciamento das aplicações distribuídas.

Nosso projeto não pretende prover a especificação de uma linguagem de configuração, visando, principalmente, estender o *framework* XCMF para cobrir as necessidades relacionadas com o gerenciamento de configuração. Visamos determinar mecanismos para o suporte ao gerenciamento interativo, principalmente nas operações de criação de componentes e *rebind* de referências. Como veremos no próximo Capítulo, também foram desenvolvidas ferramentas gráficas para possibilitar o gerenciamento interativo.

Capítulo 6

A Implementação da Facilidade

Este capítulo descreve os caminhos seguidos na implementação do modelo proposto no Capítulo 5.

6.1 O Ambiente de Desenvolvimento

Para este trabalho, fizemos uso do ambiente desenvolvido para a plataforma *Multitiware* que se constitui em estações IBM RS/600, Sun Sparc e PCs, com 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.

O ORB utilizado neste trabalho foi o *OrbixWeb* [ION98], com mapeamento IDL para Java. Java contribui muito para os trabalhos em ambientes distribuídos e heterogêneos por sua característica de portabilidade e, em se tratando de CORBA, por ser a única linguagem que possui o mapeamento para IDL especificado no OMG. Isso permitiu inclusive que a Sun incluísse na versão Java 2 SDK utilitários para comunicação com ORBs.

6.2 O ORB OrbixWeb

A implementação do ORB utilizado no trabalho foi o *OrbixWeb* 3.1 que implementa a especificação CORBA 2.0 e a revisão 1.1 da especificação de mapeamento IDL/Java. O *OrbixWeb* faz parte da família de produtos Orbix da *Iona Technologies Ltd.*

Os componentes do *OrbixWeb* são:

- Um compilador de Descrições IDL que produz código Java para o desenvolvimento dos clientes e servidores. Ele é usado para gerar os *Stubs* e *Skeletons* a partir da descrição IDL de uma interface.
- *OrbixWeb runtime* que é usado por todo programa *OrbixWeb* e consiste de um conjunto de bibliotecas Java que implementam diversos componentes do ORB incluindo o DII, o DSI e a funcionalidade do núcleo do ORB.
- *OrbixWeb daemon* (*orbixd*), que roda em cada *host* servidor e implementa algumas funcionalidades CORBA como o repositório de implementações. Um *daemon* implementado em Java também é incluído (o *orbixdj*). Estes *daemons* podem ativar também servidores C++ do *Orbix*.
- Um executável implementando o Repositório de Interfaces.

Com isso, o *OrbixWeb* se constitui fundamentalmente de duas bibliotecas, uma cliente e outra servidora, e os *daemons* de ativação. A biblioteca cliente representa um subconjunto da biblioteca servidora: enquanto a biblioteca servidora pode emitir e receber chamadas de operações remotas, a biblioteca cliente pode somente requisitar operações remotas. Os *daemons* de ativação necessitam estar presentes somente nos *hosts* servidores sendo responsáveis por (re-)lançar os servidores dinamicamente, de acordo com as políticas de ativação descritas na especificação CORBA. Eles funcionam a semelhança do daemon UNIX *inetd*.

Não existe, portanto, neste estilo de implementação, um componente central que encapsula o ORB e por onde passam todas as requisições de chamadas de métodos dos objetos. Ao contrário, estas requisições são passadas diretamente entre o código cliente e a implementação do objeto requisitada (Figura 6.1).

Com relação à implementação dos objetos CORBA, o compilador IDL gera classes Java para duas abordagens de implementação: *TIE* e *BOAImpl*. A *BOAImpl* usa o mecanismo de herança para as classes de implementação do objeto, o que limita a flexibilidade destas classes e elimina a possibilidade de reuso destas implementações. Isto se deve ao fato de Java não possuir um mecanismo de herança múltipla de implementação. A abordagem *TIE* utiliza um mecanismo misto de herança e delegação eliminando as deficiências da abordagem *BOAImpl*. Em nosso trabalho optamos pela abordagem *TIE* principalmente pelo fato do trabalho envolver o desenvolvimento de um conjunto de classes reutilizáveis. Vale lembrar que um servidor de objetos pode ter diferentes objetos CORBA com interfaces IDL distintas tendo ainda sua implementação construída através de qualquer uma destas abordagens citadas.

O *OrbixWeb* trabalha com dois protocolos de comunicação: o protocolo *Orbix*, para ambientes homogêneos *Orbix/OrbixWeb*, e o protocolo IIOP (*default*) que é

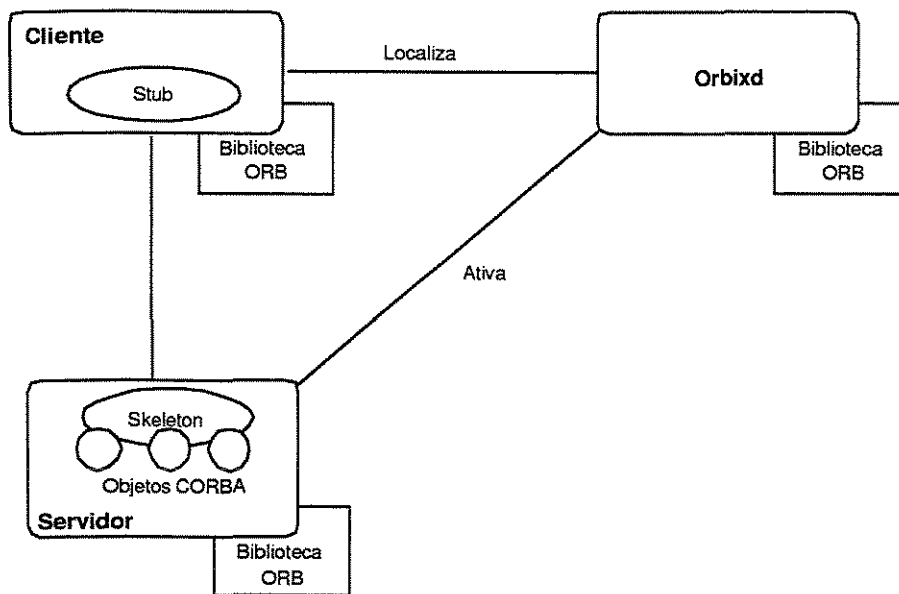


Figura 6.1: Estrutura da Implementação do ORB OrbixWeb

o padrão CORBA para interoperabilidade entre ORBs de diferentes fabricantes. Cada um destes dois protocolos define sua implementação para as referências de objetos. A referência IIOP requer o formato IOR¹, que inclui o endereço Internet do *host*, o número da porta lógica e uma referência de objeto interna do ORB envolvido.

Uma consequência muito importante do uso do protocolo IIOP é a possibilidade de se desenvolver clientes e servidores CORBA totalmente independentes de plataforma, sem nenhuma dependência externa (*self-contained*).

No *OrbixWeb* um cliente se conecta a um objeto CORBA através de sua referência de objeto obtida pelo método proprietário *bind()*. Este método, em último caso, provocará a instanciação do servidor do objeto desejado através do *orbixd* localizado no *host* servidor. Este procedimento é totalmente dependente do *daemon orbixd* e do registro do servidor no Repositório de Implementações.

6.3 A implementação da Camada XCMF

Das interfaces presentes na especificação XCMF, são necessárias para o desenvolvimento deste trabalho: as relacionadas à funcionalidade da Facilidade de Gerenciamento de Conjuntos (*Managed Sets*) e à funcionalidade da Facilidade de Gerenciamento de Instâncias (*Instance Manager*). Como nenhuma implemen-

¹Interoperable Object Reference

tação estava disponível na época de início dos trabalhos, foi necessário partirmos para a implementação destes elementos da especificação.

A principais questões resolvidas aqui foram: Como Implementar um ambiente de gerenciamento com base em XCMF? Como implementar um mecanismo que atenda ao modelo de criação de Instâncias do Serviço de Gerenciamento de Instâncias? Qual o mecanismo que permite um gerenciador de instâncias criar uma Instância?

Tomamos o cuidado neste ponto de não produzir nenhuma solução específica para uma implementação de ORB em uso (neste caso a OrbixWeb). Procurou-se produzir uma solução que fosse completamente **independente de plataforma**. Para isso, fez-se uso de *Servidores IIOP*, de *Objetos Factories* e do *Serviço de Nomes*.

Os *Factories* são objetos responsáveis pela criação e instanciação dos objetos (Instâncias) de um *tipo* específico no ambiente de gerenciamento. Eles deverão estar presentes em todos os *hosts* do ambiente onde a instanciação daquele tipo de Instância será necessário.

Os *Servidores IIOP* possuem uma implementação interna de um ORB (*self-contained*) que se comunica através do protocolo padronizado IIOP. As referências dos objetos criados nestes servidores também possuem uma característica importante: Elas são compostas pela identificação do *host* do servidor, pela *porta* de comunicação usada pelo servidor e por uma referência para o objeto em questão num formado dependente do ORB implementado no servidor IIOP.

Portanto, se após instanciados, estes objetos tiverem como publicar suas referências IIOP em um local bem conhecido e acessível, estes objetos estarão praticamente ao alcance dos demais objetos do ambiente distribuído. É aqui que identificamos a necessidade do *Serviço de Nomes*.

No entanto, no ambiente implementado neste trabalho para o gerenciamento, os objetos (os serviços de gerenciamento) não registram diretamente suas referências no Serviço de Nomes. Os serviços de gerenciamento e os *Factories* registram suas referências em um serviço criado para esta finalidade, o Serviço de Controle (*CtrlService*). Com isso, somente este Serviço de Controle necessitará ser registrado no Serviço de Nomes. O registro no *CtrlService* simplifica o processo de registro e permite que informações adicionais sejam incluídas no registro da referência. No caso dos *Factories*, são registrados também o Tipo de Instância que ele cria e o *Host* onde estas instâncias serão criadas.

A Figura 6.2 apresenta esta organização concebida para a implementação do ambiente de gerenciamento XCMF. Os outros dois elementos presentes na ilustração, o *SetBrowser* e o *SetService* serão apresentados na próxima seção.

