Um Mecanismo para Prover Interoperabilidade entre ORBs com Suporte à Transparência de Relocação

Nuccio M. S. Zuquello

Um Mecanismo para Prover Interoperabilidade entre ORBs com Suporte à Transparência de Relocação[1]

Nuccio M. S. Zuquello[2]

Instituto de Computação
IC – UNICAMP

Banca Examinadora:

- Edmundo Roberto Mauro Madeira (Orientador)[3]
- Eleri Cardozo [4]
- Rogério Drummond Burnier Pessoa de Mello Filho[3]
- Luiz Eduardo Buzato (Suplente)[3]

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

[2]O autor é Bacharel em Ciências do Mar (Habilitação em Análise de Sistemas de Armas) formado pela Escola Naval.

[3]Professor do Instituto de Computação - IC - UNICAMP.

[4]Professor da Faculdade de Engenharia Elétrica e Computação - FEEC - UNICAMP

Um Mecanismo para Prover Interoperabilidade entre ORBs com Suporte à Transparência de Relocação

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Nuccio M. S. Zuquello e aprovada pela Comissão Julgadora.

Campinas, 14 de maio de 1997.

Prof. Dr. Edmundo Roberto Mauro Madeira
Orientador

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

Tese de Mestrado defendida e aprovada em 14 de maio de 1997 pela Banca Examinadora composta pelos Professores Doutores

Prof. Dr. Eleri Cardozo

Prof. Dr. Rogério Drummond Burnier Pessoa de Mello Filho

Prof. Dr. Edmundo Roberto Mauro Madeira

Aos meus pais, que me ensinaram as coisas boas que sei.

Agradecimentos

A Deus, por tudo.

À Marinha, pela oportunidade.

Ao Prof. Edmundo, pela orientação e paciência inesgotável.

Ao Fabio e ao Augusto, pelas conversas "intermináveis" e valiosas sugestões.

A Fátima, que me ensinou a andar na chuva novamente.

A Karen e George, amizades incríveis, "... haverá sempre um instante entre o Aqui e o Agora ..."

A Verônica, Cris e Ronaldo, amigos que conquistaram um espaço especial.

A Yvonne, muito mais que amiga, sempre presente ainda que muitas vezes distante.

A tantas pessoas que aqui encontrei e que marcaram um período inesquecível de minha vida, suas lembranças permanecerão comigo para sempre. Espero que continuemos nos reencontrando e que, a cada novo encontro, nossa amizade prossiga ainda maior.

Aos amigos de Praça D'Armas: Marques, Marcus Vinicius, Lucio, Augusto, Pagliusi e Francisco, com quem tive o privilégio de compartilhar esse novo desafio, em um ambiente tão distante da vida naval.

... nem cora o livro de ombrear com o sabre,
nem cora o sabre de chamá-lo irmão ...

Castro Alves

Resumo

A capacidade de sistemas heterogêneos cooperarem é uma necessidade atual que vem se impondo cada vez mais, principalmente pelo rápido desenvolvimento da tecnologia de processamento distribuído, obrigando a busca de soluções mais apropriadas. Nesse contexto, a especificação CORBA (*Common Object Request Broker Architecture*) apresenta uma arquitetura que permite a interoperabilidade entre aplicações em ambientes distribuídos heterogêneos. Entretanto, pela sua grande flexibilidade nas decisões de implementação, surgem problemas para estender essa interoperabilidade entre dois ORBs desenvolvidos por diferentes tecnologias. Nesta dissertação são discutidos esses problemas e suas respectivas soluções, sendo apresentado um conjunto de operações para suportar o mecanismo de interceptação, estendendo as transparências de acesso e localização, consideradas fundamentais num ambiente distribuído, para além do escopo de um ORB. É proposta, também, uma forma para prover transparência de relocação, estendida para suportar interoperabilidade. Por fim, é descrita uma implementação desses mecanismos utilizando um protótipo aberto de ORB e uma implementação comercial, chamada ORBeline.

Abstract

The ability of heterogeneous systems cooperate to problem's solutions is a real need that has been growing, due to the increasing development of the distributed processing technology. This need is driving the development of more suitable solutions. The CORBA (*Common Object Request Broker Architecture*) specification presents an architecture which allows interoperability among heterogeneous distributed environment applications. However, because of the great flexibility offered in implementation decisions, many problems to extend interoperability between two ORBs developed by different technologies arise. This work discusses these problems and their solutions and presents a set of operations to support an interception mechanism extending access and location transparencies, considered as fundamental for a distributed environment, to work beyond an ORB scope. A scheme to provide relocation transparency, extended to support interoperability is also proposed. Finally, an implementation of these mechanisms over an open ORB's prototype and ORBeline, a commercial product, is described.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Siglas

ANSA	Advanced Networked Systems Architecture
ANSI	American National Standards Institute
API	Application Programming Interface
BDOO	Banco de Dados Orientado a Objetos
BEO	Basic Engineering Object
BOA	Basic Object Adapter
CAD	Computer Aided Design
CDR	Common Data Representation
CORBA	Common Object Request Broker Architecture
COSS	Common Object Services Specification
CUT	Coordinated Universal Time
DCE	Distributed Computing Environment
DII	Dynamic Invocation Interface
DIR	Dynamic Implementation Routine
DOME	Distributed Object Management Environment
DSI	Dynamic Skeleton Interface
DTP	Distributed Transaction Processing
GIOP	General Inter-ORB Protool
GUI	Graphical User Interface
IDL	Interface Definition Language
IED	Interface de Esqueleto Dinâmico
IEEE	Institute of Electrical and Electronic Engineers
IID	Interface de Invocação Dinâmica
IIOP	Internet Inter-ORB Protool
IP	Internet Protocol
IPC	Interprocess Communication
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union - Telecommunication Standard Sector
NVList	NamedValue List

oad	object ativation daemon
ODP	Object Distributed Processing
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
OSA	Object Service Architecture
OSI	Open Systems Interconnection
osagent	ORBeline's smart agent
OSF	Open Software Foundation
refobj	referência de objeto
RFP	Request for Proposal
RM-ODP	Reference Model for Open Distributed Processing
RPC	Remote Procedure Call
SQL	Structured Query Language
TCP	Transport Control Protocol
TLI	Transport Layer Interface
UNO	Universal Networked Objects
WAN	Wide Area Network
XDR	eXternal Data Representation

Capítulo 1

Introdução

A distribuição da informação é uma realidade que vem se afirmando cada vez mais. O desenvolvimento de microprocessadores, possibilitando a construção de máquinas cada vez menores, mais poderosas e de menor custo, e de redes locais de alta velocidade, garantindo a interconexão dessas máquinas, levou ao surgimento de sistemas distribuídos. Mais recentemente, tem-se o avanço da interconexão de redes de longo alcance com redes locais e o avanço proporcionado pelas tecnologias de redes de alto desempenho, permitindo o acesso e o transporte de grandes volumes de dados. Considerando-se os tradicionais sistemas centralizados, a distribuição oferece, principalmente, uma melhor relação entre custo e desempenho, maior confiabilidade e maior disponibilidade [Tan92].

Os novos desafios advindos das atuais necessidades dos usuários referem-se não apenas à busca e ao armazenamento da informação mas também à efetiva utilização dessa informação de forma distribuída. A capacidade de processar os dados obtidos localmente ou remotamente em computadores situados em localizações geográficas distintas passa a ser altamente desejável. Torna-se cada vez mais evidente a importância em distribuir tanto os dados quanto o seu processamento.

As empresas, devido à contínua emergência de novas tecnologias, têm dificuldade na integração das novas soluções às já adotadas, acabando por manter, num mesmo ambiente organizacional, vários sistemas e "softwares" incompatíveis. Isso acaba por criar "ilhas" de informações dentro da própria organização, impossibilitando o acesso e utilização desses dados de forma automática.

Apesar das grandes vantagens dos sistemas distribuídos o seu desenvolvimento se depara com problemas que se tornam críticos numa distribuição em larga escala: heterogeneidade, gerenciamento de configuração, contabilidade, possibilidade de expansão do sistema e gerenciamento da rede. Além disso, considera-se altamente desejável que os detalhes decorrentes da distribuição não sejam tratados diretamente pelo usuário, o que tornaria bastante complexo o desenvolvimento ou uso de qualquer aplicação, de onde surge o conceito de transparência de distribuição.

Nesse contexto, surge a computação de objetos distribuídos (DOC[1]), usando a tecnologia de

[1] *Distributed Object Computing.*

orientação a objetos para tratar com as dificuldades inerentes à distribuição.

Os conceitos que caracterizam essa tecnologia, como abstração, encapsulamento, reusabilidade, portabilidade e modularidade, acrescido do fato das interações entre objetos serem efetuadas exclusivamente através de mensagens, torna bastante natural sua utilização em sistemas distribuídos [Vin93, NWM93, Sol94, ISO95a].

Um dos objetivos básicos do DOC é permitir aos desenvolvedores de aplicações distribuídas a utilização de técnicas de programação usuais, como chamadas a métodos em objetos [Vin93].

Essa abordagem está expandindo rapidamente, sendo demandado bastante esforço no seu desenvolvimento, tanto em organizações internacionais, como a ISO[2] e a ITU-T[3], quanto em organizações comerciais, como a APM[4] e a OSF[5].

A complexidade advinda dos sistemas distribuídos torna-se ainda maior com o acréscimo da capacidade de abertura, permitindo que qualquer usuário (um ser humano, um componente ou uma aplicação), resguardados os requisitos de segurança, possa ser admitido no sistema.

A ISO e a ITU-T estão em fase adiantada de elaboração de um Modelo de Referência para o Processamento Distribuído Aberto (RM-ODP[6]) com o objetivo de definir uma infra-estrutura para a padronização do processamento distribuído aberto, de forma a permitir a integração, de forma transparente, de componentes heterogêneos.

Existem outras propostas, não tão abstratas quanto o RM-ODP, para o suporte ao desenvolvimento do processamento distribuído aberto, destacando-se: DCE[7], CORBA[8] e ANSA[9]. A plataforma DCE é orientada a procedimentos, enquanto que as plataformas ORB e ANSAware são orientadas a objetos. Os conceitos básicos para a criação do RM-ODP nasceram com a ANSA. A CORBA, uma iniciativa da OMG[10], a princípio totalmente independente do modelo de referência, atualmente tem seu desenvolvimento seguindo alguns dos conceitos preconizados pela ISO/ITU-T.

Está em desenvolvimento atualmente na Unicamp, uma plataforma chamada de Multiware, na qual é apresentada uma arquitetura para suporte à construção de sistemas distribuídos abertos usando como base para desenvolvimento as plataformas anteriormente citadas. Sua arquitetura provê uma camada, entre essas plataformas e as aplicações, a qual incorpora as funcionalidades preconizadas no RM-ODP e que não são fornecidas pelas plataformas utilizadas [MM94, LMM+94, Mad95].

Para início do desenvolvimento foi escolhida a CORBA, por obedecer ao paradigma da orientação a objetos e por sua característica modular, permitindo a agregação incremental de novos serviços. Além disso, é uma plataforma que está em pleno desenvolvimento, sendo despendido

[2] *International Organization for Standardization.*
[3] *International Telecommunication Union - Telecommunication Standardization Sector.*
[4] *Architecture Projects Management.*
[5] *Open Software Foundation.*
[6] *Reference Model for Open Distributed Processing.*
[7] *Distributed Computing Environment.*
[8] *Common Object Request Broker Architecture.*
[9] *Advanced Networked Systems Architecture.*
[10] *Object Management Group.*

um esforço muito grande na sua elaboração. O consórcio que forma a OMG consta de mais de 600 organizações, entre as quais IBM, HP, Sun, Digital e Microsoft.

A especificação de sua arquitetura permite um alto grau de liberdade na escolha da implementação, causando dificuldades para a *interoperabilidade*, que consiste na capacidade de um cliente, num ORB, solicitar serviços fornecidos por um objeto em outro ORB. No atual estágio das implementações, dois ORBs desenvolvidos por diferentes fabricantes interoperam de forma restrita através de protocolos proprietários ou através de um protocolo específico para TCP/IP[11], a serem apresentados adiante.

A interoperabilidade pode ser vista como a capacidade de estender as transparências de distribuição para além do escopo de um único ORB [OMG94k]. Para atender esse objetivo alguns problemas inerentes à interoperabilidade devem ser superados, principalmente no que diz respeito à correta compreensão das referências de objetos entre ORBs que as implementam de formas diferentes. Essas referências, que identificam objetos de forma confiável no escopo de um ORB, possibilitam uma grande flexibilidade para a implementação desses ORBs, permitindo aos diversos fabricantes escolherem a tecnologia que consideram mais apropriada. O projeto da estrutura e mecanismos envolvidos com a referência de objeto é considerado crucial para o desempenho do ORB [Pow93].

A especificação do ORB já provê as transparências de acesso e de localização, consideradas fundamentais em um ambiente distribuído. Como uma extensão da transparência de localização surge a transparência de relocação, escondendo do usuário as movimentações que o objeto fornecedor do serviço possa efetuar entre suas invocações. A relocação pode ser necessária para balanceamento de carga, por exemplo. Dessa forma, a transparência de relocação evita o inconveniente da aplicação tratar diretamente com a complexidade inerente a essa necessidade. Atualmente, a CORBA não aborda essa transparência completamente.

Para suportar interoperabilidade, a OMG definiu os componentes básicos necessários com as devidas alterações na especificação CORBA existente, bem como um protocolo, chamado Protocolo Geral Entre-ORBs (GIOP[12]), o qual foi padronizado para TCP/IP. Esse protocolo forma a base para a interoperabilidade provendo uma forma comum, intermediária, que deve ser obedecida por ORBs que desejam interoperar. Também está padronizado um protocolo (DCE-ESIOP[13]), otimizado para ORBs que utilizam a tecnologia DCE. Esses protocolos abstraem quais os componentes dos ORBs serão os responsáveis pelas mensagens enviadas/transmitidas [OMG94k].

Baseando-se, principalmente, nos objetos e conceitos empregados pelo RM-ODP, e respectivos mapeamentos para a CORBA, esta dissertação apresenta uma proposta para suportar a transparência de relocação bem como o detalhamento de um mecanismo para permitir interoperabilidade, as operações necessárias, e respectivas extensões às operações e serviços já existentes. Para validar essas idéias é apresentado um protótipo implementado a partir dos conceitos expostos, utilizando-se de um ORB com funcionalidades básicas, desenvolvido em [dM95], e baseado

[11] *Transmission Control Protocol/Internet Protocol.*
[12] *General Protocol Inter-ORBs.*
[13] *Environment Specific IOP* para DCE.

em RPC[14], e uma plataforma disponível comercialmente, chamada ORBeline.

1.1 Estrutura da dissertação

No Capítulo 2, onde são apresentados os conceitos sobre o Modelo de Referência para Processamento Distribuído Aberto (RM-ODP), com ênfase aos aspectos referentes à relocação, à transparência de relocação e à interoperabilidade; e a descrição da plataforma Multiware, em linhas gerais. No Capítulo 3 são expostos os conceitos utilizados pela CORBA com ênfase às necessidades decorrentes dos mecanismos de interoperabilidade e de transparência de relocação. São descritas duas plataformas comerciais que implementam a CORBA: a Orbix, em linhas gerais, e a ORBeline, de uma forma mais detalhada, pois faz parte do protótipo implementado neste trabalho. É feita, também, uma pequena comparação entre as funcionalidades da ORBeline e as preconizadas pelo RM-ODP. O estado atual de desenvolvimento dos Serviços de Objetos e das Facilidades Comuns, com um pequeno resumo das suas funções, é apresentado ao final. A interoperabilidade entre ORBs é abordada no Capítulo 4, onde são descritos os problemas a serem superados para a obtenção de interoperabilidade, algumas soluções existentes e uma descrição detalhada de um mecanismo para suportar interoperabilidade e suas respectivas operações. É apresentado, também, um modelo com as funcionalidades que podem ser desempenhadas pelo Interceptador, o qual constitui o núcleo do mecanismo. A descrição de um mecanismo para suportar a transparência de relocação num ambiente constituído por um único ORB ou por ORBs interoperantes aparece no Capítulo 5. Os mecanismos descritos nos capítulos 4 e 5 são utilizados na implementação de um caso de invocação de objetos remotos suportando interoperabilidade e transparência de relocação entre o ORB desenvolvido em [dM95] e a ORBeline. Essa implementação é detalhada no Capítulo 6. A conclusão deste trabalho, com as contribuições e sugestões para futuras pesquisas, é exposta no Capítulo 7.

Como apêndice ao trabalho foi acrescentada a descrição do Protocolo Geral Entre-ORBs (GIOP), com sua Representação Comum de Dados (CDR[15]) e os formatos padronizados das mensagens.

[14] *Remote Procedure Call* - Chamada a Procedimento Remoto.
[15] *Common Data Representation.*

Capítulo 2

Processamento Distribuído Aberto

2.1 Introdução

O uso cada vez mais intenso de computadores na execução dos mais diversos serviços, em grandes e pequenas organizações, está causando uma grande mudança nos ambientes de processamento de informação. Observa-se uma necessidade crescente do usuário em ter a capacidade de efetuar o acesso à informação onde quer que ela esteja, sem se preocupar com a forma com que isso será feito. Essas necessidades não podem ser atendidas apenas por redes, as quais provêem apenas comunicação, não garantindo a desejada cooperação entre os componentes de um sistema. Essa capacidade de dois ou mais componentes interagirem de forma a executar uma tarefa comum chama-se **interoperabilidade**, a qual possibilitará o suporte à integração de informações heterogêneas que, por sua vez, poderão estar contidas em várias máquinas também heterogêneas [Nan91, Com91].

A contínua tendência de rápido crescimento do processamento distribuído ajusta-se ao contexto atual, onde temos informações armazenadas e processadas em pontos distintos, com grande heterogeneidade de meios. Devido a esses fatores surgiram várias plataformas fornecendo serviços que possibilitam a necessária integração da informação [SHF+93]. Entretanto, essas plataformas também são heterogêneas [Ber93]. A ISO, juntamente com a ITU-T, então, propôs uma padronização, através de um modelo de referência para Processamento Distribuído Aberto (RM-ODP), que está em fase adiantada de elaboração.

2.2 Sistemas distribuídos abertos

A evolução das necessidades dos usuários, obrigando a troca de informações entre ambientes heterogêneos, e o desenvolvimento da tecnologia de comunicação, permitindo uma interconexão eficiente entre os vários ambientes, são alguns dos fatores que levam ao surgimento de sistemas abertos, os quais podem ser vistos como sistemas que podem interagir com outros sistemas. Podem ser desenvolvidos para um melhor aproveitamento dos sistemas computacionais e de comunicação, possibilitando [TMdS+93]: a) a utilização de recursos e serviços externos a esses sistemas; b) o acesso por meios externos a recursos e serviços próprios; e c) a participação em

atividades comuns e aplicações cooperativas.

As principais características de um sistema distribuído aberto são:

- **ilimitado** - público e livre de qualquer tipo de restrição geográfica, organizacional ou tecnológica;
- **de livre acesso** - qualquer tipo de usuário, componente ou aplicação pode ser admitido no sistema;
- **heterogêneo** - cada usuário tem total poder de decisão sobre sua existência, projeto, implementação, uso e gerenciamento de seus componentes;
- **autônomo** - cada usuário ou servidor de aplicações é completamente livre para determinar suas propriedades, seu comportamento, e a evolução de seus elementos. Cada entidade interage com o restante do sistema da forma que desejar; e
- **descentralizado** - o gerenciamento das propriedades e comportamentos dos componentes é efetuado em nível local, sem qualquer influência ou controle externo.

Como conseqüência dessas características observa-se que:

- cada elemento tem apenas uma visão parcial do sistema, sendo incapaz de compreendê-lo ou modificá-lo como um todo; e
- a necessidade de se garantir integridade e privacidade, já que o sistema torna-se bastante vulnerável a influências indesejáveis, leva à adoção de mecanismos mais sofisticados de controle de acesso e de segurança.

2.3 Especificação ODP

Nesta seção serão apresentados os conceitos básicos abordados no RM-ODP com uma análise mais detalhada das necessidades e funcionalidades relativas à relocação e respectiva transparência e à interoperabilidade.

Devido ao rápido crescimento do processamento distribuído, permitindo a interconexão de sistemas de computadores geograficamente separados, tornou-se bastante conveniente que tais sistemas também pudessem interagir para a execução conjunta de uma tarefa.

O RM-ODP surgiu, então, com o objetivo de fornecer uma padronização do processamento distribuído aberto, através da criação de uma estrutura básica que permita, de uma forma integrada, o suporte à distribuição, interoperabilidade, *interworking* e portabilidade.

Esse modelo de referência está sendo desenvolvido em conjunto pela ISO e pela ITU-T.

Para tratar com a grande complexidade de um sistema ODP, foram criadas cinco diferentes abstrações desse sistema original, todas com a mesma importância e gerando a base para a especificação do sistema. Essas abstrações são chamadas de pontos de vista, já que cada uma apresenta o sistema sob uma perspectiva específica, conforme observa-se a seguir [ISO95a, ISO95c, Ray93]:

- **ponto de vista de empresa** - define os requisitos organizacionais caracterizados pelos objetivos, escopo e políticas do sistema, os quais podem representar uma empresa, apenas uma parte dela ou um consórcio de empresas. Descreve como os usuários (gerentes, administradores, usuários comuns), pertencentes a uma determinada organização, a percebem. Cada gerente de departamento, provavelmente, terá uma visão diferente, associada às suas necessidades de acesso à informação.

- **ponto de vista de informação** - define as políticas de armazenamento e fluxo de informação. Descreve como os engenheiros de informação e analistas percebem o sistema. Preocupa-se com a semântica e o processamento da informação.

- **ponto de vista computacional** - preocupa-se com a decomposição funcional sem se preocupar com os detalhes de distribuição. Considera o sistema ODP como um conjunto de objetos que interagem, ignorando a forma com que essas interações são efetuadas.

  Descreve os algoritmos e o fluxo de dados que caracterizam um sistema distribuído.

  Sua linguagem esconde do programador a noção da distribuição da aplicação, embora ainda permita a execução de programas projetados para sistemas centralizados.

  Sua especificação funcional define os objetos do sistema, suas ações internas e as interações entre eles. É baseada em objetos: há encapsulamento de dados e processos; interações entre objetos só podem ser realizadas através de interfaces; e os objetos podem oferecer múltiplas interfaces.

  A linguagem computacional apresenta duas interfaces que permitem interações entre objetos [Ray93]:

  - interfaces **operacionais** - fornecem uma descrição completa dos serviços oferecidos pelo objeto: nomes das operações, tipos dos parâmetros, requisitos de qualidade de serviço e de transparências, a função do objeto (cliente ou servidor), etc;
  - interfaces **streams** - constituem-se de um conjunto de fluxos[1] que fluem de um objeto (produtor do fluxo) para outro (consumidor do fluxo). Essas interações foram definidas particularmente para o tratamento de aplicações que usam informações isócronas (como em multimídia e telecomunicações, por exemplo), devido às características próprias dos seus requisitos; e
  - interfaces de **sinal** - provêem ações de comunicações de muito baixo nível, suportando as interfaces anteriores. As primitivas dos servi'cos OSI[2] (REQUEST, INDICATE, RESPONSE e CONFIRM) são exemplos de sinal.

  O objeto responsável por manter a conexão entre outros objetos computacionais, chama-se **objeto *binding***.

  O *binding*[3] pode ser:

[1] Tradução de *flow* - uma abstração de uma seqüência de interações.
[2] *Open Systems Interconnection.*
[3] Um caminho onde seja possível a comunicação entre os objetos que pretendem interagir.

- **implícito** - ocorre automaticamente num instante entre o recebimento do **identificador da interface** do objeto fornecedor do serviço e a primeira invocação de uma operação nessa interface pelo objeto que recebe o identificador. A forma como o referido identificador é obtido e o instante preciso em que o **binding** é efetuado são questões de engenharia e, portanto, não consideradas nesse ponto de vista.

  Qualquer erro nesse processo será tratado como uma falha de infra-estrutura[4] ao ser invocada a primeira operação.

  As necessidades e capacidades dos recursos básicos existentes é que determinarão a duração do *binding*, não havendo nenhuma forma de restrição na linguagem computacional; e

- **explícito** - é criado um objeto que passa a oferecer o serviço de *binding* aos objetos que pretendem interagir. Ao contrário do anterior, possibilita ao usuário a manipulação do *binding*, permitindo a sua criação de forma específica para determinadas interações. Pode ser:

  * primitivo - quando liga dois objetos computacionais diretamente e que possuam mesmo tipo de interface; ou
  * composto - quando existe um **objeto *binding*** ligando dois ou mais objetos computacionais através de *bindings* primitivos.

- **ponto de vista de engenharia** - preocupa-se com a infra-estrutura necessária para suportar distribuição num sistema ODP. Como proporciona um ambiente compatível com a descrição computacional podemos mapear um objeto computacional para um ou mais objetos de engenharia.

  Essa infra-estrutura é definida:

  - pela identificação das funções ODP necessárias para o gerenciamento da distribuição física, comunicação, processamento e armazenamento; e
  - pela identificação das funções desempenhadas pelos diferentes objetos que suportam as funções ODP.

  As entidades principais neste ponto de vista são descritas abaixo e podem ser visualizadas na Figura 2.1:

  - **objeto**: que se divide em:
    * **objeto de engenharia básico (BEO[5])**: um objeto de engenharia que necessita do suporte de uma infra-estrutura distribuída; e
    * **objeto de infra-estrutura**: todos os objetos que contribuem para fornecer uma infra-estrutura distribuída para o BEO.

[4] Ocorre quando o **contrato ambiental** - acordo entre objetos estabelecendo parte de seus comportamentos - entre as interfaces que estão interagindo não é satisfeito.

[5] *Basic Engineering Object.*

– **canal**: permite a interação entre objetos de engenharia, criando um *binding* entre suas interfaces. É composto pelos seguintes objetos, que serão descritos posteriormente: *stub*, *binder*, protocolo e interceptador.

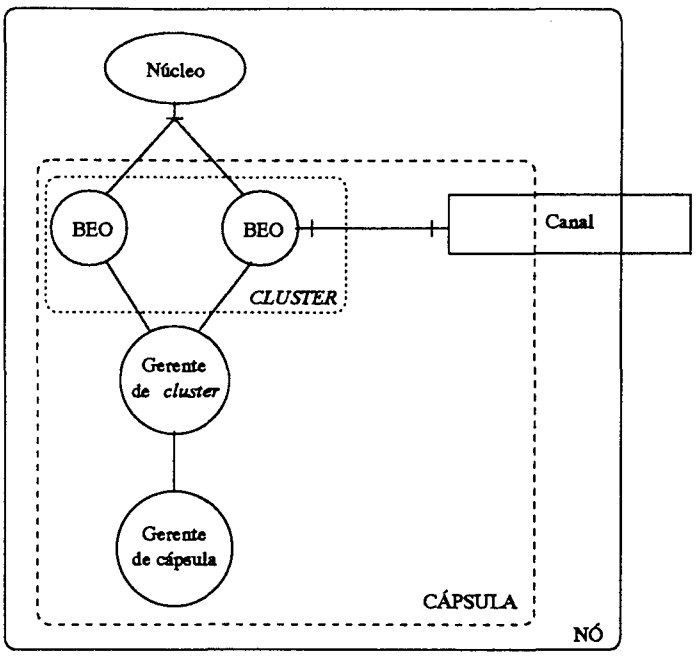

Figura 2.1: Estrutura de suporte a um BEO (ponto de vista de engenharia)

Além desses elementos, os seguintes conceitos compreendem sua estrutura:

– ***cluster***: um conjunto de objetos de engenharia básicos formando um elemento único para o propósito de:

  * criação de *checkpoint*[6] - criação de um *template* de objeto de engenharia, o qual pode ser usado para instanciar outro objeto de engenharia, que será consistente com o original no instante da criação do *checkpoint*;
  * desativação - destruição de um *cluster* após efetuar a criação do seu *checkpoint*;
  * reativação - instanciação de um *cluster*, a partir do seu *checkpoint*, em seguida a sua desativação;
  * recuperação - instanciação de um *cluster*, a partir do seu *checkpoint*, depois de uma falha ou destruição do *cluster*; e
  * migração - movimentação de um *cluster* para outra cápsula.

  É constituído por um conjunto de um ou mais BEOs, associado a um **gerente de *cluster***, o qual será responsável pela política de gerenciamento dos objetos que compõem o seu *cluster*.

[6]**Criação de *checkpoint*** é a tradução de *checkpointing*.

Um BEO, num *cluster*, terá sempre duas ligações básicas: ao núcleo (apresentado adiante) e ao gerente de *cluster*. As outras interfaces farão a conexão ou a outro BEO (pertencente ao mesmo *cluster*) ou a um canal.

Um *cluster* é responsável por sua própria segurança e pelo gerenciamento das **referências de interface de engenharia**. Entretanto, pode ser auxiliado por algumas funções ODP específicas.

Um *template* de *cluster* especifica a configuração dos objetos no **cluster** e as necessidades a serem atendidas para instanciá-los e estabelecer *bindings*. Essa instanciação é uma função do **gerente de cápsula**.

- **cápsula**: um conjunto de objetos formando um elemento único para o propósito de encapsulamento do processamento e do armazenamento.

  É constituída por um conjunto de um ou mais *clusters*, gerenciadores de *clusters*, canais e um gerente de cápsula, sendo esse último, o responsável pela política de gerenciamento dos *clusters* que compõem a cápsula.

  Um *template* de cápsula especifica a configuração inicial dos objetos na cápsula e a sua instanciação é executada pelo **núcleo**.

- **nó**: um conjunto de objetos formando um elemento único para o propósito da localização no espaço. Estes objetos compartilham um mesmo conjunto de funções de processamento, armazenamento e comunicação.

  É constituído por um conjunto de uma ou mais cápsulas e um núcleo, o qual é responsável pela política de gerenciamento do nó.

  A instanciação de um nó é executada externamente à estrutura do ODP, embora deva garantir:

  - a criação de um núcleo e suas funções associadas de processamento, armazenamento e comunicação;
  - a criação de uma função de *trading* necessária ao processo de instanciação. Essa função intermedia a oferta e a busca de serviços; e
  - a criação dos canais necessários à configuração inicial do nó.

- **núcleo**: coordena as funções de processamento, armazenamento e comunicação a serem utilizadas pelos objetos pertencentes a um mesmo nó. Administra o nó, estabelecendo as suas políticas de gerenciamento. A unidade básica para aplicação dessas políticas é a cápsula, podendo existir políticas diferentes para cápsulas diferentes num mesmo nó.

  Só existe um núcleo por nó, o qual pode comunicar-se com todos os outros núcleos.

  Embora não especifique, o RM-ODP menciona que devem existir mecanismos que permitam interações dos gerentes de *cluster* e de cápsula entre si e com o núcleo.

- **ponto de vista de tecnologia** - preocupa-se com a escolha da tecnologia para implementação do sistema (hardware e software). Descreve como os responsáveis pela sua instalação, configuração e manutenção o percebem.

Para cada um desses pontos de vista é definida uma linguagem que o especificará, com conceitos, regras e estruturas apropriadas àquela abstração.

2.3.1 Mecanismo de comunicação do ponto de vista de engenharia

Para permitir a interação entre objetos de engenharia num mesmo *cluster*, são suficientes suas interfaces. Entretanto, quando se pretende interagir com objetos situados em diferentes *clusters* isso só será possível através de **canais**, da mesma forma que surge a necessidade de um identificador não-ambíguo (as **referências de interface de engenharia**) para permitir a correta localização da interface com a qual se tenciona interagir.

Existem dois tipos de *binding* nesse ponto de vista:

- local - entre objetos dentro de um mesmo *cluster* ou entre objetos que cooperam no escopo de um nó para a construção de um canal. É implementado através de mecanismos específicos do sistema operacional; e
- distribuído - quando é suportado por um canal. Embora normalmente existente entre objetos em diferentes *clusters*, pode existir também entre objetos dentro de um mesmo *cluster*.

Canal

Permite a interação, com transparência de distribuição, entre objetos de engenharia localizados em *clusters* diferentes, criando um *binding* entre suas interfaces. Suporta, inclusive, *multicasting*. A Figura 2.2 apresenta como uma interação do ponto de vista computacional pode ser representada no ponto de vista de engenharia.

É composto pelos seguintes elementos:

- ***stub*** - fornece funções de conversão (por exemplo, *marshalling* e *unmarshalling* de dados) e tem a capacidade de efetuar controles e manter registros.

  Uma de suas interfaces será sempre utilizada para o gerenciamento da qualidade do serviço. *Stubs*, num mesmo canal, usando diferentes sintaxes de transferência, deverão utilizar interceptadores para efetuar a conversão entre os sistemas;

- ***binder*** - responsável pela manutenção e gerenciamento de *bindings* entre objetos de engenharia básicos que estejam interagindo. Gerencia a integridade fim-a-fim e a qualidade do serviço do canal.

  Uma de suas interfaces será sempre para controle, possibilitando alterações na configuração do canal ou a sua destruição (total ou parcial);

- **protocolo** - objeto que, através da comunicação com outro objeto protocolo, permite a interação entre objetos de engenharia situados em *clusters*, cápsulas ou nós diferentes. Provê, portanto, as funções de comunicação.

  Quando são de tipos diferentes, num mesmo canal, necessitam de um interceptador, para possibilitar a conversão dos protocolos; e

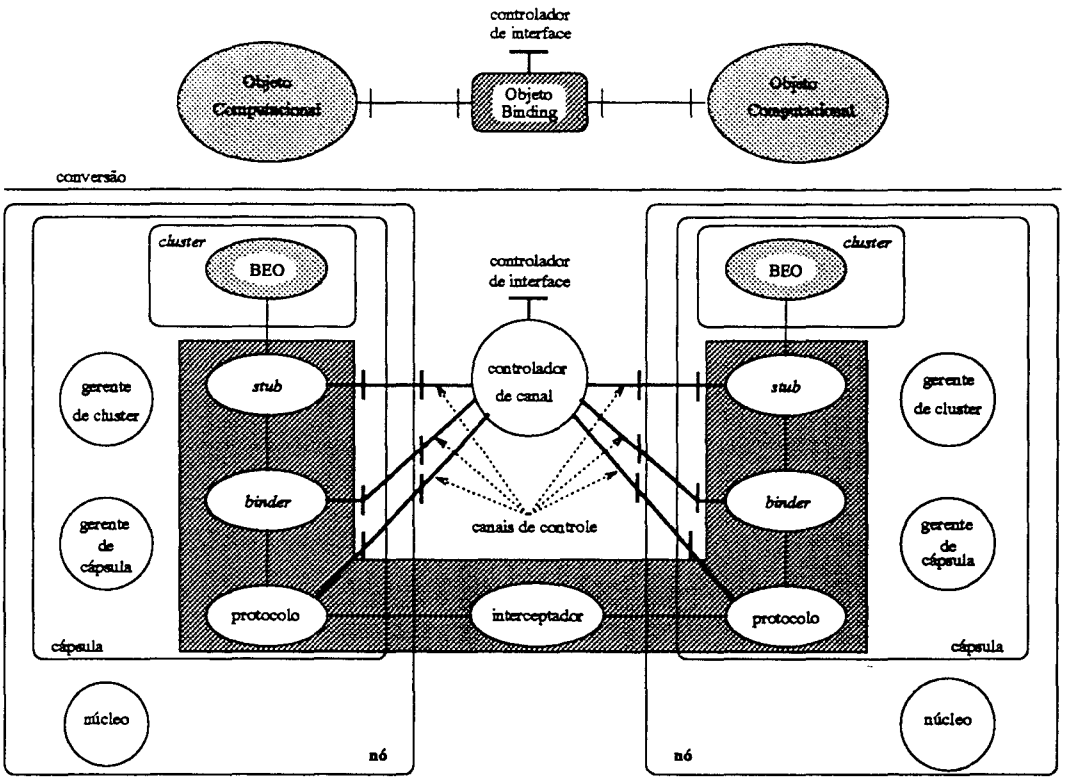

Figura 2.2: Exemplo de conversão entre o modelo computacional e o de engenharia.

- **interceptador** - possibilita a interação entre objetos de engenharia situados em domínios diferentes (administrativos e/ou tecnológicos).

  Dessa forma surgem os seguintes conceitos [HOvdL+93, ISO95a]:

  - **domínio administrativo** - conjunto de objetos cuja segurança, contabilização, monitoramento e outras funções de gerenciamento estão sob uma mesma administração. Ao ultrapassar o seu domínio, as responsabilidades de gerenciamento (as garantias de dependência ou a forma como são alocados certos recursos, por exemplo) podem ser alteradas dependendo apenas das necessidades e prioridades estabelecidas pelo administrador do novo domínio;
  - **domínio tecnológico** - conjunto de objetos que utilizam uma mesma representação de dados e funcionalidades de protocolos, nomeação e endereçamento. Para ultrapassar esses limites será necessária a conversão de protocolos e nomes; e
  - **federação** - conjunto de domínios que têm um contrato comum regulamentando as interações entre seus membros. Mesmo assim é mantida a **autonomia**: cada participante pode deixar ou ingressar no grupo a qualquer momento e por sua livre escolha.

  Para permitir interações independentemente dos domínios são utilizados **interceptadores**:

- **fixos**: posicionados entre domínios, permitem interações através de contratos entre administrações, que especificarão a base da sua federação. Possuem conceito semelhante a um *gateway*; e
- **nômades**: permitem que um objeto seja desconectado de um domínio e conectado a outro. Para tanto, um interceptador deste tipo deve ser capaz de acompanhar a trajetória de um objeto que, utilizando-se de meios externos ao sistema, se movimente de uma localização para outra.

Interceptadores administrativos descrevem as características de uma administração. Localizam-se completamente dentro dos domínios aos quais respondem e são acessados sempre que for necessário o conhecimento das possibilidades e restrições de um domínio administrativo. Podem manter comunicação com interceptadores semelhantes em outros domínios, trocando informações de maneira a assegurar as corretas conversões exigidas.

Interceptadores tecnológicos são responsáveis pela conversão de dados e protocolos, participando de todas as interações que envolvam domínios diferentes.

Interceptadores podem ser visíveis de dois pontos de vista:

- computacional: deverão ser construídos especificamente para cada tipo de interface além de se submeterem à verificação do tipo; e
- de engenharia: poderão atuar sobre quaisquer interfaces, já que são transparentes ao tipo.

Qualquer objeto que compõe um canal pode ter um *binding* (através de outros canais, inclusive) com objetos que não fazem parte do canal. Esses novos objetos normalmente são repositórios de informações que auxiliam os objetos *stubs, binders*, protocolos e interceptadores nas suas funções.

Criação de um canal entre dois objetos

- O canal começa a ser configurado através da interação de um dos objetos com o seu núcleo, para a criação dos *stubs*, *binders* e objetos protocolos e a respectiva **referência de interface**;
- A referência de interface criada é comunicada ao outro objeto, o qual interage com seu núcleo de forma a criar os *stubs*, *binders* e objetos protocolos coerentes com o tipo de canal que está sendo criado; e
- Os *binders* interagem entre si de forma a habilitar a comunicação através do canal.

Referência de interface de engenharia

Uma interface de objeto de engenharia é identificada de forma confiável, no tempo e no espaço, por uma referência de interface de engenharia, gerada pelo núcleo no momento em que a interface

é criada. Essa referência provê as informações necessárias para identificar e descrever uma interface de engenharia de forma a permitir que qualquer objeto possa estabelecer um *binding* com essa interface sem qualquer informação adicional. Através dela, podem ser determinados, por exemplo:

- o tipo da interface;
- o *template* de canal apropriado a ser instanciado, com as respectivas informações sobre seus componentes; e
- a localização das **interfaces de comunicação**[7] a serem utilizadas.

Os dados contidos numa referência de interface de engenharia podem ser tanto um ponteiro para um objeto, o qual encontra a localização da interface, quanto uma informação completa e precisa dessa localização. Podem conter, então:

- os dados propriamente ditos;
- identificadores para interfaces que permitam o acesso aos dados; ou
- uma combinação dessas alternativas.

O RM-ODP não prescreve como os diversos sistemas ODP devem implementar as referências de interface nem como devem ser extraídas as informações nelas contidas ou por elas apontadas. Caso o núcleo suporte protocolos, processos de *bindings* e sintaxes de transferência diferentes, a referência de interface de engenharia deverá indicar quais as combinações válidas para um determinado *binding* distribuído.

Essas referências são válidas no contexto de um domínio de gerenciamento de referência de interface de engenharia[8], portanto, havendo necessidade de sua utilização fora desse espaço, será também necessário a sua conversão, para a qual é utilizado um interceptador.

Quando objetos são relocados (devido à desativação/reativação ou migração, por exemplo), as referências de interfaces de engenharia permanecem válidas e os *binders* (associados à função de relocação) são responsáveis pelo mapeamento dessas referências com os identificadores de interface de comunicação.

Para monitorar essas referências faz-se uso da **função de acompanhamento**[9] **de referência de interface de engenharia**, que mantém, no escopo para a qual foi designada:

- informação sobre a posse de uma referência para um determinado tipo de interface de engenharia. Dessa forma, pode ser contabilizada a quantidade de cópias dessas referências existente fora do *cluster* que suporta a interface referenciada (Figura 2.3). Caso chegue a

[7] Uma interface de um objeto protocolo ligado a uma interface para outro **domínio de comunicação** - conjunto de objetos protocolo capazes de interagir entre si.

[8] Domínio de gerenciamento de uma referência de interface de engenharia é o conjunto de nós que formam um contexto para a geração e entendimento de referências de interfaces de engenharia.

[9] Tradução de *tracking*.

zero, o gerente de *cluster* é informado. Caso todas as cópias existentes no *cluster* sejam destruídas, o gerente de *cluster* deve comunicar à função de acompanhamento de referência; e

- informação sobre a existência ou não de uma determinada interface. Caso ocorra falha ou destruição da interface, as cápsulas que contêm objetos que possuem referências para ela deverão ser notificadas.

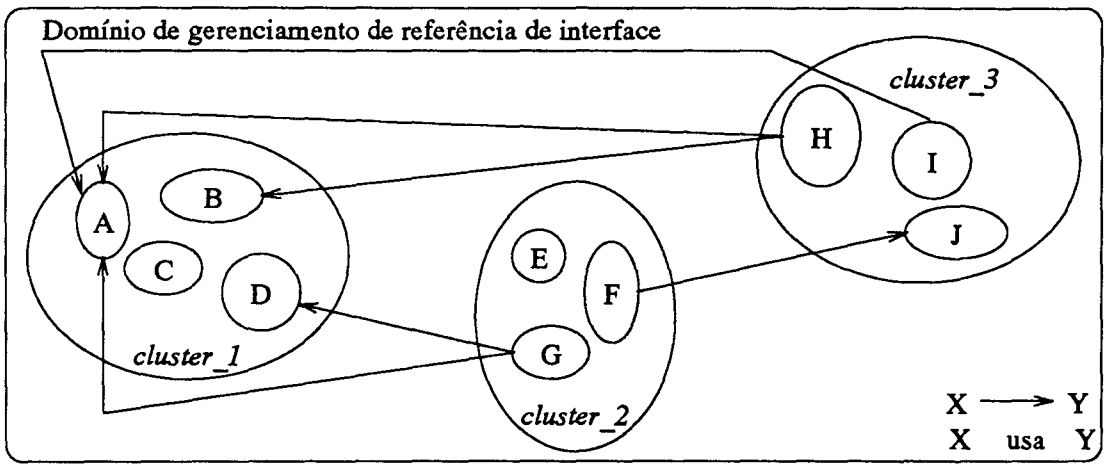

Figura 2.3: Exemplo de utilização da função de acompanhamento de referência de interface.

Num domínio de gerenciamento de referência de interface de engenharia, essa função será informada: a)através dos canais (pelos *stubs*), sempre que referências trafegarem entre *clusters*; e b)através do gerente de *cluster*, quando todas as cópias de uma determinada referência de interface de engenharia forem destruídas no seu *cluster*.

