Duas Abordagens de Acesso a Objetos em Ambientes Distribuídos

Luiz Otávio Botelho Lento

Duas Abordagens de Acesso a Objetos em Ambientes Distribuídos

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Luiz Otávio Botelho Lento e aprovada pela Comissão Julgadora.

Campinas, 06 de Janeiro de 1995.

Edmundo Roberto Mauro Madeira
Orientador

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

Duas Abordagens de Acesso a Objetos em Ambientes Distribuídos[1]

Luiz Otávio Botelho Lento[2]

Departamento de Ciência da Computação
IMECC – UNICAMP

Banca Examinadora:

- Edmundo Roberto Mauro Madeira(Orientador)[3]
- Manuel de Jesus Mendes[4]
- Maria Beatriz Felgar Toledo[3]
- Nelson Castro Machado (Suplente)[3]

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

[2]O autor é Engenheiro de Sistemas formado pela Escola Naval

[3]Professor do Departamento de Ciência da Computação – IMECC - UNICAMP.

[4]Professor do Departamento de Computação e Automação – FEE – UNICAMP.

Dedicatória

À minha esposa, Silvana, à minha filha, Maria Fernanda, ao meu pai, Waldyr, à memória de minha mãe, Maria Heloísa,

Agradecimentos

Inicialmente à "Deus" pela força espiritual, e por iluminar os meus caminhos em mais uma etapa de vida.

À Marinha pela oportunidade a mim cedida.

Ao meu orientador por ter procurado sempre, da melhor forma possível, me apoiar e ajudar na conclusão desse trabalho.

A todos os colegas de graduação e pós-graduação, e professores do DCC que me auxiliaram com seus conselhos, com sua documentação e sobretudo com sua presença quando solicitados para consultas.

À secretaria do DCC que com boa vontade e paciência procurou sempre resolver as minhas solicitações de forma eficiente.

Aos companheiros de Marinha que participaram comigo dessa luta, procurando sempre acatar as solicitações por mim feitas nesse período.

Em particular, ao amigo Nuccio que sempre dividiu comigo os momentos difíceis e felizes nessa caminhada.

Resumo

O propósito dessa dissertação é a apresentação e a análise de formas de acesso a objetos em ambientes distribuídos com o objetivo de otimizar e facilitar ao cliente o acesso a serviços disponíveis em sistemas distribuídos.

Para o desenvolvimento de nosso trabalho consideramos as especificações referentes ao processamento distribuído aberto (ODP - Open Distributed Processing), modelos de interfaces para programas de aplicação(APIs - Application Program Interface) e plataformas comerciais de serviços distribuídos.

A primeira forma propõe especificar uma interface para programas de aplicação, baseada na metodologia orientada a objetos, oferecendo um conjunto de serviços distribuídos fornecidos por plataformas comerciais existentes, para o desenvolvimento, manutenção e execução de aplicações.

A segunda forma é baseada na arquitetura CORBA (Common Object Request Broker Architecture), no qual especificamos um esquema de acesso aos objetos desta arquitetura. Este esquema é baseado em repositórios de interfaces e implementações. Este esquema possibilita que as informações referentes a estes repositórios sejam obtidas de forma mais rápida e segura. Com isso espera-se que a CORBA proporcione melhores condições à camada *Middleware* de prover facilidades no processamento distribuído para as aplicações, em especial para as aplicações da plataforma *Multiware* que está sendo desenvolvida na UNICAMP.

Abstract

The purpose of this dissertation is to present and to analyse ways to access objects in distributed environments, as a means to optimize and to facilitate to the client the access to the distributed system services.

The specifications of the open distributed processing (ODP - Open Distributed Processing), the models of the application program interfaces (APIs - Application Program Interface) and commercial distributed plataforms were considered for the development of this work.

The first proposed specifies an application program interface , using an oriented-object methodology, to offer a group of services supplied by the existent commercial distributed plataforms to the development, running and management of distributed applications.

The second proposed is based on the CORBA (Common Object Request Broker Architecture) architecture, where we specified an access scheme to the objects of this environment. This scheme is based on implementation and interface repositories. This scheme provides that the repository information is got quickly and safely. Then the Middleware layer provides distributed processing facilities to the applications, specially to the applications of the Multiware plataform, that is being developed at UNICAMP.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O uso de computadores tornou-se uma rotina no dia a dia de empresas e usuários domésticos. O seu uso abrange todos os níveis de funções, desde o mais modesto sistema de processamento de informações até as grandes organizações de sistemas. Esse fato pode ser observado em função dos esforços que vem sendo realizados em relação aos aspectos de desenvolvimento e divulgação da utilização de potentes *workstations*. Em paralelo, pode-se citar também a evolução da tecnologia de comunicação através do desenvolvimento de novas técnicas para estruturas de rede (redes de fibra ótica), possibilitando assim, uma melhora sensível no processamento e transmissão da informação para usuários locais ou remotos.

O vultoso crescimento tecnológico, citado anteriormente, bem como a necessidade de se obter uma quantidade de informações imprecindíveis à realização de diversos serviços, tornou necessário a criação de ambientes de processamento de informações distribuídas. Esses ambientes devem oferecer e viabilizar o acesso de informações seja ele local ou remoto, de uma forma transparente e homogênea para o usuário. Junto com essa metodologia começaram a aparecer problemas de distribução e comunicação, dificultando a interoperabilidade e interconectividade na integração de sistemas distribuídos[CD88, Tan92].

Para se criar um sistema distribuído, integrado e de grande escala, deve-se inicialmente observar a interconectividade entre sistemas. Esse conceito é baseado na possibilidade de dois ou mais recursos de computação trocarem mensagens entre si, garantindo assim somente a comunicação, mas não a cooperação entre esses recursos. A forma mais indicada para estabelecer interconectividade entre recursos é o modelo cliente/servidor, que organiza os sistemas distribuídos como um número de servidores posicionados distribuidamente e oferecem um conjunto de serviços para clientes através da rede. Os mecanismos que possibilitam a comunicação entre processos como, RPC[1], podem prover o acesso pelos clientes de serviços oferecidos pelos servidores. O segundo objetivo que deve ser observado é a interoperabilidade. Tal conceito baseia-se na possibilidade de dois ou mais recursos

[1]Remote Procedure Call

interagirem para a execução de um determinado serviço. Ferramentas simples como o RPC, já citado anteriormente, ajudam na execução da interoperabilidade. Formas avançadas de interoperabilidade necessitam não somente do entendimento amplo dos sistemas como um todo, mas também dos tipos de dados que compartilham essa interoperabilidade. Essa forma de interoperabilidade ampla possibilita a integração de informações heterogêneas, como aplicações multimídia, e suporta o armazenamento em repositórios integrados.

Embora sistemas distribuídos baseados no paradigma cliente/servidor suportem a interoperabilidade entre sistemas, experiências demonstram que somente esse fator não é suficinte para ativá-la entre sistemas globais, nos quais problemas como ambientes heterogêneos, gerenciamento de configuração e monitoramento da rede são críticos. A metodologia baseada na orientação a objetos é uma solução, onde a modelagem de um sistema distribuído é feita através de uma coleção de objetos interagindo entre si, proporcionando assim a melhor forma para a integração no processamento das informações distribuídas em um sistema[TLX90, BGHS91]. Essa capacidade da metodologia citada ateriormente, onde objetos formam um modelo natural para sistemas distribuídos, é explicada porque esses possuem propriedades (encapsulamento, abstração,etc) que possibilitam a interação somente através de troca de mensagens, satisfazendo o paradigma cliente/servidor utilizado em sistemas distribuídos. O uso de objetos também acomoda, de forma adequada, a heterogeneidade e a autonomia em sistemas distribuídos de grande escala. A heterogeneidade é aceita em função das mensagens enviadas aos componentes distribuídos do sistema dependerem somente de suas interfaces, não se importando com sua estrutura interna. A autonomia é obtida devido aos componentes do sistema poderem ser alterados de forma transparente e independente, sem que suas interfaces sejam alteradas[PS85, CD88, Tan92, Mul89].