Com relação à especificação XCMF, suas interfaces são organizadas em 8

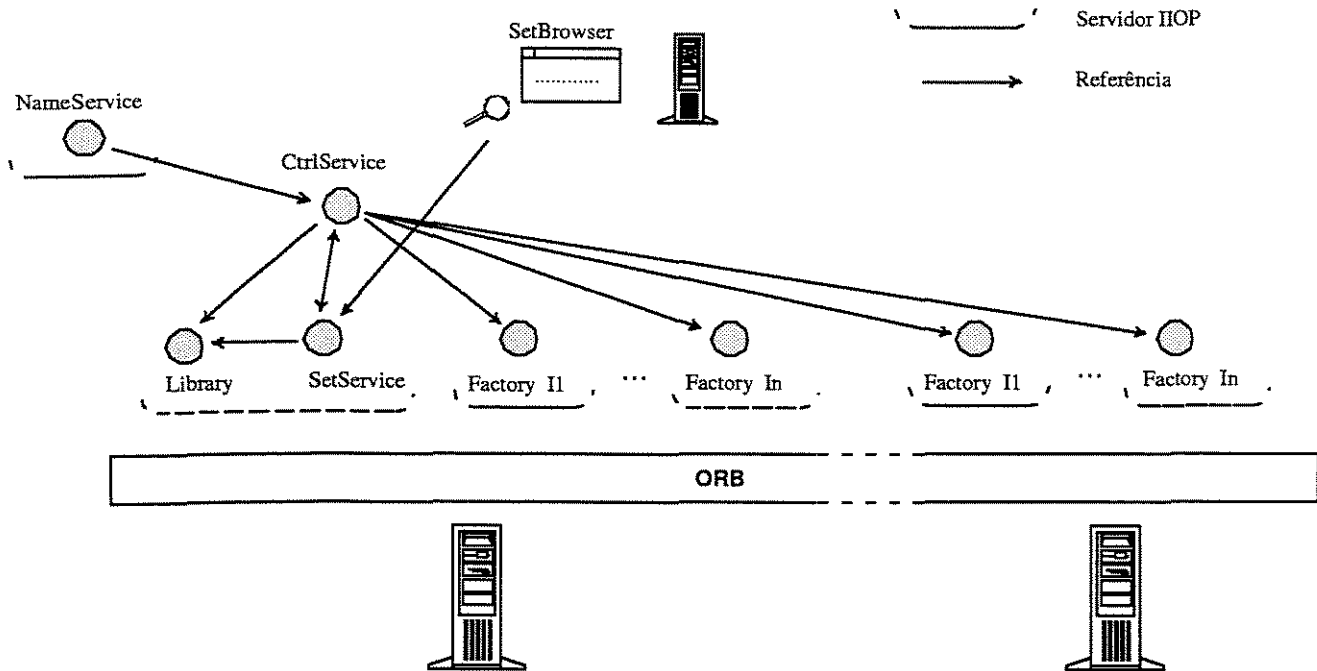


Figura 6.2: Organização do Ambiente de Gerenciamento para XCMF

módulos IDL. Estes módulos, com seus respectivos elementos implementados, são:

- *SysAdminTypes* - Define tipos e estruturas de dados referenciados nos demais módulos. Interfaces implementadas: Todos os tipos e estruturas definidas aqui foram mapeadas para classes Java.
- *SysAdminExcept* - Define as exceções usadas nos módulos. Interfaces implementadas: Todas as exceções definidas foram mapeadas para classes Java.
- *Identification* - Define a interface *Labeled* que permite um objeto ser identificado. Interface implementada: *Identification::Labeled*
- *SysAdminLifeCycle* - Define interfaces que permitem identificar uma localização específica (*Location* e *HostLocation*). No entanto, não há suporte para operações de ciclo de vida como cópia e movimentação de objetos. Este suporte foi implementado na Facilidade de Configuração. Interfaces implementadas:

– *SysAdminLifeCycle::Location*

- *SysAdminLifeCycle::HostLocation*
- *ManagedSets* - Define as interfaces que suportam a funcionalidade dos conjuntos e membros para a facilidade de gerenciamento de conjuntos. Interfaces implementadas:
 - *ManagedSets::Member*
 - *ManagedSets::Set*
 - *ManagedSets::FilteredSet*
- *ManagedInstances* - Define as interfaces para o suporte da funcionalidade de instâncias, gerenciadores de instâncias e da *library* para a facilidade de gerenciamento de instâncias. Interfaces implementadas:
 - *ManagedInstances::Instance*
 - *ManagedInstances::BasicInstanceManager*
 - *ManagedInstances::Library*
- *PolicyRegions* - Define interfaces de regiões (conjuntos) onde o comportamento dos objetos é determinado por políticas ligadas a estas regiões. Nenhuma interface foi implementada.
- *Policies* - Define as interfaces para os objetos de políticas de validação e inicialização, e a interface para o suporte ao relacionamento com estas políticas. Nenhuma interface foi implementada.

6.3.1 A Facilidade de Gerenciamento de Conjuntos

Este serviço pode ser considerado o ponto de entrada para o ambiente de gerenciamento ilustrado na Figura 5.2. Para realizarmos esta idéia, definimos e implementamos a interface *MngServices::SetServices*, incluída no módulo *MngServices* da facilidade de configuração.

Este serviço representa a Facilidade de Gerenciamento de Conjuntos sendo responsável por organizar a árvore de conjuntos (*domínios*) do ambiente de gerenciamento. A interface *MngServices::SetServices* possui a seguinte descrição IDL:

```
interface SetService : ManagedSets::FilteredSet{
    ManagedSets::Set resolve_name(in LabelTypeList name) raises (MngNotFound);
    void destroy_set(in ManagedSets::Set set) raises (MngNotFound,MngNotEmpty);
```

```

ManagedSets::Set new_set( in SysAdminTypes::LabelType label,
    in SetType st, in ManagedSets::Set ancestral)
    raises (MngInvalidLabel,MngNotFound,MngCanNotCreate );
};

```

Este serviço foi implementado em um servidor IIOP. Assim como no Serviço de Nomes, podemos construir uma federação de *SetServices*. No entanto, com relação ao gerenciamento, sempre teremos um *SetService* principal que irá montar o ambiente de gerenciamento.

Um *SetService*, quando instanciado, irá publicar sua referência no Serviço de Controle. Este Serviço de Controle, *MngServices::CtrlService* (apresentado adiante), será localizado através do Serviço de Nomes. Desta forma, como comentado anteriormente, a arquitetura garante sua independência de plataforma.

Uma ferramenta para gerenciamento gráfico, o *SetBrowser*, foi desenvolvido para o gerenciamento do ambiente XCMF. Ela permite ao gerente navegar no ambiente de gerenciamento controlado pelo serviço *SetService*. A Figura 6.3 ilustra esta ferramenta.

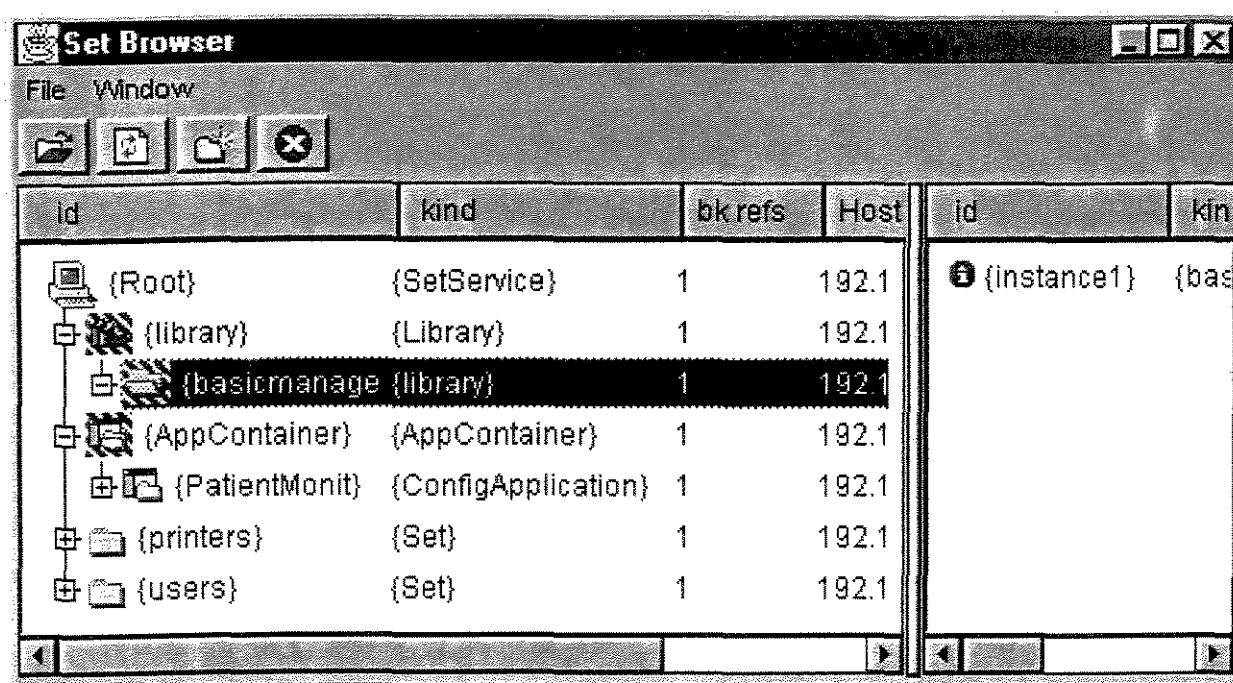


Figura 6.3: SetBrowser

Este *browser*, quando conectado ao *SetService* principal, exibe informações sobre a estrutura do ambiente de gerenciamento. Na Figura 6.3 está ilustrado

o ambiente de gerenciamento com alguns serviços instanciados como a *Library*, o *AppContainer* e o objeto *ConfigApplication* criado para o exemplo descrito na seção 6.4.3.