Com esses dados é possível identificar os recursos que não estão sendo usados e permitir a sua liberação, otimizando sua utilização.

É interessante observar que também é definido um **identificador de ponto final**[10] **de *binding***, que indica, no contexto de uma cápsula, os *bindings* nos quais os objetos de engenharia estão envolvidos, enquanto a referência de interface de engenharia é um identificador para uma interface de um objeto de engenharia disponível para *binding*. Assim, enquanto o primeiro é usado para executar interações, o segundo permite o estabelecimento de *bindings*.

O RM-ODP considera que não há necessidade da utilização dos pontos finais de *binding* fora do escopo de cápsulas e, portanto, não define nenhum mecanismo para tal.

[10]Tradução de *endpoint*

2.3.2 Funcionalidades

Segundo o RM-ODP, são necessárias, na construção de um sistema ODP (baseado nas linguagens de engenharia e computacional e nos requisitos originados na necessidade de prover transparência de distribuição), algumas funções básicas, as quais são divididas nos seguintes grupos [ISO95a, ISO95c]:

- **funções de gerenciamento** - responsáveis por criar, manter e controlar os objetos de infra-estrutura do sistema ODP [Gar94]:
  - função de gerenciamento de nó - controla as funções de comunicação, armazenamento e controle dentro de um nó e é usada por todas as outras funções;
  - função de gerenciamento de objeto - destrói e efetua a criação do *checkpoint* dos objetos;
  - função de gerenciamento de *cluster* - recupera, migra, desativa , reativa, destrói e cria *checkpoints* dos objetos; e
  - função de gerenciamento de cápsula - instancia *clusters*, desativa e efetua a criação do *checkpoint* em todos os *clusters* da cápsula, e destrói cápsulas.
- **funções de coordenação** - responsáveis pela coordenação das atividades desempenhadas pela infra-estrutura ODP:
  - função de *checkpoint* e de recuperação - coordena a criação de *checkpoints* e a recuperação de *clusters* que falharam;
  - função de desativação e de reativação - coordena a ativação e desativação de *clusters*;
  - função de notificação de evento - registra e torna disponível o histórico dos eventos ocorridos num sistema ODP;
  - função de grupo - gerencia as interações entre objetos membros de um grupo, executando um trabalho cooperativo;
  - função de migração - coordena a migração de um *cluster* de uma cápsula para outra;
  - função de replicação - considera um caso especial da função de grupo, onde todos os membros são iguais;
  - função de transação - coordena e controla um conjunto de transações para garantir os requisitos de visibilidade, recuperação e permanência; e
  - função de acompanhamento de referência de interface - monitora a transferência das referências de interface (que contêm as informações necessária para que um objeto de engenharia possa efetuar uma conexão) entre os objetos de engenharia.
- **funções de repositório** - responsáveis por garantir o armazenamento e o acesso às informações, de forma persistente:

- função de informação da organização - gerencia um repositório de objetos que mantém uma correspondência entre entidades e seus relacionamentos, num sistema ODP;
- função de relocação - gerencia informações sobre interfaces que mudaram de localização;
- função de armazenamento - gerencia um repositório para dados;
- função de *trading* - encontra os objetos fornecedores dos serviços solicitados por outros objetos; e
- função de repositório de tipos - armazena as especificações dos tipos e seus relacionamentos;

- **funções de segurança** - responsáveis por garantir as restrições de uso e de acesso às informações, de forma a manter sua integridade:
  - função de controle de acesso - impede interações não autorizadas;
  - função de auditoria - provê monitoração e coleta de informações sobre as interações;
  - função de autenticação - confirma a identidade de um objeto;
  - função de confidencialidade - impede o acesso não autorizado às informações;
  - função de integridade - previne a alteração ou destruição não autorizada de dados;
  - função de não-repúdio - impede que um objeto envolvido numa interação negue esse fato; e
  - função de gerenciamento de chave - provê facilidades para o gerenciamento de chaves criptográficas.

Algumas dessas funções serão utilizadas para prescrever as transparências de distribuição. Tais prescrições, juntamente com a linguagem computacional e a de engenharia, determinarão a arquitetura para sistemas ODP.

2.3.3 Transparência de distribuição

As transparências, cuja implementação significa que o comportamento de alguma parte do sistema não será visualizado pelo usuário, são tidas como fundamentais tanto para a computação distribuída como para a padronização ODP [SHF+93].

A transparência de distribuição torna desnecessário o conhecimento das características próprias de um sistema distribuído. Entretanto, sua implementação causa certas exigências, expressas nas linguagens de empresa, de informação e computacional, as quais deverão ser observadas na especificação das funções na linguagem de engenharia.

O comportamento dos objetos que constituem um canal são determinados pela combinação das transparências de distribuição aplicadas a esse canal.

Divide-se em [ISO95a, ISO95c]:

- **Acesso** - Esconde as diferenças entre as possíveis representações de dados e os mecanismos de invocação utilizados entre objetos interagindo. Não há diferença para o usuário se a forma de acesso foi local ou remota, por exemplo. Essa transparência é obtida através de *stubs*;
- **Localização** - O usuário não se preocupa com a localização dos serviços. Os identificadores de interface não contêm informações sobre a localização da interface. Os *binders* são responsáveis por essa transparência;
- **Migração** - Esconde, de um objeto, a habilidade do sistema de alterar a localização do próprio objeto. Possibilita o balanceamento de carga e a redução de latência;
- **Transação** - Esconde a coordenação de atividades numa determinada configuração de objetos de forma a manter a consistência;
- **Replicação** - Esconde o efeito da existência de cópias múltiplas de objetos, suportando, portanto, a mesma interface. Permite um maior desempenho e disponibilidade.
- **Falha** - Esconde, de um objeto, a falha e possível recuperação ocorrida tanto em outros objetos quanto nele mesmo;
- **Persistência** - Esconde, de um objeto, a desativação e reativação tanto de outros objetos quanto a sua própria;
- **Relocação** - Esconde a alteração da localização de uma interface das outras interfaces ligadas a ela.

2.3.4 Relocação

Um sistema ODP considera como relocação, a alteração da localização de uma interface. Esse fato pode ocorrer como conseqüência:

- da desativação e reativação de *clusters*;
- da criação de *checkpoint* e recuperação de *clusters*;
- da migração de *clusters*; e
- de alterações no domínio das comunicações.

Os três primeiros itens são efetuados sob a responsabilidade do gerente de *cluster*, através da função de gerenciamento de *cluster*. Em relação ao domínio das comunicações, as alterações são efetuadas por decisões de gerenciamento de comunicações (provavelmente através do núcleo).

A Relocação pode causar a falha do canal e pode invalidar[11] as referências de interfaces de engenharia. Os canais podem ser "reparados" se a atividade que alterou a localização do objeto de engenharia notificar corretamente a função de Relocação. Essa função fornecerá ao *binder* as informações necessárias para garantir a transparência de relocação.

[11] Uma referência de interface de engenharia válida é aquela que possui informações consistentes de maneira a permitir um *binding*.

Função de Relocação

Essa função gerencia um repositório de localização de interface e é suportada por um objeto chamado Relocador.

Para que possa exercer sua funcionalidade, o Relocador deve ser associado, de alguma forma, com as interfaces passíveis de terem sua localização alterada. Um Relocador pode ser associado a uma única interface ou a várias. Para as interfaces que pertençam a domínios de gerenciamento diferentes, a forma de acesso ao Relocador deve permitir a correta identificação do domínio a que pertence a interface movimentada.

O gerente de *cluster* deve notificar ao Relocador sobre cada interface de cada objeto existente no *cluster*, quando o *cluster* é relocado. Com isso, percebe-se que há a necessidade de que todas as interfaces em um *cluster* devem estar associadas a um Relocador, que será responsável por um determinado conjunto de interfaces.

O Relocador deve estabelecer uma política, a ser utilizada pelas funções de coordenação, que permita a reativação ou recuperação de um *cluster* para o caso em que alguma interface de algum dos seus objetos seja solicitada por outros objetos. Isso se deve ao fato do *cluster* ser a unidade básica de reativação e recuperação. Pelo mesmo motivo, para que um objeto possa ser relocado, é necessário que todos os objetos do *cluster* ao qual pertencem suportem essa capacidade.

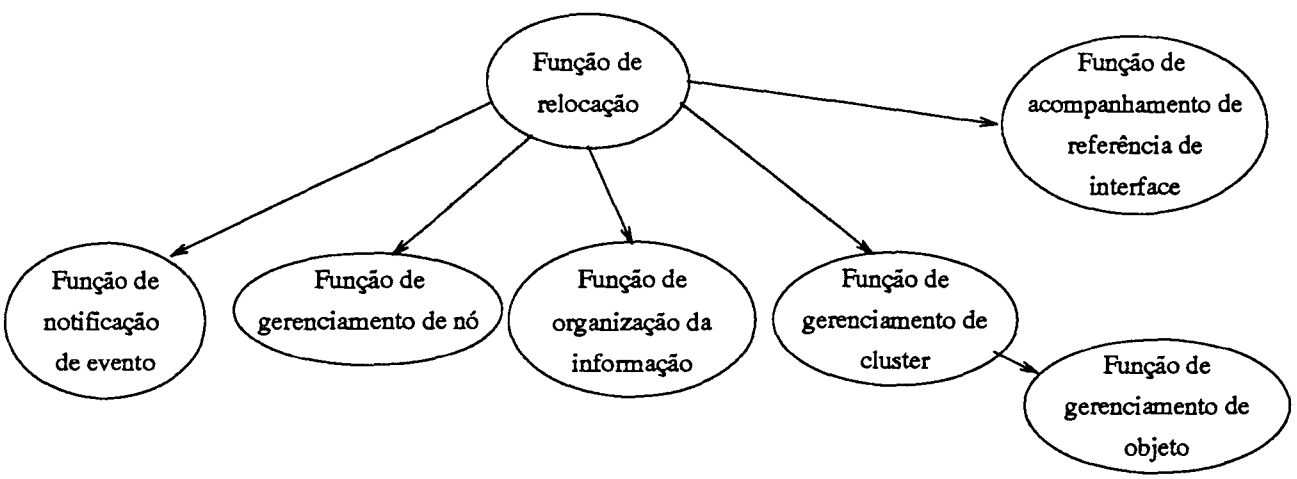

Figura 2.4: Funções relacionadas com a função de relocação.

São observados os seguintes relacionamentos necessários à função de relocação:

- função de gerenciamento de nó - essa função cria canais e referências de interfaces;
- função de organização da informação - pode prover o armazenamento de informações a serem utilizadas pelo Relocador para a validação ou atualização de referências de interface. No caso de relocação por desativação/reativação, esta função pode ser usada para manter um relacionamento entre o *cluster* desativado e a interface à qual deve ser solicitada a reativação desse mesmo *cluster*;

- função de notificação de evento - pode armazenar os eventos produzidos pelas atividades que causam relocação e, então, notificar o Relocador de sua ocorrência;

- função de gerenciamento de *cluster* - responsável pela maioria das atividades que causam relocação, cuja ocorrência deve ser informada ao Relocador. Só é possível executar suas funcionalidades caso todos os objetos contidos no *cluster* suportem, através da função de gerenciamento de objeto, as funções de destruição e de criação de *checkpoint*;

- função de desativação e reativação - estabelece as políticas a serem obedecidas pelos gerentes de *cluster* e de cápsula para a finalidade de desativação e reativação de *clusters*;

- função de *checkpoint* e recuperação - estabelece as políticas a serem obedecidas pelos gerentes de *clusters* e de cápsulas para o objetivo de criação de *checkpoints* e de recuperação de *clusters*;

- função de acompanhamento de referência de interface de engenharia - usa a função de notificação de eventos para tornar disponível os eventos citados na seção 2.3.1. Embora não seja mencionado no RM-ODP, consideramos que essa função pode ser utilizada pela função de relocação para a verificação dos seus dados.

- função de migração - estabelece políticas para a migração e usa as funções de gerenciamento de *cluster* e de cápsula.

Transparência de relocação

Caso um *binder* detecte uma falha no canal, deve tentar validar, através da função de relocação, as referências de interfaces de engenharia para as interfaces às quais o canal estava ligado e, se necessário, restabelecer o canal. Dessa maneira, um objeto que requisita uma determinada interface não perceberá se a mesma for movimentada, uma vez que suas interações serão mantidas.

O Relocador deverá estabelecer um procedimento para restauração do *cluster* contendo o objeto desativado (ou que tenha falhado mas previamente tenha sido criado um *checkpoint*) como parte do mecanismo de validação.

Para possibilitar essa transparência é necessário que:

- um relocador seja associado a cada uma das interfaces de um *cluster*;

- as informações sobre as alterações de localização de qualquer objeto sejam enviadas ao Relocador (pelo objeto que efetuou a alteração ou pelo gerente de *cluster*); e

- os *binders* tornem válidos os *bindings* suportados pelo canal, através da verificação das informações contidas nas referências de interface.

Observam-se as seguintes dependências em relação à transparência de relocação (Figura 2.5):

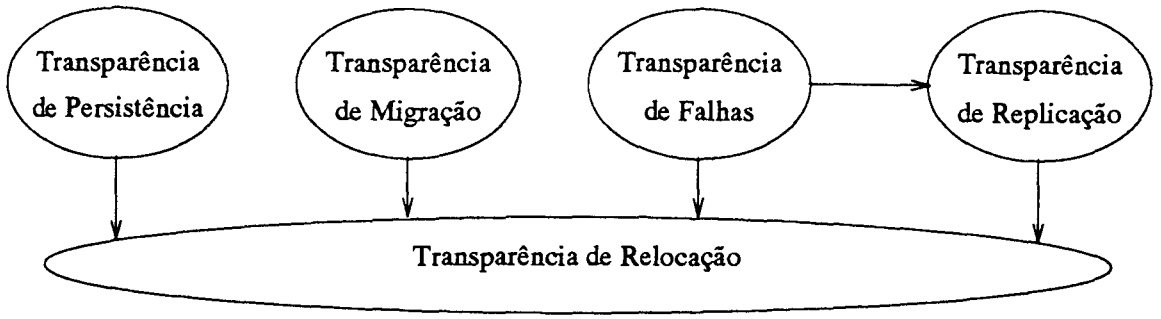

Figura 2.5: Transparências dependentes da transparência de relocação.

- Transparência de Replicação - o *cluster* deve possuir transparência de relocação, de maneira a esconder, do objeto que está interagindo com um objeto contido no *cluster*, o fato de estar, na realidade, interagindo com uma interface que suporta vários objetos com compatibilidade de comportamento;

- Transparência de Persistência - todos os canais ligados ao *cluster* devem possuir transparência de relocação, de forma que não percebam que houve a desativação/reativação do *cluster* (e, conseqüentemente, do objeto) ao qual está efetuando um *binding*;

- Transparência de Falhas - essa transparência pode ser obtida: a) através da replicação de *clusters*; b) através da criação do *checkpoint* e recuperação de *clusters*; e/ou c) alocando ao objeto uma infra-estrutura suficiente para impedir uma falha específica. As duas primeiras exigem a transparência de relocação; e

- Transparência de Migração - todos os canais ligados ao *cluster* devem possuir transparência de relocação, de maneira que as alterações na localização do objeto contido no *cluster* não sejam percebidas pelos canais, os quais deverão manter o *binding* com o objeto fora do *cluster*.

Domínio de gerenciamento de referência de interface de engenharia

Define políticas, específicas para o domínio considerado, com o objetivo de estabelecer o conteúdo, alocação, acompanhamento e validação das referências de interfaces de engenharia.

Os domínios podem ser divididos em subdomínios, assim, as referências de interface de engenharia serão organizadas como um conjunto composto por conjuntos de informações, cada qual relativo a um subdomínio.

2.4 Plataforma Multiware

Está em desenvolvimento, na UNICAMP, o projeto de uma plataforma para suporte ao processamento distribuído aberto, denominada **Multiware**. Conforme a Figura 2.6, essa plataforma é composta pelas camadas hardware e software básicos, middleware e groupware.

O conjunto de serviços formando uma camada entre o sistema operacional, acrescido dos softwares de redes, e as aplicações recebe o nome de *middleware* [SHF+93, VA93, Ber93].

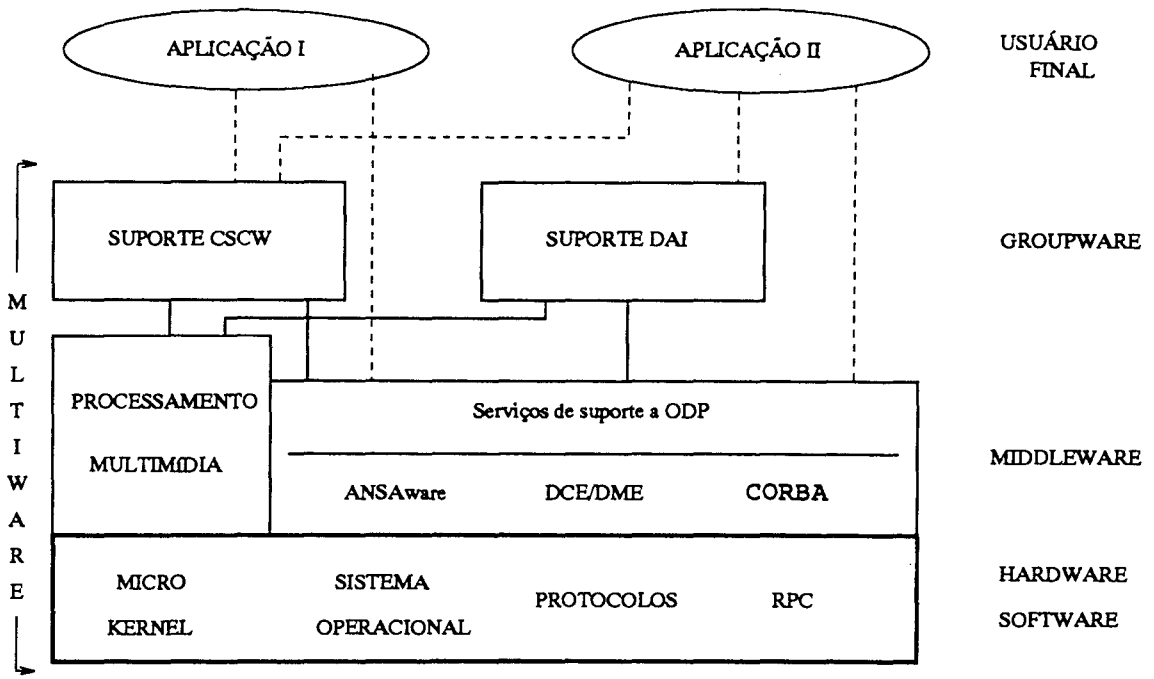

Figura 2.6: Plataforma Multiware

A camada middleware, nessa plataforma, tem como função prover um ambiente de processamento aberto distribuído às aplicações, independente de quais sejam.

Essa camada será dividida em dois níveis, cuja composição pode ser observada na Figura 2.7 [MM94]:

- inferior - composto por plataformas comerciais já conhecidas: ANSAware, DCE e ORB; e
- superior - chamado de **nível ODP**, que oferece suporte aos serviços ODP além de garantir a transparência ao usuário da plataforma que está sendo utilizada. É composto por três subníveis:
  - gerenciamento ODP - fornece os serviços básicos de gerenciamento, auxiliado pelas funções de repositório;
  - transparência/segurança - oferece essas características, de acordo com a especificação ODP; e
  - suporte a ODP - provê as funcionalidades do ODP, sendo dividida em:
    * suporte a aplicações - fornece as funcionalidades básicas de uma plataforma de serviços abertos;
    * *trading* - possibilita a negociação entre clientes e servidores. Anuncia serviços [dPLJ94, JM95];

* suporte a grupos - oferece suporte à cooperação entre membros de um mesmo grupo [Cos95, CM96];
* suporte a transações -garante as propriedades de atomicidade, consistência, isolamento e durabilidade, necessárias para uma invocação transacional;
* suporte a multimídia - permite o envio e a recepção de informação multimídia e de tempo real; e
* suporte a OODBMS - possibilita o acesso a Bases de Dados Orientadas a Objetos.

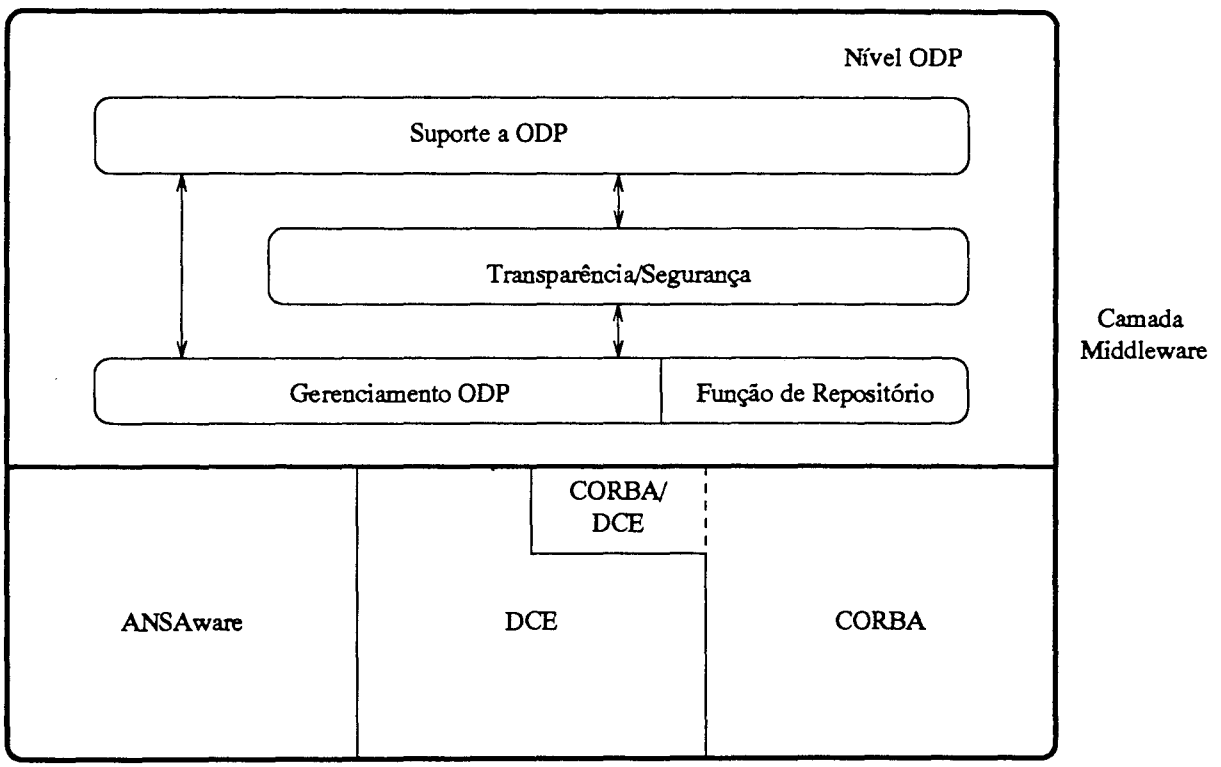

Figura 2.7: Camada *Middleware*

Observa-se que os subníveis utilizam os serviços oferecidos pelas plataformas, portanto, o esforço despendido pelos subníveis para prover certas funções será maior ou menor dependendo das características dessas plataformas.

Atualmente, em nível do projeto, está sendo efetuado o mapeamento dos conceitos de ODP para a plataforma ORB, a qual é utilizada na implementação corrente.

A opção pelo ORB é justificada pela sua orientação a objetos (de acordo com a metodologia de modelagem do RM-ODP) e pela sua simplicidade, possuindo uma estrutura que permite uma implementação inicial pequena e que pode evoluir de acordo com as necessidades do usuário, já que permite que novos objetos prestadores de serviços incorporem-se gradativamente ao sistema. A arquitetura do ORB será detalhada no capítulo seguinte.

Esta dissertação aborda as transparências de acesso, localização e relocação usando como base a plataforma ORB.

Capítulo 3

Conceitos da CORBA

3.1 Introdução

A OMG consiste num dos consórcios existentes de companhias que buscam a produção de especificações para ambientes que utilizam a orientação a objetos, voltados para sistemas distribuídos. É responsável pela definição da plataforma ORB, a qual tem por objetivo fornecer, basicamente, mecanismos pelos quais objetos fazem requisições e recebem as respectivas respostas de forma transparente.

Dentro de um mesmo ambiente ORB, um cliente, em qualquer processo, pode enviar uma requisição para um servidor, em qualquer outro processo, e receber a devida resposta, se houver. Esse mecanismo funcionará igualmente para clientes e servidores em um mesmo processo, em um mesmo processador mas em processos diferentes, ou em diferentes processadores conectados por rede.

A especificação da CORBA descreve a tecnologia escolhida pela OMG para definir uma estrutura padrão para a implementação de um ORB. Dessa forma, garante-se a portabilidade dos clientes e dos objetos prestadores dos serviços, especificando-se serviços e interfaces comuns.

Este capítulo apresenta os conceitos básicos da CORBA e uma análise dos conceitos pertinentes ao trabalho, além de uma descrição do estado atual dos Serviços de Objetos e Facilidades Comuns. São apresentadas, também, as implementações de dois ORBs: a ORBeline (um produto da PostModern) e a Orbix (um produto da IONA). Enquanto a Orbix é descrita em linhas gerais, a ORBeline é mais detalhada, uma vez que faz parte do protótipo implementado, sendo efetuada uma pequena análise comparativa dessa plataforma com o RM-ODP.

3.2 Conceitos básicos

A CORBA define um modelo de objeto bastante simples, visando facilitar o seu mapeamento para os vários modelos já existentes e implementados. Possui os seguintes elementos principais:

- **objeto** - entidade identificável e encapsulada capaz de fornecer serviços. Qualquer conjunto de serviços que possa ser representado por uma interface bem definida e passível de

ser invocado através de uma referência de objeto é um objeto na CORBA [Sol95]. Por exemplo: uma aplicação, um *daemon*, uma planilha eletrônica, etc;

- **requisição** - mecanismo através do qual um cliente solicita um serviço de um objeto;
- **cliente** - é qualquer entidade capaz de invocar uma operação em um objeto, através do ORB;
- **tipo** - classifica objetos de acordo com características semelhantes;
- **interface** - descreve para o sistema as operações que podem ser fornecidas por um objeto; e
- **operação** - um serviço que pode ser requisitado por um cliente.

Esse modelo é mais específico e prescritivo na abordagem dos conceitos significativos para o cliente. Para as implementações dos fornecedores de serviços existe uma maior flexibilidade, possibilitando às diversas tecnologias de objetos escolherem a sua própria forma de implementação.

O modelo proposto é descrito como "clássico", no sentido de que clientes enviam mensagens a servidores (objetos). Essas mensagens são compostas pela identificação do objeto invocado e zero ou mais parâmetros.

A herança múltipla é permitida, embora limitada às interfaces. Apesar de um desenvolvedor de serviços poder usar herança no projeto de uma implementação de objeto, esta dependência não é explicitada pela sintaxe da linguagem de definição de interface adotada. Não é do conhecimento do ORB se os serviços têm seu acesso através de uma hierarquia de interfaces são também relacionados por herança de implementação.

Possui duas categorias de tipos:

- básico - tipos que não são objetos e representam tipos de dados fundamentais: inteiro curto[1] e longo[2] com ou sem sinal, número de ponto flutuante (padrão IEEE[3]) com 32 e 64 bits, caracter (padrão ISO Latin-1), boolean, enumeração, cadeia de caracteres e um tipo não específico, chamado *any*, o qual pode representar qualquer tipo básico ou construído. Além desses, existe um tipo de dados especial, o octeto, composto por 8 bits, e que não sofre conversão quando é transferido de um sistema para outro; e
- construído - entidades de mais alto nível e mais complexas que as anteriores. Podem ser: estruturas, que operam de modo similar às *structs* na linguagem C; uniões, que operam como *unions* em C; seqüências, um tipo de vetor com tamanho variável que contém um único tipo de objeto, incluindo outras seqüências; e vetores, de tamanho fixo e de um único tipo.

  Um tipo importante desta categoria é o Interface, que especifica o conjunto de operações que uma instância deste tipo deve suportar.

[1] Tradução de *short*.
[2] Tradução de *long*.
[3] *Institute of Electrical and Electronic Engineers*.

Além do modelo de objeto, a CORBA incorpora os conceitos a seguir, alguns dos quais serão detalhados no decorrer deste capítulo e cuja estrutura geral pode ser observada na Figura 3.1.

- **implementação de objeto** - compreende o código e os dados que caracterizam o comportamento de um objeto. Provê a semântica do objeto.

  Pode ser constituído por um ou mais objetos fornecedores de serviços diferentes;

- **núcleo do ORB** - responsável pelos mecanismos que conduzem uma requisição de um cliente para o adaptador de objetos responsável pela implementação de objeto;

- **linguagem de definição de interface (IDL[4])** - linguagem que define as interfaces, que serão chamadas pelos clientes e fornecidas pelas implementações dos objetos. Deixa disponível, para um cliente em potencial, as informações sobre os serviços que um objeto possui e a forma de invocá-los;

- **interface de invocação estática** - para invocar uma determinada operação em um objeto, o cliente utiliza um *stub*, o qual é pré-definido e corresponde a uma operação particular em um tipo de objeto específico;

- ***stubs*** - um procedimento local correspondendo a uma única operação, a qual é invocada quando o *stub* é invocado;

- **interface de invocação dinâmica (IID)**- permite a construção dinâmica de uma requisição, possibilitando a utilização de interfaces desconhecidas do Cliente no instante em que foi compilado. A invocação é efetuada sem a utilização de **stubs**: o cliente "monta" a requisição em tempo de execução, especificando o objeto a ser invocado, a operação a ser executada e o conjunto de parâmetros que serão necessários, e os envia através do núcleo do ORB;

- **repositório de interfaces** - armazena objetos que representam as informações, inicialmente descritas em IDL, de uma determinada interface. Através da consulta a esse repositório, o cliente verifica se existe alguma interface disponível capaz de atender ao serviço solicitado. Desse objeto são retiradas as informações para a construção da requisição em uma invocação dinâmica;

- **repositório de implementações** - armazena informações que permitem a localização e a ativação de implementações de objetos;

- **interface ORB** - permite a interação de clientes e implementações de objetos diretamente com o ORB. Possui algumas poucas operações que são comuns a todos os objetos;

- **adaptador de objetos** - provê a interface principal através da qual uma implementação de objeto pode acessar os serviços fornecidos pelo ORB.

[4] *Interface Definition Language.*

- **esqueletos** - componentes definidos unicamente para uma determinada interface. São usados pelo adaptador de objetos para enviar requisições para os métodos específicos da referida interface; e
- **referência de objeto (refobj)** - identificador não-ambíguo de um objeto.

Na Figura 3.1 pretende-se realçar o fato de que a plataforma ORB é um mecanismo de comunicação entre o Cliente e o Servidor, situado logicamente entre ambos, e que possui as interfaces muito bem definidas.

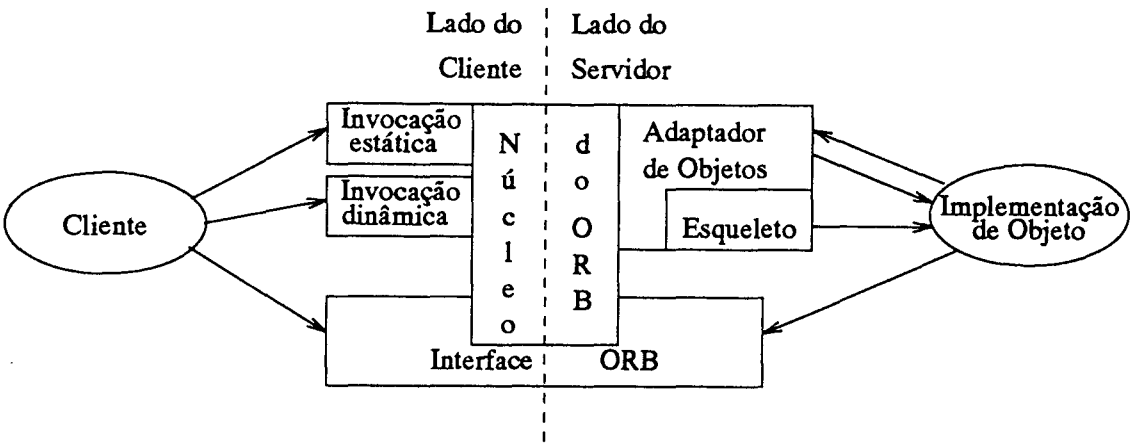

Figura 3.1: Estrutura da CORBA

3.3 Aspectos do Cliente

Um Cliente consiste em um programa ou processo que invoca uma operação em um objeto. Entretanto, uma Implementação de Objeto também pode ser cliente de um outro objeto. Na realidade, o caso de um fornecedor de serviços utilizar os serviços de outros objetos é bastante comum em um ambiente distribuído.

Para solicitar um serviço, o cliente deve, necessariamente, possuir uma referência para o objeto que o fornece. Para uma invocação, devem ser especificados: o objeto a ser invocado (alvo), a operação a ser executada e os parâmetros de entrada e/ou retorno.

A forma como a referência de objeto é obtida pelo Cliente não é especificada pela CORBA: o seu armazenamento pode tanto ocorrer em um BDOO[5] como estar contido em uma mensagem de correio eletrônico [ORB93].

Essa tarefa pode ser efetuada por um mecanismo como o Serviço de Nomes, por exemplo, com funcionalidade semelhante a um servidor de nomes, mas padronizado pela OMG e que será mais detalhado na seção 3.8.3, ou por um serviço de localização interno ao próprio ORB.

De posse da referência de objeto, o Cliente pode iniciar uma invocação, acionando o *stub* específico da operação desejada. Entretanto, como a representação da referência é dependente

[5]Banco de Dados Orientado a Objetos.

da linguagem de programação do Cliente, é necessário que o *stub* tenha acesso (através de rotinas de biblioteca, por exemplo) à representação em que o ORB implementa a referência de objeto, possibilitando a conversão necessária e a continuação da invocação (Figura 3.2).

Os *stubs* são responsáveis pelo "empacotamento/desempacotamento" e envio/recepção[6] dos dados que fluem entre o Cliente e o fornecedor de serviços.

O Cliente só tem necessidade de conhecer a interface do objeto "alvo", que se traduz no conjunto de operações oferecidos pelo objeto. A forma como os serviços são implementados não é informado e nem mesmo interessa ao Cliente.

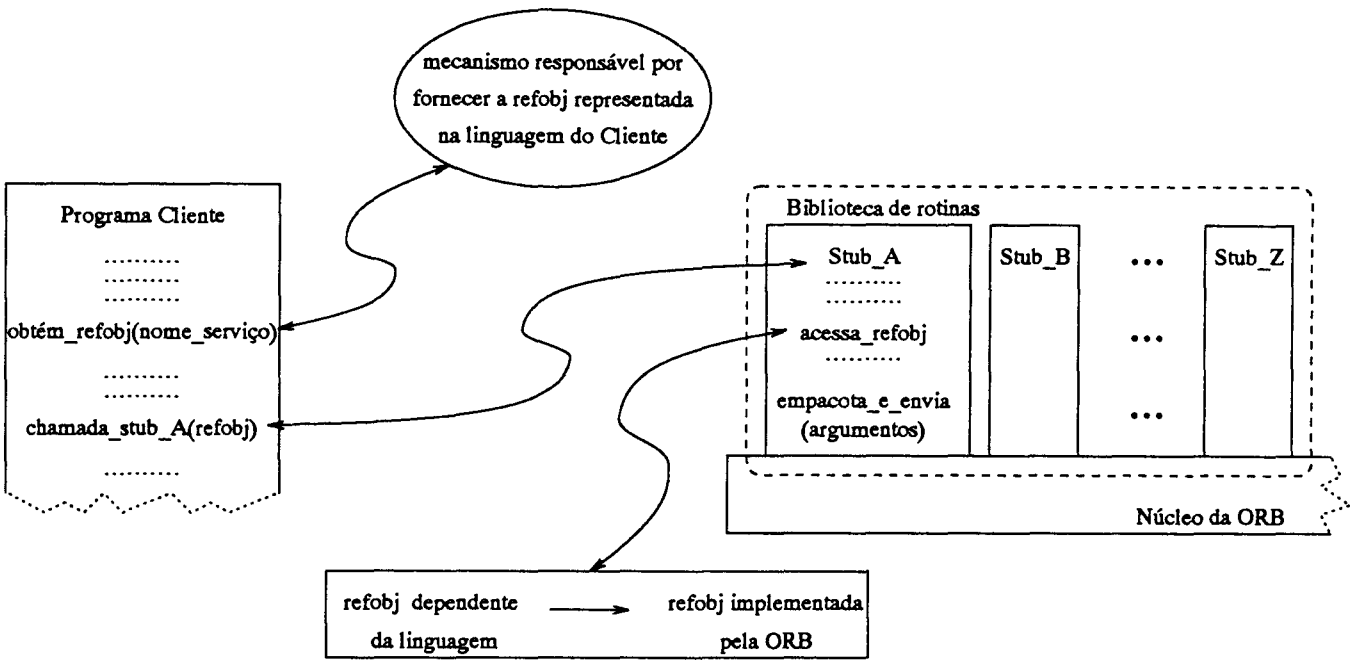

Figura 3.2: Exemplo de mecanismo de invocação do lado do Cliente

3.3.1 Invocação estática

Esse caso abrange a situação em que as interfaces, a serem solicitadas pelo Cliente, são conhecidas em tempo de compilação. A partir da definição dessas interfaces (em IDL), são criados os *stubs*, gerados pelo compilador de IDL. Cada operação dessa interface será representada por um *stub*, o qual será acessado pelo Cliente através de chamadas a rotinas de biblioteca. O cliente solicita serviços distribuídos como se fossem fornecidos por objetos locais através da simples invocação dessas funções (Figura 3.2).

Essas rotinas representam o mapeamento de linguagem entre a linguagem do cliente e a implementação do ORB. Dessa forma, as funcionalidades do ORB tornam-se disponíveis a clientes escritos em qualquer linguagem para a qual possam ser gerados *stubs* (a partir da especificação IDL da interface).

[6]Tradução de *marshalling/unmarshalling*.

A comunicação dos *stubs* com o ORB é feita através de interfaces privadas específicas para cada núcleo de ORB.

Como exemplo, observa-se uma parte do código de um Cliente para a ORBeline [Pos94], uma plataforma implementada de acordo com a especificação da CORBA e que será descrita na seção 3.9:

```
   main() {
...
classe *refobj_do_cliente = classe::_bind();
retorno = refobj_do_cliente->acrescenta_item(nome_item);
...
}
```

A operação ***_bind*** retorna uma referência de objeto (refobj_do_cliente), que é utilizada para a invocação do serviço. Nessa plataforma (implementada em C++), a referência de objeto é representada como um ponteiro para a classe que apresenta a interface desejada [Pos94, OMG94c]. Com essa referência, é possível efetuar uma invocação em uma operação da classe (acrescenta_item) com os devidos parâmetros (nome_item). É interessante notar que **acrescenta_item** representa, na realidade, um *stub*, compilado com o mesmo nome da operação oferecida pela interface em IDL, e que deverá ser implementada no servidor e apresentando-se como uma invocação normal a um objeto qualquer na linguagem C++.

3.3.2 Invocação dinâmica

Caso a interface do servidor tenha sido definida após o Cliente ter sido compilado, devemos utilizar a Interface de Invocação Dinâmica para executar essas invocações. A semântica das requisições será idêntica à utilizada com os *stubs*, de tal maneira que o lado do servidor não faça distinção entre as formas de invocação. Entretanto, será necessário explicitar as informações, as quais são intrínsecas no caso das invocações estáticas.

Os parâmetros de entrada/retorno citados são especificados: a) através de uma seqüência de chamadas a uma operação da IID (**add_arg**, em [OMG94a]) cuja função é acrescentar os novos argumentos à requisição, conforme pode ser observado na Figura 3.3; ou b) acrescentando-se diretamente uma lista (criada pela operação **create_list**) que já contém todos os parâmetros a serem utilizados na requisição.

O código do Cliente é responsável pela especificação dos tipos dos parâmetros e respostas. Estas informações podem ser retiradas do Repositório de Interfaces. Na Figura 3.3, podem ser observadas as setas numeradas: 1) é efetuada uma consulta ao Repositório de Interface em busca dos tipos dos parâmetros da operação requisitada; 2) a informação é obtida; e 3) a partir dessa informação é criado um pseudo_objeto Requisição, que funciona como um gabarito que deve ser preenchido pelos dados fornecidos pelo Cliente. Esse mecanismo, entretanto, é uma solução implementada pela ORBeline, na qual está "embutida" na operação de criação da requisição dinâmica a consulta ao Repositório de Interfaces, o que não é preconizada pela CORBA, que não se preocupa com isso, apesar de definir as operações para o acesso às informações contidas

naquele repositório.

Essa interface também está disponível sob a forma de rotinas de biblioteca.

Utilizando novamente a ORBeline como exemplo, para o caso de um cliente efetuando uma invocação dinâmica, temos:

```
main(int argc, char** argv) {
...
CORBA_DII::Request *requisição;
CORBA::Boolean retorno;
requisição = CORBA_DII::Request::create(''nome_interface'',
                        ''nome_operação'', ''nome_Repositório_Interface'');
requisição->bind();
CORBA::Value *Valor_da_Estrutura = requisição->arg(''nome_argumento'');
Valor_da_estrutura->member(''nome_A'') = argv[1];
Valor_da_estrutura->member(''nome_B'') = argv[2];
Valor_da_estrutura->member(''nome_C'') = argv[3];
requisição->invoke();
retorno = requisição->result();
...
}
```

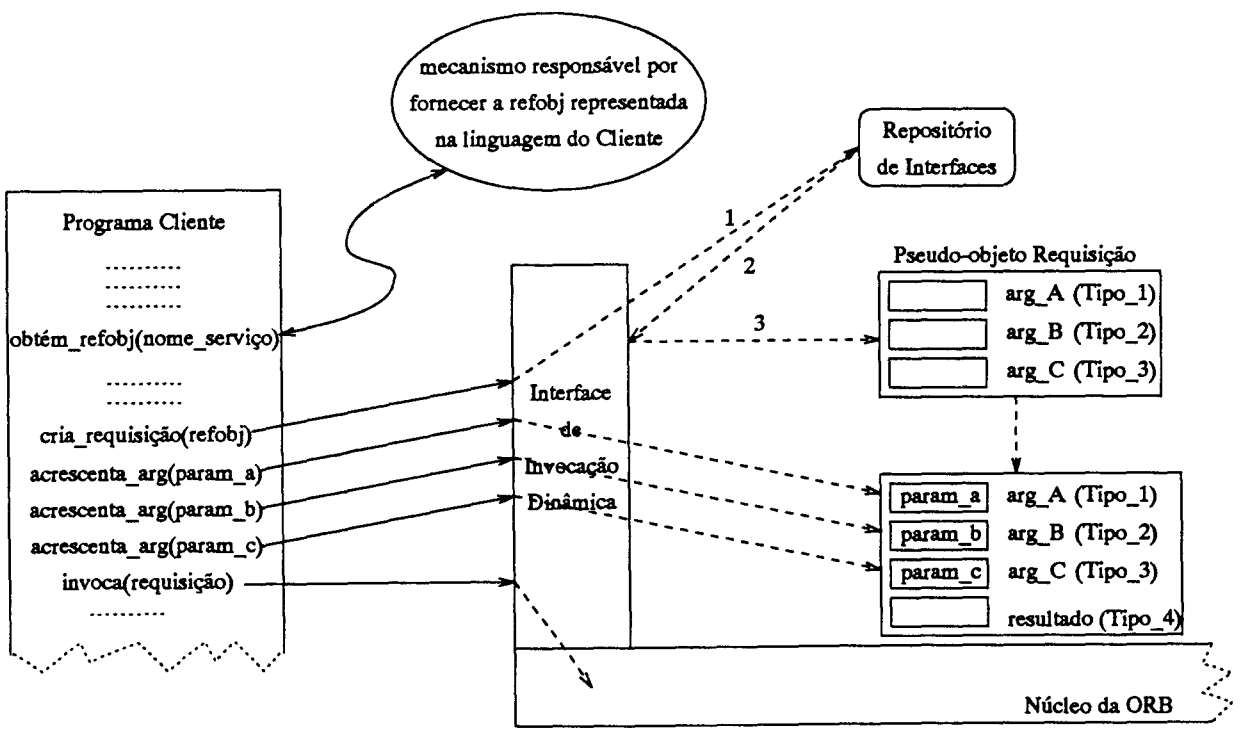

Figura 3.3: Mecanismo para uma invocação dinâmica

Inicialmente, a ORBeline cria um pseudo-objeto ***Request***, através da operação **create**, a qual recebe, como parâmetros, o nome da interface que possui a operação, o nome da operação invocada e o nome do Repositório de Interfaces que contém a descrição da interface que fornece o serviço procurado. Dessa operação retorna um apontador para a classe *Request* (requisição, no exemplo). Do repositório, são obtidos os tipos dos argumentos a serem utilizados como parâmetros da operação invocada. Em seguida, é efetuada a localização e conexão do cliente com um objeto que apresente a interface solicitada, através da função **bind()**. De posse dos tipos dos argumentos a serem enviados, estes devem ser associados aos seus respectivos valores. No exemplo, consideramos que a operação necessita de um argumento constituído por uma estrutura composta por três membros: A, B e C. Os valores desses membros são fornecidos ao cliente, cujo código está preparado para introduzi-los na requisição. Após isso, é efetuada a invocação, através do *invoke()*.