Com o decorrer do tempo, a necessidade de se obter as informações de forma rápida e precisa vem se tornando uma realidade. Baseada nessa necessidade e utilizando-se de recursos computacionais modernos disponíveis, em conjunto com o paradigma cliente/servidor, a metodologia orientada a objetos e de especificações que tentam reduzir os problemas referentes à distribuição e comunicação em sistemas distribuídos, estamos propondo duas formas de acessar e utilizar serviços, armazenados em objetos e oferecidos de forma distribuída. Tanto uma como a outra abordagem dão ao cliente tanto a precisão quanto a rapidez na obtenção das informações desejadas [MWE93].

Inicialmente especificamos uma interface para programas de aplicação (API[2]), que recebeu o nome de MID/API[3], e tem a finalidade de prover a portabilidade às aplicações, bem como oferecer um conjunto de serviços de forma homogênea e transparente para o desenvolvimento, execução e gerenciamento de aplicações distribuídas. Esses serviços de forma geral, são provenientes de plataformas comerciais (plataformas midware), como

[2]Application Program Interface

[3]Midware Application Program Interface

DCE[4] e ANSA[5], onde o cliente que não possui conhecimento pode acionar a plataforma que desejar para a execução do serviço. No capítulo três estamos especificando a estrutura dessa API, através de suas camadas, do cliente e provedora de serviços, citando alguns dos seus serviços mais importantes que podem ser invocados. Essa API baseia-se na metodologia orientada a objetos, é biderecional e utiliza do paradigma cliente/servidor. Além da especificação da MID/API, faz-se também uma breve descrição do que vem a ser camada *Middleware*, plataformas *midware* e das plataformas DCE e ANSA, como também uma breve comparação da DCE perante o processamento distribuído aberto. O processamento distribuído aberto é abordado no capítulo inicial, no qual descrevemos um acompanhamento do processamento distribuído e também do modelo de referência ODP[6] (RM-ODP) que tem por finalidade estabelecer uma estrutura padrão para o processamento distribuído aberto.

Levando-se em consideração que a CORBA[7] é uma das soluções para os problemas de distribuição e comunicação em sistemas distribídos, especificamos um esquema de acesso que tem a finalidade de acessar os objetos armazenados nos repositórios de interfaces e implementações de uma forma mais simples e eficiente. A CORBA foi a escolhida para auxiliar no desenvolvimento da camada *middleware* da plataforma *multiware*. Esta plataforma provê uma estrutura de *hardware* e *software* que possibilita o trabalho cooperativo em ambientes de serviços abertos, baseado na maioria dos conceitos do modelo de referência ODP. Essa plataforma é composta de três camadas: **Software/Hardware básica**, **Middleware** e **Groupware** (vide figura1.1). A camada *Software/Hardware* básica é composta por um sistema operacional e protocolos de comunicação. A camada *Middleware* é responsável por prover facilidades de processamento distribuído para a camada *Groupware* e para aplicações. Essa camada é composta por duas subcamadas: **Processamento Multimídia** - possibilita a troca de informações multimídia em tempo real, com uma qualidade de serviço especificada; **ODP** - composta pelos sistemas comerciais ANSA, DCE e ORB, e nível ODP que provê funcionalidades para esses sistemas. A última camada é a *Groupware*, que provê funcionalidades através de classes diferentes de aplicações, como CSCW[8], nos quais as aplicações podem entrar ou deixar o sistema dinamicamente sem a degradação da operação, DAI[9] e outras[MM94, LEM+94, EWM94]. Através do esquema de acesso acima citado, espera-se que a CORBA tenha capacidade de proporcionar à camada *Middleware* melhores condições desta oferecer à camada *Groupware* e aplicações facilidades para o processamento distribuído.

Para especificarmos o esquema de acesso, realizamos um estudo detalhado da CORBA

[4]Distributed Computing Environment
[5]Advanced Network Systems Architecture
[6]Open Distributed Processing
[7]Common Object Request Broker Architecture
[8]Computer Support Cooperative Work
[9]Distributed Artificial Intelligence

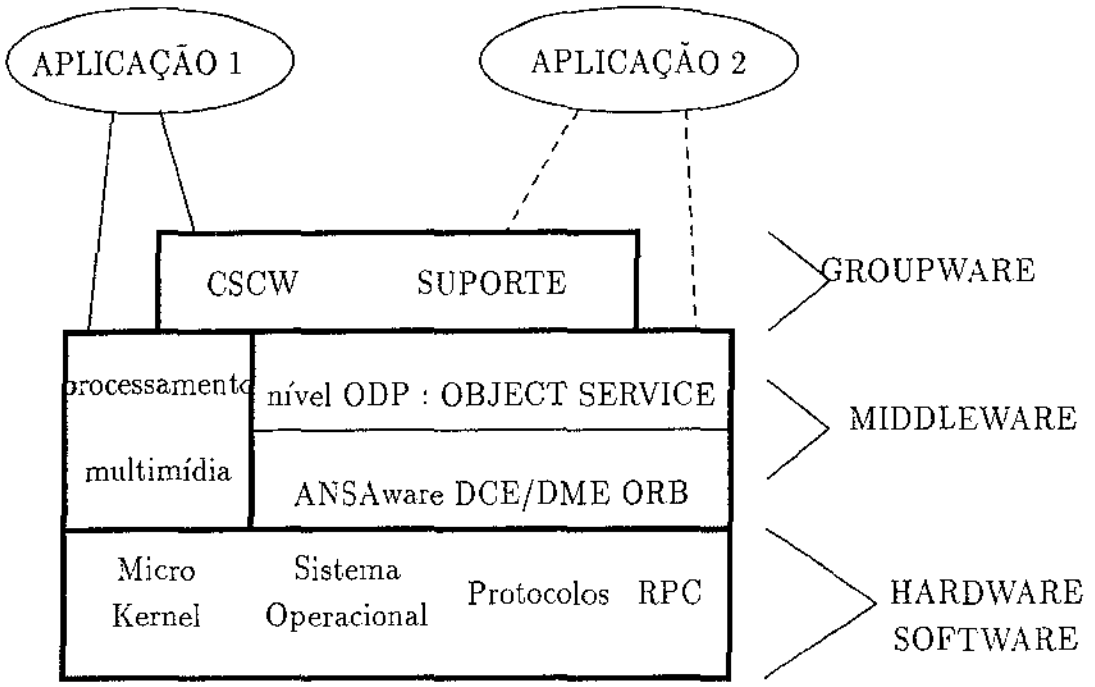


Figura 1.1: Plataforma Multiware

e fizemos no capítulo quatro uma breve descrição de sua estrutura, dos seus principais componentes e serviços que são oferecidos. Realizamos também um estudo comparativo entre a CORBA e o modelo de referência ODP, procurando descrever os pontos comuns entre as duas abordagens. Por último, realizamos uma análise comparativa entre as duas formas de acessar objetos, apresentados nesta dissertação, verificando também as vantagens e desvantagens da utilização da CORBA em relação à API na plataforma *Multiware*.