Através deste *browser*, podemos exercer operações como:

- Interagir com os serviços *SetService* criando, removendo e gerenciando conjuntos;
- Quando conectado ao serviço *SetService* principal do ambiente de gerenciamento, podemos interagir com o serviço *CtrlService* e obter informações sobre os *factories* instanciados no ambiente;
- Podemos interagir com os serviços XCMF, como a *Library* e os gerenciadores de instâncias, e usufruirmos de suas funcionalidades de gerenciamento;
- Verificar os *bindings* com cada um dos objetos instanciados no ambiente, obtendo informações a respeito de suas referências e localização.
- Interagirmos com o serviço de configuração *AppContainer* e instanciar objetos *ConfigApplication*.

6.3.2 A Facilidade de Gerenciamento de Instâncias

A Facilidade de Gerenciamento de Instâncias fornece a infra-estrutura requerida para o gerenciamento de múltiplas instâncias de um tipo de objeto. Este serviço apresenta uma especificação sintática e semântica para um serviço de **criação** baseado nas interfaces padronizadas do serviço OMG de Ciclo de Vida. O modelo de criação de objetos é definido em termos de objetos “*Factory*”.

As principais interfaces implementadas aqui foram: a *ManagedInstances::Instance* para as instâncias, a *ManagedInstances::BasicInstanceManager* para os gerenciadores das instâncias, e a *ManagedInstances::Library* para organizar e gerenciar os gerenciadores de instâncias.

A *Library* foi implementada em um servidor IIOP, e quando instanciada, localiza o Serviço de Controle e publica sua referência garantindo ao modelo a independência de plataforma. Somente uma *Library* poderá controlar um ambiente de gerenciamento. A *Library* será responsável por criar e instanciar os gerenciadores de instâncias.

Os Gerenciadores de Instâncias são instanciados no espaço de endereçamento de seu *Factory* ou da própria *Library*. Eles criam e o controlam as Instâncias por intermédio do *Factory* projetado para a instanciação daquele tipo de Instância.

6.3.3 O Mecanismo de Criação de Instâncias

Um componente importante deste mecanismo é o Serviço de Controle que funciona como um serviço central onde os serviços de gerenciamento publicam suas referências.

Este serviço também foi implementado em um servidor IIOP e quando instanciado, faz o seu registro no Serviço de Nomes com uma identificação bem conhecida. A partir deste registro, ele está disponível para todo o ambiente de gerenciamento. Os demais serviços de gerenciamento do ambiente fazem seu registro no *CtrlService* assim que instanciados. São eles: a *Library*, o *SetService* principal (de gerenciamento), o *ApplicationContainer* e todos os *Factories* de Instâncias e Gerenciadores de Instâncias.

Como mencionado anteriormente, o modelo de criação de objetos é definido em termos de objetos “*Factory*”. Portanto, quando um Gerenciador de Instâncias precisar criar uma Instância do tipo *T* no *host H*, ele deverá localizar, através do *CtrlService*, um *Factory* para Instâncias do tipo *T* disponível no *host H*.

Estão disponíveis na interface do *CtrlService* operações para:

- O registro da *Library*, do *ApplicationContainer* e do *SetService* de gerenciamento;
- Obter a referência destes serviços registrados;
- Adicionar e remover registros dos *Factories* para Instâncias e Gerenciadores de Instâncias;
- Pesquisar por *Factories* com base no tipo de Instância que o mesmo cria e em sua localização.

Note que, como comentado anteriormente, esta arquitetura garante sua independência de plataforma, combinando servidores IIOP e Serviço de Nomes.

Aqui, vale a pena analisarmos a interface *CosLifeCycle::LifeCycleObject* do Serviço de Ciclo de Vida. Esta interface, como pode ser observado no modelo da Figura 4.1, deverá ser suportada por todo objeto criado dinamicamente. Esta interface possui a seguinte descrição IDL:

```
interface LifeCycleObject{
    LifeCycleObject copy(in FactoryFinder there, in Criteria the_criteria)
        raises(NoFactory, NotCopyable, InvalidCriteria, CannotMeetCriteria);
    void move(in FactoryFinder there, in Criteria the_criteria)
        raises(NoFactory, NotMovable, InvalidCriteria, CannotMeetCriteria);
```

```
void remove() raises(NotRemovable);
};
```

Esta interface representa um objeto capaz de sofrer operações de ciclo de vida. É importante notar que o próprio objeto terá grande participação no procedimento para sua cópia, movimentação e remoção. No caso da cópia e movimentação, o parâmetro *there* representa o escopo da operação. O parâmetro *the_criteria* deverá ser passado diretamente para o *Factory*. Este processo de cópia e movimentação está ilustrado na Figura 6.4.

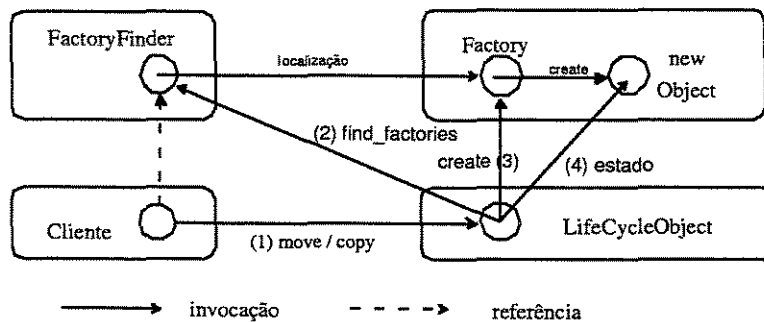


Figura 6.4: Movendo ou copiando um LifeCycleObject

Os Clientes, conhecendo uma referência para um *FactoryFinder*, iniciam o processo passando esta referência para o *LifeCycleObject* na chamada do método para a cópia movimentação (1). Este *FactoryFinder* representa o escopo para onde o objeto será movido ou copiado. De posse desta referência o *LifeCycleObject* deverá efetuar a cópia ou movimentação. Ele, através da operação “*find_factories*” (2) deverá obter uma referência para um “*Factory*” que poderá criar o novo objeto no escopo determinado (3). A operação (4) indica o momento em que o estado é transferido de um objeto para o outro através de uma interface privada.

Como já discutido, XCMF apresentou uma especificação sintática e semântica para um serviço de criação baseado na especificação abstrata do Serviço de Ciclo de Vida. No entanto, XCMF não apresentou o mesmo para as demais funcionalidades de ciclo de vida como “cópia” e “movimentação”. Estas funcionalidades tiveram a sua definição e implementação na Facilidade de Gerenciamento de Configuração.

O procedimento desenvolvido no trabalho para estas operações de ciclo de vida de um objeto *ConfigFacility::ConfigInstance* é ilustrado na Figura 6.5. Nestas operações, o primeiro argumento (o *FactoryFinder*), como definido na interface *CosLifeCycle::LifeCycleObject*, nunca será usado porque, de acordo com a especi-

ficação XCMF de Gerenciamento de Instâncias, toda Instância, conhecendo o seu Gerenciador, já dispõe do *FactoryFinder* correto para o processo.

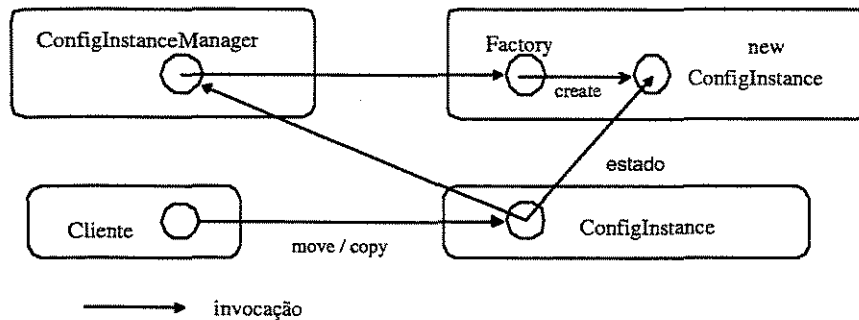


Figura 6.5: Movendo ou copiando um ConfigInstance

O escopo para a criação do novo objeto será definido no segundo argumento (*the_criteria*). A definição deste argumento se encontra na definição da interface *ConfigFacility::ConfigInstanceManager*.

Estas operações deverão produzir diversos efeitos no ambiente da aplicação distribuída. Assim, a implementação de uma biblioteca para a interface *ConfigFacility::ConfigInstance* é imprescindível.

Um problema importante, tratado durante a implementação, foi a manutenção da consistência das informações de configuração da aplicação. Podemos ilustrar esta preocupação com uma visão geral dos passos envolvidos na *movimentação* de objetos *ConfigInstance*:

```
void ConfigInstance::move(FactoryFinder finder, Criteria the_criteria)
{
```

(i) “Verificar se o objeto permite ser movido.”

(ii) “Através de seu Gerenciador de Instâncias, criar o novo objeto. Caso a operação se complete com sucesso, este novo objeto tomará o lugar daquele sendo movido. Caso ocorra algum problema, este novo objeto será removido pelo Gerenciador de Instâncias.”

(iii) “Notificar *ConfigApplication* do início da operação. *ConfigApplication* automaticamente irá suspender todas as referências para este componente.”

(iv) “Chamar a função gancho apropriada para a transferência do estado para o objeto temporário”

(v) “Transferir as referências (os relacionamentos) para o objeto temporário”

(vi) “Fazer o objeto temporário tomar o lugar do objeto sendo movido (No Gerenciador)”

(vii) “Notificar a *ConfigApplication* do fim da operação. *ConfigApplication* automaticamente atualizará todas as referências para este objeto.”

(viii) “remover este objeto”

}

Note que um “Gancho” deve ser definido, ou seja, uma função que deve ser chamada dando ao projetista a oportunidade de transferir o estado específico de seu objeto. Esta função deverá ser algo como:

```
void transfer_state( ConfigInstance newobj);
```

Todo este processo de movimentação deve ser implementado como uma ação atômica. Se algum destes passos falhar, todo o processo deverá ser cancelado sem corromper a consistência da configuração da aplicação. Esta propriedade foi garantida na implementação da interface *ConfigInstance* que compõe a biblioteca de classes desenvolvida para a Facilidade de Configuração.