3.3.3 Invocação estática x Invocação dinâmica

São observadas as seguintes vantagens, em relação à forma de invocação [Sol95]:

- **Invocação estática:**
  - Forte checagem de tipos em tempo de desenvolvimento;
  - Proteção contra modificações que possam tornar interfaces incompatíveis; e
  - Sintaxe de chamada muito mais simples que a usada na IID, o que facilita a escrita do código do cliente.
- **Invocação dinâmica:**
  - Permite uma grande flexibilidade, possibilitando a utilização de objetos cujas interfaces sejam definidas em tempo de execução;
  - Sua natural lentidão em relação à estática é reduzida pela latência da rede;
  - Permite, embora não sendo uma exigência, a checagem de tipos em tempo de execução;
  - Permite mais opções para invocação: *one-way* (o cliente não espera resposta do servidor, portanto, é liberado) e sincronismo retardado[7] (o controle retorna ao cliente tão logo a requisição é enviada, sendo função do cliente perguntar ao ORB se o servidor já forneceu a resposta).

A flexibilidade fornecida pela invocação dinâmica é essencial para aplicações que necessitam descobrir os serviços em tempo de execução [OH95], operando em objetos quaisquer que sejam seus tipos ou operações, como: navegadores[8], gerenciadores de recursos, construtores (de GUI[9], por exemplo), interpretadores, etc. Também existem os casos em que os nomes das operações a serem invocadas não são conhecidos em tempo de desenvolvimento [Sol95].

[7] Tradução de *deferred sincronization*.
[8] Tradução de *browsers*.
[9] *Graphical User Interface*.

Entretanto, essa grande vantagem da invocação dinâmica tem um custo alto em relação à estática. Para uma requisição a um objeto remoto será necessário: a) obter uma definição de interface; b) através dessa definição, obter as informações sobre a operação requisitada; c) criar um pseudo_objeto Requisição para essa operação; d) acrescentar cada argumento (ou a lista completa de argumentos, caso seja possível) da operação na Requisição; e e) efetuar a invocação. Se a operação não possuir argumentos, serão necessárias pelo menos duas chamadas a funções, das quais pelo menos uma resultará em uma chamada de procedimento remoto. A invocação estática pode ser sempre conseguida com apenas uma. Também deve ser considerada a necessidade de interpretação da Requisição. Para a maioria das aplicações, especialmente aquelas escritas em uma linguagem compilada, como C++, é muito mais eficiente efetuar requisições de forma estática do que dinâmica [Vin93].

3.4 Aspectos da Implementação de Objeto

A Implementação de Objeto contém as informações necessárias para criar um objeto e estabelecer seu comportamento. Define os dados para uma instância e o código para os métodos.

Uma implementação de objeto pode definir várias interfaces, apresentando-as, cada qual, como um conjunto de métodos. Esses métodos serão acionados pelos esqueletos através de uma chamada ascendente[10]. Cada esqueleto refere-se a um conjunto de operações que constitui uma única interface.

Durante uma invocação, a resolução do método[11] é efetuada pelo trabalho conjunto do núcleo do ORB, do adaptador de objetos e do esqueleto. O mecanismo de invocação pode ser visto na Figura 3.4.

Através da referência de objeto, o método encontra os dados do objeto que está sendo invocado.

Normalmente, uma implementação de objeto registra-se com o ORB, através do adaptador de objetos, informando quais objetos, e respectivas interfaces, podem ser acessados, bem como a forma de iniciar sua execução e efetuar sua inicialização, caso necessário.

Uma Implementação de Objeto independe do ORB, sendo, portanto, capaz de ser executada sem alterações, em qualquer dessas plataformas, desde que as mesmas implementem o mapeamento de linguagem e o adaptador de objetos adequados.

3.5 Adaptador de objetos

Essa interface, além de permitir o acesso, pela implementação de objeto, aos serviços fornecidos pelo núcleo do ORB, funciona como uma camada sobre esse núcleo, que irá suprir as deficiências do ORB que está sendo utilizada, em relação aos serviços exigidos pela implementação de objeto e não implementados pela plataforma em questão.

[10] Chamada ascendente - invocação que sai do ORB, normalmente a partir de seus esqueletos ou do adaptador de objetos.

[11] Resolução do método - seleção do método adequado para atender uma invocação de uma operação.

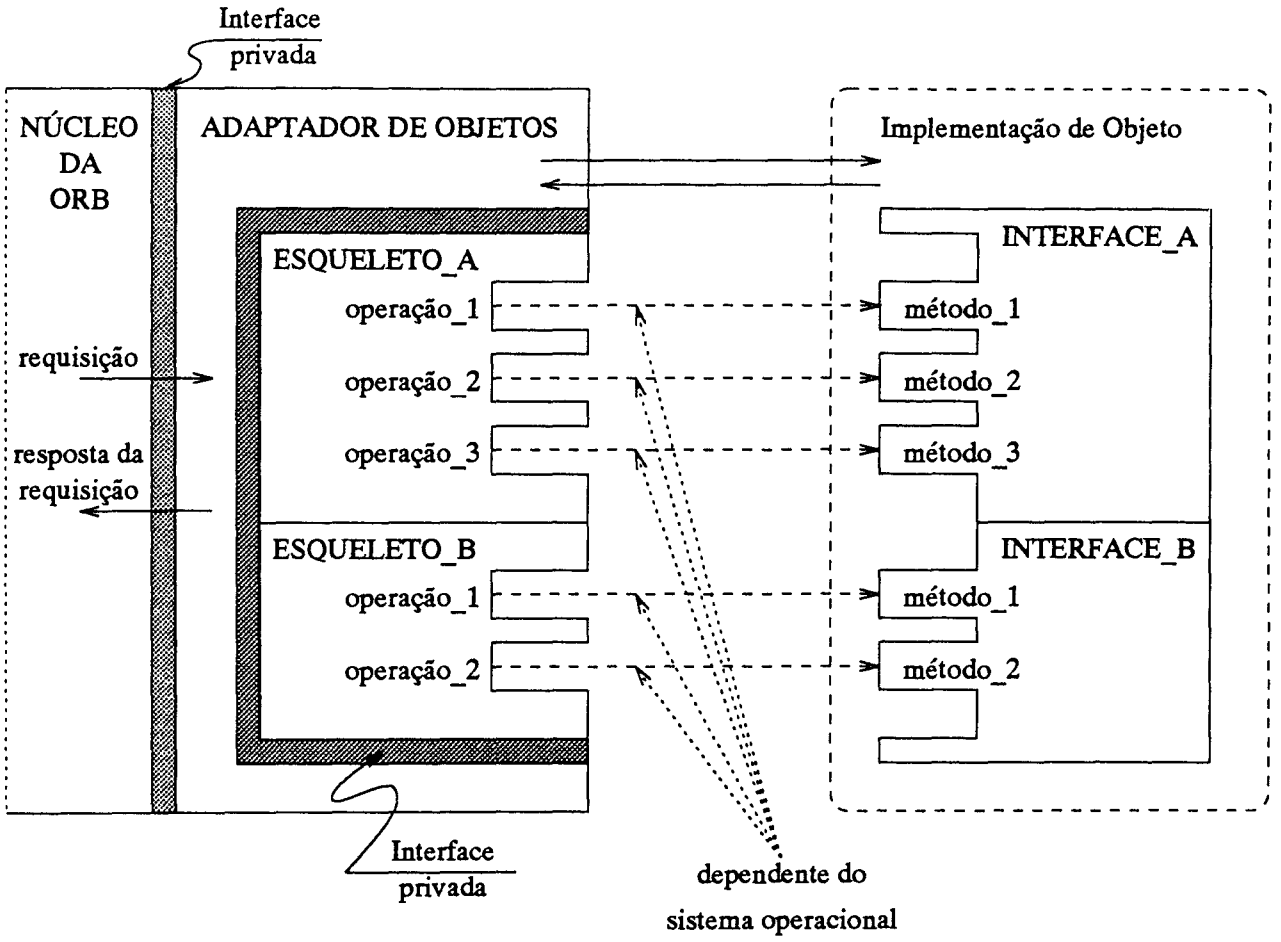

Figura 3.4: Exemplo de mecanismo de invocação do lado da Implementação de Objeto

O adaptador de objetos também permite estender o modelo de objeto, permitindo o mapeamento dos diversos modelos utilizados pelas implementações de objetos para o especificado pela CORBA [NWM93].

Durante uma invocação, o adaptador de objetos será o responsável por definir qual esqueleto será acionado para efetuar a chamada aos métodos corretos da implementação de objeto (Figura 3.4).

Um adaptador de objetos será responsável pelas seguintes funcionalidades básicas:

- **geração e interpretação de referências de objetos** - por requisição da Implementação de Objeto, o Adaptador de Objetos, juntamente com o Núcleo do ORB, efetua a criação de uma referência de objeto. Usa a seguinte operação:

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

onde,

- **ReferenceData** - é uma informação determinada pela Implementação de Objeto e , normalmente, só para ela tendo significado;
- **InterfaceDef** - objeto no Repositório de Interfaces que descreve completamente as interfaces implementadas pelo objeto; e
- **ImplementationDef** - objeto do Repositório de Implementação que especifica a implementação a ser usada pelo objeto.

- **autenticação de requisições** - o adaptador de objetos possui uma operação (**get_principal**, em [OMG94a]) que permite o acesso a informações sobre o Cliente que está invocando um serviço. Essa operação é chamada pela implementação de objeto que, analisando seu conteúdo, poderá decidir se o Cliente tem permissão ou não para solicitar seus serviços.

  Como exemplo, a ORBeline apresenta os seguintes parâmetros que compõem as informações referentes ao Cliente:

  - nome da máquina;
  - nome do cliente (aplicação), para entrada no sistema;
  - senha;
  - nome do usuário que está utilizando a aplicação;
  - identificador (*id*) do processo que executa a invocação;
  - dados do usuário: informações fornecidas pelo Cliente (aplicação) não interpretadas pelo ORB; e
  - tamanho dos dados do usuário.

- **Registro de implementação de objeto** - através das informações contidas nos repositórios de interface e de implementação, o adaptador de objetos faz uma associação entre a referência de objeto e sua respectiva interface e implementação;

- **Ativação e desativação de implementações de objetos** - a ativação é efetuada iniciando, por determinação do adaptador de objetos, a execução do programa que compreende a implementação de objeto, através de funcionalidades específicas do sistema operacional (e, portanto, fora da especificação CORBA). A ativação é automática, sendo iniciada quando o adaptador recebe uma invocação destinada a uma implementação que ainda não esteja em execução. A partir daí a própria implementação de objeto efetuará a inicialização necessária e se registrará com o adaptador de objeto, ficando pronta para receber invocações a objetos ou, caso estes não estejam ativos, chamadas às suas rotinas de ativação dos objetos.

  A ativação do objeto é transparente ao cliente. São definidas as seguintes formas de ativação:

- servidor compartilhado - múltiplos objetos são ativados em um único processo servidor;
- servidor não compartilhado - cada objeto é ativado em seu próprio processo servidor;
- servidor persistente - o processo servidor é ativado por um mecanismo qualquer externo ao ORB, registrando-se com o Adaptador de Objetos quando estiver pronto para receber uma requisição; e
- servidor por método - cada operação da interface do objeto é implementada em um processo servidor distinto.

Para a desativação, será necessário que a própria implementação de objeto faça essa solicitação ao adaptador de objetos, podendo desativar toda a implementação ou apenas alguns objetos específicos; e

- **Invocação de métodos** - é função exclusiva do adaptador de objetos prover a conexão dos métodos com os esqueletos.

Devido a existência de diferentes ORBs, fornecendo diferentes serviços, seria difícil para o núcleo do ORB fornecer uma única interface que satisfizesse convenientemente todos os objetos. O adaptador de objetos fornece às implementações de objetos o acesso a serviços que não estejam no núcleo do ORB. A CORBA pretende conseguir atender uma grande variedade de implementações através de um pequeno número de adaptadores, elaborando-os de forma suficientemente genérica.

Considerando o fato citado, a especificação CORBA apresenta um adaptador de objeto básico (BOA[12]) o qual suporta um conjunto de serviços que provê uma funcionalidade mínima,embora atenda a maioria dos tipos de implementações de objetos.

Uma possível implementação do BOA, sugerida em [OMG94a], o separa em duas partes. Uma será responsável, usando recursos não fornecidos pelo ORB, pela ativação da implementação, e outra, funcionando como uma biblioteca da implementação de objeto, estabelece a ligação entre os métodos da implementação e os esqueletos.

3.6 Linguagem de Definição de Interface - IDL

A IDL é uma linguagem declarativa, possibilitando a separação entre interface e implementação, o que é bastante enfatizado no desenvolvimento de sistemas orientados a objeto [Vin93] e em aplicações que interoperam em sistemas distribuídos heterogêneos.

Através dessa linguagem, as pessoas encarregadas do desenvolvimento especificam interfaces, que descrevem métodos e atributos de objetos que podem estar distribuídos em um sistema computacional.

O tipo de um objeto, definido através da IDL, é descrito como o conjunto de funcionalidades que um objeto é capaz de fornecer.

[12] *Basic Object Adapter.*

Pela especificação de sua interface, composta pelas operações que apresenta e os parâmetros dessas operações, os clientes de uma determinada implementação de objeto tomam conhecimento dos serviços que estão disponíveis e da forma como chamá-los.

A partir dessas definições é possível fazer o mapeamento dos objetos da CORBA para uma linguagem de programação específica.

Esse mapeamento abrange os seguintes itens, entre outros:

- definição dos tipos de dados específicos da linguagem;
- definição das interações entre as invocações de objetos e os *threads* de controle no cliente ou na implementação; e
- definição das interfaces para acessar objetos através do ORB: *stubs*, IID, esqueletos, adaptadores de objetos e a interface ORB.

A IDL é independente de qualquer linguagem de programação, embora pretenda suportar diferentes linguagens através de um conveniente mapeamento para os objetos da CORBA. Naturalmente, as facilidades oferecidas pelas linguagens tornarão esse mapeamento mais simples ou mais complexo. Atualmente, estão especificadas essas conversões para C, C++, Smalltalk, Ada e COBOL e já existem submissões para Java.

A IDL apresenta as mesmas regras básicas adotadas pelo ANSI[13]/ISO C++, enquanto que sua gramática é um subconjunto, estendido para suportar invocações.

Uma especificação em IDL pode consistir de definições de tipos, constantes, exceções, módulos e interfaces.

Os módulos permitem agrupar outras definições em unidades lógicas, formando um escopo de nomes.

As seguintes definições também criam escopos, os quais podem ser aninhados: interface, estrutura, união, operação e exceção. A partir delas, os seguintes identificadores podem pertencer a um escopo: tipos, constantes, valores de enumeração, exceções, interfaces, atributos e operações.

Identificadores são definidos apenas uma vez em um escopo, mas podem ser redefinidos em escopos aninhados.

A definição de interface consiste em um cabeçalho, composto pelo nome da interface e uma especificação opcional de herança, e um corpo, composto pelas declarações de constantes, tipos, exceções, atributos e operações.

O nome da interface é representado por um identificador.

A declaração de uma operação compõe-se do seu nome, tipo de dado de retorno, tipos de todos os parâmetros, exceções possíveis e informações referentes à resolução do método.

Interfaces permitem herança. Uma interface **derivada** herdará os elementos de uma interface **base**, podendo, caso deseje, redefini-los. Também é possível herança múltipla: uma interface pode herdar de várias interfaces bases.

[13] *American National Standards Institute.*

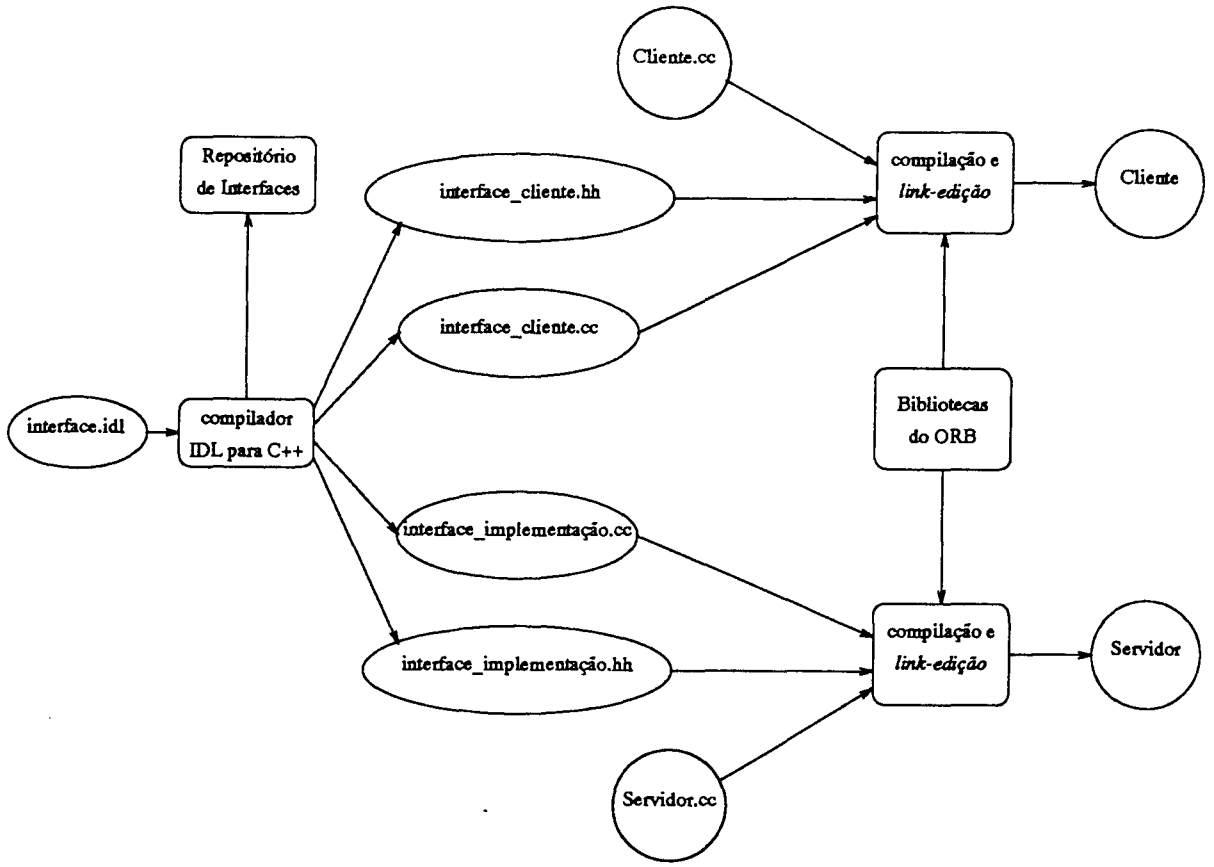

Figura 3.5: Exemplo do mecanismo utilizado através de um compilador IDL para criar um "cliente" e um "servidor", usando *stubs*

A Figura 3.5 apresenta os arquivos gerados por um compilador de IDL para C++ [Sol95]: dois arquivos de código e dois de cabeçalho. Para gerar o processo Cliente serão necessários os arquivos de cabeçalho, de código (interface_cliente.cc, contendo os *stubs*) e o programa criado pelo programador (Cliente.cc). O mesmo ocorre para o processo Servidor, com os arquivos cabeçalho, interface_implementação.cc (contendo os esqueletos) e Servidor.cc. Além disso, são necessárias, é claro, as bibliotecas do ORB. Um exemplo desse procedimento com a sua modificação para a invocação dinâmica será apresentado na seção 3.9.

Pode-se observar que o compilador é aproveitado para inserir as definições das interfaces no Repositório de Interfaces. Este mecanismo é utilizado na plataforma Orbix[14], por exemplo, sendo uma opção da operação de compilação. A ORBeline, por sua vez, possui um comando específico para essa tarefa, possibilitando, inclusive, apenas a análise sintática do arquivo em IDL.

[14]Implementação da especificação CORBA, pela IONA Technologies Ltd.

3.7 Repositório de Interfaces

Esse repositório é o responsável pelo armazenamento e gerenciamento das definições de interfaces de objetos.

Essas definições são armazenadas como objetos e o seu acesso é permitido em tempo de execução, possibilitando que o ORB invoque objetos cujas interfaces não eram conhecidas quando o cliente foi compilado.

De posse de uma referência para o objeto é possível obter a sua interface bem como todas as informações necessárias a ela relacionadas, em tempo de execução, através de funções definidas para o Repositório de Interfaces.

Cada interface é mantida como uma coleção de **objetos de definição de tipos** [OMG94g]: Repository, ModuleDef, InterfaceDef, AttributeDef, OperationDef, ParameterDef, TypeDef, ConstantDef e ExceptionDef.

Esses objetos são estruturados em uma hierarquia para a qual existe um conjunto de operações que permitem a localização de objetos no repositório.

São utilizados nomes para identificar módulos, interfaces, constantes, definições de tipos, exceções, atributos e operações. Esses nomes não são únicos no contexto de um Repositório de Interfaces, mas apenas no contexto de um módulo. Entretanto, existe um identificador de repositório (**RepositoryId**) que os identifica, unicamente, sendo definido como uma cadeia de caracteres, mas de estrutura opaca, permitindo que cada desenvolvedor de ORB o implemente da forma mais otimizada.

Um ORB pode possuir vários repositórios, com diferentes implementações e/ou objetos agrupados por motivos administrativos/tecnológicos.

O Repositório de Interfaces pode ser utilizado para:

- Prover a verificação de tipos das assinaturas das requisições;
- Auxiliar na verificação da correção do grafo de herança da interface;
- Auxiliar em prover interoperabilidade entre ORBs de diferentes implementações.

3.7.1 Códigos de Tipos

São valores que representam tipos. Consistem de um campo que identifica o tipo (caracter, estrutura, entre outros) e uma lista de parâmetros, com as informações que compõem os tipos. Possuem representação opaca.

Além da sua utilização na IID, indicando os tipos dos parâmetros, e no Repositório de Interfaces, representando as especificações dos tipos que são parte da declaração em IDL, é o componente básico do tipo **Any** (constituído por um Código de Tipo e seu valor).

Apesar de receberem uma pequena descrição na CORBA, a sua implementação é bastante complexa. Compreende, por exemplo, a maior parte do código do *Runtime* (implementa a IID, o BOA e as funcionalidades dos *stubs*) na Orbix [ORB93].

3.8 Modelo de Referência para a OMA

A Arquitetura para Gerenciamento de Objetos, proposta no documento OMA[15] Guide [Sol94], elaborado pela OMG, fornece os termos e definições a serem utilizados nas especificações.

O Modelo de Referência faz parte dessa arquitetura, identificando seus componentes, interfaces e protocolos. Compreende os seguintes componentes principais:

- **Gerenciador de Requisições de Objetos (ORB)** - conforme já comentado nas seções anteriores, provê o canal básico de comunicação através do qual objetos interagem;
- **Serviços de Objeto** [OMG95e, OHE96] - um conjunto de serviços, providos por objetos, compreendendo as funcionalidades básicas necessárias aos objetos CORBA. Juntamente com o ORB, preocupa-se com os aspectos básicos do gerenciamento de objetos distribuídos;
- **Facilidades Comuns** [OMG95a] - um conjunto de serviços, de uso mais específico que os Serviços de Objeto, que pode ser utilizado em vários tipos de aplicações; e
- **Objetos de Aplicação** - específicos para uma determinada aplicação.

A divisão nos três últimos itens compõe a estratégia da OMG para o desenvolvimento da sua padronização. Os Serviços de Objetos provêem as interfaces para os objetos envolvidos com os serviços de mais baixo nível; as Facilidades Comuns provêem as interfaces para os serviços de aplicações comuns; e os Objetos de Aplicação apresentam as necessidades específicas decorrentes de interfaces de aplicações desenvolvidas independentemente. Apesar de estar com alguns serviços já definidos, esta divisão não é rígida, podendo ocorrer de um serviço inicialmente considerado pertencente aos Serviços de Objeto migrar para as Facilidades Comuns ou, mesmo, ficar interno ao ORB. As possibilidades de invocação entre esses objetos são apresentadas na Figura 3.6.

A OMG pretende que exista um grande relacionamento de herança entre esses itens (Figura 3.7), facilitando a reutilização de interfaces padronizadas, a interação entre objetos de acordo com a base padronizada e o aumento da consistência no projeto de interfaces entre tipos de objetos.

Por exemplo, o Serviço de Objeto Ciclo de Vida define um serviço para a movimentação de objetos e deve ter sua interface herdada pelo objeto responsável por uma Facilidade Comum que pretenda possuir a capacidade de ser movimentada no sistema. Da mesma forma, um objeto qualquer, criado especificamente para uma determinada aplicação, pode herdar essa interface, incorporando a capacidade de movimentação.

3.8.1 Serviços de Objetos

Várias interfaces distintas podem definir um mesmo Serviço de Objeto. Conceitualmente, podemos classificá-las de acordo com seus clientes [OMG95e]:

[15] *Object Management Architecture.*

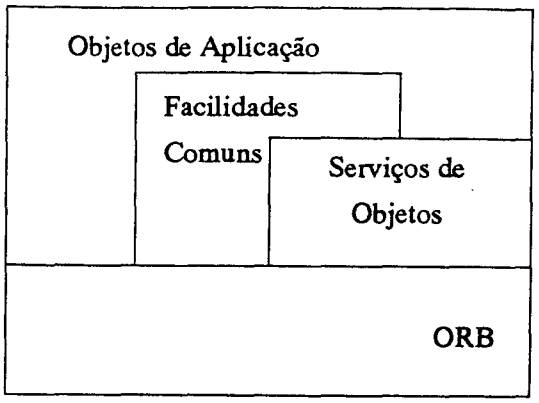

Figura 3.6: Invocações possíveis entre os componentes da arquitetura elaborada pela OMA

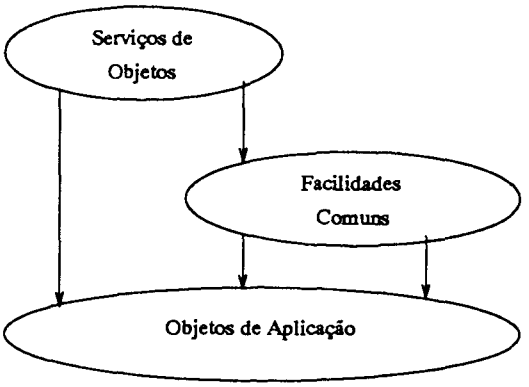

Figura 3.7: Relacionamento de herança na arquitetura OMA

- **audiência** - considerando os objetos que consomem os serviços oferecidos por uma interface. Possui as categorias:
  - **interface funcional** - define as operações invocadas pelos clientes de um Serviço de Objeto;
  - **interface de gerenciamento** - semelhante à anterior, mas específica para a comunicação com os serviços de gerenciamento do sistema; e
  - **interface de construção** - define as operações que permitem a troca de informações entre o Serviço de Objeto e os demais objetos que o auxiliarão na consecução do serviço.
- **portadora**[16] - considerando os tipos de objetos que apresentam a interface. Podem ser:
  - **objeto específico** - sua existência tem como única finalidade suportar uma parte do serviço provido por um determinado Serviço de Objeto; e

[16] Tradução de *bearer*.

- **objeto genérico** - também suporta parte do serviço, entretanto, não é o propósito de sua existência. Um objeto, já com determinada funcionalidade específica, herda uma interface de um certo Serviço de Objeto, aumentando suas capacidades.

Serviços de Objetos podem ser invocados por Objetos de Aplicação, Facilidades Comuns ou outros Serviços de Objetos, como pode ser visto na Figura 3.8 e, geralmente, se relacionam com outros Serviços de Objetos para a execução de certas tarefas obedecendo a um dos seguintes tipos de relacionamentos entre suas interfaces:

- **herança** - um Serviço de Objeto herda a interface de outro Serviço de Objeto;
- **uso** - um Serviço de Objeto usa a interface de outro Serviço de Objeto; e
- **implicação** - quando existe alguma dependência intrínseca entre os serviços fornecidos por um Serviço de Objeto e outro. Por exemplo, a existência de um repositório que suporte uma operação de busca implicará a existência de um serviço de consulta.

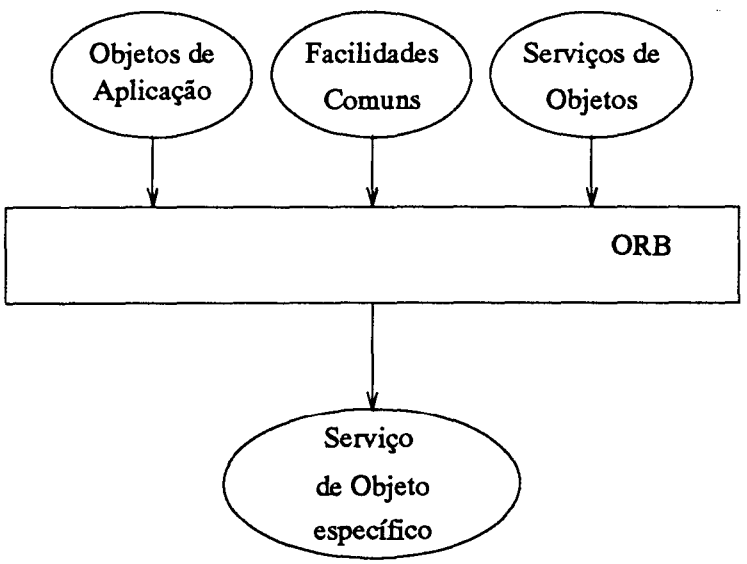

Figura 3.8: Invocações possíveis de um Serviço de Objetos

Existem vários Serviços de Objetos definidos, entretanto, muitos ainda estão em processo de adoção. Atualmente, foram adotados os seguintes:

- **COSS1**[17] [OMG94b]: adotados em dezembro de 1993.
  - **Notificação de eventos**: possibilita a troca de mensagens entre objetos sem que haja uma comunicação específica entre eles, funcionando como um "quadro de avisos". Dessa forma, os fornecedores das mensagens são isolados dos consumidores,

[17] *Common Object Services Specification*, vol. 1.

permitindo que um objeto acesse ou deixe de acessar as informações sem afetar o restante dos objetos.

O objeto **canal de eventos** é o responsável por essa troca de mensagens, sendo modelado como um objeto CORBA sendo acessado, portanto, através de requisições. Os eventos propriamente ditos não podem ser tratados como objetos uma vez que o modelo de objetos CORBA não admite a passagem de objetos por valor;

- **Ciclo de vida**: apresenta serviços para criação, destruição e movimentação de objetos;
- **Nomes**: suporta a implementação e administração heterogênea e distribuída de nomes e seus contextos. Considerando-se que as referências de objetos são totalmente opacas e o seu formato, quando passado para cadeia de caracteres, também não é conveniente para o entendimento por pessoas, o Serviço de Nomes permite uma representação de objetos compreensível, associando nomes a referências de objetos.

  Para tanto, considera-se um **contexto de nomes**, modelado como um objeto que contém um conjunto de associações nome-referência de objeto, no qual cada nome é único. Vale ressaltar que os nomes, portanto, não são globais, mas dependem do contexto. Através de uma hierarquia de contextos, possibilita a criação de uma seqüência de nomes para referenciar um objeto. Por fim, objetos podem ter múltiplos nomes e podem existir em múltiplos contextos; e
- **Persistência**: define interfaces comuns para os mecanismos usados no armazenamento e gerenciamento do estado persistente[18] dos objetos. O gerenciamento deve ser de uma maneira independente do tipo de armazenamento.

- **COSS2** [19]: adotados em dezembro de 1994.
  - **Controle de concorrência** - define como um objeto gerencia o acesso simultâneo de vários clientes a um objeto, mantendo a consistência e coerência tanto do objeto quanto de seus clientes. Provê um gerente de trancas que pode obter trancas:

    a) para uma transação - as trancas são liberadas pelo serviço de transação quando a transação termina normalmente ou aborta; e

    b) para um *thread* - a liberação das trancas é de responsabilidade do usuário. O *thread* deve estar executando fora de uma transação.

    Os clientes desses dois tipos são serializados um em relação ao outro.
  - **Externalização** - define protocolos e convenções para a externalização e internalização de objetos. A externalização consiste no armazenamento do estado do objeto em uma *stream* de dados (em memória principal ou secundária, por exemplo). Essa forma pode ser armazenada por um tempo indefinido, ser transportada por meios fora do escopo do ORB (um disquete, por exemplo) e ser internalizada (o objeto

[18] Estado a partir do qual um objeto pode reconstruir seu estado dinâmico (tipicamente em memória e de curta duração).

[19] *Common Object Services Specification*, vol. 2.

é novamente criado) em outro ORB (sem comunicação com o ORB que originou o objeto externalizado), retornando uma referência de objeto ao cliente (podemos ter um cliente para a externalização e outro para a internalização).

Um objeto externalizado pode representar um único objeto, um conjunto de objetos ou um grafo de objetos relacionados. O cliente usará sempre a mesma interface e não perceberá se a externalização de um determinado objeto implica a externalização de vários outros objetos relacionados.

Para garantir portabilidade, é definido um padrão para o formato de armazenamento dos dados.

Esse mecanismo é similar à cópia de um objeto (através do Ciclo de Vida), entretanto, nesse caso, o objeto é "congelado" para o seu transporte e só é novamente criado após ter sido internalizado.

- **Relacionamento** - provê uma forma de criar associações entre componentes e os mecanismos necessários para percorrê-las em um grupo de componentes. Pode ser usado para forçar restrições de integridade e acompanhar os vários tipos de relacionamentos (hierarquia, por exemplo).
- **Transação** - Consideradas essenciais para a construção de aplicações distribuídas confiáveis, as transações estão bastante interligadas com objetos distribuídos [OH95]. Possibilita que um cliente efetue requisições para múltiplos objetos que podem estar localizados em diferentes nós dentro do escopo da transação. O suporte a essas transações distribuídas necessita das facilidades do ORB, de forma que o contexto transacional deve ser passado, transparentemente, em cada requisição. Em [OH95], é considerado um serviço de baixo custo, apresentando um desempenho, pelo menos, tão bom quanto os serviços de transação procedural que obedecem ao padrão do X/Open DTP[20].

  Suporta transações aninhadas (uma transação embutida em outra) e agrupadas[21] (todas as operações no seu escopo são agrupadas em uma única entidade transacional).

- a partir do **RFP3**, [OMG94d]: sendo que **Temporização** foi adotado em março e **Segurança** em novembro, ambos em 1996.

  - **Segurança** [OMG94i] - a introdução desse serviço afeta o sistema distribuído como um todo, envolvendo o ORB, os Serviços de Objetos, as Facilidades Comuns e as Aplicações de Objetos. Aborda os aspectos: confidencialidade (o acesso a uma informação só é permitido por usuários autorizados), integridade (uma informação só pode ser alterada por usuários que tenham essa permissão), contabilidade (usuários têm suas ações de segurança relevantes registradas) e disponibilidade (o uso do sistema não pode ser negado a usuários autorizados).

    Provê as seguintes funcionalidades: identificação e autenticação dos requisitantes de um serviço (uma aplicação ou o próprio usuário humano); autorização e controle de

[20] *Distributed Transaction Processing.*
[21] Tradução de *flats.*

acesso; auditoria; comunicação segura entre objetos; e administração das informações de segurança sobre a identidade dos requisitantes, os objetos servidores e as opções de configuração das políticas de segurança.

- **Temporização** - provê um mecanismo para sincronização de relógios em um ambiente com múltiplas máquinas. Deve suportar uma noção sincronizada de tempo compatível com o CUT[22] e, para certos ambientes, um serviço de tempo confiável[23], o qual será o responsável por suportar as políticas e operações de um sistema (que obedece aos padrões da OMA) considerado seguro. O serviço de tempo confiável deve possuir características de integridade, confiabilidade no suporte a administração do controle de acesso e consistência (permitindo um fator de erro, desde que conhecido). Este serviço é de grande importância para a segurança, garantindo a integridade do tempo, possibilitando a confirmação da validade de uma autorização em um determinado instante, por exemplo.

- baseados no **RFP4**, [OMG94e]: adotados em novembro de 1995.

  - **Licenciamento** - provê uma infra-estrutura para especificação e gerenciamento de políticas para licenças e métricas para utilização. Considera-se que o esquema de fornecimento de licenças pode utilizar licenças flutuantes perpétuas, renováveis ou com medição de utilização do componente licenciado, por exemplo. Dessa forma, este serviço pode incorporar o valor das informações com possibilidade de acesso e estabelecer o correspondente valor a ser pago pelos requisitantes da informação. Deve abordar também o caso de uma licença expirar e, a partir daí, ser iniciado um mecanismo para medição de utilização do serviço requisitado.

  - **Propriedades** - consiste na associação de informações úteis ao estado do objeto. Por exemplo, a data em que a informação passou a existir e o nome do responsável por sua última atualização. Deve ser possível definir as propriedades por nome, enumerá-las, dar-lhes valores, recuperar os valores por nome e destruí-los.

  - **Consulta** - provê facilidades de consulta em objetos. Essas facilidades incluem operações de seleção (para informações de recuperação), de manipulação de dados (para inserção, atualização e remoção) e de definição de dados (para criação e destruição[24]).

    As especificações desse serviço são baseadas na linguagem de banco de dados SQL[25] e suas extensões para o paradigma de orientação a objetos.

- baseados no **RFP5**, [OMG95f]: adotados em outubro de 1996, com exceção do serviço de ***Startup***.

[22] *Coordinated Universal Time.*
[23] Tradução para *trusted.*
[24] Tradução de *dropping.*
[25] *Structured Query Language.*

- **Coleções**[26] - grupos de objetos os quais suportam algumas operações e exibem um determinado comportamento em função da natureza do grupo (e não do tipo dos objetos), constituem coleções. Por exemplo: pilhas, filas, listas, etc. Essas entidades possuem operações específicas, como inserção, remoção e busca de um elemento, que são independentes dos tipos que as compõem.
  Este serviço deve prover uma forma comum e genérica de criar e manipular as coleções de uso mais comum em sistemas de computação.
- ***Trading*** - provê facilidades para o oferecimento de serviços e para a descoberta incremental desses mesmos serviços de uma forma que garanta a interação com tipos consistentes. Os serviços oferecidos são registrados por uma operação de exportação (invocada pelo provedor do serviço) no *Trader*, recebendo como parâmetros as informações sobre o serviço disponibilizado. Essa operação contém a referência do objeto fornecedor do serviço, uma descrição do tipo do serviço (nome da operação e os tipos de seus parâmetros, por exemplo), informações sobre seus atributos e o contexto do *trading*. Para obter a referência a um determinado serviço, é necessário invocar uma operação de importação, informando a descrição do serviço desejado. O *Trader*, então, busca um serviço do tipo requisitado e cujos atributos preencham as restrições especificadas pelo cliente potencial.
- ***Startup*** [OMG96c] - possibilita que os Serviços de Objetos, Facilidades Comuns e outras aplicações suportadas pelo ORB possam efetuar sua inicialização. Esta funcionalidade serve tanto para o momento de inicialização do ORB, quanto durante tarefas para gerenciamento de sistemas e balanceamento de carga, por exemplo.

Alguns serviços, inicialmente considerados Serviços de Objetos, foram absorvidos pelas Facilidades Comuns: de arquivo, de cópia de segurança e restauração, e de controle operacional. Da mesma forma, alguns serviços foram transferidos para o escopo do ORB: repositório de implementação, repositório de interfaces, instalação e ativação, e replicação. Estes últimos passam a constituir, entre outros, o que se chama de Serviços do ORB, a serem discutidos na seção 4.3.1.

A Arquitetura para Serviços de Objetos (OSA[27]) deixa claro que os objetos propostos não abrangem todas as necessidades existentes e sugere alguns serviços adicionais: de localização/relocação; de índice; de regras; de Sagas (para suporte a transações não-convencionais); de armazenamento; e vários referentes ao gerenciamento de sistemas: de alarme/evento, de falha/recuperação, de operações remotas/distribuídas, de operação automática, de predição e análise de desempenho, de comunicação/rede e de capacidade/recursos, entre outros. Atualmente

3.8.2 Facilidades Comuns

Compreende duas categorias: Vertical, direcionada para áreas especializadas (como CAD[28] e sistemas financeiros, por exemplo), e Horizontal, utilizada pela maioria das aplicações, inde-

[26] Tradução de *collections*.
[27] *Object Services Architecture*.
[28] *Computer Aided Design*.

pendente de área. A facilidade Horizontal preocupa-se com necessidades "universais" e define interfaces comuns a serem compartilhadas pelas múltiplas áreas da categoria Vertical, a qual busca atender às necessidades de segmentos da indústria que envolvem computação [OMG95a].

A Facilidade Horizontal, por sua vez, divide-se nas seguintes categorias funcionais:

- **Interface do Usuário** - atende a todas as necessidades referentes à interface com o usuário. Atualmente, são descritas as seguintes:
  - Gerenciamento de Apresentação - possibilita tanto a apresentação da informação em qualquer tipo de equipamento (tela, impressora, *plotter*, etc) quanto o tratamento dessa informação (obtido do teclado, *mouse*, *scanner*, etc). Inclui suporte para: gerenciamento de janelas, bibliotecas de classes para os objetos da interface, objetos que permitem a interação com a interface (barra de menu, barra de *scroll*) e abstrações dos vários equipamentos de entrada/saída;
  - Gerenciamento de Apresentação Composta - suporta a apresentação de objetos em documentos compostos;
  - Suporte ao Usuário - visa prover ao usuário certas capacidades, de uso rotineiro e normalmente fornecidas por várias aplicações individualmente, através de facilidades comuns. Dessa forma, o usuário será beneficiado por um estilo de interface consistente e um comportamento previsível, e o desenvolvedor obterá a vantagem da reutilização do código com interfaces padronizadas. Apesar de estar prevista sua expansão para várias facilidades, espera-se atender, inicialmente, às de informações de auxílio[29] e de verificação de texto;
  - Gerenciamento de *Desktop* - provê uma infra-estrutura geral para a visualização do ambiente de trabalho do usuário; e
  - *Scripting* - suporta a criação interativa de *scripts*.
- **Gerenciamento de Informação** - possibilita a modelagem, definição, armazenamento, recuperação, gerenciamento, migração e troca de informações. Sua intenção é fornecer meios para prover a melhor utilização possível das informações. É composta por:
  - Modelagem da Informação - suporta a criação de modelos de informação e esquemas, através da definição de regras para estruturar, permitir o acesso e manter a informação;
  - Armazenamento e Recuperação da Informação - suporta o armazenamento persistente e a recuperação da informação;
  - Intercâmbio Composto[30] - suporta o armazenamento e troca de dados em documentos compostos;

[29] Tradução de *help*.
[30] Tradução de *Compound Interchange*.