No capítulo 5, especificamos o nosso esquema de acesso que tem a finalidade de reduzir os problemas bem como otimizar o tempo de acesso ao repositório de interfaces, tornando este acesso mais rápido e seguro. Ele é composto por dois objetos (acesso e repositório), pelos arquivos de acesso e pelos repositórios de interfaces e implementações. O objetivo do objeto de acesso é gerenciar as solicitações do ORB para o repositório de interfaces, e buscar a informação adicional necessária à localização da interface. O objeto repositório tem a finalidade de obter as informações do repositório de interfaces, como também interagir com o repositório de implementações à fim de obter informações necessárias à localização e ativação da implementação do objeto. O arquivo de acesso tem todas as informações sobre identificadores de tipos de interfaces, endereços e identificadores de repositórios de interface, possibilitando o objeto de acesso obter todas as informações da interface desejada. Este esquema é acionado quando o objeto de acesso recebe do ORB a solicitação da execução de um determinado serviço, junto com seus parâmetros. Esse aciona o arquivo

de acesso obtendo as informações necessárias à invocação do objeto repositório. O objeto repositório interage com o repositório de interfaces, invocando seus métodos, de forma que possa obter a informação necessária à conclusão do serviço. A resposta retorna até o objeto de acesso, que a passa para o ORB. Esse capítulo possui ainda duas seções: na primeira realizamos uma comparação do esquema de acesso com a forma como são acessadas as informações do repositório de interfaces da ORBeline; na segunda estabelecemos um paralelo entre os objetos e as funcionalidades do esquema de acesso com o modelo de referência ODP.

Com a finalidade de validarmos o nosso esquema de acesso, no capítulo seis, especificamos um protótipo composto pelos objetos repositório e de acesso, o arquivo de acesso, dois repositórios de interfaces, um repositório de implementações e três stubs. Utilizamos a ferramenta de comunicação RPC, o sistema de arquivo da rede da SUN e de plataformas SUN para implementar o nosso projeto. O repositório de interfaces está estruturado em uma árvore B, onde cada campo do nó corresponde a um tipo de interfaces, com um apontador para um arquivo, que tem o nome da interface, com o nome da operação que ela executa. O arquivo de acesso é um arquivo composto por *strings*, onde cada uma delas possui o tipo da interface, o nome do repositório onde se encontra a informação e a máquina correspondente ao repositório. O repositório de implementações possui o nome da operação, seu endereço na rede e a forma como está armazenada.Esse protótipo baseia-se no paradigma cliente/servidor, onde utiliza-se da ferramenta RPC para realizar as suas solicitações de serviço, e do NFS da SUN como sua forma de armazenamento.

O capítulo referente à conclusão apresenta as contribuições dadas pelo nosso trabalho, bem como os trabalhos futuros que podem ser desenvolvidos utilizando-se dos meios aqui analisados.

Capítulo 2

Processamento Distribuído Aberto

2.1 Introdução

A informação é uma das necessidades básicas da sociedade, pois implica no conhecimento que se pode obter em diversas fontes distintas, necessárias ao desempenho de qualquer tarefa. A localização dessas informações baseia-se no posicionamento dos recursos de armazenamento e processamento disponíveis em um sistema. Com a disponibilidade das LANs[1] ou WANs[2][Tan89, WJJJ89, Bla89], o acesso a essas informações tornaram-se mais viáveis. A comunicação entre sistemas ou usuários, passou a ser além dos limites de uma máquina, sala, prédio, ou até mesmo de um país. O volume de informações disponíveis aumentou sensivelmente, facilitando assim a realização de qualquer tarefa[VMW+93].

Devido a esses fatores, passou-se a ter um rápido crescimento do processamento distribuído, levando a criação de uma estrutura para padronizar todo esse processamento. A principal finalidade dessa padronização é possibilitar que procedimentos utilizem recursos (*hardware, software*) heterogêneos e distribuídos em um sistema, para que acessem e processem as informações disponíveis de uma forma consistente e confiável. Logo, para que essa padronização seja feita da melhor forma possível, a ISO[3] apresentou o **Modelo de Referência ODP** (RM-ODP[4])[VMW+93, ISO94a].

Nesse capítulo são descritos conceitos sobre sistemas distribuídos no geral (uma leve descrição sobre sistemas distribuídos tradicionais), e especificamente sobre sistemas abertos. Faz-se também uma descrição sucinta do que vem a ser o modelo de referência ODP.

[1] Local Area Network
[2] Wide Area Network
[3] International Standardization Organization
[4] Open Distributed Processing

2.2 Sistemas Distribuídos

Com a necessidade de se utilizar uma grande quantidade de informações na solução de uma determinada tarefa, os sistemas de processamento de informação tiveram um grande crescimento, expandindo-se além dos limites de uma máquina (estação de trabalho). Para satisfazer essa nova visão de processamento, os sistemas passaram a utilizar de recursos disponíveis, como estrutura de redes, para obter as informações necessárias na solução de tarefas. Estes sistemas de processamento de informações receberam o nome de sistemas distribuídos, definidos como um conjunto de computadores espalhados fisicamente, capazes de realizar operações individualmente, ligados entre si através de uma rede e comunicando-se por trocas de mensagens[PS85, Tan92, Mul89].

Sistemas distribuídos utilizam como princípio básico o compartilhamento de recursos, onde vários desses recursos, disponíveis na rede, são utilizados na conclusão de um determinado serviço. Pode-se citar as seguintes vantagens na utilização desses sistemas:

- **Extensibilidade** → o sistema pode ser aumentado de acordo com a necessidade do crescimento de serviços sem realizar qualquer substituição de elementos já pertencentes à antiga configuração.
- **Confiabilidade** → a alta confiabilidade é obtida através da replicação das informações em vários *sites*.
- **Disponibilidade e Tolerância a Falhas** → a alta disponibilidade e tolerância a falhas ocorre em função dos sistemas não deixarem de cumprir total ou parcialmente um serviço solicitado.
- **Desempenho** → melhora o desempenho em função das operações realizadas em paralelo de diferentes aplicações, ou de partes de uma dada aplicação.
- **Gerenciamento** → gerenciamento descentralizado, onde os donos de seus recursos podem gerenciá-los como quiserem.
- **Compartilhamento de Recursos** → todos os computadores do sistema compartilham de recursos e aplicações disponíveis.

2.2.1 Arquitetura

A arquitetura de sistemas distribuídos identifica seus principais componentes de *hardware* e *software*, devendo-se levar em conisderação alguns fatores durante a sua composição:

- **Modularidade** → a arquitetura é composta de módulos, devendo-se observar sempre as relações existentes entre eles, de forma que possa ser aumentada sem afetar a estrutura existente.

- **Equipamento** → prever os computadores que serão utilizados bem como as suas posições na rede.
- **Integração de Sistemas** → a arquitetura deve suportar a possível integração com outros sistemas de diferentes arquiteturas e diferentes recursos.
- **Consistência** → a manutenção da consistência e integração do sistema deve sempre ser observada, apesar das alterações realizadas em sua estrutura.
- **Aplicações** → definir quais as aplicações serão executadas, como também prover ferramentas no desenvolvimento de aplicações.

2.2.2 Comunicação

A comunicação entre máquinas pode ser feita de duas formas. A primeira utiliza mecanismos *broadcast*, no qual todos os processadores do sistema podem comunicar-se diretamente com todos os outros através do uso de um ou mais canais de comunicação, compartilhados pelos próprios processadores.

A segunda forma de realizar a comunicação entre processadores utiliza mecanismos do tipo ponto a ponto, no qual existe um canal exclusivo de comunicação entre cada par de processadores. Esta exclusividade elimina o problema das colisões existentes no *broadcast*, mas causa por outro lado a necessidade de mensagens serem enviadas ao longo de rotas que passam por outros processadores. A figura 2.1 mostra um sistema distribuído com os dois tipos de comunicação.

2.2.3 Sistemas Distribuídos Tradicionais

Um sistema distribuído tradicional é um sistema que de uma forma geral possui um número limitado de componentes (configuração fixa), pertencentes a uma única organização (ambiente homogêneo) que controla o projeto (controle central), instalação, operação e gerenciamento do sistema. O fator distribuição resulta exclusivamente das decisões de projeto, sendo os seus elementos fortemente acoplados através de mecanismos comuns com limitada autonomia.