Uma ilustração para o algoritmo da operação de *cópia* é:

```
LifeCycleObject copy(in FactoryFinder there, in Criteria the.criteria)
```

```
{
```

```
    (i) “Verificar se o objeto permite ser copiado.”
```

```
    (ii) “Através de seu Gerenciador de Instâncias, criar o novo objeto temporário.”
```

```
    (iv) “Chamar a função gancho apropriada para a transferência do estado para o novo objeto.”
```

```
}
```

Podemos perceber pela diferença entre os dois algoritmos que mover um objeto não terá o mesmo efeito que copiar o objeto e depois remover o antigo.

6.4 A implementação da Facilidade de Configuração

Os Módulos componentes da Facilidade de Configuração são:

- *ConfigFacility* - Define interfaces para o desenvolvimento dos objetos que compõem a aplicação distribuída configurável (*ConfigFacility::RefManager*, *ConfigFacility::ConfigObject* e *ConfigFacility::ConfigInstance*), e para os objetos criados para o gerenciamento destes componentes (*ConfigFacility::Con-*

figApplication, *ConfigFacility::ConfigInstanceManager*, *ConfigFacility::ConfigListener*).

- *MngFactories* - Define interfaces para os *Factories* de Instâncias (*ConfigFacility::InstanceFactory*) e Gerenciadores de Instâncias (*ConfigFacility::ManagerFactory*).
- *MngServices* - Define as Interfaces para os Serviços de Gerenciamento que, juntamente com a *Library*, compõem o ambiente de gerenciamento. Estas interfaces são: *ConfigFacility::SetService*, *ConfigFacility::ApplicationContainer*, *ConfigFacility::CtrlService*

A implementação produziu um conjunto de bibliotecas de programação para as interfaces *ConfigObject*, *ConfigInstance*, *InstanceFactory* e *ManagerFactory*, que devem ser especializadas pelo projetista da aplicação distribuída, e para a interface *RefManager*. Considerando que o trabalho deste projetista será basicamente o de especializar e implementar estas interfaces, um conjunto de “ganchos” (*hooks*) foi definido no desenvolvimento desta biblioteca.

Uma ferramenta para o gerenciamento gráfico de configuração, o *ConfigBrowser*, foi desenvolvido. este *browser*, ativado a partir do *SetBrowser*, trabalha sobre um objeto *ConfigApplication* permitindo ao gerente visualizar a configuração da aplicação distribuída e executar, interativamente, operações de gerenciamento de configuração. Ele implementa a interface *ConfigFacility::ConfigListener*. A Figura 6.6 ilustra este *browser* ativado para o objeto *ConfigApplication* criado para o exemplo descrito na seção 6.4.3. A Figura 6.7 ilustra outros elementos que podem estar presentes no *ConfigBrowser*, como a representação de outros tipos de referência (*FixedRef*, *OpenRef* e *CompositeRef*) e a representação dos *ConfigObject* e processos clientes componentes da aplicação.

Através deste *browser*, podemos:

- Executar as operações de ciclo de vida sobre os objetos *ConfigInstance*;
- Checar o estado do objeto da aplicação;
- Ativar ou desativar um objeto da aplicação;
- Visualizar e gerenciar a estrutura da aplicação, exercendo interativamente o *rebind* de referências;
- Instanciar um novo componente para a aplicação, com a possibilidade de se participar ativamente no processo de definição das referências deste novo objeto através do processo de criação Interativa;

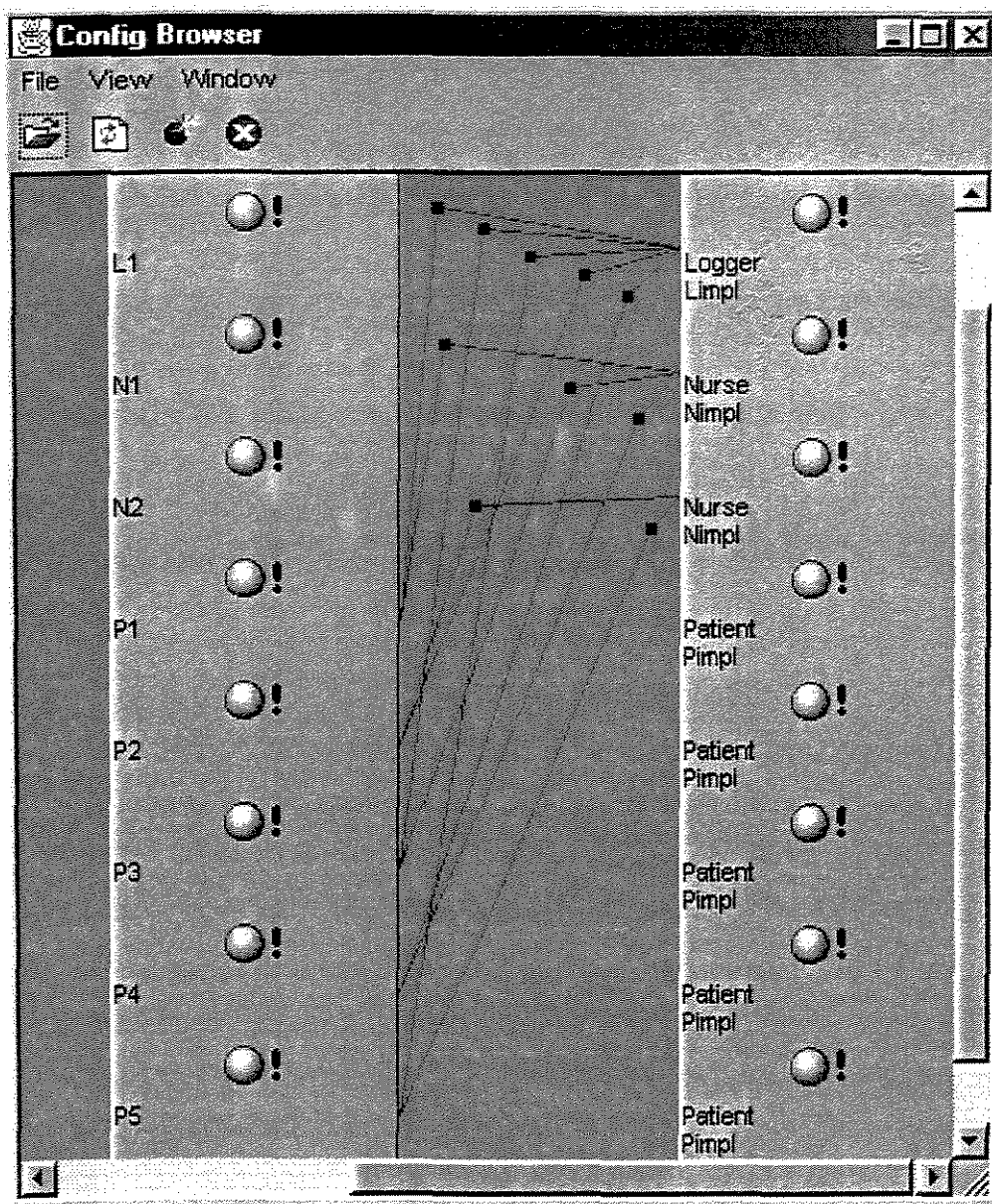


Figura 6.6: ConfigBrowser

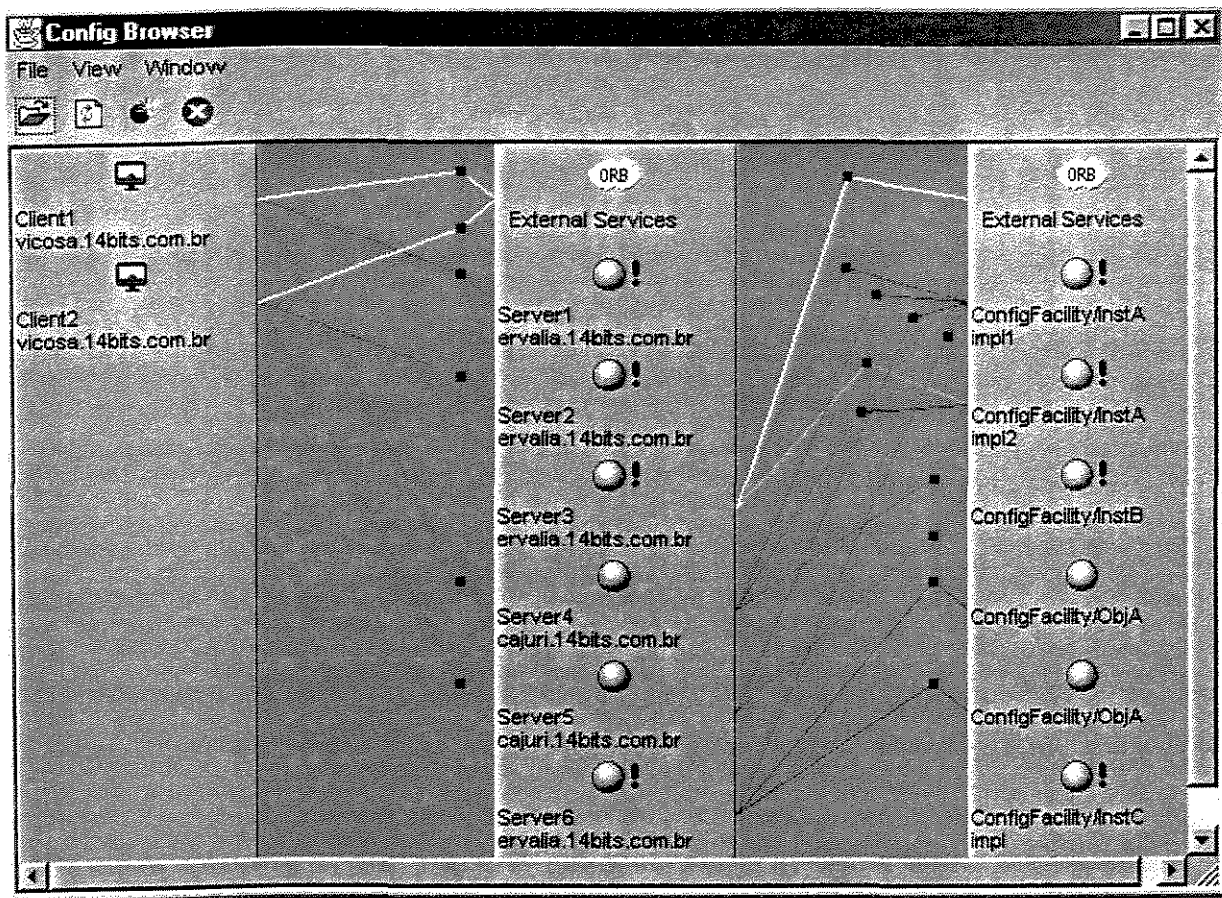


Figura 6.7: ConfigBrowser (ilustração)