- Troca de Dados[31] - permite a interação entre objetos através da troca de dados. Pode ser usado em várias formas de transferência de dados: troca de representações de objetos específicos de um domínio, troca de dados formatados, transferência de dados em grande quantidade[32], transferência de dados estruturados, troca de dados entre objetos e um software encapsulado;
- Troca de Informação[33] - permite a troca de diferentes tipos de dados e operações entre sistemas. Atua no nível mais alto de troca de informações entre sistemas, não sendo limitado à passagem de dados tradicionais ou a um conjunto estruturado de parâmetros, podendo, além desses, passar objetos de dados, eventos, notificações, etc;
- Codificação de Dados e Representação - suporta facilidades para a codificação de formatos de dados e conversões, fornecendo o acesso a serviços como compressão e descompressão de dados, e conversão para/de uma representação interna de/para uma forma canônica; e
- Operações de Tempo - suporta facilidades para a manipulação dos dados de calendário e de tempo.

- **Gerenciamento de Sistemas** - adotada pela OMG em novembro de 1996, permite o gerenciamento de sistemas de informação, fornecendo as funções bá sicas para a administração de um sistema, como controle, monitoração, segurança e configuração. É composta por:

  - Ferramentas de Gerenciamento - suporta a interoperabilidade das ferramentas de gerenciamento e o gerenciamento de coleção;
  - Gerenciamento de Coleção[34] - suporta a integração de gerenciamento de coleção e objetos gerenciados. Define operações necessárias para obter duas formas de relacionamento entre objetos, dessa forma as coleções resultantes podem ser consultadas ou ter operações aplicadas aos seus membros. Essa funcionalidade é necessária para permitir a interação natural entre administradores e aplicações; e
  - Controle - suporta o controle dos recursos do sistema e objetos gerenciados.

- **Gerenciamento de Tarefas** - trata da automação do trabalho, envolvendo tanto os processos do usuário quanto os do sistema, desde que pertencentes a um mesmo sistema de informação. Compreende:

  - *Workflow* - provê gerenciamento e coordenação de objetos que são parte de um processo de trabalho;
  - Agente - sua infra-estrutura é constituída por agentes estáticos, que podem solucionar algum problema específico (ordenar uma lista de mensagens de correio eletrônico,

[31] Tradução de *Data Exchange.*
[32] Tradução de *bulk.*
[33] Tradução de *Information Exchange.*
[34] Tradução de *Collection Management.*

por exemplo) ou prover uma nova funcionalidade a uma aplicação já existente, e agentes móveis, que representam o conceito de "mensagem inteligente", contendo dados, programas, chamadas de procedimentos e informações administrativas a serem utilizadas pelos agentes estáticos;

- Gerenciamento de regras - suporta a aquisição de conhecimento, manutenção e execução de objetos baseados em regras (tais como agentes inteligentes);
- Automatização - permite, através de convenções e interfaces, que um objeto tenha acesso às funcionalidades de outro objeto. Seus objetivos são similares aos do ORB, entretanto, seus objetos típicos são visíveis pelo usuário (um documento ou um parágrafo, por exemplo) e existe um objeto com a capacidade de referenciar outro objeto, no contexto de um objeto servidor, sem especificar uma referência de objeto (a primeira linha e/ou o segundo parágrafo, por exemplo).

- **Gerais** [OMG95b] - compreende facilidades que afetam a qualidade de um sistema de informações:

  - Internacionalização - possibilita que um usuário utilize um sistema de informações ou aplicação em sua própria linguagem (inglesa, árabe, russa, etc) e convenções culturais (forma de apresentação de datas, representação da moeda, representação de dados numéricos, etc).

    Esta facilidade implica algumas extensões na CORBA, em vários Serviços de Objetos e na maioria das Facilidades Comuns.
  - Segurança - consiste em facilidades adicionais de segurança específicas para o mercado vertical (abordado pelas Facilidades Verticais) ou para aplicações de alto nível de utilização geral (abordada pelas Facilidades Horizontais). Pode usar as facilidades fornecidas pelo ORB ou Serviços de Objetos (que constituem o núcleo de segurança do sistema), estendendo-as sem afetar suas funcionalidades (controle de acesso e criptografia, por exemplo), para atender aos requisitos de um segmento de mercado particular.

Para as Facilidades Horizontais existem as seguintes RFPs:

- RFP1 - gerenciamento de apresentação composta e facilidade de intercâmbio composto. Adotadas em março de 1996;
- RFP2 - internacionalização e operações de tempo. A adoção de uma especificação padrão para essas facilidades está prevista para o primeiro semestre de 1997;
- RFP3 - troca de dados e agentes móveis. Esta RFP foi iniciada em novembro de 1995 e a adoção de suas facilidades está prevista para o primeiro semestre de 1997;
- RFP4 - trata de **Objetos Comuns de Negócios**. Esses objetos representam as semânticas comuns na maioria do negócios. Esta RFP também trata da necessária infra-estrutura para

que esses objetos atuem de uma forma cooperativa. Foi transferida para o Domínio dos Objetos de Negócios (seção 3.8.3), com adoção prevista para janeiro de 1998;

- RFP5 - meta-objetos. Para a especificação de esquemas de informação, com o objetivo de auxiliar na integração de processos e dados. Esta RFP esta com a adoção prevista para junho de 1997;
- RFP6 - impressão. Com o objetivo de prover as necessidades de impressão de um escritório moderno distribuído. Esta RFP esta com a adoção prevista para julho de 1997;
- RFP7 - gerenciador de método de entrada[35]. Com o objetivo de prover as necessárias conversões para aceitar parâmetros de entrada em caracteres asiáticos, por exemplo; e
- RFP8 - Internet. Para prover interfaces IDL que permitam o acesso aos serviços da Internet, como: FTP, Telnet, SMTP, POP3, WAIS, Finger e Gopher.

Os serviços descritos a seguir, compreendem exemplos de áreas que podem ser exploradas pelas Facilidades Verticais:

- Imagens[36] - provê o acesso e a troca de imagens[37] e seus dados correlacionados;
- Supervias[38] de Informação - suporta aplicações de serviço de informação multiusuário no ambiente de uma WAN[39];
- Manufatura Integrada por Computador - suporta interoperabilidade entre objetos de manufatura;
- Simulação Distribuída - suporta a interação de múltiplos objetos de simulação em ambientes virtuais. Suporta a simulação distribuída de controle de tráfego aéreo ou jogos de guerra, por exemplo;
- Indústria de Exploração e Produção de Óleo e Gás - suporta interoperabilidade no mercado vertical de petróleo;
- Contabilidade - suporta transações comerciais;
- Desenvolvimento de Aplicações - suporta interoperabilidade entre objetos de desenvolvimento de aplicações. Preocupa-se com a seleção, desenvolvimento, construção e evolução das aplicações necessárias para suportar a estratégia de um sistema de informações de uma empresa; e
- Mapeamento - suporta interoperabilidade entre objetos de mapeamento. Compreende os serviços utilizados pelas aplicações que necessitam ter a capacidade de acesso e apresentação de dados geoespaciais.

[35] Tradução de *Input Method Manager*.
[36] Tradução de *Imagery*.
[37] Definido como dois ou mais vetores dimensionais de dados, os quais podem ser derivados de sensores ou produzidos artificialmente.
[38] Tradução de *Superhighways*.
[39] *Wide Area Network*.

3.8.3 Evolução da Arquitetura para Gerenciamento de Objetos

Uma visão mais atual dessa arquitetura [OMG96b, Gro97], em constante evolução, apresenta uma nova categorização, por interfaces, do modelo de referência:

- **Serviços de Objeto** - interfaces para serviços gerais e de uso freqüente;
- **Facilidades Comuns** - interfaces para facilidades horizontais orientadas ao usuário final e aplicáveis na maioria dos tipos de aplicações;
- **Interfaces de Domínio** - interfaces específicas de um tipo de aplicação; e
- **Interfaces de Aplicação** - interfaces específicas de aplicações não padronizadas.

As Interfaces de Domínio são divididas em áreas como: Finanças, Medicina, Manufatura, Telecomunicações, Comércio Eletr ·co, Negócios e Transporte. A tendência, portanto, é que essas interfaces incorporem as Fa es Verticais.

Além disso, passa a abordar o da utilização de uma interface, introduzindo o conceito de **Objeto de Infra-estrutura**. bjeto é constituído por uma coleção de objetos que cooperam entre si com o objetivo · r uma solução integrada em um determinado domínio de aplicação ou tecnologia. Dessa a, possibilitam as funcionalidades de interesse direto de um usuário final naquele domínio particular.

Os objetos que o compõem são categorizados em **Aplicação**, **Domínio**, **Facilidade** e **Objetos de Serviço**, conforme as interfaces que suportem. Cada Objeto de Infra-estrutura pode ser composto por vários objetos de várias categorias.

3.9 ORBeline

A ORBeline [Pos94] é uma implementação da CORBA, revisão 1.1. A seguir, seus principais mecanismos e características serão descritos.

Linguagem de Definição de Interface

Uma interface, em IDL, ao ser compilada, gera quatro arquivos (Figura 3.9):

- um arquivo cabeçalho (interface_cliente.hh) contendo a declaração da classe que representa o objeto alvo (fornecedor do serviço) no lado do cliente. Essa classe, com mesmo nome da interface em IDL, apresenta:
  - operações da interface - são gerados métodos para cada uma dessas operações, representando, na realidade, os *stubs*. Ao serem invocados, como uma chamada normal em C++, causam a construção de uma requisição, seu "empacotamento" e envio à implementação de objeto. Havendo resposta, efetuam o "desempacotamento" dos parâmetros (ou exceções) de retorno e os entregam ao cliente.

- operação de *binding* - essa operação é invocada para obter uma referência de objeto para o objeto em C++ que representa localmente o objeto alvo (potencialmente remoto). As invocações das operações são efetuadas utilizando-se essas referências, resultando na invocação "real" dos métodos do objeto alvo.

  Além disso, essa operação (implementada com o nome ***_bind***), localiza o objeto alvo e estabelece uma conexão entre cliente e implementação de objeto (neste contexto, é o que se chama de *binding*). Possui opções que permitem que: a) o *binding* seja efetuado apenas quando ocorrer a invocação de um dos métodos do objeto alvo; b) caso ocorra uma falha na conexão, seja efetuado (ou não) um novo *binding* transparentemente ao cliente; e c) seja especificado o número de tentativas para o *binding*.

  Em princípio, a operação *_bind* só necessita saber o nome da interface do objeto alvo. Essa informação é conhecida desde o instante da sua criação e aparece obrigatoriamente na sua invocação (por exemplo: nome_interface::_bind()). Entretanto, admite a determinação de um objeto específico (nome_interface:: _bind("nome_objeto")), a máquina que deve ser utilizada (nome_interface:: _bind("nome_máquina")) ou ambos (nome_interface::_bind("nome_objeto", "nome_máquina")).

Também são mapeados como classes os tipos complexos que fazem parte da definição da interface, como estruturas, uniões e enumerações.

- um arquivo cabeçalho (interface_implementação.hh) contendo a declaração da classe C++ do lado do servidor, com seus esqueletos apropriados.

  Os esqueletos, que correspondem aos métodos gerados para cada operação da interface, são chamados pelo núcleo da ORBeline para "desempacotar" os parâmetros associados com a invocação do método, invocar o método correspondente na implementação, "empacotar" os parâmetros (ou exceções) de retorno e devolver o controle ao núcleo, o qual envia a resposta ao cliente. O nome dos esqueletos corresponde ao nome da operação "real" do objeto precedido por "_" (por exemplo, _nome_operação).

- dois arquivos contendo a implementação dos dois arquivos anteriores (interface_cliente.cc e interface_implementação.cc).

A implementação do objeto efetua a instanciação do objeto e, então, informa ao BOA, através da operação CORBA::BOA::impl_is_ready(), que o objeto está pronto para receber requisições.

A implementação do cliente pode obter uma referência para o objeto requisitado, através da chamada à operação _bind (essa referência também pode ser obtida por outros meios, como a resposta de uma invocação a outro objeto). Os métodos serão invocados nessa referência. Por exemplo:

```
...
classe_interface* referência_objeto = classe_interface::_bind();
referência_objeto->nome_operação(parâmetro_operação);
...
```

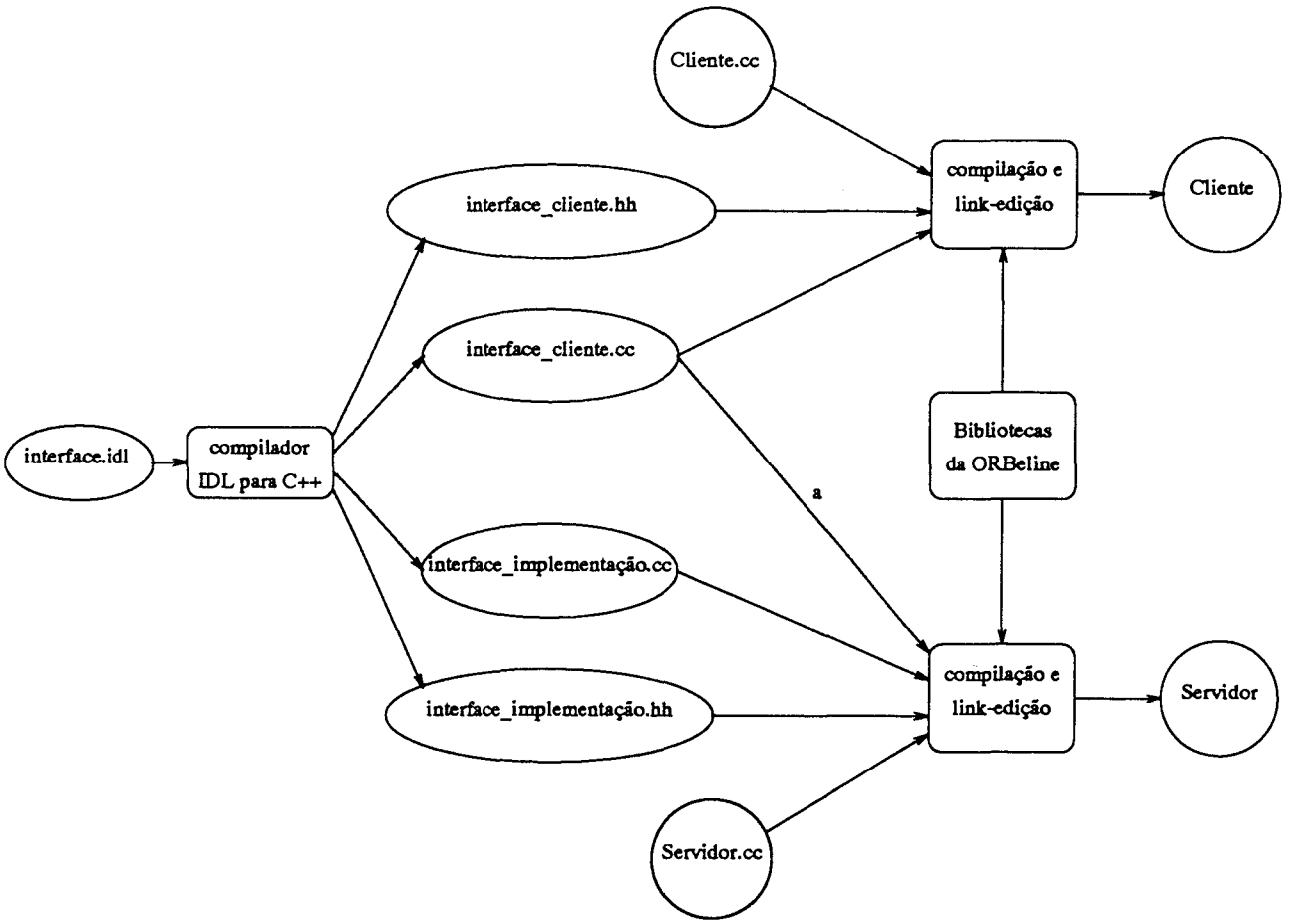

Figura 3.9: Mecanismo para a obtenção do Cliente e da Implementação de Objeto, a partir da definição de interface em IDL

Para a obtenção do executável do cliente, serão utilizados, além do programa do cliente, os arquivos de cabeçalho e o arquivo de código, gerados pelo compilador IDL para o cliente (Figura 3.9).

Para a obtenção do executável do servidor, o procedimento é idêntico para os arquivos de cabeçalho e de código gerados para a implementação. Entretanto, será necessário, ainda, o arquivo gerado para o cliente (representada pela ligação "a", na Figura 3.9).

A Figura 3.10 apresenta o mesmo mecanismo, modificado para uma invocação dinâmica. A geração do processo Servidor é idêntica a anterior, entretanto, ocorrem várias alterações para a criação do Cliente. As informações dos tipos dos parâmetros, a serem utilizados pela operação a ser requisitada, devem ser fornecidos pelo código do Cliente (Cliente.cc), através da IID. Esses dados podem ser obtidos através do Repositório de Interfaces, onde devem estar armazenadas as definições das interfaces e, portanto, as descrições das suas operações. Para a geração do Cliente, deve ser efetuada a compilação e *link-edição* do programa gerado pelo programador (Cliente.cc) e as rotinas de biblioteca do ORB. A IID pode ou não efetuar a verificação de tipos através de consulta ao Repositório de Interfaces (na ORBeline essa verificação sempre é efetuada).

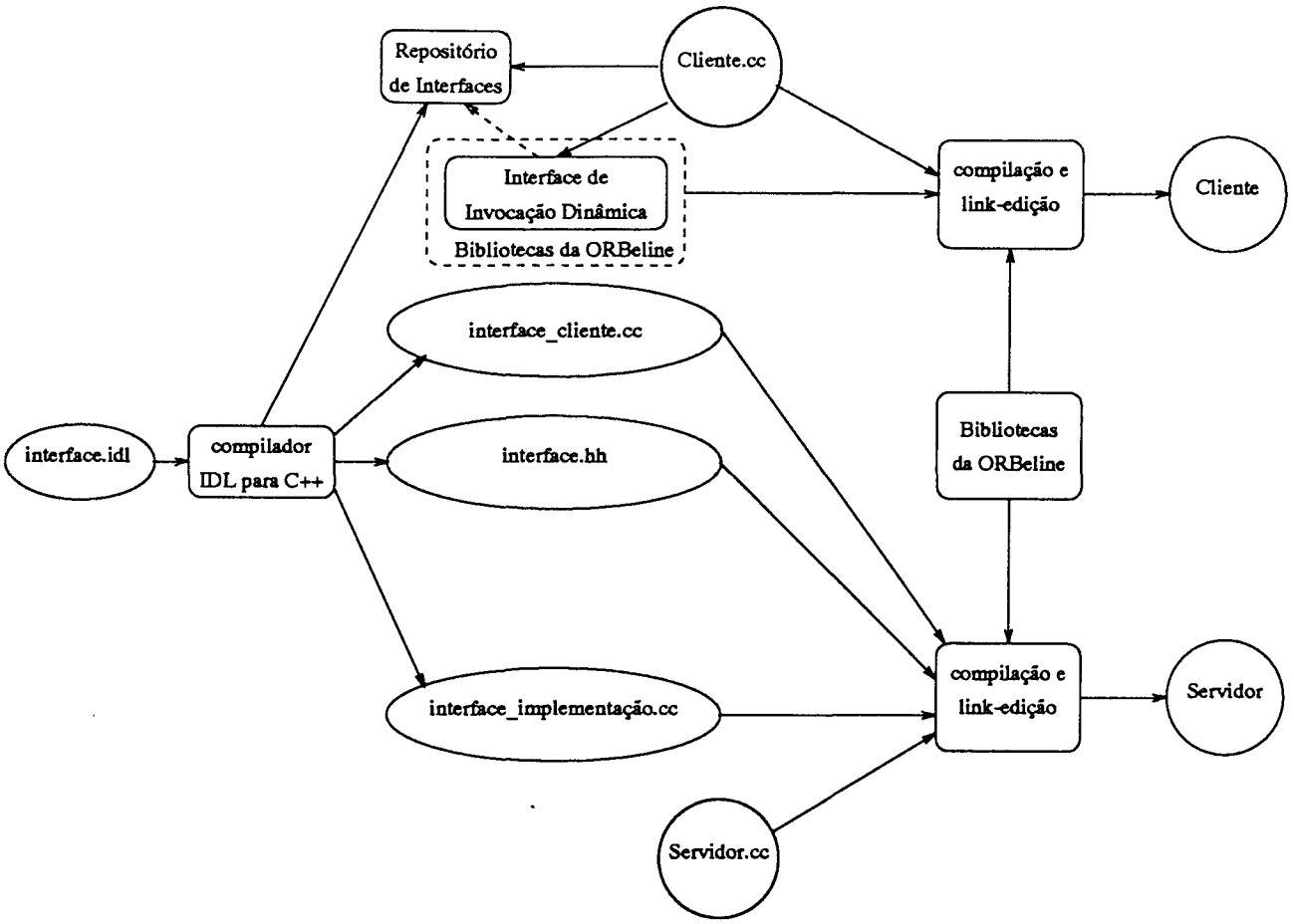

Figura 3.10: Exemplo do mecanismo utilizado através de um compilador IDL para criar um "cliente" e um "servidor", usando a Interface de Invocação dinâmica

A Figura 3.11 apresenta a hierarquia de classes obedecida pela ORBeline.

Repositório de implementações

É implementado como um objeto da ORBeline, armazenando, para cada objeto registrado, o nome da sua interface, o *path* do código executável da implementação, o modo de ativação, os dados de referência e, se necessário, as variáveis a serem utilizadas para a inicialização do objeto. Esses dados são armazenados num único arquivo, a ser determinado pelo usuário através de uma variável de ambiente. Todas as requisições para acrescentar, alterar ou destruir uma definição de implementação devem ser feitas a esse repositório.

Na ORBeline, a definição da implementação é representada por uma classe (*ImplementationDef*), cujos membros de dados constituem as informações necessárias para o registro.

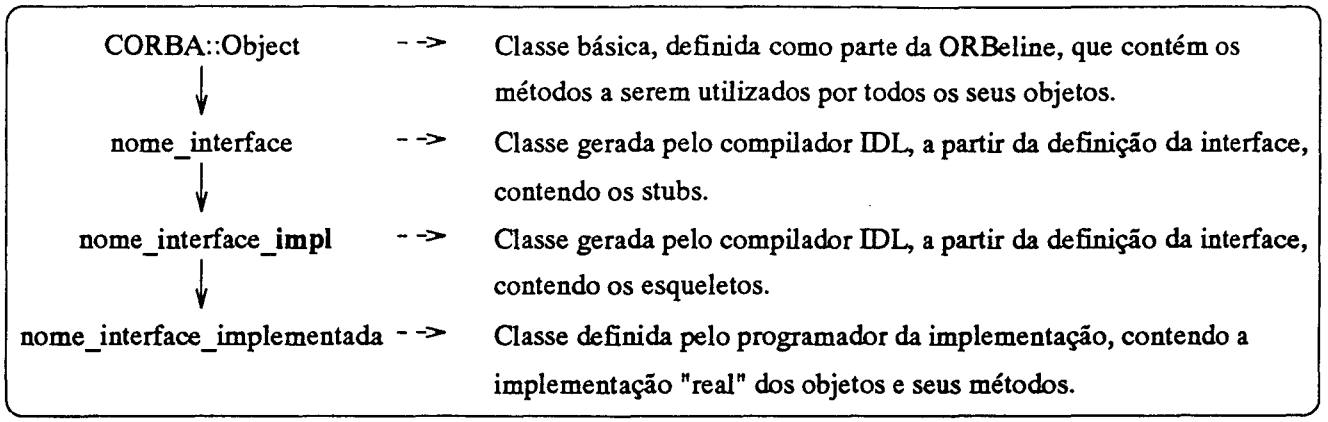

Figura 3.11: Hierarquia das classes na ORBeline

O repositório de implementações é gerenciado pelos *daemons* de ativação de objeto (***oad***[40]).

Daemon de ativação de objeto

Caso um cliente requisite um objeto ainda não ativado, esse *daemon* irá ativar o objeto automaticamente, além de auxiliá-lo na sua inicialização. Deve existir pelo menos um *oad* executando em cada máquina.

Para registrar um objeto com o *oad*, por exemplo, de nome "Soma" com uma interface "Calculo", cujo executável encontra-se no arquivo "/home/msc/user/soma", com uma política de ativação do tipo "compartilhada" e com um dado de referência "exemplo", é utilizado o comando:

```
regobj -o Calculo,Soma -f /home/msc/user/soma -p shared -r exemplo
```

O registro do objeto também pode ser efetuado através da operação ***create***, conforme especificada na CORBA (ver seção 3.3.2).

Essas informações são armazenadas no Repositório de Implementações e o *oad* informa ao ***osagent*** do novo objeto registrado.

ORBeline's *Smart Agent* (*osagent*)

Consiste de um serviço de diretório dinâmico que mantém um registro de todos os objetos na ORBeline. Além disso, faz o mapeamento entre os objetos e os *oads* com o qual foram registrados.

Dessa forma, um cliente, desejando estabelecer um *binding* com um determinado objeto, obedece os seguintes passos (Figura 3.12):

a) contacta o *osagent* do seu domínio (podem existir vários *osagents*, num mesmo ambiente, utilizando portas diferentes) para obter o endereço do servidor da implementação do objeto requisitado;

[40] *Object activation daemon.*

b) recebe, como resposta, o endereço do objeto, caso tenha sido registrado diretamente com o *osagent* (nesta situação, não é possível a ativação automática), ou o endereço do *oad*, caso contrário;

c) caso o objeto já esteja ativo, recebe o endereço do objeto sendo possível, portanto, iniciar a comunicação com ele. Senão, será necessário que o *oad* ative o objeto;

d) o *oad* retorna o endereço do objeto; e

e) pode ser efetuado o *binding*.

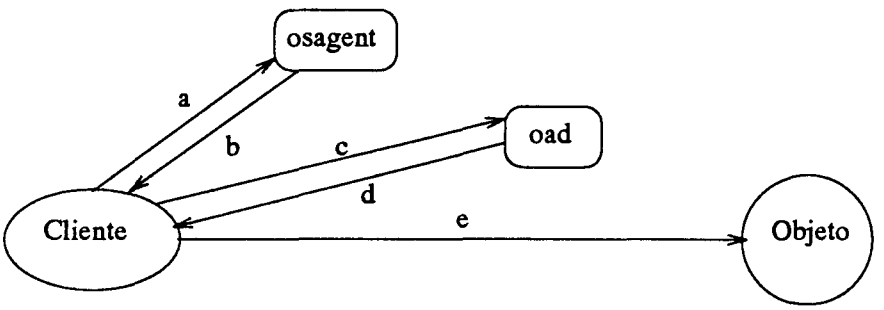

Figura 3.12: Mecanismo de ativação e *binding* usado pela ORBeline.

Repositório de Interfaces

Consiste num banco de dados contendo as definições das interfaces fornecidas pelo ambiente, especificadas em IDL. Podem haver múltiplas instâncias do servidor da interface desse repositório funcionando num mesmo ambiente (inclusive, numa mesma máquina), sendo permitido aos clientes determinarem em qual servidor e arquivo deve ser efetuada uma consulta ou atualização. Como esses servidores também são objetos da ORBeline, podem ser registrados junto ao *oad*.

Para o armazenamento das definições e descrições das interfaces, usa-se o comando **irload**, o qual faz a análise sintática dos arquivos IDL e gera as estruturas, representando as interfaces, que serão armazenadas.

É usado implicitamente pela Interface de Invocação Dinâmica para a verificação de tipos em tempo de execução.

Interface de Invocação Dinâmica

Como já comentado na seção 3.6, essa interface permite às aplicações a invocação de métodos nas implementações de objetos sem que seja necessário o acesso aos *stubs*. Na ORBeline, seus serviços são encapsulados na classe chamada *Request* (derivada da classe *Object*), contendo as operações especificadas pela CORBA para a invocação dinâmica acrescidas de operações para criar uma requisição e efetuar o *binding* com o objeto requisitado.

A operação de criação recebe um dos seguintes conjuntos de parâmetros de entrada:

- nome da interface, nome da operação e nome do repositório de interfaces: é efetuado o acesso ao repositório, onde se obtém a descrição completa da interface procurada. A partir dessa informação, é criada uma Requisição (instância da classe *Request*), inicializada como se fosse um *template* da linguagem C++. A lista de parâmetros a ser utilizada pela operação é criada, com os respectivos modos e os códigos de tipo referentes a cada parâmetro. O programador do cliente deve providenciar apenas o "preenchimento" dos valores correspondentes a cada parâmetro;

- apontador para o objeto contendo a descrição completa da interface e o nome da operação: procedimento idêntico, sendo desnecessário a consulta ao repositório de interfaces;

- apontador para o objeto contendo a definição da interface e o nome da operação: procedimento idêntico, entretanto o repositório é consultado para a obtenção da descrição completa da interface; ou

- apontador para o objeto, o nome da operação e o nome do repositório de interfaces: procedimento idêntico, entretanto o nome da interface precisa ser obtido através da referência de objeto.

Referência de Objeto

É implementada utilizando-se os nomes da interface e do objeto. Por exemplo, um objeto "nome_objeto" com uma interface "nome_interface" gera uma referência de objeto "nome_objeto nome_interface". Objetos transientes[41] terão suas referências acrescidas do endereço de rede do seu servidor.

Código de Tipos

Podem ser obtidos do Repositório de Interfaces ou gerados diretamente pelo compilador de IDL.

Os Códigos de Tipos são implementados como uma classe em C++ (classe TypeCode), possuindo métodos para o acesso à quantidade de argumentos de sua lista, o tipo predefinido que está sendo representado (*tk_null, tk_short*, por exemplo), para a verificação de igualdade (confirma se um código de tipo é igual a outro código de tipo ou não) e para a obtenção de ponteiros para os seus parâmetros.

A ORBeline define classes, derivadas da classe TypeCode, para tipos complexos (referência de objeto, estrutura, união, enumeração, cadeia de caracteres, seqüência e vetor), com os métodos e construtores específicos desses tipos.

Any e *Value*

O tipo any permite a especificação de valores que podem expressar qualquer tipo definido em IDL. É definida pela ORBeline através de um Código de Tipo, indicando o tipo do valor, e um

[41] Objetos criados sem serem registrados pelo *osagent*.

Valor, com o valor propriamente dito. É implementada como uma classe (Any), com métodos para consulta e atribuição dos Códigos de Tipos e dos Valores.

A classe Value possibilita uma forma abstrata e genérica de representação de dados, facilitando o uso da classe Any, a qual será sempre manipulada em termos de instâncias de Value.

Cada tipo de dados é definido por uma classe particular derivada de Value. Para os tipos estrutura, união, enumeração, seqüência e vetor são definidos novos métodos para prover a inicialização dos seus membros de dados.

Possibilidades da ORBeline e seu relacionamento com ODP

Essa plataforma provê algumas funcionalidades importantes:

- permite a **Relocação** (Figura 3.13):

  a) de objetos criados por instanciação - esses objetos são registrados diretamente com o *osagent*. Para efetuar a relocação, a instância do objeto em execução deve ser destruída na sua localização de origem e criada novamente na localização de destino. Essa nova instanciação ocasiona um novo registro do objeto (a). Assim, quando o cliente executar uma nova invocação (b), o servidor não será encontrado na localização esperada. Isso faz com que o núcleo da ORBeline (a parte que faz parte do processo do cliente) contacte o *osagent* (c) para obter o novo endereço e, então, prossiga com a requisição (d); e

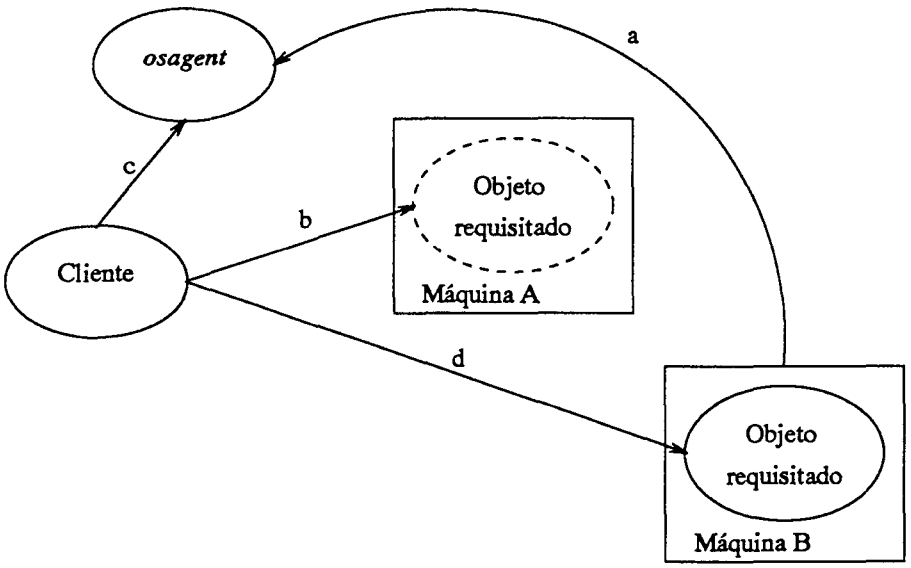

Figura 3.13: Mecanismo de relocação utilizado pela ORBeline

  b) de objeto registrados com o *oad* - nesse caso, inicialmente, deverá ser removido o registro desses objetos com o *oad* no nó de origem e efetuado novo registro no novo nó (com seu respectivo *oad*). O restante do procedimento é semelhante ao anterior, lembrando-se que o *osagent* informa o endereço do novo *oad*.

O *osagent*, nesse mecanismo, tem uma funcionalidade muito semelhante a do Relocador no ODP (ver seção 2.3.4), com uma implementação, de modo geral, mais simples: registra todos os objetos e não apenas aqueles que foram relocados (bastante natural nesse caso, uma vez que também funciona para auxilar na localização dos objetos invocados, conforme já visto). Da mesma forma, das várias possibilidades sugeridas para o escopo do Relocador, a ORBeline utiliza apenas um *osagent* por ambiente (embora possa ser replicado); e só existe um domínio de gerenciamento de referências de objetos.

O procedimento para a relocação, através de destruição e nova criação, é semelhante à função de migração utilizando as funções de desativação e reativação. Bem mais simples, considerando-se que a ORBeline não se preocupa com a manutenção do estado dos objetos, portanto, não é necessário efetuar o seu *checkpoint*.

- **Replicação** - implementada de forma bastante simples, apenas possibilitando a existência de várias cópias de objetos, em diferentes nós, não garantindo consistência. Pode ser:

  a) de objetos criados por instanciação - é efetuada pela instanciação de múltiplas réplicas em diferentes nós. Todas as instâncias serão registradas com o *osagent*, o qual será contactado pelo núcleo da ORBeline, caso a réplica em utilização deixe de atender seus clientes (pela queda do seu nó, por exemplo), para receber o endereço de uma outra réplica; e

  b) de objeto registrados com o *oad* - obedece o mesmo procedimento, sendo que os registros das réplicas são informados pelos *oads* e o endereço fornecido pelo *osagent*, em caso de falha, é o do *oad*.

- **Tolerância a falhas** - Quando um cliente (ou servidor) observa uma perda de comunicação com o servidor (ou cliente), notifica o *osagent*, o qual poderá efetuar um novo *binding*. O *osagent* também observa todos os processos numa comunicação, através do envio de sinais periodicamente. Caso detecte uma perda acidental de conexão, efetua, com o auxílio do núcleo da ORBeline, a restauração da comunicação. Para restabelecer uma conexão pode ser utilizada uma réplica do objeto que falhou.

  O *osagent* pode ser replicado em várias máquinas diferentes, todos com as mesmas informações, não sendo especificado pelo cliente qual atenderá sua invocação. Da mesma forma, os *oads* podem ser replicados em cada máquina, com mecanismo semelhante ao dos *osagents*. Assim, é incrementada, de forma transparente, a tolerância a falhas do sistema.

- **Transparências** - A ORBeline provê as seguintes transparências:

  a) acesso: apresentando a mesma semântica para invocação de objetos locais e remotos. Ambas são efetuadas em referências de objetos;

  b) localização: o cliente não precisa saber a localização do objeto requisitado, pois sua invocação será sempre efetuada sobre a referência de objeto. Permite a seletividade (preconizada no ODP) ao permitir, ao efetuar o *binding*, a determinação da máquina em que se deseja obter o serviço;

c) relocação: o cliente não percebe a movimentação do objeto com o qual está interagindo. É obtida utilizando-se o *osagent*, também sendo permitido a seletividade, através da inibição da sua capacidade de refazer o *binding* com os objetos relocados. Isto é obtido, através da seleção da opção apropriada na operação de *binding* (***_bind***); e

d) a falhas: a falha de um objeto não é percebida pelo cliente. É obtida pela combinação das características do *osagent* acrescidas das funcionalidades de replicação e relocação já descritas. A seletividade é obtida da mesma forma que para a relocação.

3.10 Orbix

A Orbix é uma plataforma desenvolvida pela IONA Technologies e obedece a especificação CORBA. Consiste, basicamente, de [ION95a, ION95b]:

- uma biblioteca a ser ligada às aplicações clientes: provendo a capacidade de invocar requisições;
- uma biblioteca a ser ligada aos servidores: provendo a capacidade de invocar e receber requisições; e
- um *daemon* de ativação, chamado de **orbixd**: tem funcionamento semelhante ao *inetd* do UNIX, embora seja muito mais complexo, obedecendo às políticas de ativação descritas na CORBA. Através do Repositório de Implementações, obtém as informações necessárias para a ativação das implementações de objetos: modo de ativação, o *path* do código executável e os parâmetros iniciais, caso existam. Essas informações são idênticas às utilizadas pela ORBeline;
- um compilador de IDL; e
- um Repositório de Interfaces.

As facilidades de transporte baseiam-se, em princípio, em TCP/IP e XDR[42] e na chamada *select* do sistema UNIX, implementando a passagem de mensagens das requisições entre objetos.

Pela sua característica de implementação, o ORB não é encapsulado em um único componente, através do qual todas as requisições devem passar. De fato, as requisições são passadas diretamente do cliente para a implementação do objeto invocado [ION95a].

Possui duas classes principais:

- ***Object*** - implementa a interface *Object*, definida pela CORBA. É a classe da qual derivam todas as classes dos objetos gerados pelo compilador IDL e que contém as informações necessárias para a comunicação com um objeto remoto; e
- ***Request*** - implementa a interface *Request*, definida pela CORBA. É utilizada tanto na invocação estática quanto na dinâmica. Caso existam argumentos na invocação, verifica

[42] *eXternal Data Representation.*

se são fornecidos pelos *stubs*, os quais sofrem o *marshalling* diretamente, ou pela IID, caso em que os argumentos são inseridos em uma lista específica e o *marshalling* aguarda até a operação *invoke* ser invocada.

Além dessas, existem as classes ***TypeCode*** e ***any***, implementando os conceitos correspondentes da CORBA.

3.10.1 Facilidades da Orbix

A Orbix oferece a possibilidade do programador do servidor alterar o *stub* gerado pelo compilador IDL, através da criação de novas classes derivadas da classe que implementa a interface inicialmente compilada. Dessas classes derivadas é permitido a criação de novos *proxies*, chamados de "espertos[43]". Esse termo advém da flexibilidade permitida por esse mecanismo, sendo uma de suas utilidades possibilitar ao cliente o armazenamento do estado do servidor, informação que pode ser utilizada para aumentar o desempenho e reduzir o número de chamadas remotas.

Outra característica útil é a possibilidade do programador especificar certos procedimentos adicionais a serem executados em vários pontos de uma invocação, como antes ou depois do *marshalling/unmarshalling*.

Esse mecanismo é chamado de **filtro**, permitindo autenticação; coleta de informações para depuração e auditoria; etc.

O Repositório de Interface, além de obedecer a especificação CORBA, também tem sua funcionalidade alterada de forma a permitir a criação de requisições através de um mecanismo usando operadores de um conjunto de cadeias de caracteres[44] semelhantes aos usados em C++.

Apesar de não apresentar um mecanismo próprio para prover persistência, oferece um Tratador de Falta de Objetos[45], ao qual pode ser acoplado um dispositivo de armazenamento persistente do qual podem ser extraídas as informações para a reconstrução do objeto procurado. Este mecanismo é implementado através de **Carregadores**[46], instâncias de uma classe específica, e que são notificados sempre que um novo objeto é registrado na Orbix. Os Carregadores, sendo avisados que foi solicitado que o processo servidor terminasse, providenciam o armazenamento do estado dos objetos pelos quais é responsável. Visando uma certa tolerância a falhas, o Carregador pode, por iniciativa própria, efetuar o armazenamento do estado de um objeto.

A Orbix não possui mecanismos próprios para a transformação do estado volátil de um objeto em dados apropriados para armazenamento persistente nem para a transformação contrária.

A referência para um objeto consiste na concatenação do nome da máquina em que foi criado, com o nome do servidor que o criou e um nome único gerado pelo próprio servidor. Para a localização e *binding* de um cliente com o serviço procurado podem ser fornecidos, pelo cliente, a referência completa; o nome da máquina e o nome do servidor; ou apenas o nome do servidor. Neste último caso, a busca do serviço é executada por um **Localizador**[47], o qual possui

[43] Tradução de *smart*.
[44] Conjunto de cadeias de caracteres - tradução de *stream*.
[45] *Object Fault Handler*.
[46] Tradução de *Loaders*.
[47] Tradução de *Locator*.

a informação de todas as máquinas que fornecem o serviço procurado, selecionando uma delas para executá-lo. Esse mecanismo ainda possui a flexibilidade de possibilitar sua alteração ou substituição por um novo Localizador considerado mais apropriado e criado pelo programador.

Ainda sobre as referências, a Orbix implementa a operação de conversão de referência de objeto para a forma de cadeia de caracteres de maneira que, ao ser convertida novamente para a forma normal possui informações suficientes para efetuar a especialização para a interface a ser utilizada. Assim, permite que a referência obtida seja utilizada diretamente para a invocação das operações implementadas pela interface que representa, sem que seja necessário efetuar explicitamente a especialização, o que obrigaria o conhecimento, em tempo de compilação, da interface a ser utilizada.

Para permitir a utilização dos serviços fornecidos por uma interface, a Orbix pode: a) utilizar a abordagem apresentada pela ORBeline (Figura 3.11, seção 3.9), na qual a classe implementada pelo servidor deve herdar a classe gerada pelo compilador IDL; ou b) o programador da classe que implementa a interface deve associá-la à classe correspondente, gerada pelo compilador IDL. Esta segunda opção[48] é mais flexível, facilitando a reutilização de classes já existentes, pois não há a necessidade de inserir o nome da interface na implementação da classe. Para o mesmo propósito seria necessária a utilização de herança múltipla no caso da primeira abordagem.

3.11 Limitações da CORBA

A CORBA permite que o desenvolvedor de aplicações preocupe-se essencialmente com os aspectos específicos da aplicação, despendendo pouco esforço com os detalhes da comunicação entre os objetos distribuídos e, portanto, eliminando uma fonte potencial de erros. Para tanto, é constituída por componentes reutilizáveis os quais elevam o nível de abstração em que as aplicações são projetadas e implementadas.