2.3 Sistemas Distribuídos Abertos

Sistemas abertos surgiram em função do avanço da tecnologia de comunicação e das necessidades do usuário. São utilizados em sistemas computacionais e de comunicação com o objetivo de usufruir de recursos e serviços externos, oferecer recursos e serviços locais para

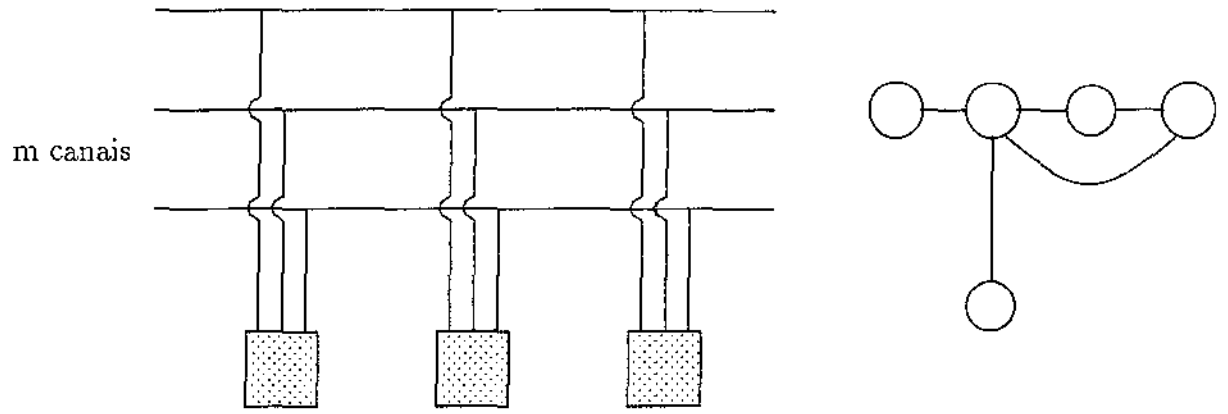


Figura 2.1: Tipos de Comunicação

a utilização externa e participar de atividades comuns e aplicações cooperativas[VMW+93, ISO94a, SLC93].

Um sistema aberto é um sistema capaz de interagir com outros em um ambiente. Tem como suas principais características os seguintes ítens:

- **Ilimitado** → o sistema aberto é livre quanto ao fator geográfico e organizacional, podendo ter a sua configuração aumentada.
- **Livre Acesso** → admite que qualquer tipo de usuário, componente de aplicação ou aplicação acesse o sistema desde que tenha autorização para tal.
- **Heterogêneo** → cada usuário decide sobre suas próprias características de existência, projeto, implementação, uso e gerenciamento de seus componentes.
- **Autônomo** → cada entidade será submetida às características de seu ambiente local. Possui liberdade para determinar suas propriedades, seu comportamento e a evolução de seus componentes.
- **Descentralizado** → possui uma funcionalidade física e lógica dos elementos gerenciados localmente, sem qualquer influência ou controle externo.

Em ambientes abertos não é possível obter-se uma visão completa do sistema, ou mesmo manter-se uma influência total sobre ele. Cada **visão é parcial** de acordo com

o observador, dando a esta um valor local e restrito. Como esta visão e o **controle** do sistema são **parciais**, fatos, eventos e atividades fora da área em questão podem não serem observados. Esta parcialidade tanto no controle quanto na visão é considerada, tendo em vista que cada informação mantém-se válida somente pelo período de tempo em que estiver sob a visão ou controle de um elemento.

A **integridade** do sistema deve ser mantida através de mecanismos de segurança, que tem o propósito de reconhecer e autorizar as entidades cadastradas para que possam utilizar todos os recursos disponíveis. A **privacidade** de cada elemento do sistema é mantida através do estabelecimento de limites quanto à observação e influência externa, garantindo assim a integridade dos recursos locais.

Sistemas abertos suportam a cooperação em ambientes que corresponde à junção de esforços e operações na obtenção de propósitos comuns, devendo operar de uma forma logicamente integrada, consistente e inteligível aos seus componentes. Essa cooperação compreenderá no estabelecimento de um contexto e de controles globais, bem como no estabelecimento de princípios homogêneos e de relações coordenadas entre os seus componentes. A cooperação em ambientes abertos pode ser estabelecida de maneira permanente ou através de federações limitadas no desenvolvimento de ações coordenadas na obtenção de metas comuns.

A arquitetura de sistemas abertos deve possuir tanto as características locais quanto as globais dos seus componentes. Alguns aspectos podem descrever essa arquitetura:

- **Ambiente Operacional Local** → corresponde aos elementos técnicos locais (estações de trabalho, sistemas operacionais, linguagens e bibliotecas) e aos serviços, ferramentas e características de suporte a aplicações (serviços de processamento e transmissão de arquivos, qualidade de serviços, capacidade de recursos, etc).

- **Ambiente de Aplicações Locais** → refere-se à estrutura local para projeto, implementação, operação, gerenciamento e administração de aplicações, além das considerações de confiabilidade, disponibilidade e desempenho operacional.

- **Terminologia Local e Global** → corresponde à compreensão de termos, nomes, serviços e protocolos pelos componentes e interfaces.

- **Mecanismos e Formas de Cooperação** → é o conjunto de metodologias para o desenvolvimento de negociações e estabelecimentos de concordâncias sobre atribuições, recursos comuns, atividades e decisões conjuntas e níveis de autoridade.

O trabalho cooperativo, exercido nos sistemas abertos, leva a formação de grupos cujos elementos possuem alguns atributos técnicos, comportamentos, serviços e áreas de influência em comum. A junção de grupos forma um domínio. A inter-relação entre os

componentes de grupos, ou entre grupos ou domínios deve ser feita de uma forma cooperativa e coerente, através de uma infraestrutura para interação, mantendo-se a autonomia e privacidade dos elementos.

2.4 RM-ODP

O **Modelo de Referênncia ODP** foi criado em função do rápido crescimento do processamento distribuído, e tem a finalidade de prover uma estrutura para padronizar o processamento distribuído aberto. O modelo de referência ODP tenta englobar a heterogeneidade de ambientes, a distribuição física e lógica de equipamentos de computação e de aplicações, a interoperabilidade entre sistemas heterogêneos e a portabilidade das aplicações nos diversos ambientes. Sendo assim, este modelo deve possuir os seguintes requisitos básicos[K.R93, ISO94a]:

- Prover um modelo consistente para o desenvolvimento em separado de diversos padrões.
- Definir áreas que necessitem de padrões ODP e o relacionamento entre eles.
- Descrever um conjunto básico de ferramentas a serem utilizadas na definição de padrões.
- Prover um conjunto consistente de conceitos e terminologias comuns.

O modelo ODP não é simplesmente uma ferramenta descritiva, pois estabelece uma base e uma estrutura para projetos e implementações de sistema. Os seus padrões especificam a comunicação e a coordenação da informação distribuída para uma determinada empresa. Esses padrões podem ser divididos nas seguintes categorias:

- Um modelo de referência ODP geral definindo os conceitos e os identificadores de funções comuns.
- Modelos de referência específicos que cobrem tipos especiais de empresa, usando conceitos e funções comuns do RM-ODP, definindo detalhes conceituais e funções específicas.
- Padrões para a realização de funções comuns identificadas no RM-ODP.
- Padrões para a realização de funções específicas necessárias a determinadas aplicações.

O modelo de referência ODP baseia-se no conceito de orientação a objetos, tendo em vista que a utilização de objetos é a forma mais indicada para a especificação ODP. Seus objetos possuem propriedades como a abstração, encapsulamento, distribuição, interação, falha, replicação, segurança, hierarquias, hierarquias de herança e restrições de ambiente. O objeto modelado de um forma geral pode ser aplicado em várias situações, pois engloba todas as propriedades já citadas e pode ser implementado em várias linguagens. A modelagem utiliza-se de alguns conceitos básicos, como ações, comportamentos e assinaturas de interfaces.