O *ConfigBrowser* apresenta três painéis: O primeiro apresenta os processos clientes que compõem a aplicação e que instanciaram um *RefManager* para o gerenciamento de suas referências. O segundo e o terceiro listam todos os objetos da aplicação. Os objetos enfatizados com um sinal de exclamação (!) ao seu lado são do tipo *ConfigInstance* e os outros, do tipo *ConfigObject*.

Entre cada um destes painéis está uma área destinada aos relacionamentos. Na primeira são traçados os relacionamentos dos processos clientes para os objetos da aplicação e na segunda área, os relacionamentos entres os objetos da aplicação.

Os tipos distintos de referências, descritas no capítulo anterior, são identificadas no *browser* pela cor:

- *ManageableRef* → São traçadas com a cor Verde Escuro;
- *FixedRef* → São traçadas com a cor Verde Claro;
- *OpenRef* → São traçadas com a cor Branca e estão sempre direcionadas para o símbolo da ORB;
- *CompositeRef* → São traçadas com a cor Azul;

Cada uma das referências possui um controle (*Tracker*) que permite o reposicionamento das mesmas e a execução dos *rebinds*.

6.4.1 Organizando o Ambiente de Gerenciamento

Podemos resumir assim a organização do ambiente de gerenciamento. Como resultado do trabalho, temos os seguintes serviços compondo o ambiente de gerenciamento:

- O *Naming Service*, utilizado para se localizar o *CtrlService*;
- O *CtrlService* que conhece e controla os serviços de gerenciamento do ambiente;
- O *SetService* que controla a estrutura em forma de árvore do ambiente de gerenciamento;
- O *ApplicationContainer* que organiza e gerencia todos os objetos *Config-Application* do ambiente;
- A *Library* que organiza e gerencia todos os gerenciadores de instâncias do ambiente;

- Os objetos *ConfigApplication* que executam diretamente o gerenciamento de configuração das aplicações.

Podemos encontrar ainda os serviços de *factories* instanciados no ambiente. São eles:

- Os objetos *ManagerFactory*, que instanciam os Gerenciadores de Instâncias. Estes últimos poderão estar ausentes pois a própria *Library* tem condições de instanciar estes *factories*;
- Os objetos *InstanceFactory*, que instanciam as Instâncias e as Instâncias Configuráveis.

Estes *factories* deverão ser desenvolvidos pelo projetista da aplicação distribuída mas, para isto, ele contará com as bibliotecas do projeto.

As duas ferramentas de gerenciamento também compõem o ambiente: O *SetBrowser* e o *ConfigBrowser*. A Figura 6.8 ilustra a organização deste ambiente, as ligações entre os serviços e o caminho percorrido para se localizar estes serviços de gerenciamento. Nesta figura, as setas indicam referências de objetos contidas nos componentes do ambiente de gerenciamento.

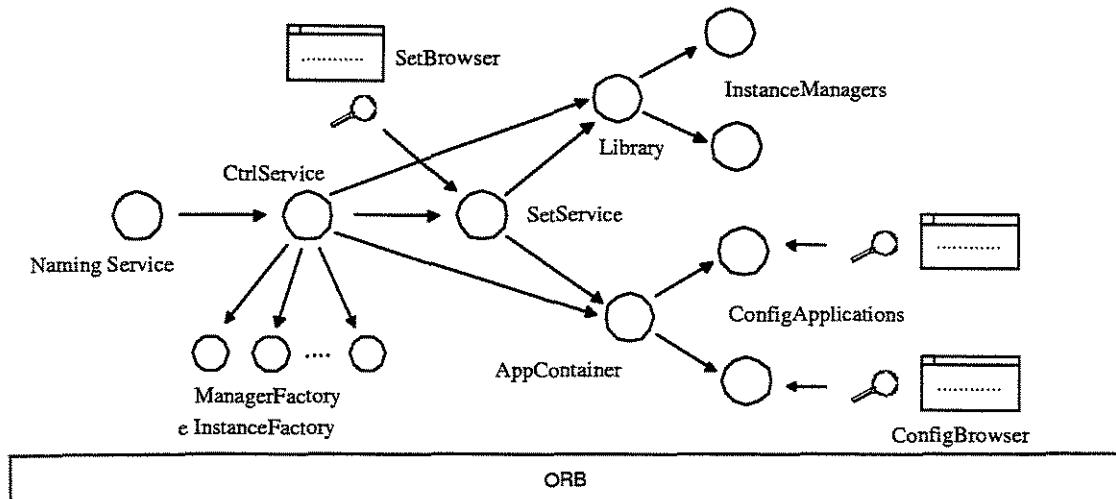


Figura 6.8: Organização do Ambiente de Gerenciamento.

6.4.2 Trabalhando com a Facilidade de Configuração

O processo de desenvolvimento de uma aplicação distribuída, segundo os moldes da Facilidade, requer os seguintes passos:

- projeto dos componentes da aplicação. Estes serão especializações das interfaces *ConfigInstance* ou *ConfigObject*;
- para cada especialização de *ConfigInstance* desenvolvida, desenvolver seu *factory* e um servidor para instanciá-lo. O Gerenciador de Instâncias procurará pelo *factory* daquele tipo de instância sempre que precisar criar uma destas instâncias.

Para o processo de instalação, será necessário:

- criar um objeto *ConfigApplication* para a aplicação através do serviço *ApplicationContainer*;
- registrar os tipos especializados de *ConfigInstance* no objeto *ConfigApplication*;
- registrar os *factories* no Serviço de Controle, o *CtrlService*. Os *factories* desenvolvidos através da biblioteca de classes do projeto se registram automaticamente no momento de sua instanciação.

O objeto *ConfigApplication* prepara o ambiente de gerenciamento criando um objeto *ConfigInstanceManager* para cada tipo de *ConfigInstance* registrado. Este Gerenciador de Instâncias será responsável, de acordo com o *framework* XCMF, pela criação das instâncias daquele tipo que gerencia. Este trabalho, como já foi dito, é feito por intermédio do *factory* específico desenvolvido e instalado pelo projetista da aplicação.

Agora, através da interface do objeto *ConfigApplication*, objetos *ConfigInstance* poderão ser instanciados dinamicamente. Objetos *ConfigInstance* poderão ser instanciados em modo *Normal* ou em modo *Requesting*. No modo *Normal* os objetos são responsáveis pela obtenção de suas referências, estabelecendo seus relacionamentos. No modo *Requesting* o objeto *ConfigInstance* poderá requerer do objeto *ConfigApplication* um conjunto de referências pré-estabelecidas. Deste modo, um gerente externo será capaz de participar, através da interface de *ConfigApplication*, do estabelecimento dos relacionamentos destes objetos.

Como já foi dito, processos clientes modelados como componentes da aplicação poderão ter seus relacionamentos gerenciados por um objeto *RefManager*. A instanciação de tais objetos, ou mesmo, a instanciação de objetos *ConfigObject* deverá ser notificada ao objeto *ConfigApplication* que controla a estrutura da aplicação.

A estrutura da aplicação poderá ser construída graficamente por um gerente externo obtendo do objeto *ConfigApplication* todas as informações necessárias

para tal. Poderá, ainda, manter estas informações atualizadas através de notificações de atualização recebidas do objeto *ConfigApplication*.

As operações de ciclo de vida dos objetos *ConfigInstance* precisam ser invocadas através da interface do objeto *ConfigApplication* que elimina muitos detalhes envolvidos na sintaxe de tais operações.

6.4.3 Exemplo de Implementação

Podemos resumir os passos para o desenvolvimento de uma aplicação distribuída segundo as regras da Facilidade de Configuração em:

- Projeto dos componentes da Aplicação. Os componentes da Aplicação deverão ser especializações de *ConfigObject* e *ConfigInstance*. Para tanto, o projetista usará diversas bibliotecas produzidas para a Facilidade de Configuração.
- Para cada **ConfigInstance**, projetar o seu *Factory*.
- Instanciar os *Factories*.
- Criar o objeto da Aplicação (*ConfigApplication*).
- Registrar todos os tipos de componentes no objeto da aplicação.

O Desenvolvimento da Aplicação

Vamos ilustrar estes passos no desenvolvimento de um sistema simples de monitorização de pacientes como o descrito em [Mag94]. Este sistema será composto por três tipos de objetos principais:

- **Patient** - Representa e monitora um paciente recebendo tratamento;
- **Nurse** - Representa um enfermeira responsável por um conjunto de pacientes;
- **Logger** - Entidade responsável por armazenar notificações e estados sucessivos de **Patient**.

Objetos **Patient** sempre serão executados em uma estação próxima ao paciente real. Se este paciente for transferido para um leito coberto por uma outra estação, o objeto **Patient** deve também ser transferido para esta outra estação.

Objetos **Nurse** e **Logger** seriam executados em uma máquina responsável pela monitorização dos pacientes.