Entretanto, apresenta algumas desvantagens no que diz respeito à [SV95]:

- Aprendizado - a introdução de novos conceitos (referência de objeto, *proxy* e adaptadores de objeto, por exemplo), novos componentes (como a IDL e seus compiladores para as várias linguagens) e novas características (tratamento de exceções e herança de interface) tornam necessário um tempo razoável para seu entendimento. Conseqüentemente, é esperado uma demora para sua utilização em todo o seu potencial. Entretanto, o tempo despendido deve ser naturalmente diminuído com a aquisição da experiência ao longo de vários desenvolvimentos;

- Portabilidade - existe dificuldade para a portabilidade de aplicações entre ORBs, que deve ser alcançada de acordo com a aceitação, por parte dos desenvolvedores de aplicações, em se adaptarem aos padrões recentemente aprovados pela OMG e que influenciam a portabilidade: o mapeamento de linguagem de IDL para C++, o Serviço de Nomes e a inicialização de um ORB, por exemplo.

[48] Essa abordagem é chamada de ***tie*** ("amarrar", em Português)

Outro fator fundamental para a portabilidade é a necessidade de uma especificação mais completa e definida de certas áreas críticas abrangidas pelo Adaptador de Objetos. Por exemplo, registro de objetos , ativação de implementação, ativação de objeto e a definição e conteúdo da definição de implementação (***ImplementationDef***) [OMG95g].

- Desempenho - considera-se que algumas aplicações, incrementadas manualmente com código, podem ter melhor desempenho que as desenvolvidas para CORBA, em virtude:
  - da necessidade de invocações remotas adicionais para nomeação;
  - da sobrecarga decorrente do *marshalling/unmarshalling*;
  - da necessidade da cópia dos dados;
  - do uso de técnicas conservativas de gerenciamento de memória; e
  - da necessidade de efetuar uma busca entre as operações de uma interface para selecionar a chamada apropriada.

- Segurança - é uma necessidade importante para qualquer plataforma a especificação, [OMG94i], de seu Serviço de Objeto foi aprovada pela OMG em de novembro de 1996. Entretanto, uma especificação que tratasse desse aspecto levando em consideração a capacidade de interoperabilidade entre ORBs só foi completamente adotada em março de 1997.

- Características - neste item são colocados vários pontos inerentes à arquitetura:
  - não é possível a passagem de objetos por valor. Referências para objetos são passadas por referência. Além disso, todas as invocações remotas são "repassadas" à máquina que efetivamente possui o objeto requisitado. Entretanto, as estruturas que obedecem ao estilo da linguagem C e as uniões discriminadas podem ser passadas por valor;
  - não é definido suporte para operações não-bloqueantes. Considerando-se como um serviço a ser fornecido pela implementação;
  - a IDL não suporta diretamente versões (como tem-se em DCE, por exemplo); e
  - as implementações atuais de CORBA não possuem um suporte eficiente para transferência de dados em grande quantidade.

- Interoperabilidade - a arquitetura descrita na especificação da CORBA 1.2 (e atualmente implementada na maioria dos produtos comercialmente disponíveis), é bastante incompleta sobre esse item, apenas dando algumas sugestões para a sua solução [NWM93]. A nova especificação (CORBA 2.0) define como a interoperabilidade pode ser alcançada, embora ainda deixando alguns aspectos a serem considerados [OMG96a]. Esse item é discutido no decorrer dos próximos capítulos.

Capítulo 4

Interoperabilidade entre ORBs

4.1 Introdução

Como previsto pela OMG, existe atualmente uma variedade grande de produtos que obedecem à especificação CORBA, cada qual procurando atender da sua maneira os problemas que lhe são apresentados, buscando prover as funcionalidades e interfaces preconizadas. As implementações de cada ORB diferem, portanto, refletindo as soluções peculiares de cada fabricante, não só pela utilização de diferentes mecanismos para a obtenção das mesmas funcionalidades preconizadas pela CORBA, como também através do acréscimo de novas funções, consideradas importantes para o usuário final que o fornecedor está pretendendo satisfazer.

Este fato corresponde às decisões técnicas de implementação relativas, por exemplo, ao tempo despendido por uma requisição, ao escopo abrangido por um ORB (uma única aplicação ou todo o ambiente envolvido por uma empresa com conexões internacionais, por exemplo), aos níveis de segurança e ao uso de determinados protocolos.

No entanto, existem outros fatores, embora desejáveis, que também podem motivar a divisão de um ambiente em ORBs diferentes:

- segurança - em muitos casos é conveniente que certos objetos, em um ORB, não possam ser utilizados por outro ORB, mesmo que ambos tenham mesma implementação;
- desenvolvimento - isolando-se o ambiente de desenvolvimento é possível testá-lo restringindo-se os possíveis danos causados por suas falhas, ao mesmo tempo em que permite a utilização segura de seus serviços já comprovados;
- gerenciamento - permitindo um controle mais eficaz sobre os objetos gerenciados; e
- outros - tempo de vida do objeto, sua proximidade ou não da aplicação e o escopo em que pode ser utilizado.

Observa-se então, que tais plataformas não permitem um sistema distribuído realmente aberto, uma vez que não conseguirão interagir de forma a obter um objetivo comum.

Dessa necessidade surge o conceito de **interoperabilidade**, que é definido pelo RFP[1]2.0 [OMG94m], como:

"A habilidade de um cliente num ORB A invocar uma operação, definida em IDL, num objeto num ORB B, onde ORB A e ORB B são desenvolvidos independentemente".

Essa capacidade deve ser atingida sem que as plataformas tenham a necessidade de conhecer os mecanismos específicos utilizados em cada uma, ao mesmo tempo em que são mantidas todas as funções definidas pela especificação do ORB e são preservados o conteúdo e a semântica das informações, inerentes a um ORB qualquer e enviadas para outro.

A interoperabilidade, definida de uma forma mais geral em [Com91], como a habilidade de diversos sistemas de computação cooperarem na resolução de um problema, é uma necessidade fundamental para a obtenção de um sistema distribuído aberto.

Não se deve pensar, entretanto, apenas na situação em que são conhecidos os detalhes de implementação, fato que, embora quase certamente deixe a interação entre os ORBs mais simples e mais eficiente nem sempre é possível. Certamente, essa abordagem apresentará soluções restritas às plataformas consideradas.

Neste capítulo, serão apresentados os conceitos e problemas básicos advindos dos novos requisitos impostos pela interoperabilidade. São propostas algumas operações para suportar essa capacidade, considerando soluções já apresentadas, dando ênfase à resolução do problema originado pelas diferentes formas de implementação das referências de objeto. Por fim, é apresentado um modelo para o Interceptador, com a descrição dos mecanismos e funcionalidades dos diversos objetos que o compõem.

4.2 CORBA2

A OMG reuniu as várias especificações dos componentes que compõem a versão atual da especificação da CORBA, definindo a chamada CORBA2, a qual é composta por:

- **CORBA2/CORE**[2] - composto pelos itens a seguir, considerados essenciais à implementação de qualquer ORB:
  - revisão 1.2 da CORBA sem o mapeamento para a linguagem C;
  - extensões necessárias para suportar os Serviços de Objetos de Concorrência, Externalização, Relacionamento, Temporização e Transação;
  - extensões ao Repositório de Interfaces [OMG94g];
  - extensões para a inicialização do ORB; e
  - extensões para o suporte ao interceptador inter-ORBs.

[1] *Request for Proposal.*
[2] *Common Object Request Environment.*

- **CORBA2/Interoperabilidade** - possui um componente específico para a interoperabilidade, chamado de CORBA2/IIOP[3], entretanto, para garantir que um ORB seja interoperável, faz-se necessário que possa suportar também a CORBA2/CORE. Aqui podem ser acrescentados, opcionalmente, os ESIOPs[4], conforme forem sendo padronizados, bem como outros protocolos proprietários. A obrigatoriedade para a conformidade com o padrão existe apenas para o IIOP. Esse componente será tratado com mais detalhe na seção 4.8;

- **CORBA2/C** - provê o mapeamento para a linguagem C;

- **CORBA2/C++** - provê o mapeamento para a linguagem C++;

- **CORBA2/SmallTalk** - provê o mapeamento para a linguagem Smalltalk;

O objetivo é permitir que novos componentes venham a ser agregados aos já existentes, o que é bastante esperado no que diz respeito às linguagens, por exemplo. Dessa forma, aumenta-se a característica modular da CORBA, permitindo, a partir de um núcleo (CORBA2/CORE), a flexibilidade de cada implementação da plataforma poder selecionar os componentes que melhor se adaptem às necessidades/potencialidades tanto do fornecedor quanto do usuário para o qual foi desenvolvida.

4.3 Domínios

Pode-se dizer que interoperabilidade está basicamente associada com uma mudança transparente de **domínio**. Considera-se domínio como um escopo em que certas características e/ou regras comuns são preservadas [OMG95d, OMG94k].

Há uma tendência de que esses domínios sejam, basicamente, de cunho administrativo e/ou tecnológico, sendo que os mesmos não correspondem, necessariamente, aos limites de um ORB instalado:

- Administrativo - nomes, grupos, gerenciamento de recursos, segurança e outras características dependentes de decisões administrativas; e

- Tecnológico - protocolos, sintaxes, redes e outras características dependentes de decisões tecnológicas.

Domínios possibilitam o particionamento de um ambiente em grupos de objetos que tenham características em comum. Um determinado objeto pode, portanto, fazer parte de mais de um domínio, desde que satisfaça a todos os seus requisitos. Com isso, vemos que os conjuntos de objetos que determinam os vários domínios podem ter várias intersecções, como podemos observar na Figura 4.1. Considera-se o limite de um domínio como o limite de um escopo

[3] *Internet Inter-ORB Protocol.*
[4] *Environment Specific IOP.*

no qual uma determinada característica tem algum significado. O próprio domínio pode ser modelado como um objeto e ser membro de outros domínios.

Segundo [OMG95d], os domínios podem ser relacionados em federações (agrupados de acordo com algum motivo comum) ou em hierarquias[5] (um ou vários domínios são contidos dentro de outro).

Exemplos de domínios:

- **referência** - escopo de uma referência de objeto;
- **representação** - escopo de uma sintaxe de transferência de mensagem;
- **endereçamento de rede** - escopo de um endereço;
- **conectividade de rede** - escopo potencial de uma mensagem de rede;
- **segurança** - escopo de uma política de segurança;
- **tipo** - escopo de um identificador de tipos;
- **transações** - escopo de um serviço de transações qualquer.

Interoperabilidade só será possível através de uma perfeita conversão entre os domínios. É essencial que todos os conceitos empregados num domínio sejam transformados em conceitos compreensíveis pelo outro.

Um mecanismo que realize esse tipo de mapeamento deve ser introduzido, conceitualmente, nos limites dos domínios.

4.3.1 Serviços do ORB

Os domínios podem ser associados aos diferentes aspectos das funcionalidades ORBs, consideradas cada uma independentemente, caracterizando o que se chama de Serviços do ORB. Entretanto, não é prescrita qualquer forma de decomposição das funcionalidades ORB (e a capacidade de interoperabilidade) em Serviços do ORB e os correspondentes tipos de domínios.

Essa nova noção de serviço esta diretamente relacionada com as próprias funcionalidades do ORB, através da decomposição destas em funções bem definidas [OMG94l], podendo tratar de segurança, capacidades transacionais, recuperação, replicação, etc [OMG95d].

Dentro de um mesmo ORB, esses Serviços serão invocados implicitamente, afetando a forma como essa requisição será comunicada ao servidor, pelo núcleo do ORB, sem que o cliente tome conhecimento desse fato. Nesse ponto, observa-se a necessidade de um mecanismo que selecione os serviços a serem utilizados. Esse mecanismo é semelhante ao que deve existir entre ORBs diferentes.

Esses Serviços, quando envolvidos com interoperabilidade, deverão ser unicamente identificados, de forma a permitir a perfeita correspondência entre suas funcionalidades nos diversos ORBs. Deve-se notar que essa correspondência não pode ser considerada, necessariamente,

[5]Tradução livre de *containment*.

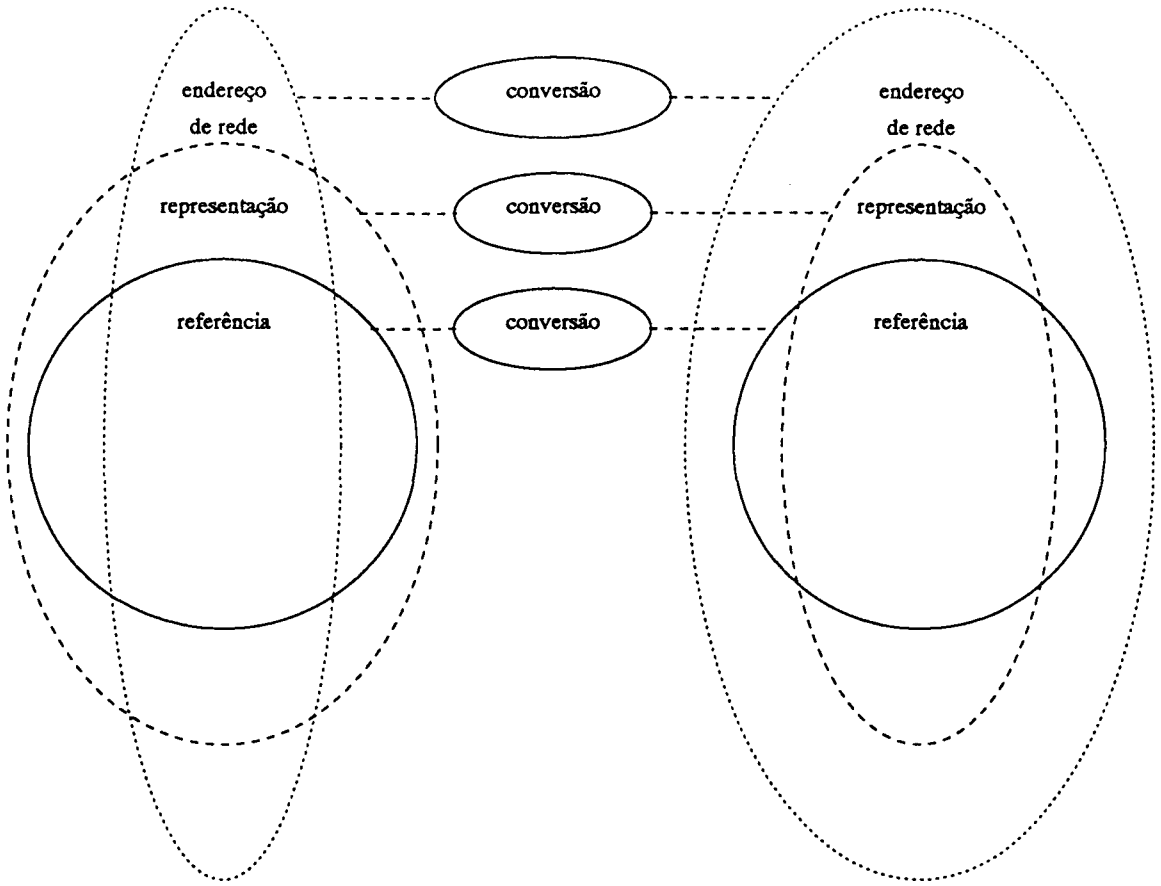

Figura 4.1: Exemplos de domínios com possíveis sobreposições

biunívoca (como apresentado na Figura 4.3), uma vez que um desses Serviços, em um ORB, pode fornecer, sozinho, uma funcionalidade que necessitará mais de um Serviço, em outro ORB.

Pela sua natureza, freqüentemente estão relacionados com uma transparência de distribuição:

- codificação/decodificação de mensagem[6] - provê uma transparência de representação da requisição; e
- resolução de referência[7] - provê uma transparência de localização.

Embora a sua diferença com os Serviços de Objetos seja o fato de estarem abaixo da aplicação (observando-se a arquitetura descrita pela OMA) e a invocação implícita de suas operações, muitos Serviços do ORB são constituídos por componentes invocados explicitamente. Da mesma forma, um serviço pode ser fornecido pela combinação dos Serviços do ORB e os de Objetos.

Na Figura 4.2 pode-se notar que esses serviços podem ser incorporados diretamente no núcleo do ORB ou formando uma camada sobre ele.

[6] O "empacotamento/desempacotamento" de uma requisição para/de uma forma passível de transmissão.
[7] Possibilita a localização do objeto referenciado.

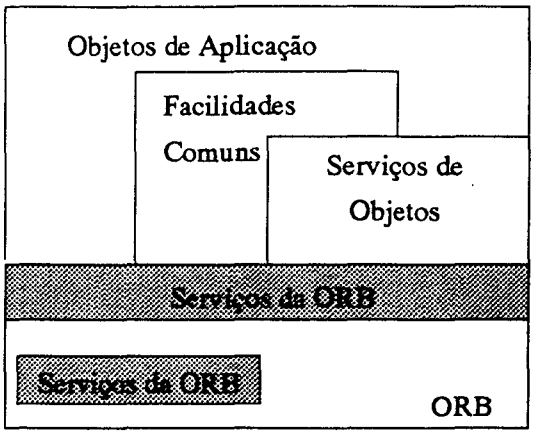

Figura 4.2: As posições possíveis dos Serviços do ORB

Será necessário a existência de mecanismos que permitam que um cliente, em um ORB, faça uma requisição em outro ORB, especificando quais Serviços do ORB serão necessários. Pode ocorrer que o ORB servidor não possua esses serviços, ocasionando um atendimento deficiente ou, até, a impossibilidade da interconexão.

Como a decisão na utilização de cada Serviço é independente da escolha de outros, esses podem ser invocados da melhor forma que se adapte aos requisitos das aplicações, as quais podem selecionar as transparências desejadas. Até o momento, a CORBA não permite o acesso às suas transparências.

Segundo [OMG94l], são obtidas as seguintes vantagens, construindo-se os Serviços do ORB obedecendo a orientação a objetos e apresentando interfaces especificadas em IDL, exatamente como ocorre com os Serviços de Objetos:

- simplificação da conversão entre componentes heterogêneos;
- otimização dos conversores, os quais poderão ser utilizados entre as "camadas" mais apropriadas (Figura 4.3);
- simplificação da sintaxe de transferência da mensagem;
- especialização dos Serviços do ORB, através da herança de interfaces padronizadas; e
- extensão da verificação de tipos para os Serviços do ORB.

4.3.2 Referência de objetos

Uma **referência de objeto** é um valor que identifica um objeto específico, de forma confiável.

Possibilitar aos clientes a utilização dessas referências de objetos de forma a invocar operações em objetos em outros ORBs é fundamental para a interoperabilidade. Deve ser transparente ao cliente o fato de estar utilizando referências para objetos localizados no ORB ao qual pertence ou implementados em outros ORBs.

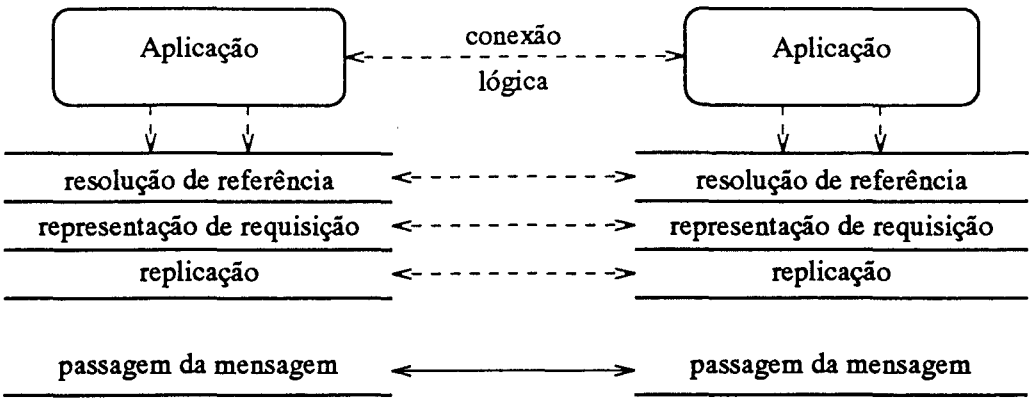

Figura 4.3: Exemplos de Serviços do ORB, que podem ser visualizados como "camadas"

Para atender esses requisitos, existem alguns esquemas básicos para permitir a passagem de referências de objetos entre ORBs [OMG94j]:

- **conversão da referência de objeto** - o interceptador armazena a referência de objeto original, faz o mapeamento para o formato do ORB destino e vice-versa;
- **encapsulamento da referência** - para evitar o armazenamento do caso anterior, pode-se acrescentar um identificador de domínio à cada passagem da referência;
- **conversão das referências dos domínios** - combinação dos dois anteriores. Cada vez que se ultrapassa um novo domínio, substitui-se o identificador de domínio existente por um outro que encapsula todos os domínios já ultrapassados; e
- **referência canônica** - semelhante ao anterior, porém usa um identificador global no lugar dos diversos encapsulamentos.

4.3.3 Referência de Objeto Interoperável (IOR)

Com a finalidade de permitir a compreensão de uma referência de objeto por outros ORBs além daquele que a criou, foi padronizada uma referência específica para utilização em interoperabilidade, chamada de Referência de Objeto Interoperável (IOR[8]), constituída pelas informações consideradas críticas para permitir que a invocação de serviços possa ultrapassar os limites de um ORB:

- **tipo do objeto** - de forma a preservar a integridade do sistema de tipos;
- **protocolos suportados** - no caso das referências de objetos serem válidas em vários domínios de referências, o conhecimento das opções de protocolos permitirá ao cliente optar pelo que oferecer maiores vantagens;

[8] *Interoperable Object Reference.*

- **Serviços do ORB** - uma invocação pode envolver vários desses serviços, cujas informações podem ser analisadas possibilitando a redução ou eliminação das transformações aplicáveis; e
- **referência de objeto nula** - caso a referência seja nula, não deverá receber invocações, só sendo passível de transmissão. É uma opção que pode ser usada no tratamento de exceções e para otimização de recursos.

A estrutura estabelecida possui a seguinte interface:

```
module IOR {
typedef unsigned long PerfilId;
const PerfilId TAG_INTERNET_IOP = 0;
const PerfilId TAG_MULTIPLOS_COMPONENTES = 1;
...                    ...
...                    ...
const PerfilId tag de um protocolo = n; // "0", "1" e "n" são estabelecidos pela OMG.

struct TaggedPerfil {
PerfilId tag; // identificador do perfil do protocolo.
sequence<octet> perfil_data; // descrição do perfil.
}
struct IOR {
string tipo_id; // tipo da interface do objeto fornecedor do serviço.
sequence<TaggedPerfil> perfis; // conjunto de perfis suportados pelo ORB solicitado.
}
```

Cada protocolo suportado torna-se conhecido através do seu **perfil**, o qual encapsula todas as informações necessárias para possibilitar que uma invocação atinja o objeto requisitado. O perfil do IIOP é apresentado na seção 4.7.1 e recebe tag = 0. Além disso, contém a versão do IIOP utilizada, o endereço da máquina, um número de porta nessa máquina e uma chave (um valor opaco usado para a identificação do objeto solicitado).

Os tags têm valores numéricos únicos, especificados pela OMG, para cada protocolo.

A OMG estabeleceu que o tag = 1 indica que a descrição de um determinado perfil é constituída por vários componentes, cada qual representando um Serviço do ORB suportado pelo protocolo especificado. Dessa forma, é possível ao cliente escolher o protocolo mais vantajoso às suas necessidades. Ressalta-se que os Serviços do ORB receberão uma forma de identificação diferente da utilizada para os perfis.

4.4 Interoperabilidade entre ORBs

Segundo a arquitetura especificada pela OMG (chamada de OMA), um ORB provê os mecanismos pelos quais objetos, **transparentemente**, fazem requisições e recebem respostas. Dessa

forma permite a interoperabilidade entre aplicações situadas em máquinas diferentes em ambientes distribuídos heterogêneos.

A interoperabilidade entre ORBs estende essa definição para os casos em que os objetos que fazem as requisições e os que as atendem encontram-se em ORBs diferentes.

O mapeamento das informações utilizadas por um ORB para uma forma compreensível por outra ORB pode ser efetuado, basicamente [OMG94k]:

- de forma **intermediada** - os elementos de interação são convertidos, entre os ORBs, para uma forma intermediária, de conhecimento de ambos. Essa forma pode ser específica (provavelmente otimizada) para dois ORBs ou ser um padrão universal;
- de forma **imediata** - os elementos da interação são convertidos diretamente da forma utilizada pelo ORB cliente para a utilizada pelo ORB servidor. Apesar de potencialmente ser ótima, uma vez que não perde tempo com transformações intermediárias e pode ser desenvolvida de forma a aproveitar as vantagens e semelhanças mais convenientes entre os ORBs em questão, perde tanto a flexibilidade quanto a generalidade.

4.4.1 Interceptador

O problema básico para se conseguir interoperabilidade pode ser traduzido em como fazer para que um objeto Y, no ORB B, apareça como um objeto X, no ORB A, de maneira que este última seja capaz de utilizar X da mesma forma que faria com um objeto qualquer que fosse, de fato, implementado por ele. Além disso, todas as funcionalidades fornecidas pelo ORB B devem ser acessíveis pelo ORB A através de X. Com isso, uma requisição em X deve ser transformada numa requisição em Y e, para tanto, devemos ser capazes de criar X através da passagem de Y para o ORB A. O objeto X será, então, um representante de Y no ORB A, recebendo a denominação de *proxy* de Y [OMG94j]. A Figura 4.4 apresenta esse aspecto.

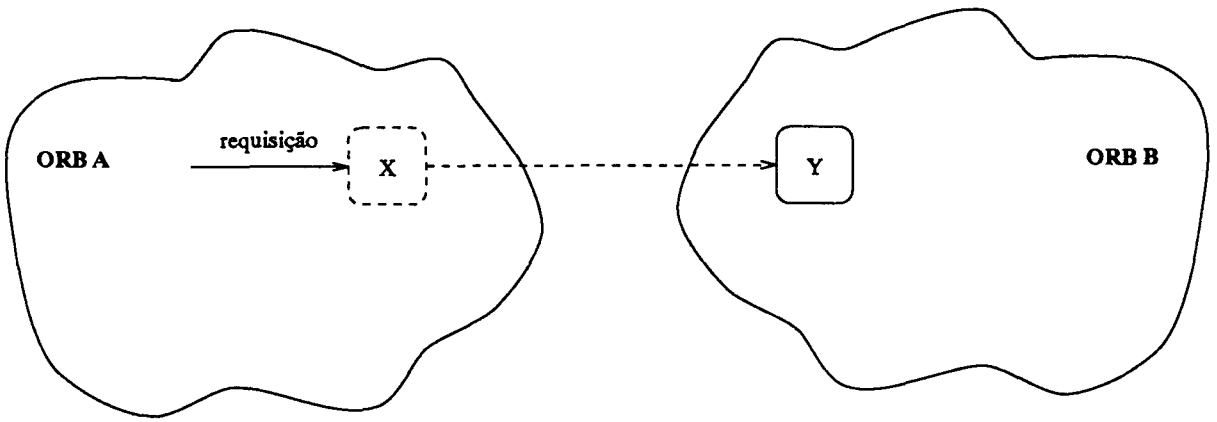

Figura 4.4: **X** é o *proxy* de **Y**.

Como já foi mencionado, durante a conversão da requisição pode ser necessário o mapeamento de outros domínios, além do definido pela referência de objeto. Múltiplos domínios podem

estar sendo ultrapassados simultaneamente e cada conversão será igualmente necessária para o completo entendimento pelo ORB destino.

Uma forma de se conseguir a interação entre dois ORBs é através da introdução de um **Interceptador** entre eles. Esse mecanismo possibilita que uma invocação iniciada em um ORB (requisição em X, na Figura 4.4) seja atendida por um objeto em outro ORB (objeto Y, na Figura 4.4).

Logicamente (computacionalmente) um Interceptador é posicionado entre os dois domínios que tem por objetivo unir. Por exemplo, dois ORBs com implementações diferentes. Entretanto, apresenta uma grande flexibilidade de implementação, podendo ser introduzido em muitos pontos num sistema distribuído (desde o processo ou máquina do cliente até uma máquina específica para exercer suas funções). Esses pontos são escolhidos ponderando-se as diversas relações de compromisso considerando-se aspectos de segurança, desempenho, utilização de recursos, administração e capacidade de utilização, por exemplo.

O interceptador é visto:

- pelo ORB cliente - como sendo a implementação do objeto requisitado: e
- pelo ORB servidor - como sendo o cliente do objeto "real" requisitado.

Um interceptador pode ser implementado de duas maneiras, considerando-se sua localização em relação aos ORBs [OMG95d, OMG94k]:

- em linha[9] - as conversões necessárias são efetuadas internamente ao ORB, sendo fundamental, portanto, o conhecimento sobre sua implementação, a qual sofrerá modificações para suportar os mapeamentos; e
- em nível de requisição - o mapeamento é executado externamente aos ORBs, compondo uma camada sobre o mesmo.

Tendo em vista que o interceptador em linha tem necessidade de um conhecimento disponível, normalmente, apenas caso haja uma predisposição dos fabricantes para que seja possível a interoperabilidade, será feita uma abordagem mais detalhada da segunda maneira, mais genérica.

Interceptador em nível de requisição

Pode ser implementado como:

- meio interceptador[10] - cada ORB possui metade do interceptador. Estes dois meio-interceptadores comunicam-se de alguma forma privada independente das plataformas em que se encontram (usando memória compartilhada ou IPC[11], por exemplo), permitindo a conversação entre ambientes de execução distintos; ou

[9] Tradução de *in-line*.
[10] Tradução de *half bridge*.
[11] *Interprocess Communication*.

- interceptador completo - para o caso de existir apenas um ambiente de execução pode-se usar um interceptador que executa as conversões de uma maneira específica àquele ambiente.

O interceptador em nível de requisição pode, ainda, ser classificado como:

- específico - restringe-se às interfaces predeterminadas, uma vez que o interceptador utilizará os *stubs* e esqueletos gerados pela compilação da definição IDL dessas interfaces (Figura 4.5); e

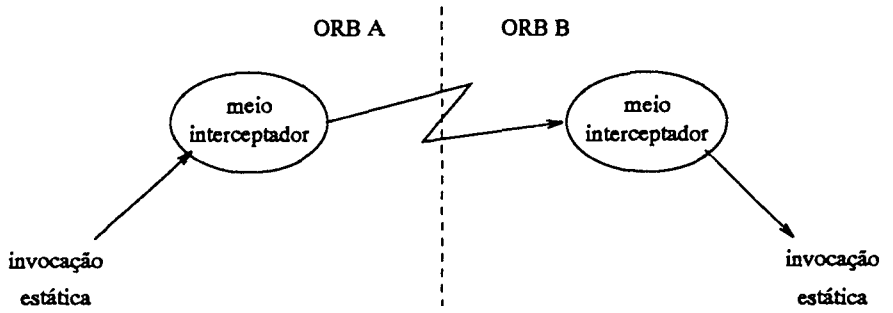

Figura 4.5: Invocação através de um Interceptador em nível de requisição específico.

- genérico - permite a conversão da requisição para qualquer interface definida em IDL, mesmo que não tenha sido conhecida em tempo de compilação (Figura 4.6).

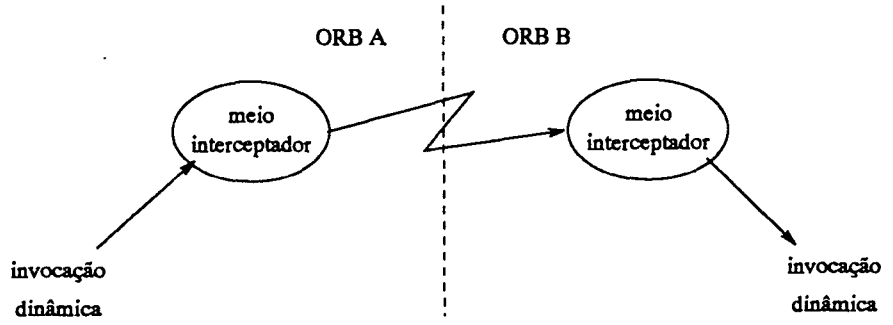

Figura 4.6: Invocação através de um Interceptador em nível de requisição genérico.

O princípio de funcionamento desse tipo de interceptador obedece as seguintes etapas:

1 - A requisição original é passada para um objeto *proxy* no ORB cliente;

2 - O objeto *proxy* converte o conteúdo da requisição para uma forma compreensível pelo ORB servidor;

3 - O *proxy* invoca a operação requisitada no que se lhe apresenta como o objeto servidor;

4 - Caso exista, o resultado da operação é devolvido pelo caminho inverso.

Quando existem dois ambientes de execução, o interceptador fica assim dividido (Figura 4.7):

- A metade localizada no ORB cliente corresponde à implementação do objeto *proxy*; e
- A metade localizada no ORB servidor corresponde ao cliente do objeto "real".

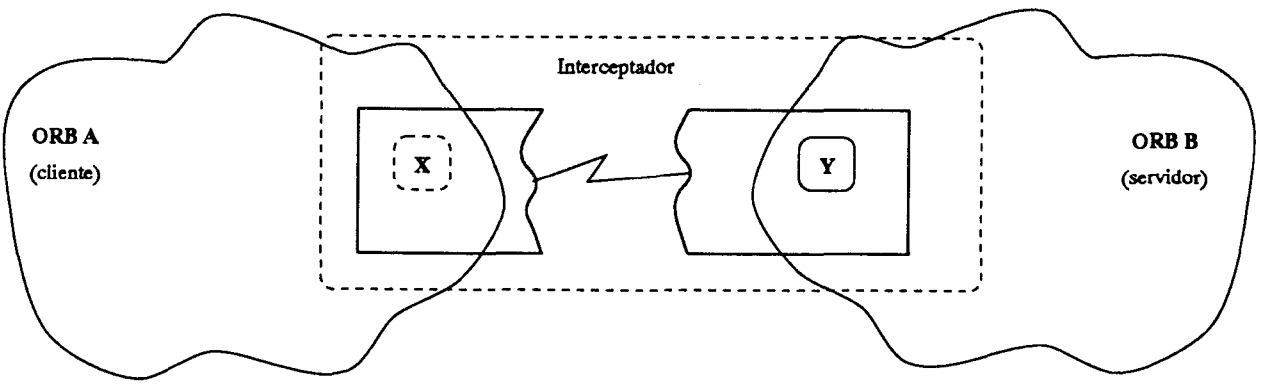

Figura 4.7: **Y** é real, enquanto que **X** é um *proxy*.

Entretanto, para que o interceptador no ORB cliente possa adquirir as informações necessárias para retransmitir a requisição para o objeto no outro ORB é necessário que se possa estabelecer dinamicamente o nome da operação e os tipos dos parâmetros. O mecanismo responsável por isso é a Interface de Esqueleto Dinâmico[12] (IED, detalhada na seção 4.4.2), que possibilita enviar requisições para uma implementação de objeto, mesmo sem o conhecimento, em tempo de compilação, do tipo de objeto que está implementado.

A Figura 4.8 apresenta os componentes dos ORBs envolvidos numa invocação com esse tipo de interceptador.

Para desempenhar as suas funções, um interceptador, em nível de requisição genérico, necessita de várias funcionalidades apresentadas pelas interfaces do ORB:

- Interface de Invocação Dinâmica - pelas características já conhecidas, permite ao interceptador a invocação de objetos cujos tipos não são de seu conhecimento em tempo de compilação;
- Adaptador de Objetos - permite a criação de *proxies* tanto durante a inicialização do interceptador quanto na passagem de uma referência de objeto de um ORB para outro;
- Repositório de Interfaces - fornece, consultado pelo interceptador, as informações necessárias a respeito dos códigos de tipo dos parâmetros das operações, dos valores de retorno e das exceções.

  Para possibilitar a correta compreensão de uma requisição, faz-se necessário que esses repositórios (um ou mais em cada ORB) sejam consistentes em relação aos tipos dos objetos que se decidiu tornar disponíveis para utilização entre ORBs.

[12] Tradução de *Dynamic Skeleton Interface*.

Para conseguir essa consistência, a maneira mais simples é através da definição de um identificador único para cada interface, compartilhado por todos os ORBs que a utilizarão, retirando do interceptador a responsabilidade por mais essa conversão [OMG94k, OMG94g]. Embora manter o mapeamento das interfaces seja uma solução mais genérica, torna-se mais complexa, não só pelas transformações exigidas mas, também, por requerer que o interceptador, ao iniciar sua execução, tenha a necessidade de adquirir os valores específicos de cada ORB a serem mapeados [OMG94j].

Além do tipo da interface, o tratamento dos identificadores de operações e os de exceções também é crítico, sendo resolvido da mesma forma.

- Interface de Esqueleto Dinâmico - permite que o interceptador receba invocações em objetos *proxy* que não são conhecidos durante seu desenvolvimento ou instalação. Esta interface será mais detalhada adiante;

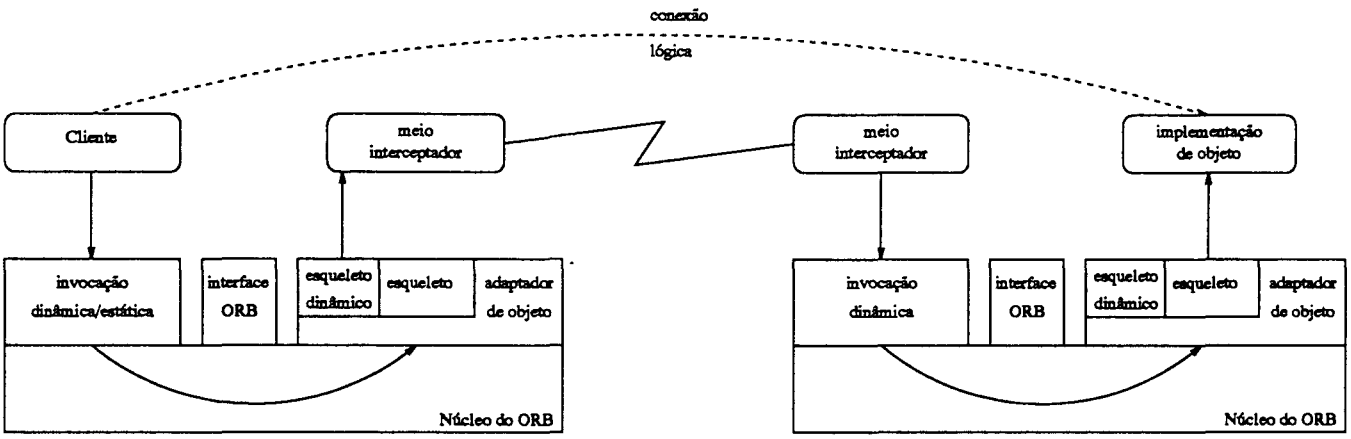

Figura 4.8: Invocação através de um interceptador genérico em nível de requisição.

- Referência de Objeto - permite obter a descrição completa da interface que referencia. Devido à grande quantidade, em potencial, de referências de objetos a serem manipuladas, é necessário um mecanismo que permita o seu gerenciamento. Tal suporte, baseado na determinação de identificadores para as referências de objetos, é utilizado, primordialmente, para permitir a criação de tabelas que efetuam o mapeamento entre referências de objetos e *proxies*. Além disso:

  - para possibilitar a detecção de objetos já destruídos, permitindo que também sejam removidas as suas referências armazenadas. No caso do interceptador, será possível identificar os *proxies* de objetos que não mais existem e eliminá-los (auxiliando no processo de "coleta de lixo[13]"); e
  - para a verificação da existência de registros que referenciam uma determinada referência de objeto. Uma das vantagens é permitir a redução do caminho percorrido

[13]Tradução de *garbage collection*.

por uma requisição a um objeto, cujo *proxy* seja referenciado por outro *proxy*, mas utilizando-se do mesmo interceptador.

- Para o mapeamento de linguagem - todos os tipos de dados devem ser obrigatoriamente suportados pelo tipo **Any**, definido pela IDL; e deve ser possível examinar qualquer tipo de dados retornado por uma operação, incluindo, principalmente, exceções.

Mapeamento de Referências de Objeto

Um problema que deve ser considerado consiste na possibilidade de ser enviada, como um dos argumentos da requisição, uma referência para um objeto que pertence ao ORB cliente (argumento "Z", na Figura 4.9). Por esse motivo, o interceptador deve ser capaz de criar um *proxy* desse objeto no ORB servidor, assim permitindo que o mesma compreenda esse argumento e possa efetuar as requisições necessárias sobre o *proxy*.

Entretanto, para fazer isso, deve haver uma maneira de detectar que um argumento, na realidade, representa um objeto (ou seja, é uma referência de objeto). Efetuando o acesso ao Repositório de Interfaces, é possível descobrir os tipos dos argumentos e, caso se perceba um tipo complexo que possa conter uma referência de objeto, será necessário analisar o argumento a fim de extrair a referência e transformá-la em um *proxy*.

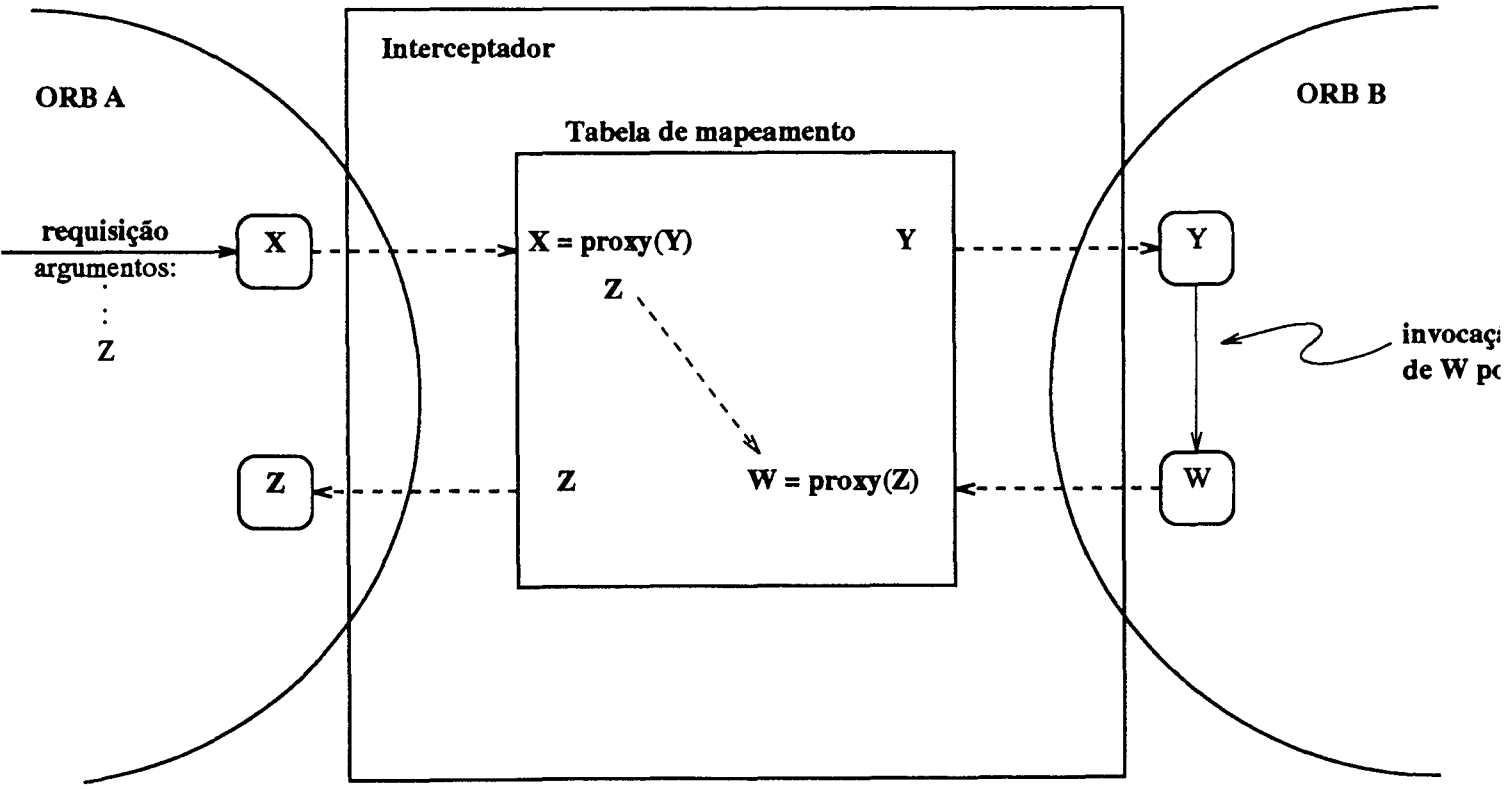

Figura 4.9: Esquema de um interceptador em nível de requisição, passando um argumento (Z) que contém uma referência para um objeto no ORB que originou a requisição inicial

Na Figura 4.9, pode-se notar que Y é um fornecedor de serviços, invocado por um cliente no ORB A através de X (*proxy* de Y). Entretanto, Y também é um cliente de W, que é um *proxy* criado no ORB B para representar Z (que se encontra no ORB A), cuja referência foi passada como argumento na invocação de Y.