A utilização dessa metodologia foi influenciada por alguns fatores básicos:

- **Estrutura e Especificação** - Inicialmente a utilização de objetos é adequada aos propósitos de estruturação e especificação, pois eles embutem idéias de modularidade e abstração de dados.
- **Interação** - A interação entre objetos também é outro fator, pois são envolvidos especificamente os seus comportamentos não afetando suas estruturas internas.
- **Propriedades de Distribuição** - O último fator preponderante é que os objetos ajudam na captura das propriedades essenciais de distribuição, proporcionam a separação através da identificação de objetos distintos, isolamento e autonomia através do conceito de encapsulamento.

2.4.1 Pontos de Vista

A especificação completa de um sistema distribuído aberto envolve uma gama enorme de informações. Para abranger todos os aspectos referentes ao sistema em uma simples especificação torna-se praticamente impossível. Para resolver esse problema, o modelo de referência ODP utiliza-se de uma metodologia que estabelece a separação de áreas de interesse através da identificação de cinco pontos de vista. Cada um desses pontos de vista está associado a uma linguagem, no qual são expressa as regras referentes a uma determinada área de influência[ISO94a, ISO93, ISO94b]. O exemplo utilizado para descrever os pontos de vista foi retirado de [K.R93].

Empresa

O ponto de vista empresa possibilita aos seus membros saber onde e como o sistema de processamento de informações está colocado dentro de uma empresa, bem como os objetivos que deseja-se alcançar. Este ponto de vista não está restrito a uma empresa como um todo, ele pode ver partes de uma empresa ou várias delas, referente às necessidades comerciais que justificam e governam o projeto do sistema. As políticas organizacional e social podem ser definidas através de:

- **agentes** - são objetos ativos que iniciam ações de desempenho (ex: gerentes de banco, clientes, caixa).
- **artefatos** - são objetos passivos que não iniciam uma ação (ex: dinheiro, conta concorrente).
- **comunidades** - é um agrupamento de agentes e artefatos (ex: uma agência de um determinado banco).
- **regras de agentes, artefatos e comunidades** - essas regras são expressas em forma de políticas:
  - **permissão** - o que pode ser feito (ex: permissão para abertura de contas).
  - **proibição** - o que não pode ser feito (ex: clientes não devem retirar mais do que 500 reais por dia).
  - **obrigação** - o que deve ser feito (ex: o gerente de banco deve avisar o cliente de alterações de regras bancárias).

A linguagem do ponto de vista de empresa está baseada em ações que podem alterar políticas, tal como a criação de uma obrigação ou a revogação de uma permissão. Como exemplo pode-se citar a mudança da taxa de juros de um banco, ela é uma ação deste tipo, pois cria para o gerente do banco, obrigações referentes aos clientes. Portanto, a obtenção do saldo não é uma ação, pois obrigações, permissões e proibições não são afetadas. Logo, a especificação de empresa de um banco não inclui essa funcionalidade, sendo identificada no modelo computacional.

Informação

O ponto de vista de informação modela as estruturas e o fluxo de informação, além das regras e restrições que devem governar a sua manipulação. Esse ponto de vista é definido através de esquemas, relações e regras de integridade. O esquema descreve o estado e a estrutura de um objeto (ex: a conta bancária consiste do saldo mais as retiradas ou depósitos do dia). O esquema pode ser classificado de três formas:

- **estático** - captura o estado e a estrutura do objeto de uma instância (por exemplo o saldo da conta ao fim do dia é igual a 50 reais).
- **invariante** - restringe o estado e a estrutura de um objeto por todo o tempo (a retirada do dia é sempre menor que 500 reais).
- **dinâmica** - define um mudança no estado e na estrutura de um objeto (retirada de x reais ou o depósito de x reais). O esquema dinâmico é sempre restrito aos esquemas invariantes.

A relação descreve as associações entre objetos, por exemplo, associa o cliente com a conta. A regra de integridade é uma assertiva sobre o esquema, por exemplo, todo cliente deve ter ao menos uma conta e toda a conta deve ter ao menos um cliente.

A linguagem de informação é usada para definir a semântica de informação de um sistema ODP, no qual é verificado o sentido que um ser humano atribui aos dados armazenados ou trocados entre componentes do sistema, bem como as necessidades de processamento de informações neste sistema.

Computacional

O ponto de vista computacional está interessado na descrição do sistema como um conjunto de objetos interagindo, sem levar em consideração os detalhes de como é feita esta interação, provendo um conjunto de funções para aplicações suportadas por um infraestrutura. Esta infraestrutura é um ambiente criado por um núcleo e um conjunto de funções comuns. Este ponto de vista descreve os algorítimos e os fluxos de dados que provêem a função do sistema distribuído, e também é usado para especificar a funcionalidade de uma aplicação ODP distribuída de maneira transparente, baseada em objetos (objetos computacionais) que:

- encapsulam dados e seu comportamento.
- oferecem interfaces para a interação com outros objetos.
- podem oferecer múltiplas interfaces.

Os objetos na especificação computacional podem ser aplicações ou parte da infraestrutura ODP (ex: trader). O RM-ODP provê três formas de interação entre objetos, operacional, orientada a fluxo e transacional. As interfaces operacionais provêem um modelo cliente/servidor para computação distribuída, onde clientes invocam operações nas interfaces dos objetos servidores. Estas interfaces consistem de operações com parâmetros, terminações, requisições e resultados, e possuem atributos de transparências, qualidade de serviço e regras aplicadas às operações da interface, que indicam se o objeto comporta-se como um cliente ou servidor.

Interfaces de fluxo provêem fluxos contínuos de informação entre objetos clientes e servidores. Clientes conectam-se às interfaces de fluxo do servidor ou vice-versa, e vários fluxos podem ser agrupados em uma interface (ex: fluxo de áudio, vídeo). Interfaces de fluxo são incluídas no RM-ODP para fornecer aplicações multimídia e telecomunicações.

Interfaces transacionais são utilizadas quando as interações são sinais. Sinal é definido como uma interação entre um objeto computacional e um objeto *binding* (é um objeto que suporta o *binding* entre um conjunto de objetos computacionais). As seguintes propriedades transacionais podem ser observadas [K.R93]:

- **Visibilidade** - o grau para o qual os efeitos intermediários de uma operação são visíveis para outras opearções.
- **Consistência** - deve ser verificada na terminação de uma operação.
- **Recuperação** - é o estado após a falha de uma operação.
- **Permanência** - as conseqüências da falha da operação em operações completadas.
- **Dependência** - as regras que determinam se a operação teve ou não sucesso, baseada em operações que relatam o sucesso ou falha.

O ponto de vista computacional também define as ações que são possíveis dentro de um objeto computacional. Essas ações podem ser compostas seqüencialmente ou em paralelo. Se composta em paralelo, as atividades podem ser dependentes (uma atividade pode ser bifurcada e deve ser subseqüentemente junta em algum ponto de sincronismo) ou independentes (uma atividade é espalhada e não pode ser junta). As ações são descritas a seguir:

- criar e destruir um objeto
- criar e destruir uma interface
- trading para uma interface
- binding para uma interface
- ler e escrever o estado de um objeto
- invocar a operação em uma interface operacional
- controlar as formas em que os fluxos em várias interfaces são transformados e interligados.

A linguagem computacional é responsável por:

- Mascarar o grau de distribuição das aplicações de seus programadores.
- Não permitir o uso de *softwares* projetados para sistemas centralizados em sistemas distribuídos, porém, admite o encapsulamento dessas aplicações como componentes de uma applicação distribuída de grande porte.

Engenharia

O ponto de vista de engenharia é usado para descrever os aspectos de distribuição num sistema ODP, definindo um modelo para a infraestrutura dos sistemas distribuídos. Esse ponto de vista não está preocupado com a semântica das aplicações ODP, exceto na determinação das necessidades para a distribuição e suas transparências.