Tendo sido tomada a decisão de se utilizar a Facilidade de Configuração para o gerenciamento de nossa aplicação, as regras da Facilidade passam a influenciar o projeto. Isto é visto na modelagem da aplicação ilustrada na Figura 6.9 .

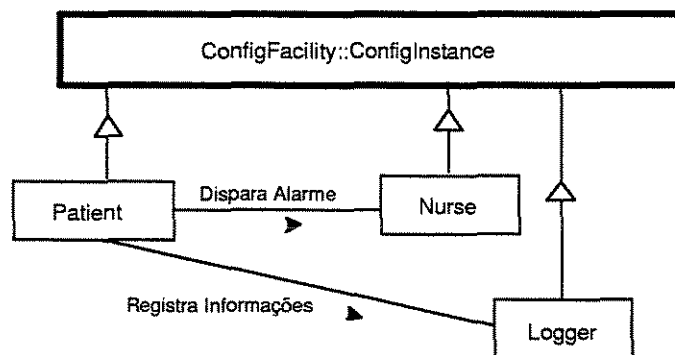


Figura 6.9: Modelagem do Sistema de Monitorização de Pacientes.

A Figura 6.10 ilustra os relacionamentos entre os componentes da aplicação que descrevem a sua estrutura e que devem ser gerenciados.

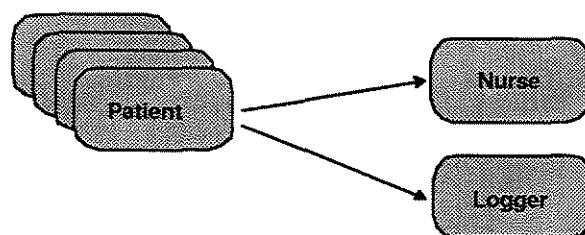


Figura 6.10: Relacionamentos Gerenciados no Sistema.

Os componentes da aplicação são especializações de *ConfigInstance*. Portanto, serão capazes de sofrerem operações de ciclo de vida.

A descrição IDL da interface **Logger** poderia ser simplesmente:

```

interface Logger: ConfigFacility::ConfigInstance
{
    // Utilizada por Patient para o registro de seu estado
    // em dado instante.
}
  
```

```

    void record (in Patient p, in DATA dat);
};

```

A descrição IDL das interfaces **Patient** e **Nurse** poderiam ser:

```

interface Patient: ConfigFacility::ConfigInstance
{
    void report_status (out Status st);
};

interface Nurse: ConfigFacility::ConfigInstance
{
    // Utilizada por Patient para notificar a ocorrência de
    // problemas
    void alarm (in Patient p);
};

```

O projetista pode desejar, para a implementação da interface **Patient**, que seus objetos, quando instanciados em modo *Requesting*, solicitem interativamente do objeto da aplicação as referências para o objeto **Nurse** e para o objeto **Logger**. É responsabilidade do projetista fazer com que o objeto **Patient** registre estas referências no seu *RefManager* para que as mesmas participem do mecanismo de gerenciamento de referências de *ConfigApplication*.

Como se tratam de objetos *ConfigInstance* é necessário que o projetista implemente a operação definida para a transferência de estado (a função gancho) entre dois objetos daquele tipo. A implementação dos *factories* respectivos é também necessária.

Após a implementação dos *factories* é preciso instanciá-los em todos os *hosts* do ambiente computacional. No caso de um ORB operando em um ambiente heterogêneo será preciso implementar versões destes *factories* para cada um destes ambientes. Note, no entanto, que a necessidade destes *factories* distintos pode ser eliminada no caso do servidor ser implementado em Java.

Por fim, deve-se criar um objeto *ConfigApplication*, por intermédio do serviço *ApplicationContainer* e registrar neste objeto todos os *tipos* de componentes da aplicação. Este será o representante da aplicação no ambiente de gerenciamento.

O Gerenciamento da Aplicação

Uma vez desenvolvida e instalada, a aplicação poderá ser gerenciada. Vamos supor que a aplicação apresente a configuração apresentada na Figura 6.11 em um dado instante.

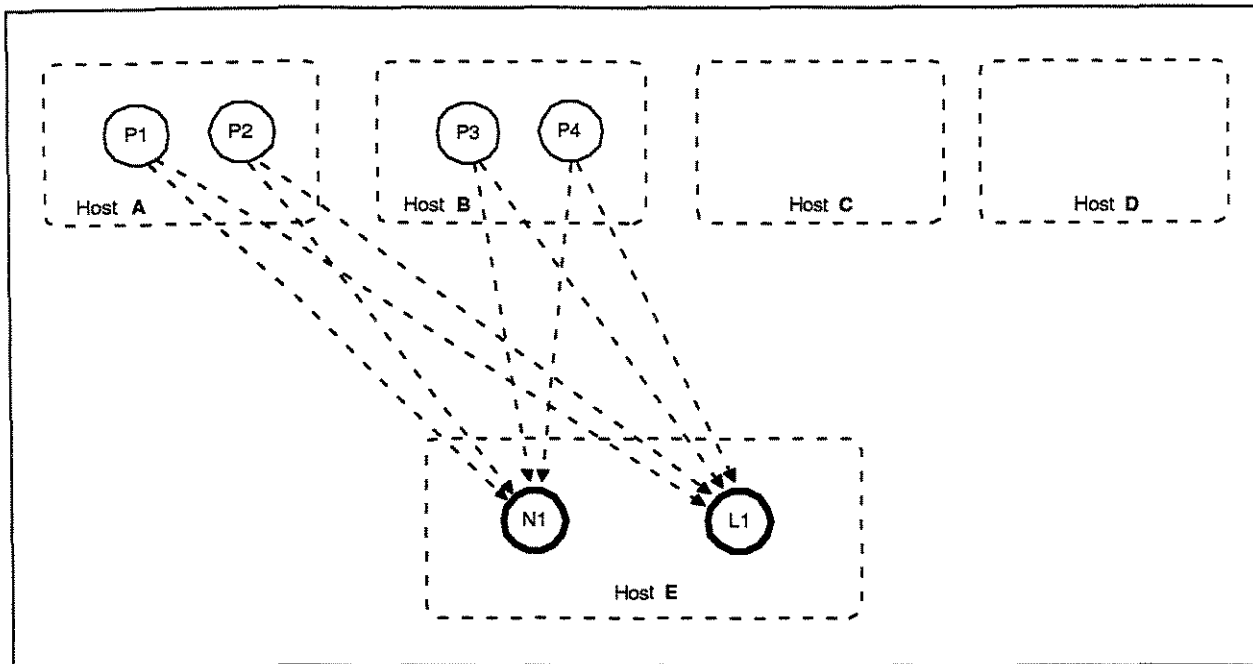


Figura 6.11: Configuração da Aplicação no Instante 1.

A configuração apresenta quatro pacientes (P1, P2, P3, P4) sendo monitorados por uma enfermeira (N1) e registrando informações no **Logger** L1.

A partir desta configuração, as seguintes operações de gerenciamento poderiam ser aplicadas:

- Criação de um novo objeto **Nurse** (N2) para balancear a carga de N1.
- Mediante a transferência de um paciente (P2) para um leito monitorado pela estação C, moveu-se P2 para a estação C.
- A monitorização de P2 passa a ser responsabilidade de N2. Para tanto, efetua-se um “*rebind*” de referências, substituindo-se P1 por P2.
- Chegada de um novo paciente em um leito monitorado pela estação D. Por isso foi criado um novo objeto **Patient** em D. Este paciente foi criado interativamente em modo de *Requesting* e a referência para N2 foi passada para ele.

O resultado para esta configuração final pode ser visto na Figura 6.12.

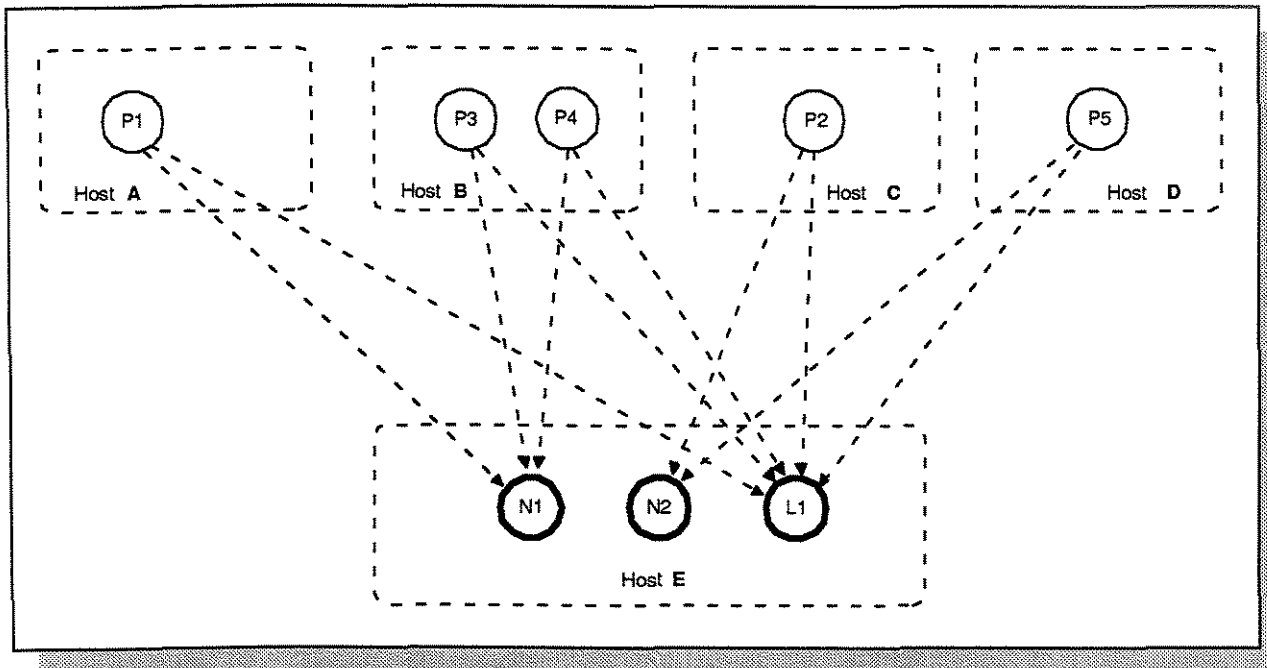


Figura 6.12: Configuração da Aplicação no Instante 2.