Para a criação do *proxy*, através do Adaptador de Objetos, pode-se utilizar o **id** (contém os dados da referência, introduzidos a critério do interceptador, já que este representa, para todos os efeitos, a implementação do objeto a ser criado) como um índice para identificar a referência de objeto numa tabela de mapeamento [OMG94j] (Figura 4.10).

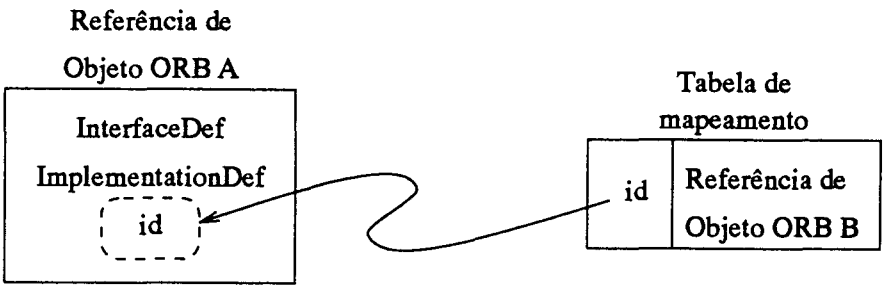

Figura 4.10: Mapeamento de uma referência de objeto utilizando-se o id

4.4.2 Interface de Esqueleto dinâmico (DSI)

Pode-se considerar uma implementação de objeto como um conjunto de métodos, cada qual preparado para ser conectado num esqueleto definido para o tipo específico de objeto que está sendo implementado. Deve haver um método na implementação de objeto para cada operação definida na interface do objeto. Entretanto, necessitamos de um mecanismo que permita receber a invocação a um objeto cujo tipo não seja conhecido em tempo de compilação, caso no qual não existirá um esqueleto preparado para recebê-lo.

Daí surge a idéia de se introduzir uma interface, semelhante a de invocação dinâmica, do lado do servidor. Uma única rotina (chamada de Rotina de Implementação Dinâmica (DIR)[14]), substituindo o conjunto de métodos, é preparada para ser conectada num esqueleto dinâmico e tratar de todas as requisições. Esse esqueleto, mais flexível, será suficiente para permitir requisições em qualquer tipo de objeto, passando para esta rotina uma operação como se fosse um parâmetro (Figura 4.11), além dos parâmetros normalmente passados.

O ORB invoca o DSI[15] quase da mesma forma que um objeto normal, com a diferença de que terá a necessidade de passar uma requisição como parâmetro. O pseudo-objeto Requisição, portanto, deverá ser modificado de forma a permitir o acesso aos seus argumentos.

Semelhante ao pseudo-objeto Requisição, da DII, é criado o pseudo-objeto RequisiçãoDoServidor, contendo todas as informações necessárias à invocação e apresentando a seguinte interface [OMG94k]:

[14] *Dynamic Implementation Routine.*
[15] *Dynamic Skeleton Service.*

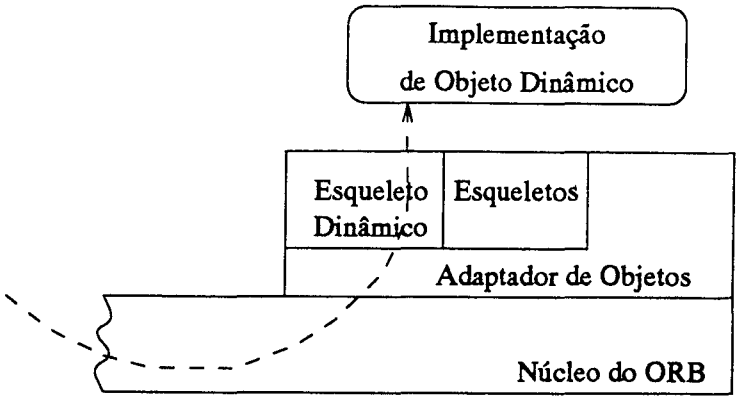

Figura 4.11: A requisição é enviada à Implementação de Objeto através de um esqueleto dinâmico

```
pseudo interface RequisiçãoDoServidor {

  Identifier      obtem_operacao();
  OperationDef    obtem_definicao_operacao();
  Context         obtem_contexto();
  void            obtem_parametros(inout NVList params);
  NamedValue      resultado();
                  }
```

onde,

- **obtem_operacao** - retorna o nome da operação sendo invocada, o qual deve ser único dentro de uma interface. O nome da operação, juntamente com a definição da interface (InterfaceDef), é utilizado para a obtenção da definição da operação, no Repositório de Interfaces;

- **obtem_definicao_operacao** - esta operação retorna diretamente uma entrada para o Repositório de Interfaces correspondente à definição da operação. De fato, esta operação é apresentada para evitar que esse dado seja obtido através do nome da operação (resultante da operação obtem_operacao) e da definição de interface (InterfaceDef);

- **obtem_contexto** - retorna as informações de contexto da operação, definida em IDL, caso existam;

- **obtem_parametros** - recebe os códigos de tipos dos parâmetros da operação e retorna os seus valores; e

- **resultado** - indica o espaço alocado para armazenar o valor de retorno da invocação.

As etapas da invocação são as seguintes:

- A DIR recebe uma chamada, através do BOA, e usando **obtem_operacao** extrai o nome da operação da requisição;
- Com esse nome, faz uma busca no Repositório de Interfaces para obter a definição da operação (OperationDef);
- Através dessa definição, cria uma NVList[16] com os códigos de tipos (internos a cada *any*) de todos os parâmetros da operação, independente do modo, e a passa para o ORB através da chamada a **obtem_parametros (NVList)**;
- Os parâmetros de modo **in** e **inout** são preenchidos, pelo ORB, com seus respectivos valores. Não é necessário a verificação de tipos dispensando, portanto, a consulta ao Repositório de Interfaces, a não ser que o ORB decida por fazê-la;
- Através do *proxy*, o qual (na visão do ORB origem) é o objeto que está sendo invocado, a DIR encontra o objeto real a ser solicitado o serviço;
- O DIR/Interceptador efetua a conversão dos dados obtidos para executar a invocação dinâmica no ORB destino;
- Os dados de retorno (**resultado**, **out**, **inout**) executam caminho inverso.

4.5 Repositório de Interfaces

Um dos problemas que devem ser superados para que seja possível prover interoperabilidade consiste na necessidade de um ORB compreender certos tipos gerados por uma outra, e que são representados por identificadores dependentes do ORB. Esses tipos são: interface, operação e exceção.

Para a criação de um objeto é necessário o conhecimento da sua definição de interface, a qual é utilizada como parâmetro de entrada na operação de criação. Da mesma forma, não será possível criar um *proxy*, sem que a definição da interface do objeto que está sendo "interceptado" seja conhecida. Para tanto, o interceptador, que é o responsável por solicitar ao Adaptador de Objetos a criação do *proxy*, deverá ser capaz de determinar uma interface, no ORB destino, equivalente à do ORB origem. Isto pode ser feito de duas maneiras:

- Através da conversão dos tipos no próprio interceptador: existem várias formas para realizá-la, normalmente utilizando identificadores para contextos de nomes [OMG94k]. Esse esquema possui algumas desvantagens:
  - necessariamente mantém tabelas para o mapeamento dos tipos;
  - torna necessária a introdução de interfaces nos repositórios de todos os ORBs para as quais se pretende disponibilizar seus serviços; e

[16] Pseudo-objeto, definido na CORBA, que representa uma lista de *NamedValues*, os quais são constituídos pelo nome do argumento, pelo seu valor (representado como um *any*),pelo seu tamanho e pelo seu modo (in, out ou inout).

- impossibilita a criação de *proxies* através de *proxies* dos objetos contidos no Repositório de Interfaces de outro ORB. Por exemplo, a própria definição de interface, que é representada como um objeto dentro do repositório (objeto InterfaceDef) pode ser um *proxy* com o qual é possível criar um *proxy* do objeto que ela define.

- Através de identificadores comuns entre ORBs interoperantes: dessa forma, os identificadores podem ser diretamente comparados, já que têm o mesmo significado nos ORBs. É uma solução que envolve possíveis restrições na implementação dos repositórios, uma vez que obriga o atendimento de características específicas, bem como a necessidade de uma padronização.

Está em processo de aprovação uma nova especificação do Repositório de Interfaces a qual, além de manter um identificador único dentro de um repositório (usando escopo de nomes), também define um identificador global, chamado de **identificador de repositório**[17], a ser utilizado entre Repositórios de Interface tanto homogêneos como heterogêneos [OMG94g, OMG94h].

Esses identificadores são independentes de ORB e válidos dentro de uma federação (ou hierarquia). Definidos como uma cadeia de caracteres, podem ser manipulados usando diretamente rotinas da linguagem, independentemente da estrutura do seu valor.

Dessa forma, são usados para uma verificação de equivalência de tipos. Se um objeto de definição de interface (*InterfaceDef*, por exemplo), possui o mesmo identificador em dois ORBs, esses identificadores são considerados idênticos, possuindo, portanto, mesmos atributos, operações e a assinatura dessas operações. O escopo de nomes, portanto, não é restrito, não tendo necessidade de ser o mesmo.

Para atender os requisitos, o identificador de repositório deve ser [OMG94h]:

- padronizado, de forma a suportar sua alocação entre vários fornecedores de interfaces, num ambiente distribuído;
- definido pelo próprio fornecedor de interfaces, gerenciando os identificadores para ele alocados;
- passível de ser federado; e
- de conteúdo compreensível.

Concluindo, num ambiente de identificadores definidos globalmente, apesar dos identificadores dos tipos que trafegam no escopo de cada ORB serem dependentes de suas respectivas plataformas, o interceptador não tem necessidade de conhecê-los, manipulando apenas os identificadores globais.

A existência de múltiplos repositórios num mesmo ambiente ORB pode ser considerada um fato natural, decorrente de: compartilhamento de dados, "escalabilidade", disponibilidade, tolerância a falhas e motivos administrativos. Pode ser usado, por exemplo, como auxílio para desenvolvimento, restringindo um repositório apenas para interfaces em estudo, mas mantendo o acesso aos outros repositórios [OMG94h] que contêm serviços já bem definidos.

[17]Tradução de *RepositoryID*. Note que este mesmo nome é utilizado para identificação, apenas no contexto de um repositório, na especificação CORBA 1.2.

4.5.1 Identificador de repositório

São definidos três possíveis formatos para a estrutura de um identificador de repositório, a fim de possibilitar o seu gerenciamento de forma eficiente:

- formato IDL - constituído por três componentes, na seguinte ordem:

  - os caracteres "I", "D", "L", ":";
  - uma lista de identificadores, separados por uma "/", compondo uma seqüência de caracteres diferentes de "/" e ":"; e
  - o número da versão, composto por dois números, em formato decimal, separados por um ".", representando as alterações mais significativas e as menos significativas.

  Por exemplo, um identificador com o seguinte formato:

  IDL:ORG/TESTE/ESTATISTICA/ACHA_VARIANCIA:1.0

  Significa que uma determinada organização (ORG), define um módulo (TESTE) que contém a primeira versão da interface ESTATISTICA, a qual possui uma operação chamada de ACHA_VARIANCIA.

- formato DCE UUID[18] - composto pelos caracteres "D", "C", "E", ":", o UUID, dois pontos (":") e o número das alterações menos significativas. Por exemplo: **DCE:0050de9e-9a0d-1d77-b049-0000c0c27047:1.**

- formato local - consiste apenas dos caracteres "L", "O", "C", "A", "L", ":" seguidos por uma cadeia de caracteres, a qual não possui uma padronização uma vez que este formato deve ser usado apenas localmente a um repositório. Pode ser usado num ambiente de desenvolvimento, visando evitar possíveis conflitos entre interfaces em avaliação e outras já em utilização. Por exemplo: **LOCAL:nome_interface.**

4.5.2 Operações de acesso a atributos

A CORBA 1.2 não especifica uma operação que possibilite o acesso aos valores dos atributos, impedindo que os mesmos possam ser utilizados pelas IID e DSI. Portanto, acrescentam-se definições de operações (OperationDefs) que descrevem operações de acesso a cada atributo numa interface [OMG94g, OMG94h, OMG94k]. Essas novas operações são, inclusive, tratadas como operações normais numa invocação efetuada através de uma mensagem de requisição do GIOP[19] (seção 4.8).

[18] *Universal Unique Identifier.*
[19] *General IOP.*

4.6 Código de tipos

Para suportar a interoperabilidade, através de interceptadores, é necessária uma API[20] que possibilite a criação do **código de tipo**[21] para descrever um novo tipo de dado desconhecido em tempo de compilação, usando apenas as informações estruturais descritas pelo código de tipo (extraídos através de **kind()** e **parameter()**) [OMG94k]. Na especificação CORBA 1.2, um código de tipo só pode ser obtido através dos códigos de tipos gerados pelo compilador de IDL ou através de consulta ao Repositório de Interfaces [OMG93].

Essa construção é obtida através de novas operações, acrescentadas à interface do pseudo-objeto ORB, similares às utilizadas pelo Repositório de Interfaces [OMG94g]. Por exemplo, para criar um código de tipo que represente uma estrutura, temos:

```
TypeCode create_struct_tc(
  in RepositoryId      id,
  in Identifier        name,
  in StructMemberSeq   member
                       );
```

Essas operações não são necessárias para códigos de tipos primitivos, para os quais já existem esses códigos definidos como constantes.

Tal API pode ser dispensada com a utilização de soluções específicas para um ORB, como é o caso da construção dos códigos de tipos a partir do formato para transferência através do GIOP (usando octetos)[OMG94k].

4.7 Funções do Interceptador

Para que um Interceptador possa suportar as diversas funcionalidades exigidas para permitir interoperabilidade, propõe-se as seguintes operações [ZM96]:

A) Para a criação de um *proxy* - para permitir que um objeto, instalado em um ORB B, seja acessado por um cliente, em um ORB A, deve ser criado um representante desse objeto no mesmo ORB em que se encontra o cliente. Para satisfazer esse requisito, deve existir um certo objeto, em B, responsável por enviar as informações necessárias para a criação do *proxy* (o Ciclo de Vida, por exemplo). O Interceptador, ao receber essa informação, providencia o mapeamento da referência de objeto e aciona o Adaptador de Objetos para registrar o objeto (seu *proxy*) e gerar uma referência de objeto compatível com o ORB em que se encontra no momento. Deve-se notar que, para garantir as transparências de localização e de acesso, é fundamental que o cliente não possa perceber que está invocando, na verdade, um objeto implementado numa plataforma diferente daquela do próprio cliente.

Em [OMG94f], é apresentada uma operação para realizar essas funções, recebendo, como dados de entrada, o ORB_origem, o BOA_destino e o objeto (sua referência) e tendo acesso

[20] *Application Programming Interface.*
[21] Tradução de *TypeCode.*

a múltiplos ORBs. Através da referência, esta operação efetua uma busca no ORB origem a fim de encontrar as respectivas definições de interface (*InterfaceDef*) e de implementação (*ImplementationDef*), a seguir faz o mapeamento dessas definições para o ORB destino (supõe-se que, se o BOA_destino é conhecido, implicitamente se sabe o ORB_destino) e aciona o BOA destino de forma a criar uma referência de objeto através de uma operação chamada de *create_generic*, a qual substitui a operação básica (*create*), especificada na CORBA1.2. Apesar dos dados de entrada serem os mesmos (*InterfaceDef*, *ImplementationDef*, já convertidos para o novo ORB, e os dados da referência, introduzidos localmente pelo próprio Interceptador), a operação *create_generic* cria uma referência de objeto que tem um *binding* para uma função que efetua o tratamento de "métodos genéricos", ou seja, métodos pertencentes a interfaces não definidas em tempo de compilação.

A operação cria_proxy é definida da seguinte maneira:

cria_proxy (

in Identifier	ORB_origem	//identificação do ORB_origem
in Identifier	BOA_destino	//identificação da BOA_destino
inout Object	refobj	//referência de objeto *proxy* e real
		);

Entretanto, a conversão das definições de interface torna-se desnecessária a partir da introdução de um identificador único para as interfaces e a manutenção de Repositórios de Interfaces consistentes (ver seção 4.5), o mesmo ocorrendo com as conversões das implementações, uma vez que é possível a utilização da implementação do próprio Interceptador para substituir a do objeto do qual se está criando um *proxy* (conforme sugerido em [OMG94j]).

Na verdade, o que se cria é uma referência (de objeto) para o próprio Interceptador, contendo a informação necessária que permitirá a identificação do objeto "real", cuja respectiva referência foi mapeada pelo Interceptador. Para efetuar o mapeamento, pode ser utilizado o mecanismo apresentado na seção 4.4.1: o Interceptador, que possui as propriedades de uma Implementação de Objeto, determina o **id** a ser associado à referência de objeto (do *proxy*), criada pelo Adaptador de Objetos do ORB em que está sendo criado o *proxy* (Figura 4.10). Esse **id** corresponde à referência de objeto original numa tabela de mapeamento.

Uma proposta simples, considerando-se um Interceptador entre dois ORBs apenas, exige somente a passagem da referência de objeto. A invocação de meio interceptador, no ORB A, torna claro que existe a intenção de criar um *proxy* no ORB B, dispensando a informação de ORB origem e destino. Da mesma forma, o Adaptador de Objetos não tem a necessidade de ser indicado, podendo ser sempre o mesmo (provavelmente o BOA), já que a Implementação de Objeto será sempre o Interceptador, ou seja, o estilo da implementação não afeta a escolha do Adaptador de Objetos pelo ORB que recebe o *proxy*. As necessidades características de cada implementação serão enviadas e atendidas pelo Adaptador de

Objetos onde foi registrado o objeto "real", tendo sido escolhido justamente por suportar essas exigências.

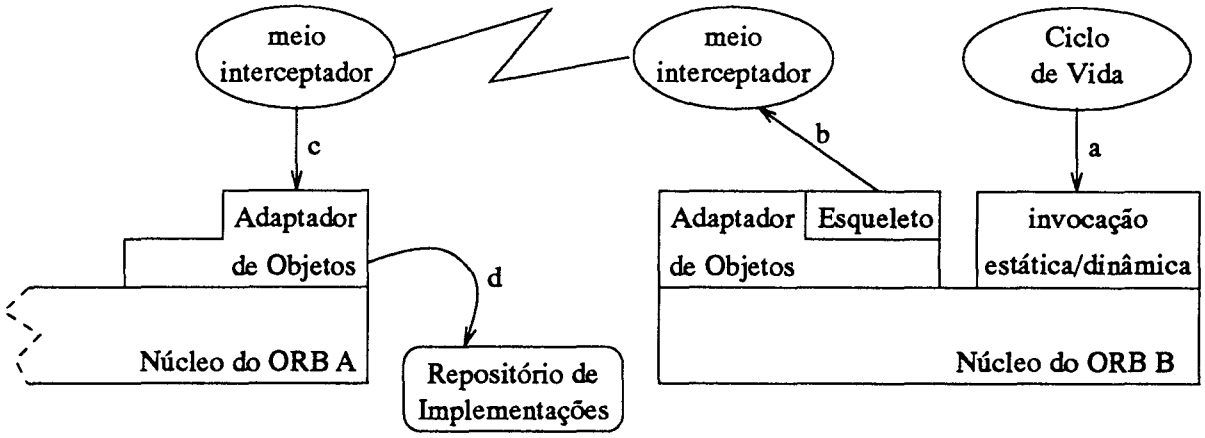

Figura 4.12: Criação de um *proxy*.

Na Figura 4.12, são identificados os seguintes passos:

a) Um objeto qualquer (no exemplo, o Ciclo de Vida) invoca o Interceptador para criar um *proxy* de um determinado objeto no ORB A. Apesar da figura apresentar a possibilidade de invocação estática ou dinâmica, é esperado que esse tipo de invocação seja, na maioria das vezes, estática, considerando-se que a interface do Interceptador é conhecida em tempo de compilação;

b) Um esqueleto aciona o método do Interceptador para a criação de um *proxy*. Da mesma forma que o item anterior, é esperado que a maioria das invocações seja estática;

c) O Interceptador registra o *proxy*, através do Adaptador de Objetos; e

d) O Adaptador de Objetos cria a referência do objeto e introduz no Repositório de Implementações a descrição da implementação (aqui representada pela descrição do Interceptador).

Para o caso do Interceptador estar servindo a mais de dois ORBs, a informação sobre o ORB_destino será necessária para permitir a identificação de qual plataforma está aceitando criar um *proxy*.

Portanto, a função proposta apresenta a seguinte forma:

```
cria_proxy (
  in Identifier   ORB_destino   //identificação do ORB_destino
  inout Object    refobj        //referência de objeto proxy e real
                                );
```

Ao receber uma invocação dirigida para um *proxy*, o Adaptador de Objetos envia ao esqueleto dinâmico (capaz de tratar com qualquer tipo de interface) uma RequisiçãoDoServidor (ver seção 4.4.2).

B) Para satisfazer as exigências da interoperabilidade, o Interceptador utilizará as seguintes operações[22]:

1 - Definidas pela interface *Object*, da qual derivam todos os objetos CORBA, portanto, podem ser invocadas em qualquer dos seus objetos:

- **get_interface** - operação definida na CORBA que retorna um objeto (InterfaceDef) que representa a definição de interface do objeto em que é invocada;
- **get_implementation** - operação definida na CORBA que retorna um objeto (ImplementationDef) que descreve a implementação do objeto em que é invocada;

2 - Definidas pela Interface de Invocação dinâmica:

- **create_operation_list** - esta operação, definida na CORBA pela interface ORB, retorna uma NVList[23], inicializada com as descrições dos argumentos para a operação a ser invocada. Recebe, como parâmetro de entrada, a definição da operação, a qual pode ser retirada do Repositório de Interfaces, sendo conhecidos o nome da operação e a definição da interface que a provê. Por ser uma estrutura parcialmente opaca, a NVList só pode ser alocada através das operações create_operation_list ou create_list.

```
create_operation_list (
        in Operationdef   operacao,   // definição da operação
        out NVList        nova_list   // definição dos argumentos da lista
                                      );
```

- **create_request** - esta operação, definida na CORBA pela interface *Object*, cria um pseudo_objeto (chamado **Requisição**) para o objeto a ser invocado (ver seção 3.3.2). É invocada no ORB destino, tendo como argumentos de entrada os dados convertidos pelo interceptador:

```
create_request (
 in Context          contexto      // informações diversas
 in Identifier       operation_id  // identificação da operação
 in NVList           parâmetros    // lista contendo os argumentos da
                                      operação
 inout NamedValue    resultado     // resultado da operação
 out Request         requisição    // a requisição criada
                                   );
```

- **invoke( )** - operação definida na CORBA, pela interface *Request*, que efetivamente executa a invocação após a requisição ter sido "montada".

3 - Definidas para o próprio interceptador:

- **insere_vetor** (in ReferenceData id, in Object refobj)- operação proposta para prover o mapeamento entre referências de objeto. Insere num vetor com índices iguais aos **ids**, as referências de objetos correspondentes;

[22] Apesar de apresentarmos as novas operações na língua portuguesa, optou-se, por uma questão de coerência e a fim de evitar confusões, manter as operações contidas na especificação CORBA em Inglês.

[23] *NamedValueList.*

- **converte_ref** (in String refobj_str) - operação proposta para converter uma referência de objeto de um ORB para outro. Recebe um objeto *proxy* (ou "real"), obtém seu **id** (utilizado como índice num vetor ou como entrada numa tabela de mapeamento), faz a busca e retorna o objeto "real" (ou *proxy*) correspondente. Para conseguir essas informações, pode utilizar:
  - get_id (Object refobj_proxy) - operação definida pela CORBA pela interface BOA, que retorna **id** de um objeto, tendo, como parâmetro de entrada, a sua referência de objeto; e
  - busca_refobj (ReferenceData id) - operação proposta que retorna a referência para o objeto do qual se possui o **id**, efetuando a busca numa tabela, por exemplo, em que está armazenado o mapeamento entre os **ids** e as suas referências de objetos.
- **converte_parâmetros** - os parâmetros são passados como uma estrutura (*NVList*) constituída por *NamedValues*, os quais deverão ter seus códigos de tipos transformados para uma forma compreensível pelo novo ORB, para que seja possível, a partir daí, a construção de uma nova *NVList*;
  Caso haja parâmetros de retorno (inout,out) também haverá necessidade da sua conversão para o ORB origem, o mesmo ocorrendo caso haja um resultado da operação.
- **converte_contexto** - por convenção [OMG94a], o contexto contém informações sobre o cliente, o ambiente ou sobre a própria requisição (referentes ao Serviço de Objeto ou Serviço do ORB a ser utilizado, por exemplo), as quais são consideradas inconvenientes de serem passadas como parâmetros;
  Essas funções poderão efetuar a conversão dos dados dos parâmetros (e do resultado) da operação e do contexto, para uma forma intermediária padronizada. Para tanto, pode ser utilizado o padrão para Representação Comum de Dados (CDR[24]), detalhado no apêndice A.1. De uma forma mais restrita mas potencialmente mais eficiente, pode ser efetuado o mapeamento específico desses dados entre as representações de dois ORBs particulares que pretendem interagir entre si.

4 - Definidas pelo Adaptador de Objetos:

- **cria_modificada** (ReferenceData id, InterfaceDef intf, ImplementationDef impl) - Cria uma referência de objeto no ORB destino (um *proxy*), registrando-o, entretanto, com um esqueleto dinâmico.

5 - Definidas pelo objeto RequisiçãoDoServidor:

As operações suportadas por esse objeto (obtem_operacao, obtem_definicao_operacao, obtem_contexto, obtem_parametros e resultado) já foram definidas na seção 4.4.2 e permitirão que o Interceptador as utilize para a extração dos dados necessários para a construção de uma requisição no ORB destino.

[24] *Common Data Representation.*

Na Figura 4.13 observa-se que a RequisiçãoDoServidor tem suas informações extraídas e convertidas de forma a serem utilizadas para a construção de uma requisição no ORB de destino, através da IID e, portanto, com o auxílio do Repositório de Interfaces.

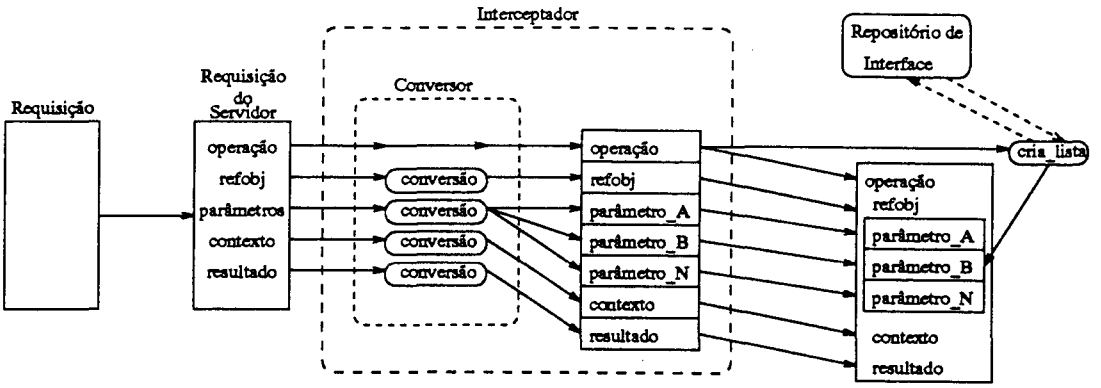

Figura 4.13: Mecanismo para prover interoperabilidade

4.7.1 Alterações no BOA

O BOA, ao receber uma invocação num objeto *proxy*, aciona uma implementação de objeto dinâmico.

```
class Implementação_Dinâmica        // C++
public:
        virtual void invoke (
                                CORBA::RequisiçãoDoServidor        requisição
                                )
```

A operação *invoke* será a responsável por acionar todas as demais operações anteriormente descritas que colaboram com os objetivos de efetuar a transmissão de uma requisição de um ORB para outro e de prover o mapeamento necessário de seus dados (*principal*, códigos de tipos, referências de objetos, etc). Como dado de entrada recebe uma RequisiçãoDoservidor, de onde serão extraídas todas as informações necessárias para compor uma nova requisição (através da DII) no ORB destino.

4.7.2 Mapeamento de objetos entre ORBs

Como mencionado na seção 4.1, a especificação da CORBA, revisão 1.2, não define completamente algumas partes do sistema, permitindo a sua especialização para diferentes aplicações e tecnologias. Dessa forma, para que dois ORBs possam cooperar, é necessária a conversão das referências de objetos, códigos de tipos, *principals*, contextos e contextos de serviços. Os três primeiros são definidos como dados opacos; o contexto não permite o acesso a todas as informações nele contidas; não existe uma interface definida para o *principal*; e o contexto de serviço, também indefinido, passou a existir apenas na revisão 2.0 da arquitetura e carrega informações sobre os Serviços do ORB (seção 4.3.1).

Esse mapeamento pode ser realizado por um módulo específico do Interceptador (ou, mesmo, externo a ele [SUZ96]), ao qual chamaremos de **Conversor**, dividindo-o em submódulos para:

- **referência de objeto** - já discutido na seção 4.4.1;
- **código de tipo** - sua conversão é fundamental para permitir a interoperabilidade, podendo usar o mapeamento através da CDR;
- ***principal*, contexto e contexto de serviço** - contêm informações que necessitam de um acordo em alto nível para manter sua semântica. O *principal* possui informações sobre o Cliente (ou Clientes potenciais) de uma invocação, as quais são analisadas pela Implementação de Objeto, possibilitando o controle de acesso, por exemplo. Algumas informações úteis de serem armazenadas no *principal* podem ser vistas na seção 3.9, sobre a ORBeline.
- **interface/operações** - apesar das interfaces, bem como suas operações, terem um identificador único, mesmo entre ORBs [OMG94g], é necessário que seja efetuada uma consulta ao Repositório de Interfaces para a obtenção desses identificadores, uma vez que, normalmente, a requisição utiliza o nome da operação a ser invocada. O identificador de repositório obtido pode, então, ser enviado para o outro ORB, onde será compreendido;
- **tipos de dados** - em princípio, todos os tipos de dados devem ser convertidos, podendo obedecer ao padrão CDR.

4.7.3 Um modelo de Interceptador

A Figura 4.14 apresenta os módulos que podem existir num Interceptador, mapeando-se conceitos já definidos no RM-ODP (ver seção 2.3.1). A função de acompanhamento de referência de objeto, obedece aos conceitos da função de acompanhamento de referência de interface (seção 2.3.4). Seu domínio, entretanto, passa a ser o de um ORB instalado pois é o que define, em CORBA, a abrangência de uma referência de objeto. Como todas as referências, obrigatoriamente, deverão passar pelo Interceptador, esse é o melhor lugar para efetuar tal tipo de controle.

Antes de efetuar a remoção de um objeto, o Interceptador pode ser consultado para verificar se o objeto foi tornado disponível em outro ORB, caso em que o *proxy* desse objeto deverá ser removido do outro ORB também. Naturalmente, podem ser criadas políticas para recusar ou não a remoção de um objeto, em virtude do seu interesse pelo outro ORB.

Outro caso a ser considerado é quando a referência do objeto a ser removido foi passada como um argumento para outro ORB, gerando automaticamente um *proxy* (Figura 4.9, seção 4.4.1). Essa informação também pode ser armazenada no Interceptador, indicando que existe um Cliente potencial, provavelmente no próprio ORB que pretende remover o objeto.

A Rotina de Implementação Dinâmica corresponde ao método "genérico" que recebe todas as requisições aos *proxies* e, então, envia seus dados ao Conversor.

O Interpretador de código de tipo é sugerido em [OMG94j] e tem como objetivo facilitar a navegação em estruturas de dados arbitrárias em busca de referências de objetos, as quais devem

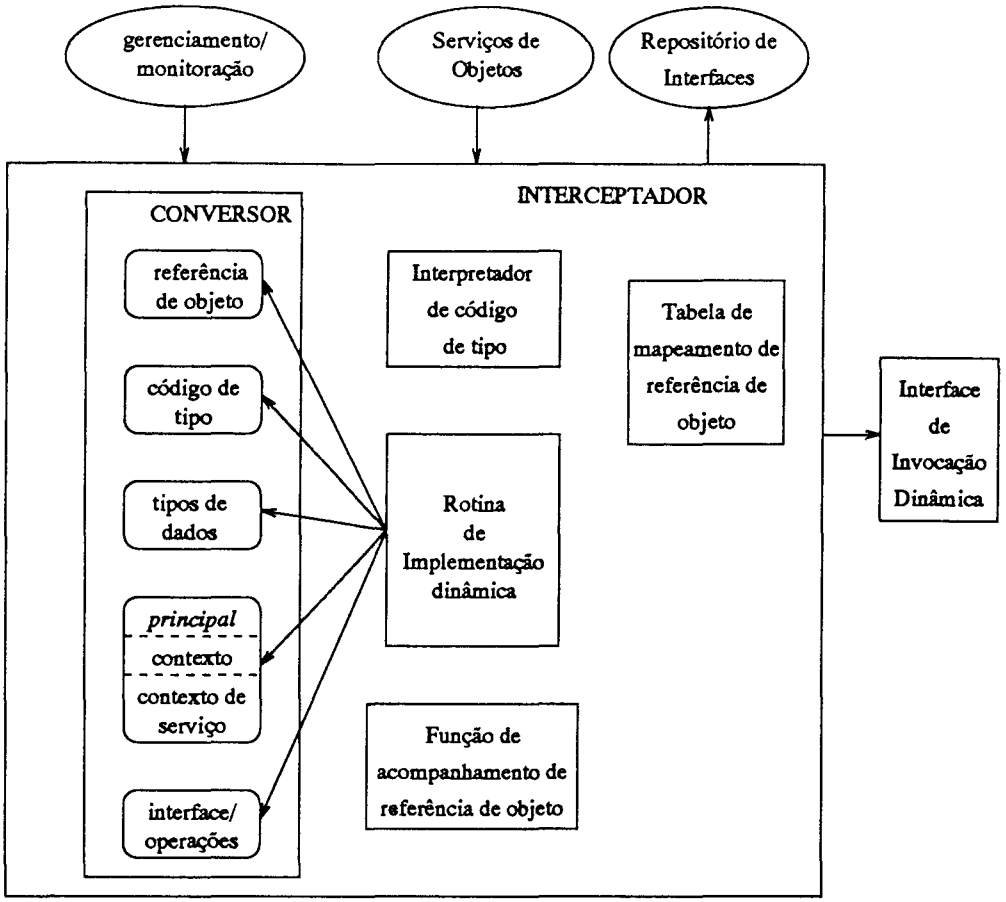

Figura 4.14: O Interceptador e seus "componentes"

ser substituídas por *proxies* antes de serem enviadas à IID no outro ORB. É necessário quando ocorre a passagem de uma referência de objeto como argumento numa invocação.

A Tabela de mapeamento de referências de objetos armazena os *proxies* e seus identificadores no escopo do Interceptador.

Pode ser observado, também, os Serviços de Objetos, que podem utilizar as informações do Interceptador e/ou que o próprio Interceptador pode utilizar: Ciclo de Vida, Persistência, Segurança, etc.

4.8 Protocolo geral entre ORBs

O **GIOP**[25] [OMG95d] tem como objetivo prover o suporte à interoperabilidade entre ORBs, em nível de protocolo, de forma genérica e a baixo custo. Para tanto, buscou-se atender as seguintes propriedades:

[25] *General Inter-ORB Protocol.*

- **disponibilidade** - baseia-se em TCP/IP[26], um dos mais utilizados mecanismos de transporte atualmente;
- **simplicidade** - acrescenta o mínimo necessário para atingir os objetivos, permitindo uma implementação variada e, ainda assim, compatível;
- **escalabilidade** - atende às demandas atuais e futuras;
- **baixo custo** - despende de pouco investimento para sua implantação ou desenvolvimento;
- **genérico** - pode ser usado com qualquer camada de transporte que obedeça a certos requisitos mínimos; e
- **arquitetura neutra** - independe totalmente da forma como o ORB irá acessar esse mecanismo.

A especificação desse protocolo baseia-se na padronização da sintaxe de transferência e do formato das mensagens trocadas entre ORBs, além da definição de algumas características mínimas necessárias à camada de transporte.

A CDR é uma sintaxe de transferência que efetua o mapeamento dos tipos de dados, representados em IDL (OMG), para uma forma bicanônica, a qual será utilizada para a comunicação entre ORBs. Abrange a forma de ordenação dos *bytes*, o alinhamento de tipos, o mapeamento de pseudo-objetos, além de permitir a definição de novas representações, caso surjam novos tipos (ver apêndice A.1).

O GIOP padroniza também um pequeno conjunto de mensagens, que possibilitam a comunicação entre ORBs interoperantes sem que, com isso, se perca qualquer das funcionalidades da CORBA: requisição, resposta, cancelamento de requisição, requisição de localização de objeto, resposta de requisiçao de localização de objeto, término de conexão e mensagem de erro (ver apêndice A.2).

4.8.1 Camada de Transporte

O GIOP considera que o protocolo de transporte utilizado possui um conjunto mínimo de características:

- Orientado à conexão;
- Confiável;
- Provê informações sobre perda de conexão; e
- Permite o mapeamento do seu modelo de conexão para o do TCP/IP.

Além disso, devem ser obedecidas as seguintes regras:

[26] *Transmission Control Protocol/Internet Protocol.*

- **Conexões não simétricas** - como só clientes podem originar conexões, o protocolo torna-se mais simples;

- **Multiplexação de requisições** - como o objeto fornecedor do serviço é inequivocamente identificado, várias requisições e respectivas respostas, podem trafegar na mesma conexão;

- **Requisições não ordenadas** - devido a um identificar não-ambíguo para as requisições enviadas (o **request_id**), não há necessidade da preservação da ordem em que estas são enviadas ou recebidas, o mesmo acontecendo com suas respostas. Da mesma forma, um cliente não precisa esperar uma resposta para enviar uma nova requisição; e

- **Término da conexão** - pode ser iniciado: pelo servidor, através de uma mensagem confiável, *CloseConnection*, desde que tenha respondido a todas as requisições naquela conexão (ou tenha recebido as respectivas mensagens de *CancelRequest*); ou pelo cliente, encerrando a conexão.

  Se o cliente recebe um pedido de término de conexão, deve considerar que as suas requisições ainda não respondidas foram recebidas pelo servidor após este já ter enviado o pedido. Significa que essas invocações não serão processadas, devendo serem novamente enviadas numa próxima conexão.

4.8.2 Contexto de Serviços ORB

Algumas especificações de serviços (Serviços do ORB e de Objetos) exigem que informações do seu contexto sejam fornecidas quando de sua requisição ou resposta.

Para atender essa necessidade, é definido um mecanismo que possibilita a transmissão desses dados como parâmetros opacos, encapsulados em octetos, de forma a serem tratados pelos ORBs sem a necessidade de serem "desempacotados", assim nem os tipos de dados terão que ser conhecidos.

O conjunto de serviços a ser utilizado numa invocação (ou na sua resposta) será enviado no cabeçalho das mensagens de requisição (*Request*) e de resposta (*Reply*), na forma de uma seqüência de octetos. Caso existam vários serviços ORB, será enviada uma lista contendo cada um deles.

```
module GIOP {
  typedef unsigned long          ServicoId;
  struct                         CtxServiço {
                                  ServiçoId          contexto_id;
                                  sequence<octet> dados_contexto;
                                 };
  typedef sequence<CtxServico>   ListaContextoServiço;
  const ServiçoID                ServicoTransacoes = 0; // exemplo
                                 }
```

Na estrutura apresentada observa-se que o Serviço de Transações recebeu um identificador igual a zero, estabelecido unicamente pela OMG. Atualmente, só está definido o identificador para este serviço.

4.8.3 Protocolo entre ORBs para a Internet (IIOP)

O mapeamento de GIOP para TCP/IP é chamado de IIOP (*Internet* IOP). O **perfil** desse protocolo, descrito numa IOR, apresenta a seguinte forma:

```
struct ProfileBody {
  Version            iiop_version;
  string             host;
  unsigned short     port;
  sequence<octet>    object_key;
                     }
```

onde,

- **iiop_version** - indica a versão do IIOP que o cliente suporta;
- **host** - identifica a máquina que está preparada para receber as requisições do cliente, através do seu endereço completo na Internet (por exemplo, "marumbi.dcc.unicamp.br" ou "143.106.23.10"). Atualmente não há suporte para endereços classe D (*multicast*);
- **port** - identifica o número da porta na qual o servidor está preparado para aceitar requisições para conexão (está "escutando"); e
- **object_key** - gerado pelo servidor, identifica o fornecedor do serviço que atenderá a requisição do cliente.

4.9 Localização do objeto

O GIOP considera que o endereço utilizado (endereço IP + número da porta TCP, se for considerado o TCP/IP, por exemplo) corresponde à existência de um "agente", o qual possibilitará a abertura de uma conexão e receberá requisições, abstraindo-se completamente do fato desse agente fazer parte ou não da CORBA.

Tal entidade deve reagir a uma requisição numa das formas a seguir:

a - Aceita a invocação e retorna uma resposta. É interessante observar que, para fazer isso, o agente pode ter que repassar a requisição para outro ORB e depois reencaminhar a resposta. Tal fato não afeta o GIOP;

b - Não aceita a requisição de nenhum objeto e retorna uma exceção (causada pela operação solicitada) ou o endereço novo ao qual a requisição deve ser reenviada;

c - Aceita a invocação de alguns objetos (e fornece as devidas respostas) e recusa a de outros, nesse último caso, retornando o respectivo endereço novo; e

d - Aceita a invocação de alguns objetos (e fornece as devidas respostas) num determinado instante e, mais tarde, para os mesmos objetos, retorna um endereço novo a ser chamado.

Os agentes (no lado do servidor) não têm obrigatoriedade em implementar um mecanismo para informar o novo endereço do objeto requisitado. Nesse caso, aceitam a requisição ou retornam uma exceção.

Já os clientes, devem sempre ser capazes de interpretar uma resposta que envolva um redirecionamento, uma vez que qualquer ORB pode decidir implementar um serviço de localização (caracterizado em (b)). Os clientes, percebendo esse fato, passam a enviar apenas mensagens de *LocateRequest*, com menor custo.

4.10 Trabalhos relacionados

Atualmente, pode-se notar um esforço muito grande por parte da OMG para definir completamente as soluções encontradas para permitir a interoperabilidade entre ORBs. Esse esforço resultou na especificação UNO[27] [OMG95d], a qual preconiza a necessidade do suporte ao mecanismo de interceptação inter-ORBs bem como apresenta o GIOP (seção 4.8), que suporta interoperabilidade em nível de protocolo através da especificação de uma sintaxe de transferência comum (CDR) e da padronização do formato das mensagens trocadas entre ORBs. Como uma especialização dessa especificação surgiu o IIOP, que consiste num GIOP sobre TCP/IP. Também foram criados os ESIOPs, que são protòcolos específicos para um determinado ambiente. Até o momento, só está especificado o ESIOP da DCE.