Os principais elementos que definem o ponto de vista de engenharia são os objetos divididos em objetos de engenharia básico (BEO) e objetos de infraestrutura (são responsáveis em dar suporte à distribuição do sistema). O BEO (corresponde ao objeto computacional na especificação computacional, que constitui a aplicação propriamente dita) é um objeto de engenharia que necessita do suporte da infraestrutura de distribuição. Este objeto pode ser visto como uma unidade de execução do modelo, sendo que este pode conter vários *threads* simultaneamente, ou seja, *multithread*. Além disso, são definidos canais que correspondem a *bindings* na especificação computacional. O canal provê mecanismos de comunicação que contém ou controla as funções de transparência necessárias aos objetos de engenharia básicos. A figura 2.2 mostra um canal, composto de objetos *stubs*, *binders* e protocolos.

- **Stub** - É um objeto com funções de conversão, isto é, realiza o empacotamento (*marshalling*) e o desempacotamento (*unmarshalling*) de dados para suportar a interação transparente de acesso entre objetos de engenharia. Com isso, o *stub* contribui para possibilitar a transparência de distribuição através da adaptação de trocas de informações. *Stubs* interconectados com diferentes sintaxes de transparências, é utilizado um interceptador que transforma os dados de uma sintaxe para outra.

- **Protocolo** - Tem a finalidade de permitir a interação entre objetos de engenharia em outros nós, através da comunicação com outros objetos protocolos. Quando os objetos protocolos em um canal são de diferentes tipos, é utilizado um interceptador para realizar a conversão entre eles.

- **Binder** - É um objeto que tem a finalidade de gerenciar a integração fim a fim, bem como a qualidade de serviço no canal. Este objeto pode interagir com objetos fora do canal para obter informações referentes às localizações de dados.

O ponto de vista de engenharia prescreve a estrutura do sistema ODP, através de unidades básicas de engenharia (vide figura 2.3 que mostra a estrutura de um nó). Estas unidades estão descritas abaixo de forma sucinta:

- **cluster** - O *cluster* é um conjunto de objetos de engenharia básico, associados a um gerente de *cluster*, formando uma unidade que tem o propósito de desativação, *checkpoint*, reativação, recuperação e migração. O *cluster* consiste por si só numa

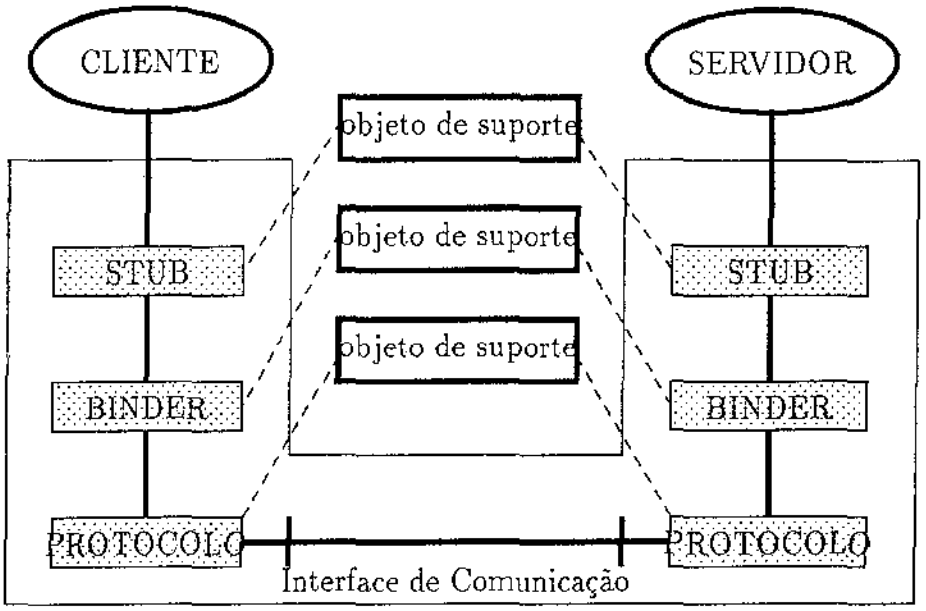


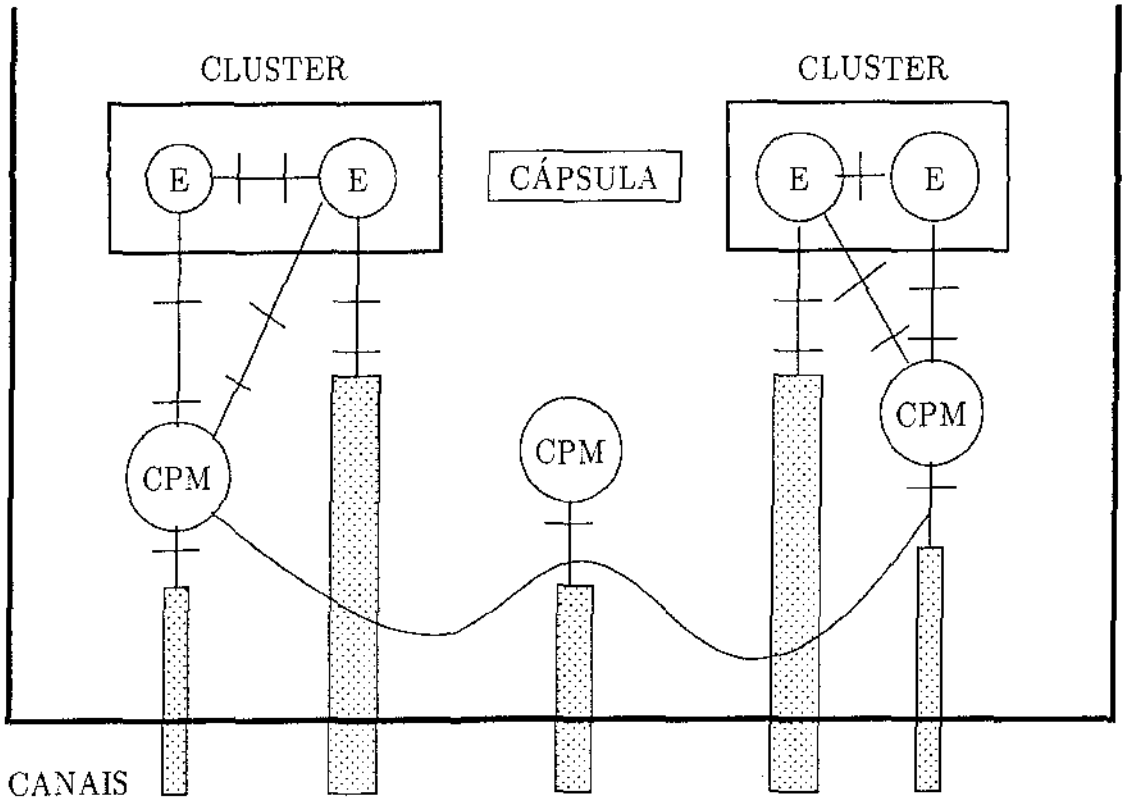
Figura 2.2: Estrutura de um canal

parte da aplicação posicionada em um segmento de memória. A comunicação entre objetos pertencentes ao mesmo *cluster* é feita diretamente, porém, canais são utilizados quando essa comunicação é feita em diferentes *clusters*.

- **cápsula** - é uma unidade de alocação de recursos, contida em um nó, composta por um conjunto de *clusters*, um gerenciador de *cluster* para cada *cluster*, um gerenciador de cápsula e partes do canal que conectam suas interfaces. Se o *cluster* é parte de uma aplicação, a cápsula (processo) é uma aplicação que desempenhará um serviço ou um conjunto de serviços.

- **núcleo** - pode ser considerado como um sistema operacional que dá suporte ao ODP. Este objeto provê um conjunto de interfaces de recursos para cada cápsula do nó. Estas interfaces consistem de: operações de escalonamento para uso de *threads*; operações de controle de comunicação para controlar canais; operações para realizar chamadas remotas via canais; operações de *binding* para estabelecer caminhos de comunicação na criação de canais.