Capítulo 7

Conclusões

Nesta dissertação procuramos analisar os diversos aspectos do problema de gerenciamento de sistemas distribuídos, em particular, no contexto das aplicações CORBA. Como comentado, este tipo de atividade está prevista para ser foco das atenções na segunda fase da retomada dos trabalhos em gerenciamento no OMG.

Realizando o objetivo principal do trabalho, projetamos e implementamos uma Facilidade de Gerenciamento de Configuração contribuindo com novos conceitos que permitiram o seu desenvolvimento a partir da especificação XCMF.

A especificação XCMF apresenta um *framework* para desenvolvimento e modelagem de recursos do ambiente de computação capazes de serem gerenciados. Com relação às operações de ciclo de vida destes objetos modelados (Instâncias), a especificação XCMF trata quase que exclusivamente dos problemas envolvidos com o processo de criação. Nesta especificação, as instâncias destes objetos são gerenciadas individualmente.

Este trabalho apresentou o projeto de uma Facilidade para desenvolvimento de aplicações de gerenciamento que estende o *framework* XCMF desenvolvendo um mecanismo que possibilita o gerenciamento conjunto de um grupo destas instâncias. Desta forma foi possível completar o suporte a operações de ciclo de vida que afetam o ambiente de gerenciamento, como a movimentação, e providenciar o suporte para operações necessárias ao controle da configuração deste grupo de objetos que passam agora a compor uma aplicação distribuída.

Podemos, portanto, resumir a principal contribuição deste trabalho: A extensão da especificação XCMF para permitir o gerenciamento dos objetos em grupo com a definição de todas as operações de ciclo de vida destes objetos. Estes grupos de objetos, identificados agora como aplicações distribuídas serão capazes de receber as operações de gerenciamento de configuração dinamicamente e interativamente. Os objetos serão capazes de:

- sofrer operações de ciclo de vida (cópia, movimentação e remoção);
- fornecer informações a respeito de seu estado e de seus relacionamentos, permitindo ainda que tais relacionamentos sejam administrados;
- sofrer operações específicas de configuração tais como a substituição de implementação e o “*rebind*” de referências.

Podemos resumir como principal resultado dos trabalhos de implementação do modelo: O padrão CORBA fornece um modelo de objetos rico para a especificação de interfaces de objetos, mas não suporta adequadamente a configuração estrutural dos objetos. Neste contexto, o processo desenvolvido para gerenciar a configuração de um conjunto de objetos inter-relacionados é adequado e eficiente.

7.1 **Trabalhos Futuros**

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

- Desenvolvimento de um mecanismos de controle e recuperação de falhas incorporados à estrutura definida para o ambiente de gerenciamento;
- Integração da Facilidade de Monitorização já desenvolvida para a plataforma Multiware à estrutura montada para o ambiente de gerenciamento;
- Substituição do modelo de suporte XCMF, adaptando a Facilidade para padrões atuais como os *CORBA Components*.

Referências Bibliográficas

- [BC95] S. Bandeira e P. Cunha. Suporte à Execução do Ambiente DisCo. *XIII SBRC - Simpósio Brasileiro de Redes de Computadores*, 1995. pp. 619-636.
- [Bat00] T. V. Batista e N. Rodriguez. Configuração de Aplicações Distribuídas no LuaSpace. *XVIII SBRC - Simpósio Brasileiro de Redes de Computadores*, 2000. pp. 169-182.
- [BEA] BEA System. *ObjectBroker and Iceberg*. <http://www.beasys.com>.
- [Bor] Borland/Visigenic. *VisiBroker*. <http://www.visigenic.com>.
- [CM96] F. M. Costa e E.R.M. Madeira. An object group model and its implementation to support cooperative applications on CORBA. *Distributed Plataforms. Chapman & Hall*, 1996. pp. 213-229.
- [Endl92] M. Endler e J. Wei. Programming Generic Dynamic Reconfigurations for Distributed Applications. *Proc. of the International Workshop on Configurable Distributed Systems*, IEE Março 1992. pp. 68-79.
- [Fos97] H. Fossa. Interactive Configuration Management for Distributed Systems. 1997. PhD Thesis. Dept. of Computing, Imperial College, London. <ftp://dse.doc.ic.ac.uk/dse-papers/management/FOSSA-THESIS.PS.GZ>.
- [HS98] Paul Haggerty e Krishnan Seetharaman. The Benefits of CORBA-Based Network Management. *Communications of the ACM*, 1998. Vol. 41, n. 10, pp. 73-79.
- [HA94] H.G. Hegering e S. Abeck. Integrated Network and System Management. *Data Communication and Network Series*, 1994.
- [IBM] IBM. *Component Broker*. <http://www.software.ibm.com/ad/cb>.

- [ION] IONA Technologies. *Orbix e OrbixWeb*. <http://www.iona.com>.
- [ION98] IONA Technologies. *OrbixWeb Reference Guide*, Setembro 1998.
- [jac] jacORB. *Projeto jacORB*. <http://www.inf.fu-berlin.de/brose/jacorb>.
- [Jav] JavaSoft. *Java IDL*. <http://www.javasoft.com>.
- [Lan94] A. Langsford. OSI Management Model and Standards. *Network and Distributed Systems Management*, 1994. Cap. 4, pp. 69-93.
- [LM95] L. A. P. Lima e E.R.M. Madeira. A Model for a Federative Trader. *ICODP - International Conference on Open Distributed Processing*, Fevereiro 1995. pp. 155-166.
- [LM97] F. H. S. Lima e E.R.M. Madeira. Qualidade de Serviço Multinível baseada em ATM para a Plataforma Multiware. *XV SBRC - Simpósio Brasileiro de Redes de Computadores*, Maio 1997. pp. 366-382.
- [LMM⁺94] W.P.C. Loyolla, E.R.M. Madeira, M.J. Mendes, E. Cardozo, e M.F. Magalhães. Multiware Platform: An Open Distributed Environment for Multimedia Cooperative Applications. *IEEE Computer Software and Applications Conference. COMPSAC'94*, Novembro 1994. Taipei, Taiwan.
- [Mag94] J. Magee. Configuration of Distributed Systems. *Network and Distributed Systems Management*, 1994. Cap. 18, pp. 483-497.
- [MaS97] Projeto MaScOTTE. Introduction to MaScOTTE - white paper, version 2.0. Maio 1997. <http://www.esrin.esa.it/MAScOTTE>.
- [MIC] MICO. *Projeto MICO (MICO Is CORBA)*. <http://www.mico.org>.
- [MP95] B. Meyer and C. Popien. Flexible Management of ANSAware Applications. *In Proceedings of ICODP'95*, 1995. pp. 271-282.
- [OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Distributed Objects - Survival Guide*. John Wiley & Sons, Inc, 1996.
- [OMG00] Distributed Instrumentation and Control For ORBs and Services Request For Proposal. Março 2000. <http://www.omg.org/cgi-bin/doc?test/00-03-01.doc>.
- [OMG97b] Object Management Group. CORBAfacilities: Common Facilities Architecture, rev. 4.0. Novembro 1997.

- [OMG97c] Object Management Group. CORBAservices: Common Object Services Specification. Março 1997.
- [OMG97a] Object Management Group. A Distributed Object Management Architecture. Janeiro 1997.
- [OMG98] Object Management Group. The Common Object Request Broker: Architecture and Specification. Fevereiro 1998. Revision 2.2.
- [Pav99] Juan Pavón. Building Telecommunications Management Applications with CORBA. *IEEE Communications Surveys*, 1999.
- [Que97] João Augusto Gomes Queiroz. Gerência de Sistemas Distribuídos Heterogêneos: Facilidade de Monitorização em um Ambiente CORBA. Dezembro 1997. Dissertação de Mestrado. Instituto de Computação - UNICAMP, Campinas - SP.
- [Ray95] K. Raymond. Reference Model of Open Distributed Processing (RM-ODP): Introduction. *In Proceedings of ICODP'95*, 1995. pp. 3-14.
- [Rop99] Patrícia Ropelatto. Gerência de Monitorização de Sistemas Distribuídos em um Ambiente CORBA usando Agentes Móveis. Junho 1999. Dissertação de Mestrado. Instituto de Computação - UNICAMP, Campinas - SP.
- [SLC95] Luiz Fernando G. Soares, Guido Lemos, e Sérgio Colcher. Das LANs MANs WANs às Redes ATM. 1995. Segunda Edição.
- [Slo95] M. Sloman. Management Issues for Distributed Services. *IEEE Second International Workshop on Services in Distributed and Networked Environments (SDNE 95)*, 1995. <ftp://dse.doc.ic.ac.uk/dse-papers/management/sdne95.ps.Z>.
- [SM99a] C. M. Silveira e E. R. M. Madeira. A Configuration Facility for CORBA Applications. *Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems - DAIS'99*, Junho 1999. Helsinki, Finland. pp. 243-257
- [SM99b] C. M. Silveira e E. R. M. Madeira. Uma Facilidade de Gerenciamento de Configuração para Aplicações CORBA. *XVII SBRC - Simpósio Brasileiro de Redes de Computadores*, Maio 1999. Salvador, Bahia. pp. 483-498

- [SMTK93] M. Slomam, J. Magee, K. Twidle, e J. Kramer. An Architecture for Managed Distributed Systems. *Proceedings 4th IEEE Workshop on Future Trends of Distributed Computing Systems*, Setembro 1993. pp. 40-46.
- [Vas99] Francisco J. S. Vasconcellos. Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA. Abril 1999. Dissertação de Mestrado. Instituto de Computação - UNICAMP, Campinas - SP.
- [XOp97] System Management: Common Management Facilities. 1997. (ISBN: 1-85912-174-8, C423). <http://www.omg.org/cgi-bin/doc?formal/97-06-08>.