O GIOP permite a interoperabilidade entre ORBs desde que esses sejam compatíveis com o protocolo. A idéia é de que qualquer ORB possa comunicar-se com outro desde que ambos suportem o GIOP. Os mecanismos que serão implementados por cada ORB para tratar esse protocolo, entretanto, é deixado em aberto, ficando sob a decisão de implementação dos fabricantes. Além disso, apesar de considerar, nas mensagens padronizadas, a possibilidade da relocação de um serviço, apenas informa seu novo endereço, não definindo como essa informação será encontrada ou tranferida para o GIOP, ou tratada pelo ORB receptor da informação.

A especificação CORBA 2.0 aborda essas especificações e foi parcialmente detalhada neste trabalho.

Além dos esforços despendidos pela OMG só foi possível encontrar um trabalho relacionado, apresentado em [SUZ96], o qual descreve uma estrutura para a construção de um mecanismo de interceptação, obedecendo aos conceitos definidos pela OMG e provendo interoperabilidade entre duas implementações de ORB: a Orbix, da IONA Technologies, e a DOME, da Object-Oriented Technologies.O núcleo do mecanismo consiste em um objeto (*InterORB_Proxy*) o qual é responsável pela ligação entre cada um dos ORBs interoperantes e um terceiro que os interliga (servindo como "espinha dorsal" [OMG95c]). Utiliza as interfaces padronizadas da CORBA para efetuar a conversão de uma requisição de uma ORB cliente pra um ORB servidor. Outro objeto básico (*Half-Bridge Object Adapter*) tem como função efetuar o encapsulamento das referências de objetos de outros ORBs no *InterORB_Proxy*, sendo construído sobre o BOA. O mecanismo

[27] *Universal Networked Objects.*

descrito é semelhante, em linhas gerais, ao apresentado nesta dissertação, uma vez que ambos são baseados nos conceitos da OMG, entretanto não apresenta as operações utilizadas nem contempla a transparência de relocação.

Capítulo 5

Transparência de Relocação

Através da transparência de localização, objetos têm seus serviços solicitados sem que o cliente necessite possuir qualquer indicação de sua posição. A transparência de relocação estende esse conceito para objetos que se movimentaram entre invocações[1].

Como já mencionamos, o ORB garante transparência de localização. Será apresentado, então, um mecanismo que permita transparência de relocação, a qual não é, atualmente, totalmente contemplada pela especificação CORBA [Mad95].

Ocorrendo a situação em que um objeto, fornecedor de um determinado serviço, altera sua localização num ambiente ORB, torna-se necessário um mecanismo que possibilite que as novas requisições sejam enviadas ao objeto na sua nova localização. Para tanto, esse mecanismo deve prover meios para que o ORB detecte a alteração e mantenha o correto direcionamento das invocações ao objeto movimentado, onde quer que este se tenha tornado disponível naquele momento.

Para suportar essa transparência, apresentamos um esquema considerando as duas situações possíveis [ZM96, ZM97]:

- no escopo de um mesmo ORB - a própria funcionalidade de relocação ainda não foi completamente tratada pela OMA, portanto, ainda não existe uma interface específica definida; ou
- no escopo de ORBs diferentes - aos problemas inerentes para prover transparência de relocação são acrescidos as novas necessidades impostas pela interoperabilidade.

5.1 Relocação no escopo de um mesmo ORB

Nesse escopo, introduzimos um objeto, ao qual chamamos de **Relocador**, baseado nos conceitos do RM-ODP (seção 2.3.4), responsável por estabelecer o mapeamento entre os identificadores dos objetos provedores de serviços e a localização para onde foram movimentados.

[1] O caso da movimentação de objetos durante sua invocação é tratada pela transparência de **migração** [DBM92], a qual se torna muito mais complexa por envolver a necessidade do armazenamento do estado dos objetos.

O objeto responsável pela relocação, que pode ser: Ciclo de Vida, Externalização, Instalação e Ativação ou qualquer outro objeto que venha a ser desenvolvido com essa funcionalidade[2], também será o responsável pela atualização do Relocador. Esses objetos, portanto, devem ser estendidos de maneira a fornecer as informações necessárias que permitam o correto estabelecimento e manutenção do mapeamento desejado, tendo a capacidade de acionar o Relocador sempre que ocorrer uma movimentação.

O Relocador é constituído por uma tabela na qual são armazenados os identificadores dos objetos movimentados (sua referência de objeto ou nome, por exemplo) e suas correspondentes novas localizações (Figura 5.1).

REFOBJ A	LOCALIZAÇÃO: endereço IP (ou chave) *path* do arquivo

Figura 5.1: Exemplo da tabela mantida pelo Relocador, associando uma referência de objeto a cada localização

O Relocador proposto possui a seguinte interface:

```
interface relocador {                                    // PIDL
        string localiza (in Object refobj);
        void insere_inicio_movimentacao (in Object refobj);
        void insere_nova_localizacao (in Object refobj, in string NovaLocalizacao);
        void remove (in Object refobj);
      };
```

As operações prescritas nesta interface têm a seguinte funcionalidade:

- **localiza** - recebe como parâmetro de entrada um identificador correspondente ao objeto que efetuou, ou está efetuando, uma movimentação e retorna um dos seguintes parâmetros:
  - a nova localização do objeto movimentado, no formato utilizado pela plataforma para localizar um objeto: seu novo endereço IP e o *path* do arquivo (executável ou de dados), por exemplo;
  - uma mensagem informando que o referido objeto ainda se encontra em movimentação, caso o Relocador, tendo sido comunicado que o objeto estava iniciando uma relocação, ainda não recebeu a atualização de sua localização; ou
  - um valor nulo, caso o objeto não esteja armazenado no Relocador, caso em que ocorreu uma falha.

[2] A OSA 8.1 [OMG95e] sugere a criação de um Serviço de Objeto específico para relocação.

- **insere_inicio_movimentacao** - recebe como parâmetro de entrada um identificador do objeto que iniciou a movimentação.

  Inicialmente, chama a operação **localiza** para verificar se já não existe uma nova localização correspondente a esse objeto. Essa situação pode ocorrer devido a sucessivas movimentações no escopo de um mesmo ORB, portanto, apenas a localização existente deve ser substituída por um sinal de início de movimentação. Assim, impede-se a duplicação de objetos no armazenamento, um dos quais estará incorreto. Tendo a operação **localiza** retornado um valor nulo, significa que pode ser inserido o objeto, juntamente com o sinal de início de movimentação;

- **insere_nova_localizacao** - recebe como parâmetros de entrada um identificador do objeto movimentado e a sua localização atual e, daí, executa uma busca usando esse identificador para introduzir a nova localização do referido objeto.

- **remove** - recebe como parâmetro de entrada um identificador do objeto movimentado, retirando-o do Relocador juntamente com suas informações correspondentes. Esta função pode ser utilizada, por exemplo, em decorrência da atualização do Repositório de Implementações, ou das informações contidas em qualquer que seja o dispositivo utilizado pelo ORB para efetuar a localização dos objetos (Serviço de Nomes, *Trading*, etc). Deve ser efetuada obedecendo uma política coerente com as necessidades que originaram, inicialmente, a relocação dos objetos.

O Relocador pode ser considerado como um Serviço de Objeto, uma vez que pode ser implementado como um objeto que fornece os serviços descritos na sua interface e que pode ter seu acesso efetuado por qualquer outro objeto que possua uma referência de objeto para ele. Inclusive, existe a possibilidade de ser agregado a sua interface uma operação que permita ao cliente decidir por ter ou não a transparência. Com essa operação, portanto, é acrescentada a seletividade da transparência, que é uma característica preconizada pelo RM-ODP. A invocação a um determinado objeto pode, por exemplo, enviar, como parâmetro, a referência de objeto ao Relocador associado aquele objeto, bem como uma lista de máquinas para as quais o cliente deseja a transparência (por uma questão de custo, talvez). Caso ocorra relocação para uma máquina que não conste da lista a operação retornará um erro.

Por outro lado, o Relocador também pode ser implementado como um Serviço do ORB, tendo seus parâmetros de chamada implícitos na invocação e, a princípio, totalmente transparentes ao cliente. Podendo ser acionado pelo Núcleo do ORB, que é o responsável por garantir a comunicação das requisições.

Deve ser lembrado, entretanto, que, em qualquer dos casos, antes de qualquer iniciativa é necessário que o objeto do qual é desejado a transparência de relocação herde o objeto Ciclo de Vida, alterado de acordo com a proposta aqui descrita. Apenas desta forma, será possível ao Relocador ter condições de prover essa capacidade ao cliente.

Além desse novo objeto, chamado de Relocador nesta proposta, serão necessárias algumas alterações em objetos já definidos pela CORBA e que serão utilizados para possibilitar a transparência de Relocação:

- Ciclo de Vida - consiste num Serviço de Objeto que define serviços e convenções para criar, destruir, copiar e mover objetos [OMG94b]. Esse objeto será o responsável por disparar o mecanismo de transparência de relocação, uma vez que tem como uma de suas funções possibilitar a movimentação de objetos. Para que isso se torne possível, sua funcionalidade deve ser estendida de forma a informar ao Relocador sobre a localização para a qual está sendo movimentado o objeto servidor.

  Para mover um objeto é necessário que o mesmo possua uma interface que suporte essa capacidade, ou seja, é necessário a "colaboração" do objeto "alvo" (objeto que será movimentado).

  Esse Serviço de Objeto é composto pelas seguintes interfaces:

  a) **ObjetoCicloDeVida**: define as operações de cópia, movimentação e remoção. Vamos nos preocupar com a de movimentação:

  ```
  void move (in LocalizadorDeFabrica local_escolhido,
             in Criterio criterio_escolhido)
         raises (FabricaInexistente, NaoMovivel,
             CriterioInvalido, CriterioNaoEncontrado);
  ```

  Essa operação move o objeto para o escopo de um **localizador de fábricas**[3]. Fábrica é definida como um objeto com a capacidade de criar outro objeto, fornecendo ao cliente as operações necessárias para criar e inicializar novas instâncias. Apesar de não ser estabelecido um padrão para sua interface, já que estas são dependentes e específicas para cada implementação de objeto a ser movimentado, é definida uma interface para uma fábrica **genérica**, a qual suporta uma operação de criação, cujos parâmetros podem ser definidos dinamicamente, possibilitando a inclusão de vários critérios. Para criar um objeto, um cliente deve, necessariamente, possuir uma referência de objeto para uma fábrica que atenda as características do objeto em movimentação.

  Fábricas não são objetos especiais, possuindo as mesmas características de um objeto qualquer em CORBA. Tanto a implementação quanto a definição das interfaces das fábricas são consideradas parte do desenvolvimento da aplicação.

  Caso esse parâmetro seja introduzido como uma referência de objeto nula então o próprio localizador de fábricas deverá encontrar uma localização que suporte sua movimentação.

  O parâmetro **Criterio** consiste numa seqüência de pares de valores, a qual suporta o acréscimo de novos pares e permite que objetos apenas os repassem sem interpretá-los. Sua função é apresentar à fábrica quais as necessidades e preferências do cliente (quem está determinando a movimentação). São apresentadas as seguintes sugestões:

  - **inicialização** - contém valores específicos para inicialização;

[3]Tradução de **factory finder**.

- **filtro** - permite restringir as localizações novas possíveis do objeto, possibilitando influir na alocação de certos recursos. Por exemplo, só efetuar a movimentação para onde existir um sistema operacional UNIX;
- **localização lógica** - permite situar logicamente o objeto através de seus relacionamentos e respectivos objetos (por exemplo, X usa Y e é contido por Z); e
- **preferências**: possibilita, por exemplo, indicar o tipo de máquina com a qual se quer trabalhar ou, mesmo, determiná-la especificamente (pelo nome ou endereço IP).

b) localizador de fabrica - é definido da seguinte forma:

```
interface LocalizadorDeFabrica {
        Fabricas localiza_fabricas (in Chave chave_fabrica)
        raises (FabricaInexistente);
       };
```

Como vemos, possui uma operação que encontra a fábrica que será utilizada para mover o objeto. Tem como parâmetro de entrada uma chave usada para identificar a fábrica desejada. Pode ocorrer de várias fábricas possuírem a mesma chave, nesse caso a operação localiza_fabricas retornará um conjunto de fábricas, com suas respectivas localizações. Um dispositivo de armazenamento específico numa máquina específica, por exemplo, pode ser representado por uma localização.

As exceções até aqui apresentadas são acionadas nos seguintes casos:

- **FabricaInexistente**: a fábrica apropriada não é encontrada;
- **NaoMovivel**: o alvo recusa-se a mover (por motivos de implementação, por exemplo);
- **CriterioInvalido**: o critério não é compreendido pelo alvo; e
- **CriterioNaoEncontrado**: o alvo não tem condições de satisfazer o critério solicitado.

Como mencionamos inicialmente, a funcionalidade do Ciclo de Vida deve ser estendida para suportar relocação. Para tanto a sua função de movimentação deve acionar as seguintes operações no Relocador:

- **insere_inicio_movimentacao** - usando como parâmetro de entrada o identificador do objeto a ser movimentado;
- **insere_nova_localizacao** - a nova localização pode ser extraída da localização da fábrica que será utilizada na movimentação do objeto.

- **Repositório de Implementações** - uma das opções para implementação consiste em armazenar a localização do Relocador neste repositório.

Quando ocorre do ORB não encontrar o objeto selecionado na localização esperada, é feita nova consulta ao Repositório de Implementações para se encontrar o endereço do Relocador, no qual estará a nova localização do objeto migrado.

Uma proposta, apresentada em [DBM92], para um ambiente ANSA (o qual obedece à padronização ODP), sugere que o endereço do Relocador esteja contido na própria referência de interface, entretanto, o mapeamento dessa solução para a CORBA obrigaria, a princípio, numa perda de flexibilidade no que diz respeito à implementação das referências de objetos, obrigando que sua estrutura seja constituída pelo endereço do Relocador. A liberdade de escolha para a forma da referência de objeto pelos diversos ORBs é uma de suas mais importantes características, e ficaria enfraquecida com este tipo de solução. Além disso, esse procedimento aumenta o tamanho da referência, cuja taxa de transferência pela rede podemos supor alta uma vez que qualquer objeto que deseje interagir com outro deverá possuir uma referência de objeto.

Com a natural evolução dos sistemas, usando replicação e estabelecendo domínios para os Relocadores, percebe-se que essa solução causa uma sobrecarga ainda maior, pelo acréscimo de novos endereços às referências. Ainda em [DBM92], tem-se que cada endereço acrescentado à referência de interface aumenta o seu tamanho em cerca de 24 bytes.

Uma solução pode ser a associação de mais esse dado à referência de objeto, embutida nos **ids**. Todo objeto que sabe que pode ser movimentado (pela herança da interface do objeto Ciclo de Vida, por exemplo), ao registrar sua definição de implementação de objeto (que ficará armazenada no Repositório de Implementações) acrescenta o endereço do Relocador do domínio ao qual pertence.

De posse de uma referência de objeto, o cliente inicia uma requisição. Caso o objeto fornecedor do serviço não seja encontrado na localização esperada (obtida a partir da referência de objeto, de alguma forma), é feita uma consulta ao Relocador para verificar se este possui uma referência para o objeto solicitado. Se possui a referência e já existe uma localização correspondente então é enviada nova requisição com a nova localização. Se o Relocador já possui a referência e ainda permanece o sinal de movimentação, deve ser observada uma política específica. Por exemplo, aguardar um determinado período de tempo e tentar novamente até que a informação seja obtida. Caso não exista o objeto no Relocador, o cliente receberá uma exceção ("implementação não existe", por exemplo).

5.2 Relocação entre ORBs com diferentes implementações

Com o acréscimo da capacidade de interoperabilidade entre ORBs com diferentes implementações, faz-se necessário estender o esquema anteriormente descrito, de maneira a garantir a correta identificação dos objetos movimentados entre esses ORBs. Considera-se o caso em que um objeto, inicialmente num ORB A, é movimentado para um ORB B. Um cliente, necessitando dos serviços oferecidos por aquele objeto, deve ser capaz de efetuar uma invocação normalmente, sem preocupar-se ou, mesmo, perceber, que o objeto requisitado alterou sua localização para além do escopo do ORB em que o próprio cliente está trabalhando.

Para permitir essa movimentação de forma transparente, são necessárias as seguintes extensões, cujos objetos que as suportam estão representados na Figura 5.3:

- **Objeto que efetua a relocação** - deve enviar, junto com as informações necessárias à relocação, uma **chave** que identifique, de forma única e confiável, o objeto sendo movimentado (seta **a**, na Figura 5.4). Tal identificador não necessita ser interpretado pelo ORB destino, sendo compreendida apenas pelo ORB que a originou, portanto, deve ser transmitida como um dado opaco.

  Essa chave é enviada ao Interceptador (seta **b**, na Figura 5.4), onde será armazenada para futura verificação, durante o mapeamento dos objetos.

  Essa chave também é introduzida no Relocador (seta **c**, na Figura 5.4), onde será interpretada como um sinal representando que o objeto ao qual corresponde está em processo de movimentação (Figura 5.2).

REFOBJ A	LOCALIZAÇÃO: chave
	nulo
	nulo

Figura 5.2: Exemplo da tabela mantida pelo Relocador, agora com a indicação de que o objeto foi relocado para outro ORB

Ao ser movimentado, pode-se considerar que o objeto está sendo instalado no novo ORB, portanto, deve ser registrado, através do Adaptador de Objetos, a fim de tornar-se disponível para utilização pelo sistema e gerar uma nova referência de objeto, compatível com a nova plataforma. É responsabilidade do objeto que efetuou a relocação providenciar que o objeto migrado torne-se disponível para o seu ORB de origem.

Para tanto, o Objeto que efetua a relocação deve ser capaz de prover meios para que seja efetuada uma invocação ao **Interceptador**, a partir do ORB destino da movimentação, de forma a possibilitar o mapeamento entre as referências de objetos das plataformas em questão. Juntamente com as referências é necessário que sejam enviadas as chaves.

Usando as definições para o Ciclo de Vida, podemos agregar essa funcionalidade ao objeto **fábrica**, o qual tem a capacidade de criação do objeto sendo movimentado. A referência de objeto decorrente dessa criação, juntamente com a chave, é enviada ao Interceptador (através da operação cria_proxy). A chave pode ser inserida no **id** da referência.

Neste ponto, convém lembrar a possibilidade de ultrapassarmos limites administrativos durante as movimentações e, portanto, a conseqüente necessidade da permissão do Administrador para oferecer um serviço, agora fornecido por um objeto de sua responsabilidade,

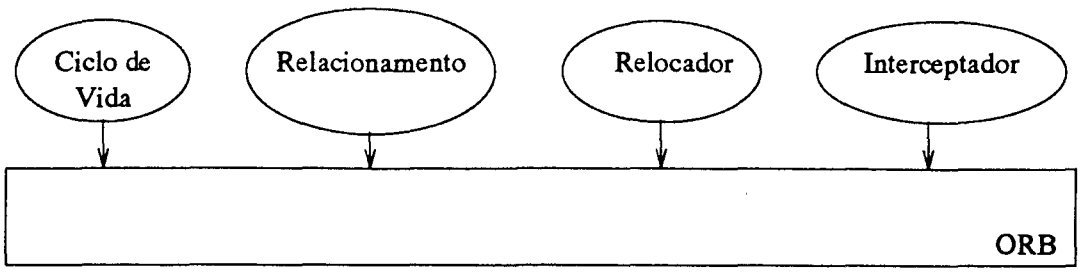

Figura 5.3: Objetos necessários para suportar a transparência de relocação entre ORBs

ao outro ORB. Considera-se essa decisão inerentemente administrativa e, no caso, existe uma autorização implícita, uma vez que a própria relocação exigiu, anteriormente, um acordo entre as administrações.

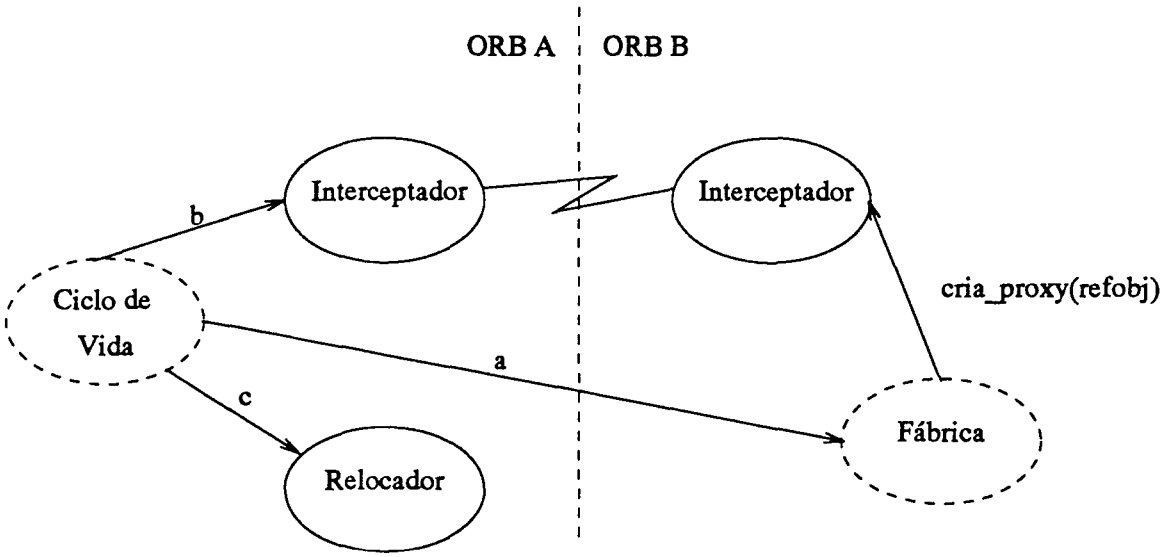

Figura 5.4: Mecanismo para suporte à transparência de relocação

- **Interceptador** - ao ser invocado para prover o mapeamento entre as referências de objetos utilizadas por um ORB e outro, o Interceptador verifica se existe uma chave agregada à referência a ser mapeada, o que indica que o objeto que está sendo apresentado, na verdade, pertenceu, em algum momento, ao próprio ORB para a qual está tendo sua referência convertida. Percebendo esse fato, o Interceptador pode:

  a - introduzir no Relocador, respectivamente à referência de objeto indicada pela chave, a sua própria localização, acrescida de um sinal informando tratar-se de um *proxy* (Figura 5.5). Como já mencionamos (seção 4.4.1), o Interceptador funciona como um representante local do objeto remoto, para o qual repassa as mensagens referentes a uma invocação e do qual recebe as respostas, caso existam, entregando-as ao cliente. A operação **insere_nova_refobj** é a responsável pela inserção.

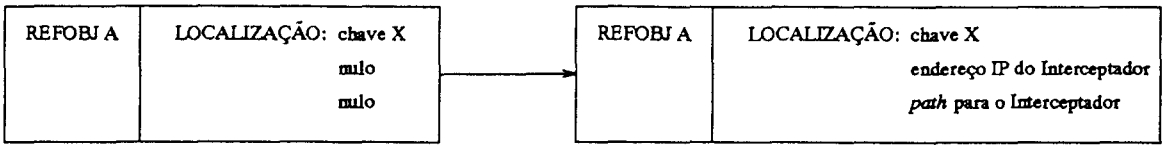

Figura 5.5: Inserção da localização do Interceptador (tendo sido detectado mesma chave) na tabela gerenciada pelo Relocador

O esquema de envio da requisição, portanto, permanece o mesmo, sendo que a invocação é desviada para o respectivo Interceptador e, ao chegar no Adaptador de Objetos, recebe, ainda no ORB cliente, o tratamento de um *proxy*, obedecendo o descrito na seção 4.4.1.

b - usar a referência de objeto contida no Relocador para efetuar a troca da definição de implementação de objeto indicada por essa referência pela definição do próprio Interceptador. Isto pode ser efetuado pela operação do Adaptador de Objetos ***change_implementation***, que pode ser chamada normalmente pelo Interceptador. Este procedimento é mais genérico uma vez que pode usar o mesmo mecanismo de localização adotado pelo ORB. Outra diferença é que a funcionalidade da operação cria_proxy, ao detectar um objeto relocado pela presença da chave, não mais cria um novo objeto (que seria um *proxy*), mas altera a definição de implementação associada a referência de objeto. O Interceptador deve, então, avisar ao Relocador que sua referência de objeto continua válida. Dessa forma, é necessário uma nova resolução de referência para efetuar nova requisição.

A requisição, em ambos os casos será enviada ao Interceptador.

- **Relocador** - além das funções descritas, deve passar a suportar:

```
void        insere_nova_refobj (
                    in Object refobj          // referência do objeto
                                                 iniciando a movimentação
                    in string NovaRefobj        // refobj atual do
                                                   objeto
            );
```

Essa operação é usada para a inserção dos identificadores dos objetos. É acionada pelo Interceptador, que é o objeto que "conhece" quais os objetos que devem ter suas localizações inseridas no Relocador.

```
void        substitui_refobj (
                    in Object refobjproxy       // referência do objeto
                                                   proxy
            );
```

Essa operação faz a substituição da referência de objeto, que estava sendo utilizada para a invocação, por outra, fornecida pelo Relocador, representando um *proxy* do objeto movimentado. Para tanto, deverá poder acessar a requisição que está sendo enviada, a fim de executar essa alteração e iniciar a "montagem" de uma **Requisição do Servidor**.

- **Objeto Relacionamento**- a existência de um objeto que referencie outros é uma situação comum num ambiente distribuído. Portanto, quando se pensa em movimentação de objetos para outros ORBs, surge a preocupação com as conseqüências desse fato, pois um fornecedor de serviços pode utilizar serviços de outros objetos, os quais permanecem no ORB origem e, portanto, inacessíveis ao objeto agora deslocado para outro ORB.

  Esse objeto é constituído por uma tabela que contém as referências de objetos para todos os objetos que têm a capacidade de movimentação em um ORB. Para cada uma dessas referências é armazenada a correspondente relação de todos os outros objetos aos quais referencia.

  Operações que deve suportar:

  - **insere_objeto (in refobj, in** $refrefobj_1$**, in** $refrefobj_2$**, ...)** - insere objeto com respectivos objetos referenciados;
  - **insere_ref (in refobj, in** $refrefobj_5$**, in** $refrefobj_6$**, ...)** - insere novas referências para um objeto já existente (para o caso de ter havido alguma alteração no objeto);
  - **deleta_objeto (in refobj)** - retira o objeto e suas referências da tabela;
  - **busca_ref (in refobj)** - busca o objeto e retorna todas as suas referências; e
  - **remove_ref (in refobj, in** $refrefobj_2$**, in** $refrefobj_6$**, ...)** - retira as referências $refrefobj_2$, $refrefobj_6$, ..., do objeto refobj. Para o caso de algumas das referências utilizadas pelo objeto deixarem de existir, o que pode acontecer quando o objeto não utiliza todos os objetos referenciados e alguns deles têm sua funcionalidade agregada a outros. Neste caso, esta operação pode ser utilizada em conjunto com **insere_ref** para efetuar uma substituição dos objetos referenciados.

  Quando um determinado objeto for movimentado o Relocador verifica, no objeto relacionamento, quais são as suas referências e, então, as envia ao Interceptador para que este crie um *proxy* para cada um dos objetos referenciados (através da operação **criar_proxy ()**). Com tal procedimento, esses objetos poderão ser acessados, pelo objeto movimentado, como se estivessem todos fazendo parte de um mesm ORB.

5.3 Domínio de um Relocador

Podem ser considerados vários domínios[4] para a distribuição de Relocadores:

- Apenas um para todos os objetos no escopo de um ORB específico;

[4]Conjunto de objetos, os quais se relacionam com um objeto controlador (no caso, o Relocador) através de um mesmo tipo de relacionamento (no caso, localização) [ISO95b].

- Um para todos os objetos em cada máquina;
- Um para todos os objetos de um mesmo cliente;
- Um para todos os objetos de uma determinada interface; ou
- Uma combinação das formas acima, podendo formar, inclusive, uma hierarquia.

Dependendo da política de movimentação de objetos, inerente às necessidades do ambiente em que está sendo implementada, decide-se por uma das possibilidades apresentadas.

Para cada caso, poderá ser efetuada uma forma de identificação do Relocador responsável, por exemplo:

- O Relocador é conhecido pelo ORB em tempo de instalação;
- Ao ser instanciado, o objeto é cadastrado junto ao Relocador responsável pela máquina em que foi efetuada a instanciação;
- O cliente, ao obter sua primeira referência de objeto, é cadastrado em um Relocador e, a partir daí, todas as referências solicitadas por esse cliente, serão associadas ao mesmo Relocador;
- A Implementação de Objeto insere a identificação do Relocador na referência de objeto, através da referência de dados (*id*) ou, mesmo, através da definição da implementação (***ImplementationDef***), a qual não está completamente definida pela CORBA;
- O Adaptador de Objetos, ao criar uma referência de objeto para determinada interface, a associa a um Relocador específico.

De qualquer forma, o Relocador introduz um componente crítico para o sucesso da operação e, portanto, um ponto de falha potencial. É possível torná-lo mais confiável, por exemplo [DBM92]:

- introduzindo transparência de falhas - através de pontos de verificação[5] os quais mantêm consistente um Relocador secundário, pronto para ser acionado caso o principal falhe, provendo uma perda mínima de informação; e/ou
- introduzindo transparência de replicação - através da replicação dos Relocadores, assumindo uma réplica, caso ocorra falha do Relocador em uso. Observa-se que a replicação implicará num baixo custo em relação à latência, uma vez que é esperada uma maior quantidade de leituras do que atualizações, considerando-se que, normalmente, a freqüência de movimentações num sistema não deve ser muito alta.

[5]Tradução de *checkpoint*.

Capítulo 6

Implementação de um caso de invocação de objetos remotos suportando transparência de relocação

6.1 Introdução

Neste capítulo será apresentada a implementação de um mecanismo para possibilitar a invocação de objetos remotos, com capacidade de relocação, de uma forma transparente ao cliente. Para tanto, serão utilizados os conceitos e mecanismos apresentados nos capítulos 3 e 4, referentes às necessidades para prover interoperabilidade e transparência de relocação em ORBs distintos.

Como estrutura básica, é utilizado um protótipo apresentado em [dM95], que compreende um subconjunto da funcionalidade de um ORB, e a ORBeline, descrita na seção 3.9.

6.2 Características da implementação ORB apresentada em [dM95]

O protótipo descrito em [dM95] provê um mecanismo para a invocação de operações em objetos remotos, tendo como base o sistema de RPC[1], da Sun, e implementado na linguagem C++.

Cada invocação de operação num objeto é realizada por uma chamada em RPC, formada pelos parâmetros próprios da invocação e a referência para o objeto fornecedor do serviço.

O sistema RPC implementa um mecanismo de comunicação lógica entre cliente e servidor, possibilitando que o cliente efetue uma chamada a um procedimento remoto como se fosse local, através da invocação de procedimentos locais, chamados ***stubs***, os quais ocultam os detalhes

[1] *Remote Procedure Call* - Chamada de Procedimento Remoto.

Tipo de dado CORBA	Tipo de dado XDR
short	*integer*
long	*integer*
unsigned short	*unsigned integer*
unsigned long	*unsigned integer*
float	*floating-point*
double	*double-precision floating point*
char	*opaque*
string	*string*
boolean	*boolean*
octet	*opaque*
enum	*enum*
any	*discriminated union*
struct	*structure*
sequence	*variable-length array*
union	*discriminated union*
array	*fixed-length array*

Tabela 6.1: Mapeamento de tipos de dados CORBA x XDR

de funcionamento da rede, tornando transparente ao cliente todos os aspectos de passagem de parâmetros e de troca de mensagens entre os *stubs* do cliente e os seus correspondentes no servidor.

Para permitir a passagem de dados entre máquinas que possuem diferentes representações desses dados existe o XDR, o qual provê um conjunto de convenções para uma representação de dados padronizada. O módulo XDR é implementado como um conjunto de rotinas de biblioteca as quais são utilizadas pelo RPC.

O mapeamento entre os tipos de dados XDR e CORBA pode ser observado na Tabela ?? [dM95].

A estrutura básica do sistema implementado pode ser observada na Figura ??.

O *portmapper* tem a finalidade de identificar a porta na qual o servidor está escutando, recebendo como parâmetros a localização do servidor e o identificador da classe.

O Repositório de Implementações é usado para localizar um servidor, recebendo como parâmetro a classe dos objetos por ele implementados. É implementado como um arquivo contendo um registro para cada servidor existente na rede, composto pelo identificador da classe que implementa e o endereço IP da máquina em que se encontra. A classe, neste sistema, consiste na interface do serviço a ser solicitado.

O Repositório de Objetos consiste numa lista de referências cruzadas entre as referências de objetos e seus ponteiros. O Adaptador de Objetos é acionado pela implementação para gerar uma referência para aquele objeto. Ao receber uma solicitação de serviço, o adaptador de objetos busca a referência de objeto utilizada na requisição e, utilizando o ponteiro correspondente,

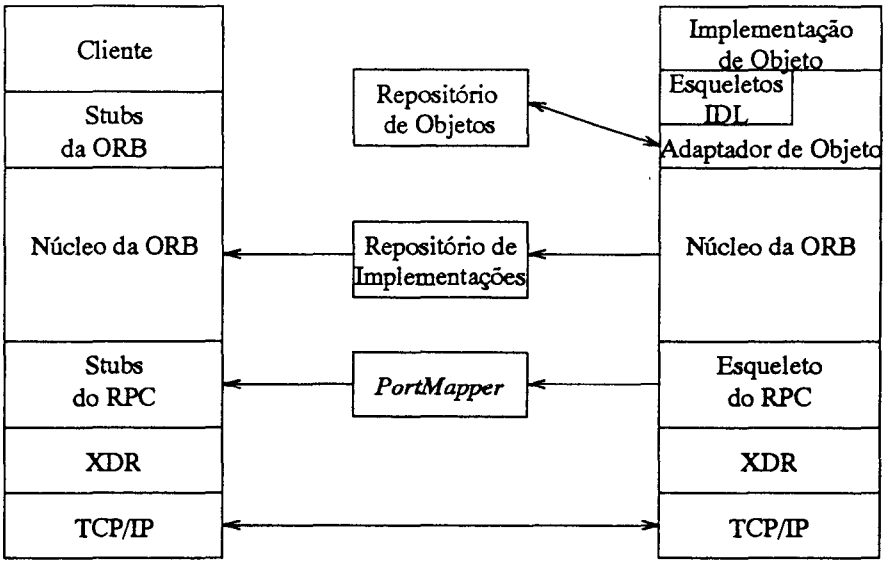

Figura 6.1: Arquitetura básica do protótipo apresentado no ORB desenvolvido em [dM95]

efetua a chamada ao método adequado.

Os *stubs* do ORB têm a finalidade de repassar a invocação e seus parâmetros para o sistema RPC. O *stub* RPC prepara os parâmetros para serem utilizados pela rotina de suporte do RPC responsável pela invocação, a qual efetua a conversão dos parâmetros (através das bibliotecas XDR) e os transmite. Processo inverso ocorre com a resposta, sendo convertida do formato XDR para uma forma compreensível pelo *stub* RPC e, então, enviada ao *stub* ORB.

Processo semelhante ao da resposta, ocorre no lado do servidor. Os parâmetros são convertidos do formato XDR para o formato compreensível pelo *stub* RPC, o qual aciona o esqueleto ORB correspondente ao método desejado.

A referência de objeto foi implementada como uma estrutura contendo o endereço IP da máquina, um identificador da classe (a interface do objeto) e um identificador do objeto dentro da classe a qual implementa.

Cada método, da classe que está sendo implementada, será associado a um *stub* e um esqueleto, gerados em linguagem C++, os quais deverão ser integrados aos processos do cliente e do servidor, respectivamente.

A implementação descrita em [dM95] apresenta a funcionalidade básica mínima da CORBA, não tendo sido implementados a Interface de Invocação Dinâmica, o Repositório de Interfaces, e alguns recursos da IDL, como exceções e contextos. Também só foi implementado o modo *in* dos parâmetros das operações e o seu retorno. A única política de ativação da Implementação existente é a de um processo para cada classe de objeto (seção 3.5).

6.3 Implementação do protótipo

O protótipo descrito nesta dissertação utiliza a implementação apresentada na seção anterior e a ORBeline (seção 3.9), e trata o caso do cliente, localizado no ORB desenvolvido em [dM95],

efetuar a invocação a um serviço implementado por um objeto localizado no ambiente da ORBeline. Considera também o caso do objeto fornecedor do serviço estar inicialmente localizado no ORB de [dM95], mas ter sua localização alterada para a ORBeline (Figura ??). Para possibilitar essa capacidade foram agregados ao ORB desenvolvido em [dM95], o Relocador, o Interceptador e a Interface de Esqueleto Dinâmico, e à ORBeline, o Interceptador.

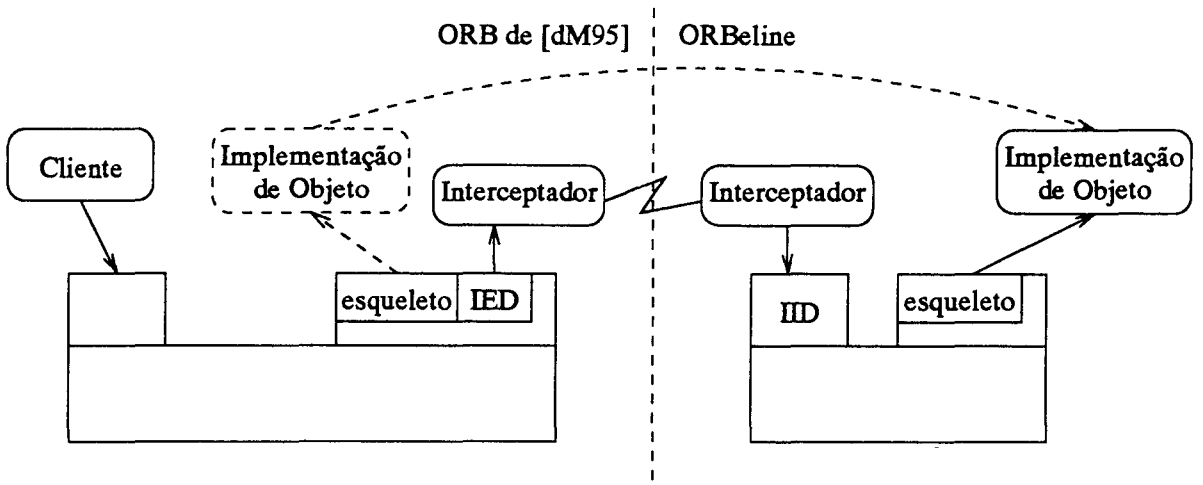

Figura 6.2: Mecanismo utilizado pelo protótipo

A implementação proposta possibilita, com algumas restrições:

- a transparência de relocação no protótipo de [dM95];
- a invocação, de forma transparente, a partir do protótipo, de serviços que tenham sido registrados na ORBeline e colocados disponíveis ao protótipo; e
- a transparência de relocação de um objeto que tenha se movimentado do protótipo para a ORBeline.

Além disso, são estendidas as transparências de acesso e localização para os serviços fornecidos por objetos na ORBeline e tornados disponíveis ao ORB desenvolvido em [dM95].

A Figura ?? apresenta a estrutura básica do ORB de [dM95] modificada com os novos objetos necessários para suportar interoperabilidade e transparência de relocação. Envolvido pela linha pontilhada temos o ORB, abrangendo, pela forma como foram implementados, o Relocador e o Repositório de Implementações, podendo ter seu acesso efetuado diretamente pelo Núcleo do ORB. Da mesma forma, o Núcleo do ORB pode ser expandido para a linha tracejada, envolvendo o mecanismo de RPC, pois o Núcleo, pela definição da CORBA, já possui as funcionalidades fornecidas pelo RPC.

6.3.1 Implementação da transparência de relocação

O Relocador foi implementado como um objeto que gerencia uma tabela contendo: a referência de objeto (ou seja, o nome da máquina na qual o objeto estava originalmente, o identificador

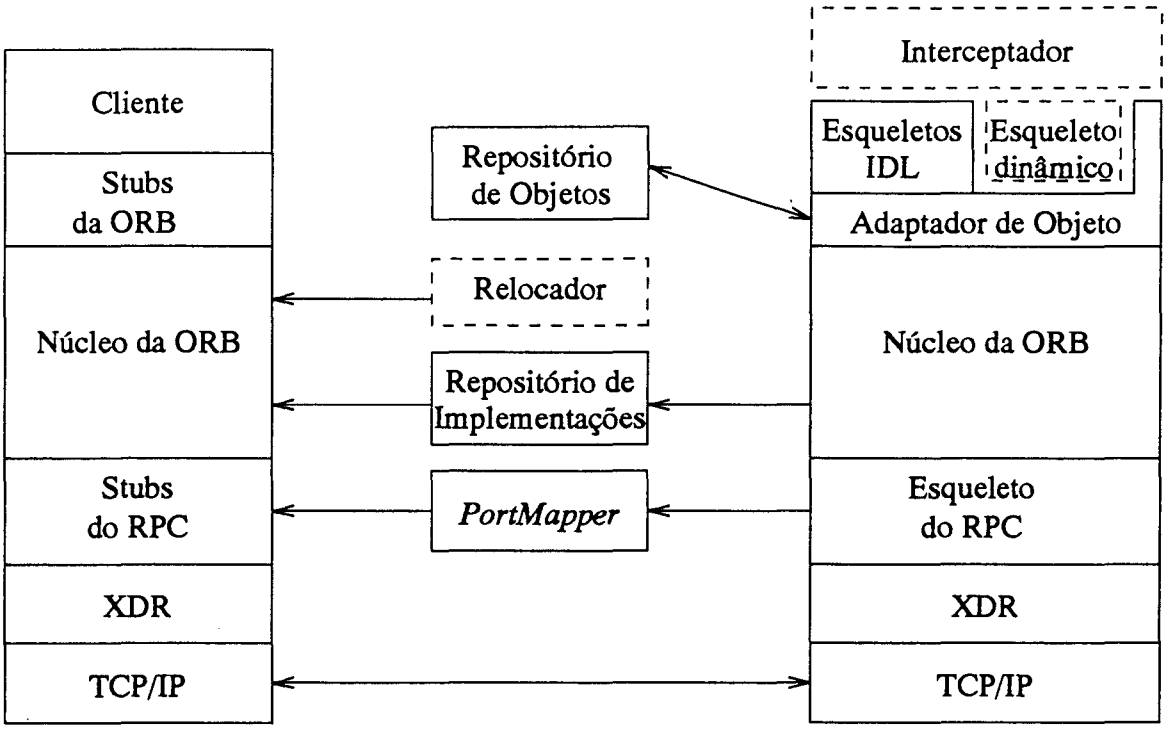

Figura 6.3: Arquitetura do sistema, apresentando o Relocador e o Interceptador (parte do cliente)

da interface do objeto e o identificador do objeto) e o nome da máquina para a qual o objeto foi movimentado. Quando se faz necessário uma consulta, basta efetuar a verificação pelo nome da máquina origem da movimentação. Neste protótipo esta tabela é implementada como uma lista em um arquivo; para uma implementação envolvendo uma utilização "real" recomenda-se a implementação numa tabela de *hashing*, por exemplo.

Todas as requisições são enviadas ao Adaptador de Objetos, que foi implementado juntamente com os esqueletos e o Interceptador, num só processo. Ao receber uma invocação, o adaptador define qual esqueleto será utilizado, desviando-a para o esqueleto dinâmico, se constatar que o objeto sendo invocado é um *proxy*.

São obedecidos os seguintes passos, no decorrer de uma invocação:

- Inicialmente, busca-se, no repositório de implementações, a localização de um servidor que forneça a interface desejada;

- De posse dessa localização, é verificado se o objeto não é um *proxy*, a partir do seu identificador de objeto e, então, é efetuada uma invocação ao BOA respectivo, o qual determina o esqueleto para onde será enviada, caso seja um objeto "real". Caso contrário, a requisição será tratada pelo esqueleto dinâmico, o qual recebe as informações da requisição e aciona a Rotina de Invocação Dinâmica que fará a extração dos dados necessários para o Interceptador repassar a invocação para a ORBeline.