- **nó** - poderia ser considerado como um processador do sistema. O nó pode também ser visto como uma estação de trabalho, no qual residem clusters e cápsulas. Um nó é administrado pelo núcleo, provendo a função de gerenciamento de um nó, no qual cada um deste possui um espaço de endereçamento distinto.



E : objeto de engenharia básico

CLM : gerente de cluster

CPM : gerente de cápsula

Figura 2.3: Estrutura de um nó

Tecnológico

O ponto de vista tecnológico descreve a implementação do sistema e a informação necessária para realizar testes. Os principais usuários são aqui responsáveis pela configuração, instalação e manutenção do *hardware* e *software*.

2.4.2 Funções ODP

O modelo de referência ODP identifica uma coleção de funções que podem ser utilizadas de forma que assegurem a execução eficaz e sem problemas (distribuição) do processamento distribuído. As funções possibilitam a construção de sistemas, através de blocos, por elas criadas, reaproveitando os componentes já existentes. A especificação de uma função descreve os objetos e as interfaces que coletivamente provêm uma função. As funções são classificadas da seguinte forma[ISO94a, K.R93]:

- **Repositório** → O RM-ODP define um número de funções repositórios, que são responsáveis pela manutenção de classes específicas de informação em uma base de dados.

  - **Armazenamento** - Esta função tem a finalidade de gerenciar um repositório de dados (base de dados). Este repositório possui interfaces que possibilitam o acesso aos seus dados, interfaces *containers*. Estas interfaces suportam interações para a: obtenção de cópias de dados associados a interface; modificação do dado associado a interface; deleção da interface e o dado a ela associado.
  - **Repositório de Relacionamento** - Esta função tem a finalidade de gerenciar o relacionamento entre objetos e interfaces contidos em um repositório. Esta função inclui a: instanciação de novos relacionamentos através de um *template* origem; consulta para verificar quais os relacionamentos que satisfazem uma determinada característica; deleção de relacionamentos.
  - **Repositório de Tipos** - Na maioria dos sistemas de computadores, as definições de tipo não são mantidas explicitamente dentro do sistema. De fato, os tipos são documentados em manuais ou definidos em algum local pré-definido. Os sistemas ODP devem tornar acessíveis estas informações, suportando a verificação de tipos durante o *trading* e o *binding* de interfaces. Com isso, esta função tem a finalidade de gerenciar o repositório de tipos. O repositório de tipos é um registro para as definições de tipo, particularmente para tipos de interface, mantendo a hierarquia de tipos e outros parentescos entre esses tipos.
  - **Trading** - seu propósito é suportar **bindings** dinamicamente através da importação e exportação de serviços que são descobertos em tempo de execução.

O trader é um repositório que contém serviços, os quais são oferecidos por servidores. O serviço especifica o tipo da interface e os seus atributos. Os servidores manipulam os serviços dentro do trader através de suas operações de retirada, modificação e exportação. Os clientes escolhem os seus serviços através da especificação do tipo e dos atributos nas operações de busca e seleção e da lista de detalhes.

- **Relocador** - O relocador é um repositório que contém as localizações das interfaces (tem um diretório com as alterações das localizações das interfaces) necessárias para o oferecimento da transparência de localização. A função relocador tem a finalidade de gerenciar este repositório, incluindo as localizações das funções de gerenciamento para o *cluster*.

- **Transparência** → O RM-ODP define um conjunto de transparências de distribuição, e descreve uma infraestrutura necessária para prover essas transparências. O objetivo dessas transparências é deixar de lado as complexidades referentes ao sistema distribuído utilizado pelos projetistas de aplicações. As transparências definidas no RM-ODP são:

  - **Transparência de Acesso** - A transparência de acesso esconde as diferenças existentes entre as representações de dados e invocações das operações dos objetos de comunicação. Essa transparência é obtida utilizando-se de *stubs* nas interfaces operacionais. O cliente interage com o *stub*, que age como um procurador para o servidor, oferecendo a interface do servidor localmente. O cliente obtém as informações necessárias (nome da operação e seus parâmetros) para compor a sua requisição. A requisição é enviada ao *stub* referente ao servidor, que desempacota essas informações, faz uma chamada local, recebe a resposta e a envia para a origem. A resposta chega ao stub cliente, que decodifica e entrega ao cliente. Como foi observado, são os objetos *stubs* que possibilitam essa transparência de acesso.

  - **Transparência de Localização** - Essa transparência liberta o objeto de engenharia básico e seu programador de conhecer a localização de algum objeto com quem ele interage. A transparência de localização entre dois objetos de engenharia básicos requer um objeto **binder** nos extremos de um canal, com a finalidade de realizar duas funções. A primeira é informar ao realocador a posição da interface que ele suporta, e a segunda é obter a localização da interface com que ele está interagindo.

  - **Transparência de Falha** - Essa transparência esconde alguma falha durante a execução de algum serviço. Ela envolve dois objetos *binders*, que periodica-

mente realizam verificações do servidor, e recupera o servidor após a última verificação, caso ele tenha falhado.

- **Transparência de Federação** - esconde os limites entre domínios administrativos e/ou tecnológicos. Essa transparência provê a integração de sistemas e recursos, cobrindo por exemplo, a integração de sistemas com diferentes arquiteturas, e sistemas com diferentes recursos e desempenho. Ela também provê a modularidade, admitindo o crescimento do sistema sem afetar as aplicações existentes, e também o compartilhamento, integração ou particionamento dos recursos e aplicações através de sistemas diferentes, gerenciando domínios e posições em resposta as necessidades dos usuários.
- **Transparência de Grupo** - consiste no mascaramento do uso de um grupo de objetos no suporte de uma interface computacional. Essa transparência admite a replicação de objetos para possibilitar a disponibilidade, confiabilidade e tolerância a falhas, ou para melhorar o desempenho do sistema ou ainda para admitir a modularidade.
- **Transparência de Migração** - mascara a heterogeneidade dos componentes do sistema, de forma que possibilite a migração de funções e aplicações do sistema entre seus componentes.
- **Transparência de Replicação** - mantém a consistência de um grupo de objetos replicados com uma interface.
- **Transparência de Recurso** - mascara a alocação e desalocação de recursos para os componentes do sistema, ativando os objetos que tem recursos para serem executados.
- **Transparência de Transação** - mascara a coordenação das operações que realizam transações. Essa transparência possibilita a manutenção da integridade e consistência dos dados, confiabilidade e tolerância a falhas e melhora o desempenho através de operações que podem ser executadas em paralelo.

- **Gerenciamento** → Estas funções tem a finalidade de prover e gerenciar o modelo de engenharia. Estas funções são divididas em: gerenciamento de nó (realiza o gerenciamento do nó criando cápsulas e canais, controlando *timers* e *threads*); gerenciamento de objeto (oferece operações básicas de *checkpoint* e remoção de objetos básicos de engenharia); gerenciamento de *cluster* (realiza o gerenciamento em um *cluster*); gerenciamento de cápsulas (realiza o gerenciamento em cápsulas); gerenciamento de referência de interfaces de engenharia (realiza o gerenciamento em referências de interfaces de engenharia).

- **Segurança** → Estas funções são divididas em: controle de acesso (garante a validação, autorização e autenticação de acesso a objetos, evitando as interações com

objetos que não estejam autorizadas); auditoria de segurança (provê monitoração e coleta de informação sobre ações relacionadas à segurança e análise destas); autenticação (assegura a autenticação de um objeto); integridade (garante que não haja criação, alteração e remoção de dados não autorizados); confidencialidade (evita a descoberta de informações não autorizadas); não-repudiação (evita a recusa de um objeto envolvido em uma interação de ter participação em toda ou parte da interação); gerenciamento de chaves (provê facilidades para a geração e manutenção de chaves criptográficas).