Apêndice A

Descrições IDL da Facilidade de Configuração

A.1 Modulo ConfigFacility

```
//-----  
//typedefs  
//-----  
enum ComponentType { cfg_refmanager, cfg_object, cfg_instance};  
enum RefClass { open_ref, manageable_ref, fixed_ref, composite_ref };  
enum RefEstate { normal , suspended};  
typedef sequence < ConfigObject > ConfigObjectList;  
  
struct Reference{  
    string local_id;  
    string obj_label;  
    Object obj_ref;  
    RefClass ref_class;  
    RefEstate ref_estate;  
};  
  
typedef sequence <Reference> ReferenceList;  
  
struct Component{  
    RefManager refmanager;  
    string label;  
    ComponentType type;
```

```

    string id_interface; //principal interface IDL
    string id_impl;
};

typedef sequence <Component> ComponentList;

struct Dependence{
    Component comp;
    Reference reference;
};

typedef sequence <Dependence> DependenceList;

struct RequestRef{ //ReadOnly
    string local_id;
    RefClass ref_class;
    string id_interface;
    string id_implementation;
    string host;
};

typedef sequence <RequestRef> RequestRefList;

struct InterfaceImpl {
    string id_interface;
    string id_implementation;
};

typedef sequence <InterfaceImpl> InterfaceImplList;
typedef sequence <ConfigListener> ConfigListenerList;

//-----
//exceptions
//-----
exception CFGNotification {string reason;};
exception CFGInvalidLabel {string reason;};
exception CFGNotFound {string reason;};
exception CFGNotEmpty {string reason;};
exception CFGInvalidType {string reason;};

```

```

exception CFGCannotCreate {string reason;};
exception CFGCannotOperate {string reason;};
exception CFGTimeout {string reason;};
exception CFGInvalidCriteria {string reason;};
exception CFGInvalidRequest {string reason;};

//-----
//ConfigApplication
//-----

interface ConfigApplication: ManagedSets::FilteredSet {

    //Administração de Labels
    boolean is_valid_label(in string label);
    string get_valid_label(in string base_label);
    string reserve_label(in string base_label);

    //Registro de informações
    void register_ConfigInstanceType( in string id_interface, in string id_implementation)
        raises (CFGInvalidType);

    //Controle de Componentes
    void add_client(in Component comp)
        raises (CFGInvalidLabel, CFGInvalidType);
    void add_config_object(in Component comp)
        raises (CFGInvalidLabel, CFGInvalidType);
    void add_listener(in ConfigListener listener);
    void remove_client(in Component comp)
        raises (CFGNotFound);
    void remove_config_object(in Component comp) raises (CFGNotFound);
    void remove_listener(in ConfigListener listener) raises (CFGNotFound);

    //Recuperação de informações
    ManagedInstances::Library get_library();
    InterfaceImplList get_registred_types();
    ConfigListenerList get_listeners();
    ComponentList get_all_components();
    ComponentList get_components(in ComponentType type) raises (CFGNotFound);
    DependenceList get_dependence_list(in Component comp) raises(CFGNotFound);
    ReferenceList get_reference_list(in Component comp) raises(CFGNotFound);
    Reference get_reference(in Component comp, in string local_id)
        raises(CFGNotFound);

```



```
Component localize_component(in string id_label) raises (CFGNotFound);
Component get_uptodate_comp(in string label)
    raises (CFGNotFound, CFGCanNotOperate);
ComponentList localize_components (in string id_interface, in string
    id_implementation, in string host_name) raises (CFGNotFound);

//Operações de Configuração
Component create_instance( in string id_label, in string host_name,
    in string id_interface, in string id_implementation)
    raises (CFGCanNotCreate,CFGInvalidLabel, CFGInvalidType);
Component create_instance_ex( in string id_label, in string host_name,
    in string id_interface, in string id_implementation,
    in CosLifeCycle::Criteria criteria)
    raises(CFGCanNotCreate,CFGInvalidLabel,CFGInvalidType,CFGInvalidCriteria);
Component create_interative_instance( in string id_label, in string host_name,
    in string id_interface, in string id_implementation, in ConfigListener listener)
    raises(CFGNotFound,CFGCanNotCreate,CFGInvalidLabel,CFGInvalidType);
void move_instance (in ConfigInstance ci, in string host_name)
    raises(CFGCanNotOperate);
void copy_instance (in ConfigInstance ci, in string label, in string host_name)
    raises(CFGCanNotOperate, CFGInvalidLabel);
void rebind (in RefManager rm, in Reference reference, in Component newcomp)
    raises(CFGCanNotOperate);
void replace_implementation(in ConfigInstance obj, in string implementation)
    raises(CFGCanNotOperate);

//Operações Auxiliares as operações de Configuração de ConfigObject,
//ou mesmo de ConfigInstance
boolean suspend_all_dependences(in Component comp);
void restore_all_dependences(in Component comp);

//Operações para notificação de eventos ocorridos nos componentes
void notify_endmove(in Component comp, in ConfigInstance newobj)
    raises (CFGNotification);
void notify_endcopy(in Component newcomp, in ReferenceList reflist)
    raises (CFGNotification);
void rm_notify_register_ref(in Component comp, in Reference ref)
    raises (CFGNotification);
void rm_notify_remove_ref(in Component comp, in Reference ref)
    raises (CFGNotification);
void rm_notify_remove_all(in Component comp)
    raises (CFGNotification);
```

```

void ci_notify_iterative_init(in Component comp, in ReferenceList refflist)
    raises (CFGNotification);
void ci_notify_remove(in Component comp);
};

//-----
// RefManager
//-----

interface RefManager : ManagedSets::Member {

    //Recuperação de informação
    ReferenceList get_all_references();
    ReferenceList get_references(in RefClass refclass);
    Reference get_reference(in string name) raises(CFGNotFound);
    ConfigApplication get_config_app();
    Component describe_component();

    //Funções de Recuperação
    void update_references() raises (CFGCanNotOperate);
    void update_reference(in Reference reference)
        raises (CFGCanNotOperate, CFGNotFound);

    //Operações de notificação
    oneway void ap_notify_suspend_ref(in Reference reference);
    oneway void ap_notify_restore_ref(in Reference reference);
    oneway void ap_notify_remove(in Reference reference);
    oneway void ap_notify_update_ref (in Reference oldreference,
        in Component newcomp);
    void ap_notify_rebind (in Reference oldreference, in Component newcomp)
        raises (CFGNotification);

    //Administração de Referências. Estas operações devem estar
    //disponíveis na implementação do RefManager.
    //Elas somente estão ilustradas aqui e
    //não devem estar presentes na declaração IDL para acesso remoto.
    void register_ref(in Reference reference)
        raises (CFGInvalidLabel, CFGCanNotOperate, CFGNotification);
    void remove_ref(in string name) raises(CFGNotFound, CFGNotification);
    void remove_all() raises(CFGNotification);

```

```

    boolean ready_to_call(in string name);
    long set_timeout(in long timeout);
    Object get_uptodate_ref(in string name)
        raises(CFGNotFound, CFGTimeout);
};

//-----
//ConfigObject
//-----

interface ConfigObject : RefManager {

    //Funções “Ganchos” chamadas durante as operações de
    //ciclo de vida para dar ao usuario a oportunidade de transferir o
    //estado atual da instancia para o novo objeto.
    //Estas operações devem estar disponíveis na implementação do
    //ConfigObject. Elas somente estão ilustradas aqui e
    //não devem estar presentes na declaração IDL para acesso remoto.
    boolean move_estate(ConfigInstance new_instance);
    boolean copy_estate(ConfigInstance new_instance);

    //Responsável por limpar e liberar o objeto antes da sua remoção.
    void clear_estate();

    void notify_set_referencelist(in ReferenceList reflist);
    string get_owner_label();
};

//-----
//ConfigInstance
//-----

interface ConfigInstance : ManagedInstances::Instance, ConfigObject {

    void replace_implementation(in ConfigInstanceManager new_manager,
        in CosLifeCycle::Criteria the_criteria)
        raises (CFGInvalidType, CosLifeCycle::NotMovable,
            CosLifeCycle::NoFactory, CosLifeCycle::InvalidCriteria);
    boolean is_initialized();
    void cancel_interative_creation();
};

```

```

RequestRefList get_interative_request();
void set_interative_request(in ReferenceList refflist,
    in ConfigListener listener)
    raises (CFGInvalidRequest, CFGCannotOperate);
};

//-----
//ConfigInstanceManager
//-----

interface ConfigInstanceManager : ManagedInstances::BasicInstanceManager
{
    InterfaceImpl get_interface_impl();
    ConfigInstance create_temporary_instance(in string label,
        in CosLifeCycle::Criteria the_criteria)
        raises (CFGNotFound, CosLifeCycle::NoFactory,
            CosLifeCycle::InvalidCriteria, CosLifeCycle::CannotMeetCriteria);
    void replace_temporary_to_instance(in string instance_label,
        in ConfigInstance temporary );
    void elimine_temporary(in ConfigInstance temporary );
};

//-----
//ConfigListener
//-----

interface ConfigListener {
    oneway void notify_new_component(in Component comp);
    oneway void notify_remove_component(in Component comp);
    oneway void notify_rebind(in Component comp, in Reference ref,
        in Component new_refcomp);
    oneway void notify_move(in Component comp);
    oneway void notify_copy(in Component comp, in ReferenceList refflist);
    oneway void notify_register_reference(in Component comp,
        in Reference ref, in Component refcomp);
    oneway void notify_remove_reference(in Component comp,
        in Reference ref);
};

```

```
oneway void notify_remove_all_references(in Component comp);  
oneway void notify_remove_instance(in Component comp);  
oneway void notify_interative_init(in Component comp,  
    in ReferenceList refflist);  
};
```