Como a invocação segue diretamente ao esqueleto, a responsabilidade por verificar se um objeto é ou não um *proxy* é do *stub*, o qual efetua também a construção da RequisiçãoDoServidor. Tendo acesso aos parâmetros para a invocação, assumiu-se que os códigos de tipos são obtidos pela consulta ao Repositório de Interfaces, o qual não foi implementado, e, então, são acrescentados à requisição.

- Ao se tentar uma conexão com o objeto fornecedor do serviço (vamos chamar de **servidor**), pode ocorrer deste ter efetuado uma movimentação, impossibilitando, portanto, o acesso aos seus serviços. Detectando a situação do servidor não responder uma chamada, é efetuada uma consulta ao Relocador para buscar a nova localização do objeto, caso o mesmo tenha, de fato, movimentado e não apenas falhado; e

- De posse da nova localização, o procedimento ocorre como anteriormente descrito, verificando se o objeto é ou não um *proxy* e agindo de acordo.

A movimentação do objeto, fora do escopo desse trabalho, é realizada através da destruição do processo que fornece o serviço solicitado, na localização em que está sendo invocado, seguido de sua instanciação em outra localização. Durante o processo de instanciação, a própria implementação de objeto efetuará o registro do objeto no novo ORB, através do Adaptador de Objeto, e no Interceptador, tornando-o disponível para acesso pelo ORB de origem. Utilizou-se um objeto, fazendo o papel do Ciclo de Vida, que obtém a referência do objeto a ser movimentado e a referência para a fábrica que vai registrá-lo na ORBeline. O Ciclo de Vida insere a referência do objeto a ser movimentado e sua chave junto ao Relocador e ao Interceptador. De posse da referência para o objeto fábrica, no caso um *proxy*, uma vez que o objeto, de fato, está localizado na ORBeline, é solicitada a sua instanciação na ORBeline, enviando a chave identificadora do objeto sendo movimentado. A fábrica registra o objeto na ORBeline, normalmente, e invoca o Interceptador para a criação de um *proxy* no ORB desenvolvido em [dM95].

A chave é obtida de um gerador de chaves que fornece uma estrutura composta pelo endereço IP da máquina, uma marca de tempo (obtida através do comando **time** do UNIX) e um número próprio incrementado unitariamente.

6.3.2 Implementação do Interceptador

O Interceptador é composto por duas partes, uma com as propriedades de uma Implementação de Objeto no ORB desenvolvido em [dM95] e outra implementada como um cliente na ORBeline. A transmissão das informações entre esses meio-interceptadores pode ser realizada por várias formas, como comentado na seção 4.4.1, entre as quais, podemos citar: *sockets* [Sun90], TLI[2], RPC ou, mesmo, um outro ORB. Para a nossa implementação escolheu-se *sockets*.

Vamos chamar de Interceptador-cliente a parte do Interceptador posicionado no ORB em que se encontra o cliente do serviço solicitado, e de Interceptador-servidor a parte que se encontra no mesmo ORB em que está o objeto fornecedor do serviço solicitado. Esta nomenclatura é simplesmente conceitual, estabelecida no sentido de facilitar a apresentação do mecanismo de

[2] *Transport Layer Interface.*

interceptação. As duas partes, na realidade, deveriam possuir as funcionalidades tanto de cliente quanto de servidor.

Comportamento do Interceptador no ORB cliente

O Interceptador-cliente é implementado juntamente com o Adaptador de Objetos e o esqueleto dinâmico. Ao receber uma invocação, o Adaptador detecta tratar-se de um *proxy*, pela chave, e aciona o esqueleto dinâmico, que irá extrair as informações da requisição que chega e fornecê-las ao Interceptador-cliente. Este último, usa a Rotina de Implementação Dinâmica para extrair as informações da requisição, através das operações definidas para a RequisiçãoDoServidor, e enviá-las à conversão adequada. Foi implementada apenas a conversão para as referências de objetos. Os dados convertidos são enviados para o Interceptador-servidor através de *sockets* e, nesse ponto, seria necessário a utilização de um protocolo para a transmissão dos dados, como o IIOP ou o DCE-ESIOP, o que não foi implementado, sendo efetuada apenas a transmissão da requisição e a recepção da resposta.

A Tabela de mapeamento de referência de objeto é a tabela gerenciada pelo Interceptador constituída pela referência do objeto "real" e um identificador gerado pelo próprio Interceptador, ou a chave, caso seja um objeto que tenha sido movido. O Interpretador de código de tipo e a Função de acompanhamento de referência de objeto não foram implementados.

Comportamento do Interceptador no ORB servidor

O Interceptador-servidor recebe os dados enviados pelo Interceptador-cliente e efetua a conversão desses dados para a forma utilizada pelo ORB ao qual pertence (a ORBeline, nesta implementação). Para isso, o Interceptador usa o Conversor diretamente. Esse Interceptador, como qualquer cliente do ORB, deverá consultar o Repositório de Interfaces para extrair os dados necessários para efetuar uma invocação dinamicamente.

Inicialmente, o Interceptador-servidor consulta a Tabela de mapeamento de referência de objeto (implementada neste protótipo como um arquivo) contendo a chave recebida do Interceptador-cliente e a respectiva referência de objeto (Figura ??).

Decidiu-se armazenar a referência de objeto no formato de uma cadeia de caracteres de maneira a possibilitar um futuro suporte à persistência. Por outro lado, quando o objeto é destruído, é conveniente a atualização da tabela.

Portanto, o Interceptador-servidor será o responsável pela conversão da cadeia de caracteres para uma referência de objeto (através da operação *_string_to_object*, da interface ORB). Essa operação retorna um objeto da classe base em Object que servirá como parâmetro de entrada para a operação de criação da requisição dinâmica. Para essa criação usaremos como parâmetros a referência de objeto, o nome da operação e o nome do Repositório de Interfaces, onde se encontra a interface.

Essa operação, na ORBeline, facilita o trabalho dos programadores das aplicações por efetuarem automaticamente a consulta ao Repositório de Interfaces, evitando a necessidade da utilização de um navegador, por exemplo.

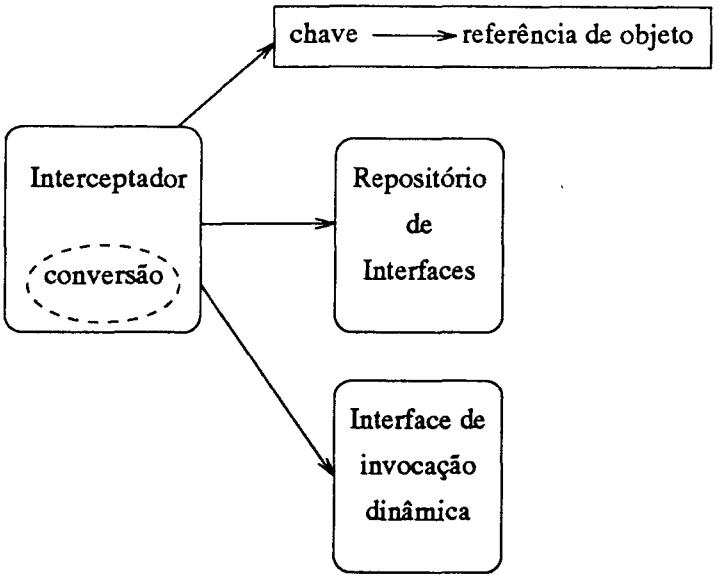

Figura 6.4: Acessos do Interceptador-servidor

A referência de objeto é utilizada para a obtenção do nome da interface, a qual será usada como parâmetro de busca no Repositório de Interfaces. Além disso, essa operação já constrói uma espécie de "gabarito"[3], com seus campos inicializados: o nome da interface, a identificação da operação, o modo (in, out, inout) e o código de tipo de cada parâmetro. A partir daí, o programador deve "preencher" os valores correspondentes aos parâmetros (no caso, só os que possuem modo in ou inout).

Os valores dos parâmetros recebidos, após serem convertidos para o formato da ORBeline, são introduzidos no gabarito. Entretanto, para que isso seja possível, todos os parâmetros devem ser inicializados numa instância de uma classe, específica para cada tipo, derivada da classe base *Value*. Essa classe consiste numa forma genérica para representação de dados. Para cada tipo de dados, existe uma classe que deriva de *Value*, possuindo operações para acesso aos dados. Através dos códigos de tipos são construídas instâncias apropriadas. Através de uma fábrica, que recebe como parâmetro um código de tipo (por referência ou por ponteiro) e retorna um ponteiro para um objeto da classe *Value*, para o tipo especificado, já alocando a quantidade necessária de memória.

De posse do código de tipo, retirado do Repositório de Interfaces, podemos identificar o tipo do parâmetro e, então, usar o valor recebido do Interceptador-cliente e convertido para o formato da ORBeline, para inicializar uma instância da classe *Value*. Encapsulado dessa forma, o valor recebido pode ser introduzido na requisição. Por exemplo, para os tipos primitivos:

```
...
CORBA_DII::REQUEST* REQ = CORBA_DII::REQUEST::CREATE(/* ARGUMENTOS */);
CORBA::TYPECODE::TCKIND TIPO = CODIGO_TIPO.KIND();
CORBA::VALUE *VALOR = REQ- >ARG("NOME_PARAMETRO");
```

[3]Tradução de *template*.

```
VALOR->ULONGVALUE(VALOR_ULONG);
...
```

A partir de uma requisição (req), obtém-se um ponteiro para uma classe, derivada de *Value*, construída para o tipo determinado pelo parâmetro "nome_parametro". Para cada tipo primitivo existe uma operação para inserir o seu valor nas suas respectivas classes. No exemplo, por tratar-se de um inteiro longo sem sinal (representado por *ulong*), utiliza-se a operação correspondente ulongValue(), que deve receber um valor de entrada compatível.

Também seria possível esse resultado pela obtenção do código de tipo do parâmetro "nome_parametro" (através da operação arg_type), a ser utilizado na construção de um *Value* para o tipo especificado (operação factory(CORBA::TypeCode& codigo_tipo)). A operação de acesso a ser utilizada pode ser descoberta através da operação kind(), a qual retorna o tipo do parâmetro, que coincide com o nome da operação acrescida da palavra *Value* (ulongValue(), no exemplo).

Depois de "montada", a requisição pode ser enviada ao objeto fornecedor do serviço, observando-se todo o procedimento normal de uma invocação dinâmica.

Capítulo 7

Conclusão

Neste trabalho foi apresentado um mecanismo para permitir a interoperabilidade entre ORBs implementados com tecnologias diferentes e uma forma de prover suporte à transparência de relocação, estendida para o caso de ORBs interoperantes. Apesar da especificação CORBA não tratar completamente essa transparência, algumas plataformas comerciais já a implementam. A capacidade de interoperabilidade, por sua vez, só tornou-se disponível a partir do segundo semestre de 1995, por umas poucas empresas.

As contribuições deste trabalho consistem na apresentação de um modelo para a construção de um Interceptador, visando possibilitar a interoperabilidade, e, através de uma abordagem envolvendo a transparência de relocação e a interoperabilidade, de um mecanismo para a movimentação de serviços de forma transparente não apenas no escopo de um ORB mas, também, entre ORBs.

Também foi descrita uma forma de implementação desses mecanismos, utilizando-se uma plataforma comercial e um protótipo construído sobre RPC.

A partir do trabalho desenvolvido pode-se concluir que:

a) Em relação ao mecanismo para suportar interoperabilidade:

- O esquema apresentado estende, de forma natural, as transparências de acesso e de localização para o escopo de ORBs interoperantes. Isso é obtido, principalmente, devido à manutenção das características da referência de objeto, representando localmente o objeto fornecedor do serviço, estando ele local ou remoto ao cliente, e permitindo, em conjunto com os *stubs* e esqueletos, um mecanismo de acesso idêntico para ambos os casos;

- O mapeamento das requisições e seus parâmetros entre ORBs, ou de um ORB para o formato intermediário, é totalmente dependente da implementação dos ORBs. Este mapeamento pode ser realizado por um módulo separado do Interceptador [SUZ96]. As conversões do contexto, do contexto do serviço e do *principal*, apesar de serem simples, exigem um acordo de alto nível (administrativo, por exemplo) para garantir sua semântica;

- Em virtude da grande flexibilidade fornecida por um Interceptador genérico, sua complexidade é bastante aumentada, resultando numa queda de desempenho

razoável, uma vez que cada invocação resultará em várias consultas (e invocações) efetuadas pelo ORB e pelo Interceptador, semelhante ao que acontece com a IID [Vin93, SV95]. Essa desvantagem é bastante reduzida com a utilização de um Interceptador específico, embora os serviços a serem solicitados fiquem restritos aos previamente estabelecidos. Portanto, é necessário uma análise de qual seria a opção mais vantajosa, observando-se as necessidades atuais e futuras dos ORBs em questão;

- O mapeamento das referências de objetos, permitindo sua compreensão no escopo de ORBs com diferentes tecnologias de implementação, é fundamental para a interoperabilidade e pode ser obtido através de um mecanismo relativamente simples;

- Sem a consistência dos Repositórios de Interfaces dos ORBs o problema ficaria muito mais complexo, obrigando o mapeamento e a passagem das descrições das interfaces entre os ORBs;

- O GIOP, apesar de permitir a interoperabilidade entre ORBs, não define completamente como esse protocolo será tratado pelas diversas implementações de ORBs existentes;

b) Considerando-se o mecanismo para suportar transparência de relocação:

- É possível um mapeamento das funcionalidades do Relocador no RM-ODP para um objeto semelhante na CORBA;

- Apesar do Relocador tornar-se um "gargalo" (em potencial), não é esperado que ocorram muitas movimentações entre ORBs interoperantes e, tampouco, no escopo de um ORB.

c) Considerando-se o desenvolvimento e implementação do protótipo:

- O protótipo teve grande parte desenvolvida sobre RPC devido à utilização da ORB desenvolvida em [dM95], que o utiliza como base para implementação. Entretanto, percebe-se que várias das implementações existentes atualmente utilizam esse mecanismo apesar de haver restrições no que diz respeito à transmissão de grandes volumes de dados e altas taxas de transferência [SHAS95, GS96]. Neste aspecto específico, ainda não foram completamente apresentadas soluções que permitam um desempenho adequado para aplicações de tempo real, multimídia e redes de alto desempenho.

- A existência de um navegador "embutido" na IID e a classe *Value* da ORBeline facilitou bastante a programação do Interceptador;

7.1 Extensões e trabalhos futuros

Observando-se as desvantagens de um Interceptador genérico, uma extensão deste trabalho pode ser a descrição de um Interceptador originalmente específico mas que permita ser estendido, em virtude da evolução dos objetivos dos ORBs os quais comunica, para tratar interfaces desconhecidas até então. Para isso, deve adquirir as características de um Interceptador genérico. Esta extensão deve ter flexibilidade suficiente para escolher uma invocação dinâmica ou estática no ORB A e, independente dessa decisão, uma invocação dinâmica ou estática no ORB B. Dessa

forma, é possível criar um Interceptador híbrido, agregando às vantagens de um Interceptador específico a flexibilidade de um genérico, além de permitir a evolução das interfaces fornecidas pelo ORB servidora sem exigir alterações nos clientes.

Além disso, pode-se:

- Estender os mecanismos com capacidade de movimentação de objetos, como o Serviço de Objeto de Externalização, de forma que passem a "colaborar" para o suporte à transparência de relocação;
- Identificar as necessidades e estudar as possíveis extensões aos Serviços do ORB, Serviços de Objetos e Facilidades Comuns requeridos para atende ou suportar a interoperabilidade, tendo em vista que, à princípio, apenas o Objeto de Transações [OMG95c, OHE95] foi especificado com a capacidade de atuar em um ambiente com múltiplos ORBs.

Apêndice A

Protocolo Geral entre ORBs

O GIOP consiste em uma especificação genérica de protocolos para permitir a interoperabilidade entre ORBs. Padroniza a sintaxe de transferência, através da CDR, e o formato das mensagens.

A.1 Representação Comum de Dados (CDR)

Cada estrutura de dados é transformada para um conjunto de octetos (*octet stream*), configurando o que se chama de **encapsulamento**. A estrutura encapsulada é, então, representada por uma seqüência de octetos (*CORBA::Sequence <octet>*). Um octeto, composto por oito bits, é um tipo não interpretável, ou seja, não sofre qualquer conversão durante seu transporte. Para resolver o problema da ordenação dos bytes, o primeiro octeto sempre indicará: 0 (Big-Endian) ou 1 (Little-Endian). Os tipos primitivos são sempre codificados em múltiplos de octetos. De forma a permitir a correta manipulação dos dados em um conjunto de octetos, deve-se garantir que o espaço por ele ocupado seja igual ao seu tamanho em octetos, garantindo o alinhamento. Para tanto, são estabelecidas as seguintes correspondências:

Tipos primitivos	número de octetos
char	1
octet	1
short	2
unsigned short	2
long	4
unsigned long	4
float	4
double	8
boolean	1
enum	4

Para os tipos primitivos temos:

- **Tipos de dados inteiros** - os tipo inteiros com sinal (*short* e *long*) são representados como complemento a dois, e os tipos inteiros sem sinal (*unsigned short* e *unsigned long*) como números binários sem sinal (Figura A.1);

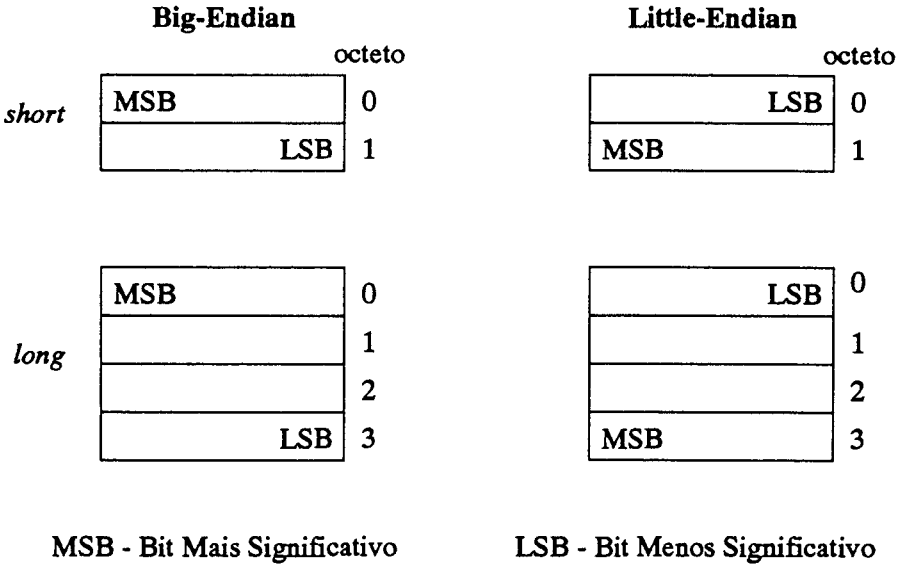

Figura A.1: Tamanho e ordem dos bits de tipos de dados inteiros

- **Tipos de dados de ponto flutuante** - obedece ao padrão ANSI/IEEE. A Figura A.2 mostra os componentes desses tipos: o bit de sinal (**s**), o expoente (**e**) e a parte fracionária da mantissa (**f**). Para precisão simples, **e** usa 8 bits (e1 + e2) e **f** usa 23 bits. Para precisão dupla **e** usa 11 bits (e1 + e2) e **f** usa 52 bits;

- **Octeto** - são valores não interpretados de oito bits, cujo conteúdo não sofre qualquer conversão durante a sua transmissão. Podem ser considerados como valores inteiros de oito bits sem sinal;

- **Caracter** - representado como um octeto, codificado como definido pela ISO Latin-1; e

- **Booleana** - representada como um octeto, onde "verdadeiro" equivale ao valor 1 e "falso" equivale ao valor 0.

Para os tipos estruturados, são mantidas as restrições de alinhamento e as conversões definidas para os tipos primitivos que os compõem:

- **Estrutura** - seus componentes são codificados na ordem em que são declarados;

- **União** - codificado como o discriminante do tipo especificado na declaração da união seguido pela representação do membro selecionado;

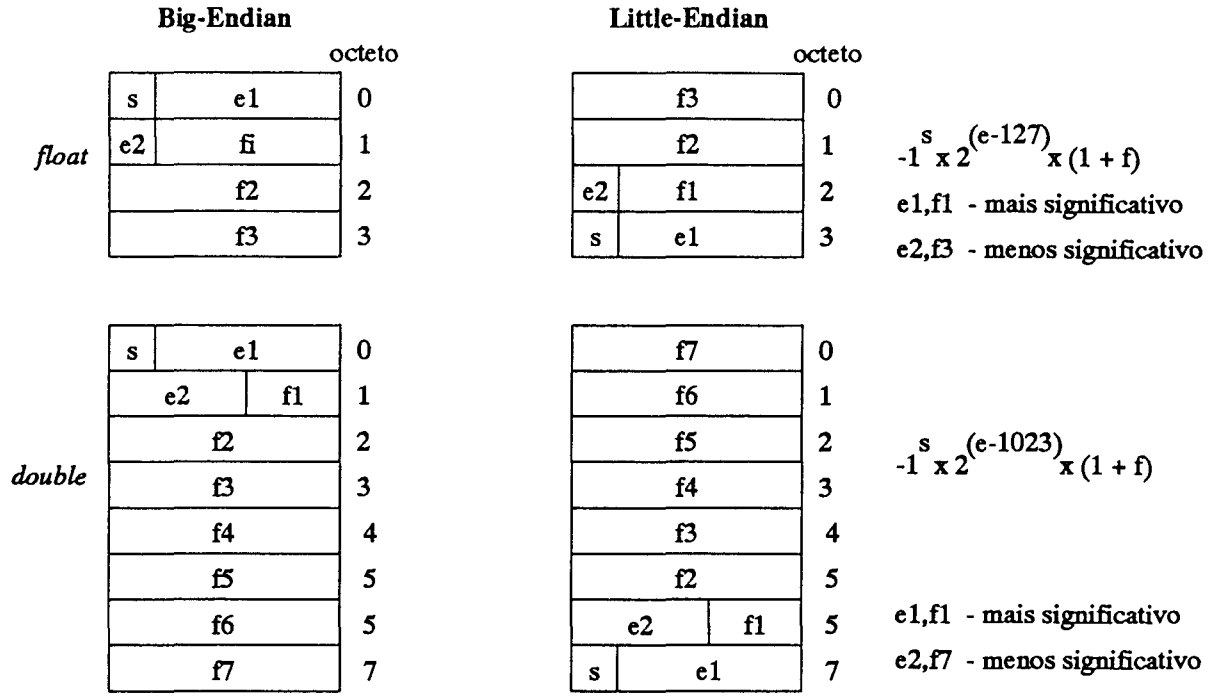

Figura A.2: Tamanho e ordem dos bits de tipos de dados de ponto flutuante

- **Vetor** - codificado como a seqüência dos seus elementos;
- **Seqüência** - codificada como um valor inteiro longo sem sinal (representando o número de elementos que compõem a seqüência) seguido pelos seus elementos;
- **String** - codificada como um inteiro longo sem sinal (contendo o tamanho da string), seguido pelos caracteres que a compõem; e
- **Enumeração** - codificada como inteiros longos. Os valores numéricos associados aos identificadores das enumerações são determinados pela ordem em que aparecem na declaração, com valores crescentes a partir de zero.

Para os pseudo-objetos serão obedecidas as seguintes regras:

- **Código de tipo** - é codificado usando-se o *TCKind* seguido por zero ou mais parâmetros. A lista de parâmetros (Tabela A.1) pode ser:
  - Vazia - a codificação do código de tipo corresponde apenas ao *TCKind*;
  - Simples - a codificação corresponde ao *TCKind* seguido pelo parâmetro codificado conforme descrito na Tabela A.1; ou
  - Complexa - a codificação corresponde ao *TCKind* seguido por uma seqüência de octetos originada do encapsulamento em CDR dos parâmetros. A representação dos parâmetros descritos na Tabela A.1 obedece ao formato: ***tipo do parâmetro (significado do parâmetro)***. Os códigos de tipos para estrutura, união, enumeração

e exceção contêm uma seqüência de tuplas com o número de tuplas determinado pelo **contador**. Essas tuplas (para estrutura, enumeração e exceção) devem estar na mesma ordem em que foram definidas em IDL. O **identificador** corresponde ao identificador de repositório (seção 4.5), sendo obrigatório para referências de objetos e exceções, podendo ser representado por uma cadeia de caracteres vazia para os outros tipos. Os nomes dos parâmetros e os nomes dos membros não são significativos, podendo ser omitidos e também codificados como uma cadeia de caracteres vazia. Em uma união o **padrão usado** indica qual tupla na seqüência descreve o caso a ser escolhido na situação em que nenhuma das alternativas preestabelecidas é adequada. Um código de tipo para vetores multidimensionais são construídos pelo aninhamento de um *tk_array* dentro de outro, um para cada dimensão.

O código de tipo permite a recursividade através da **indireção**, provida pela CDR. Essa indireção é aplicada apenas para códigos de tipos aninhados em outro código de tipo de alto nível, só existindo dentro dele. Outra restrição é que apenas a partir da segunda referência para um determinado código de tipos, no mesmo escopo, pode ser usada a indireção, pois a primeira referência deve ser codificada com as regras normais. A indireção consiste em um inteiro longo representando a quantidade de octetos que sofreram deslocamento[1] dentro do escopo do código de tipo inicial.

- ***Any*** - são codificados como um código de tipo (convertido como descrito anteriormente), seguido pelo valor codificado definido pelo próprio código de tipo;

- ***Principal*** - são codificados como uma seqüência de octetos ("sequence<octet>"), dessa forma, seus elementos não são passíveis de conversão;

- **Contexto** - são codificados como uma seqüência de cadeia de caracteres ("sequence<string>"). As cadeias de caracteres são apresentadas aos pares, de forma que a primeira representa o nome de uma propriedade do contexto e a segunda o seu valor associado; e

- **Exceção** - são codificadas como cadeias de caracteres seguidas pelos membros da exceção, os quais são codificados da mesma forma que uma estrutura.

As referências de objetos são codificadas em IORs, e o padrão GIOP especifica que a descrição dos **perfis** deverá conter:

- o número da versão da especificação do protocolo de transporte que o servidor suporta;

- o endereço de um ponto final para o protocolo de transporte em uso; e

- um dado opaco, de conhecimento apenas do servidor do serviço referenciado.

[1] Tradução de *offset*.

TCKind	Valor	Tipo	Parâmetros
tk_null	0	vazio	nenhum
tk_void	1	vazio	nenhum
tk_short	2	vazio	nenhum
tk_long	3	vazio	nenhum
tk_ushort	4	vazio	nenhum
tk_ulong	5	vazio	nenhum
tk_float	6	vazio	nenhum
tk_double	7	vazio	nenhum
tk_boolean	8	vazio	nenhum
tk_char	9	vazio	nenhum
tk_octet	10	vazio	nenhum
tk_any	11	vazio	nenhum
tk_TypeCode	12	vazio	nenhum
tk_Principal	13	vazio	nenhum
tk_objref	14	complexo	string (identificador),string (nome)
tk_struct	15	complexo	string (identificador), string (nome), ulong (contador) {string (nome do membro), TypeCode (tipo do membro)}
tk_union	16	complexo	string (identificador),string (nome), TypeCode (tipo do discriminante), long (padrão usado), ulong (contador) {tipo do discriminante (valor do lable), string (nome do membro), TypeCode (tipo do membro)}
tk_enum	17	complexo	string (identificador), string (nome), ulong (contador) {string (nome do membro)}
tk_string	18	simples	ulong (tamanho máximo)
tk_sequence	19	complexo	TypeCode (tipo do elemento), ulong (tamanho máximo)
tk_array	20	complexo	TypeCode (tipo do elemento), ulong (tamanho)
tk_alias	21	complexo	string (identificador),string (nome), TypeCode
tk_except	22	complexo	string (identificador), string (nome), ulong (contador) {string (nome do membro), TypeCode (tipo do membro)}
- nenhum -	0xffffffff	simples	long (indireção)

Tabela A.1: Código de tipo

A.2 Formato das mensagens

São estabelecidos os seguintes identificadores para cada tipo de mensagem:

Tipo da mensagem	Origem	Identificador
Request	cliente	0
Reply	servidor	1
CancelRequest	cliente	2
LocateRequest	cliente	3
LocateReply	servidor	4
CloseConnection	servidor	5
MessageError	ambos	6

Além desses identificadores, estabelece o formato a ser seguido por cada uma dessas mensagens, iniciando por um cabeçalho com a seguinte estrutura:

```
struct MessageHeader {
  char          magic[4];
  version       GIOP_version;
  boolean       byte_order;
  octet         message_type;
  unsigned long message_size;
                }
```

onde,

- **magic** - identifica o protocolo da mensagem, através dos caracteres "G","I","O","P", codificados em ISO Latin-1 (8859.1);
- **GIOP_version** - contém o número da versão do protocolo em uso;
- **byte_order** - 0 (Big-Endian) ou 1 (Little-Endian);
- **message_type** - identificador do tipo de mensagem; e
- **message_size** - contém o tamanho da mensagem, em octetos, após o cabeçalho.

A.2.1 Mensagem Request

Representa uma invocação de objeto em CORBA. Além do cabeçalho geral, possui:

- cabeçalho específico :

```
struct RequestHeader {
  ServiceContextList   service_context;
  unsigned             long request_id;
  boolean              response_expected;
  sequence<octet>      object_key;
  string               operation;
  Principal            requesting_principal;
                       }
```

onde,

- **service_context** - contém os dados específicos exigidos por alguns serviços;
- **request_id** - gerado pelo cliente, permite a associação das mensagens enviadas com as mensagens recebidas;
- **response_expected** - será "0", se espera uma resposta do servidor, e será "1", caso contrário (operação *oneway* ou invocação feita através da DII com sinal de INV_NO_RESPONSE);
- **object_key** - dado com significado apenas para o servidor. Não pode ser interpretado nem modificado pelo cliente;
- **operation** - contém o nome da operação sendo invocada; e
- **requesting_principal** - contém a identificação do *principal*.

- corpo da mensagem - contém, nesta ordem:
  - todos os parâmetros **in** e **inout**, na mesma ordem em que aparecem na definição da operação; e
  - um Contexto opcional.

A.2.2 Mensagem Reply

Constituída por parâmetros **inout**,**out** e resultados, caso existam. Também podem retornar exceções. Além do cabeçalho geral, possui:

- um cabeçalho específico - com a seguinte estrutura:

```
struct ReplyHeader {
  ServiceContextList   service_context;
  unsigned long        request_id;
  ReplyStatusType      reply_status;
                       }
```

onde,

- **service_context** - contém os dados específicos exigidos por alguns serviços;

- **request_id** - gerado pelo cliente, permite a associação das mensagens enviadas com as mensagens recebidas; e
- **reply_status** - seus valores determinam parte do corpo da mensagem.

- corpo da mensagem:
  - caso **reply_status** = NO_EXCEPTION, o corpo será representado em uma estrutura constituída, nesta ordem, pelo valor de retorno, se houver, e pelos parâmetros **in** e **inout**, na ordem em que aparecem na definição da operação;
  - caso **reply_status** = USER_EXCEPTION ou SYSTEM_EXCEPTION, o corpo conterá as exceções ocorridas; e
  - caso **reply_status** = LOCATION_FORWARD, o corpo conterá uma IOR, devidamente codificada.

A.2.3 Mensagem CancelRequest

Enviada pelo cliente cancelando uma requisição específica feita anteriormente. Além do cabeçalho geral, possui outro:

```
struct CancelRequestHeader {
    unsigned long request_id;
}
```

onde,

- **request_id** - permite identificar a mensagem que está sendo cancelada.

A.2.4 Mensagem LocateRequest

Tem por finalidade determinar se a referência de objeto é válida, se o servidor é capaz de atender diretamente a requisição ou qual o novo endereço para o qual a requisição deve ser enviada. Pode-se notar que essas respostas também são fornecidas pela mensagem Reply, entretanto, dessa forma, evita-se o trabalho de enviar todos os dados e depois (no caso de um redirecionamento) ter que enviá-los novamente. Apresenta a seguinte estrutura:

```
struct LocateRequestHeader {
  unsigned long          request_id;
  sequence <octet>       object_key;
                         }
```

onde,

- **request_id** - gerado pelo cliente, permite a associar uma LocateReply com uma LocateRequest; e
- **object_key** - identifica o objeto sendo procurado.

A.2.5 Mensagem LocateReply

É a resposta da mensagem anterior. Além do cabeçalho geral, é composta por:

- um cabeçalho específico:

  struct LocateReplyHeader {
  unsigned long **request_id;**
  LocateStatusType **locate_status;**
  }

  onde,

  - **request_id** - usado para associar as mensagens de resposta com as de requisição; e
  - **locate_status** - determina a existência ou não do corpo da mensagem.

- corpo da mensagem:

  - caso **locate_status** = UNKNOWN_OBJECT, o objeto especificado é desconhecido do servidor. Não existe corpo;
  - caso **locate_status** = OBJECT_HERE, o servidor atende diretamente a requisição. Também não existe corpo; e
  - caso **locate_status** = OBJECT_FORWARD, o corpo existe e contém uma IOR.

Bibliografia

[Ber93] P. A. Bernstein. Middleware: An Architecture for Distributed System Services. Technical Report CRL 93/6, Digital Equipment Corporation, março de 1993.

[CM96] F.M. Costa e E.R.M. Madeira. An Object Group Model and its Implementation to Support Cooperative Applications on CORBA. Na *International Conference on Distributed Platforms (ICDP) - IFIP/IEEE*, pp. 213–229, Dresden - Alemanha, fevereiro de 1996.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP - Volume 1; principles, protocols, and architecture.* segunda edição, 1991.

[Cos95] Fabio M. Costa. Suporte a Grupos Cooperativos em Ambiente Distribuído Aberto. Dissertação de Mestrado, Universidade Estadual de Campinas - DCC, 1995.

[DBM92] N. Davies, G.S. Blair, e J.A. Mariani. Supporting persistent re-locateble objects in the ANSA architecture. Technical report, Computing Department, Lancaster University, 1992. MPG-92-04.

[dM95] A. M. V. de Mello. Um Protótipo de Negociador de Requisições de Objetos. Dissertação de Mestrado, Universidade Estadual de Campinas - FEE, 1995.

[dPLJ94] Luiz A. de P. Lima Jr. Um Modelo para a Implantação de Federação de Trades. Dissertação de Mestrado, Universidade Estadual de Campinas - DCC, 1994.

[Gar94] Claudio M. Garcia. Uma proposta para modelagem de Funções de Gerenciamento para Processamento Distribuído Aberto. Dissertação de Mestrado, Universidade Estadual de Campinas - DCC, 1994.

[Gro97] OMG (Object Management Group). A Discussion of the Object Management Architecture. Obtido em http:// www.omg.org/library/omaa.htm, janeiro de 1997.

[GS96] A. Gokhale e D. C. Schmidt. Measuring the Performance of Communication Middleware on High-Speed Networks. Na *ACM SIGCOMM Conference*, Califórnia - EUA, agosto de 1996.

[HOvdL+93] A. Herbert, D. Otway, R. van der Linden, A. Watson, e I. Domville. ORB Interoperability. Technical Report APM/TR.43.00, Architecture Projects Management Limited (APM), 1993.

[ION95a] IONA Technologies Ltd. *The Orbix™ Architecture,* versão 1.3, janeiro de 1995.

[ION95b] IONA Technologies Ltd. *Programmer's Guide,* versão 1.3.1, fevereiro de 1995.

[ISO95a] ISO/IEC — ITU-T. *Draft Recommendation X.901: ODP Reference Model - Part 1. Overview,* maio de 1995.

[ISO95b] ISO/IEC — ITU-T. *Recommendation X.902: ODP Reference Model - Part 2: Foundations,* novembro de 1995.

[ISO95c] ISO/IEC — ITU-T. *Recommendation X.903: ODP Reference Model - Part 3: Architecture,* novembro de 1995.

[JM95] L.A.P. Lima Jr. e E.R.M. Madeira. *Open Distributed Processing: Experiences with Distributed Environment - A Model for a Federative Trader.* Chapman & Hall, pp. 173-184, 1995.

[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. Na *IEEE Computer Software & Applications Conference - COMPSAC 94,* Taipei - Taiwan, novembro de 1994.

[Mad95] E.R.M. Madeira. Multiware Platform: Some Issues about the Middleware Layer. Na *7th IASTED International Conference on Parallel and Distributed Computing and Systems,* Washington - Estados Unidos, pp. 162–166, outubro de 1995.

[MM94] M.J. Mendes e E.R.M. Madeira. Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware. No *12o. Simpósio Brasileiro de Redes de Computadores - SBRC 94,* Curitiba - PR, pp. 93–109, maio de 1994.

[Nan91] B. Nancy. Interoperability today. *Byte,* pp. 187–196, novembro de 1991.

[NWM93] J.R. Nicol, C.T. Wilkes, e F.A. Manola. Object Orientation in Heterogeneous Distributed Computing Systems. *IEEE Computer,* pp. 57–67, junho de 1993.

[OH95] R. Orfali e D. Harkey. Client/Server with Distributed Objects. *Byte,* pp. 151–162, abril de 1995.

[OHE95] R. Orfali, D. Harkey, e J. Edwards. Intergalactic Client/Server Computing. *Byte,* pp. 108–122, abril de 1995.

[OHE96] R. Orfali, D. Harkey, e J. Edwards. *The Essential Distributed Objects: Survival Guide.* 1996.

[OMG93] OMG. *Object Request Broker Architecture,* julho de 1993. OMG TC Document 93.7.2.

[OMG94a] OMG. *Common Object Request Broker: Architecture and Specification,* dezembro de 1994. OMG TC Document 93.12.29.

[OMG94b] OMG. *Common Object Services Specification,vol. 1*, março de 1994. OMG TC Document 94.1.3.

[OMG94c] OMG. *IDL to C++ Language Mapping*, 1994. OMG TC Document 94.8.2.

[OMG94d] OMG. *Object Services RFP3*, julho de 1994. OMG TC Document 94.7.1.

[OMG94e] OMG. *Object Services RFP4*, abril de 1994. OMG TC Document 94.4.18.

[OMG94f] OMG. *OMG ORB2.0 Interoperability and Initialisation RFP Response*, março de 1994. BNR Submission - OMG TC Document 94.3.4.

[OMG94g] OMG. *OMG RFP Submission - Interface Repository*, novembro de 1994. OMG TC Document 94.11.7.

[OMG94h] OMG. *OMG RFP Submission - Interface Repository*, maio de 1994. OMG TC Document 94.5.2.

[OMG94i] OMG. *OMG White Paper on Security*, abril de 1994. OMG TC Document 94.4.16.

[OMG94j] OMG. *ORB 2.0 RFP Submission - ORB Interoperability*, março de 1994. IONA/Sun Submission - OMG TC Document 94.3.1.

[OMG94k] OMG. *ORB 2.0 RFP Submission - Universal Networked Objects*, setembro de 1994. OMG TC Document 94.9.32.

[OMG94l] OMG. *ORB Interoperability*, março de 1994. ICL submission - OMG TC Document 94.3.3.

[OMG94m] OMG. *RFP Interoperability and Initialization*, setembro de 1994. OMG TC Document 93.9.15.

[OMG95a] OMG. *Common Facilities Architecture*, novembro de 1995. revisão 4.0. CORBAfacilities.

[OMG95b] OMG. *Common Facilities Roadmap*, janeiro de 1995. revisão 3.2. OMG TC Document 95.1.32.

[OMG95c] OMG. *Common Object Request Broker: Architecture and Specification*, julho de 1995. OMG TC Document 96.3.4.

[OMG95d] OMG. *CORBA2.0/ Interoperability - Universal Networked Objects*, março de 1995. OMG TC Document 95.3.xx[REVISED 1.8jm].

[OMG95e] OMG. *Object Services Architecture*, janeiro de 1995. OMG Document 95.1.47 rev. 8.1.

[OMG95f] OMG. *Object Services RFP5*, julho de 1995. OMG TC Document 95.10.7.

[OMG95g] OMG. *ORB Portability Enhancement RFP*, junho de 1995. OMG TC Document 95.6.2.

[OMG96a] OMG. *CORBA Interoperability Revision*, maio de 1996. OMG TC Document 96.5.1.

[OMG96b] OMG. *Object Management Architecture*, agosto de 1996. OMG TC Document ab/96-08-01.

[OMG96c] OMG. *Object Startup Service*, fevereiro de 1996. OMG TC Document 96.2.3.

[ORB93] The Orbix Architecture. *C++ Report*, agosto de 1993.

[Pos94] PostModern Computing Technologies, Inc. *ORBeline User's Guide*, 1994.

[Pow93] M. L. Powell. *Objects, References, Identifiers and Equality - White Paper.* OMG, julho de 1993. OMG TC Document 93.7.5.

[Ray93] K. A. Raymond. Reference Model of Open Distributed Processing: a Tutorial. Na *International Conference on Open Distributed Processing - ICODP'93*, pp. 3 –14, setembro de 1993.

[SHAS95] D. C. Schmidt, T. Harrison, e E. Al-Shaer. Object Oriented Components for High-speed Network Programming. Na *USENIX Conference on Object-Oriented Technologies*, junho de 1995.

[SHF+93] J. Slonim, J. W. Hong, P. J. Finnigan, D. L. Erickson, N. Coburn, e M. A. Bauer. Does midware provide an adequate distributed application environment? Na *International Conference on Open Distributed Processing - ICODP'93*, Berlin - Alemanha, pp. 34 – 46, setembro de 1993.

[Sol94] Richard M. Soley. *Object Management Architecture Guide.* segunda edição, 1994.

[Sol95] Richard Soley. Transparências de palestra realizada na *International Conference on ODP - ICODP'95.* Brisbane - Austrália, fevereiro de 1995.

[Sun90] Sun Microsystems. *Network Programming Guide*, março de 1990.

[SUZ96] M. Steinder, A. Uszok, e G.K. Zieliński. *Distributed Platforms - A Framework for Inter-ORB Request Level Bridge Construction.* Chapman & Hall, pp. 86-99, primeira edição, fevereiro de 1996.

[SV95] D. C. Schmidt e S. Vinoski. Object Interconnections - Comparing Alternative Client-side Distributed Programming Techniques. *C++ Report*, 7(4), maio de 1995.

[Tan92] Andrew S. Tanembaum. *Modern Operating Systems.* 1992.

[TMdS+93] V. Tschammer, M. J. Mendes, W. L. de Souza, E. R. M. Madeira, e W. P. de Loyolla. Processamento Distribuído Aberto e o Modelo RM-ODP/ISO. No *11o. Simpósio Brasileiro de Redes de Computadores - SBRC 93*, Campinas - SP, pp. 93–109, maio de 1993.

[VA93] F. Vogt e C. Andrae. Middleware for Distributed Aplication Suport: ODP and/or CORBA. Na *International Conference on Open Distributed Processing-ICODP'93*, Berlin - Alemanha, pp. 405–410, setembro de 1993.

[Vin93] S. Vinoski. Distributed Object Computing With CORBA. *C++ Report*, julho/agosto de 1993.

[ZM96] N.M.S. Zuquello e E.R.M. Madeira. Obtendo Interoperabilidade entre ORBs. No *14o. Simpósio Brasileiro de Redes de Computadores - SBRC 96*, Fortaleza - CE, pp. 653–672, maio de 1996.

[ZM97] N.M.S. Zuquello e E.R.M. Madeira. A Mechanism to Provide Interoperability between ORBs with Relocation Transparency. No *Third International Symposium on Autonomous Decentralized Systems - ISADS'97*, Berlin - Alemanha, abril de 1997. (Aceito para publicação).