- **Coordenação** → Tem a finalidade de coordenar as atividades desempenhadas pela estrutura ODP. Estas funções estão divididas em: *checkpoint* e recuperação (coordena o *checkpoint* e a recuperação de *clusters* falhos); desativação e reativação (coordena a desativação e reativação de *clusters*); notificação de eventos (realiza registros e torna disponível informações sobre eventos); grupo (realiza o gerenciamento de *bindings* entre objetos); migração (realiza a coordenação de migração de um *cluster* de uma cápsula para outra); replicação (assegura que um grupo de objetos apareça para outros objetos como se fosse um único objeto); transação (coordena e controla um conjunto de transações para alcançar um nível específico de visibilidade, restabelecimento e estabilidade).

Capítulo 3

Um modelo de API para a Camada Middleware

3.1 Introdução

Com a finalidade de otimizar os recursos disponíveis em plataformas comerciais e oferecê-los de uma forma consistente e segura para o desenvolvimento, execução e gerenciamento das **aplicações distribuídas**[1], estamos apresentando um ambiente de programação através da especificação de um modelo geral de API[2][New92, Mye92, Fle92a, Fle92b]. Esta API recebeu o nome de MID/API[3], e tem como principal objetivo possibilitar a portabilidade às aplicações do sistema, bem como oferece um conjunto de serviços providos por plataformas *midware*[4][Ber93]. A MID/API junto com as plataformas *midware* compõem uma camada *Middleware*[Ber93], que tem a finalidade de reduzir os problemas referentes à heterogeneidade e distribuição em ambientes distribuídos.

O nosso projeto consiste na especificação da MID/API, no qual os serviços oferecidos são específicos a qualquer plataforma *midware*, porém, para fins de desenvolvimento inicial do projeto, trabalhamos somente com as plataformas DCE[5] e ANSA[6]. A utilização do ORB no sistema virá em função do aparecimento de serviços de objetos (vide a introdução do capítulo 4). A figura 3.1 mostra uma visão geral da arquitetura de suporte às aplicações, utilizando-se da camada *Midlleware*.

[1] Componentes espalhados de forma distribuída

[2] Application Program Interface - uma interface para programas de aplicação que define uma biblioteca de funções com um conjunto de serviços a ser oferecido ao cliente

[3] Midware Application Program Interface

[4] São plataformas que aumentam o nível de abstração, oferecendo um conjunto uniforme de serviços e mascarando as diferenças físicas e de software do sistema

[5] Distributed Computing Environment

[6] Advanced Networked Systems Architecture

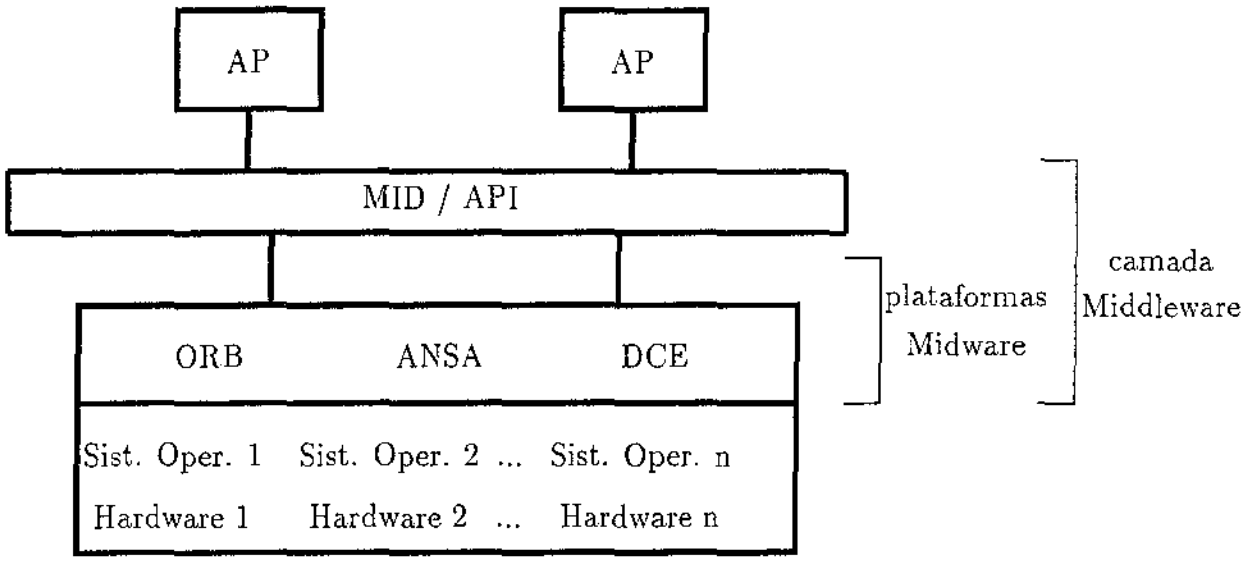


Figura 3.1: Arquitetura da Camada Middleware

3.2 Camada Middleware

Como citado acima, há a necessidade de se resolver os problemas referentes à heterogeneidade e à distribuição em sistemas distribuídos. Assim, projetistas de sistemas desenvolveram uma camada *Middleware* (tecnologia de *software*) composta por um conjunto de serviços *middleware* (serviços para suportar sistemas distribuídos) oferecidos através de interfaces de programação , possibilitando a comunicação local ou remota entre clientes e servidores sem a preocupação com sistemas operacionais e arquiteturas. O nome de *Middleware* vem em função de seu posicionamento acima do sistema operacional e do *software* da rede e abaixo da camada de aplicação[Ber93, JJP+93, FC93].

3.3 Plataformas Midware

Com a evolução da computação distribuída acoplada à necessidade de prover a conectividade e interoperabilidade entre sistemas de computação distribuída, tem-se estimulado a atividade no desenvolvimento de plataformas de computação distribuída, como DCE, ANSA, etc. O objetivo dessas plataformas *midware* é prover serviços necessários às aplicações distribuídas, bem como facilitar o seu desenvolvimento. Em particular, estas plataformas visam tornar as aplicações distribuídas independente de sistemas operacionais e hardware. Para obter essa funcionalidade, as plataformas provêem serviços em tempo de execução suportando várias formas de transparência, tais como acesso e loca-

lização[Ber93, PZTT92].

3.3.1 Sistema DCE

Para resolver a questão da interoperabilidade entre ambientes heterogêneos ligados em rede, a OSF[7] criou um ambiente de computação distribuída chamado DCE. Este sistema, além de prover comunicação, também incrementa a oferta de serviços para aplicações independente da localização do usuário, aplicação ou recursos necessários. Logo pode-se dizer que o DCE é um conjunto amplo e integrado de serviços que suporta o desenvolvimento, o uso e a manutenção de aplicações distribuídas.

Para uma melhor visualização do DCE, são citadas algumas de suas principais características[OSF90b]:

- Executa comunicação entre usuários da rede, independente das localizações de origem e destino, sem expor a complexidade da rede ao usuário, gerenciador do sistema ou programador de aplicação.
- Possui um conjunto de serviços essenciais para satisfazer o modelo cliente/servidor.
- Possui uma arquitetura que permite incorporar tecnologias futuras de conecção.
- Oferece algum suporte para padrões internacionais, que contribuem para a interconectividade (Directory Service da ISO[8]; X-500 da ITU-T[9]).
- Independência da rede e sistema operacional. O DCE pode ser usado com quase todos os *hardware* de redes e *software* de transporte, incluindo o Ethernet, X.25, TCP/IP[10] e ISO/OSI[11], tão bem quanto outros produtos similares. Em função de sua independência de transporte, ela pode ser usada com outros *softwares* de computação distribuída sem problemas. Sua implementação portável faz uso de interfaces padrões tais quais as especificadas pela POSIX e X/OPEN para serviços do sistema operacional.
- O DCE tem ainda como benefícios os seguintes itens:
  - **Fácil utilização** - sua tecnologia de construção é baseada em conceitos familiares (ex: RPC[12], DFS[13].

[7]Open Software Foundation
[8]International Standardization Organization
[9]International Telecommunication Union
[10]Transmission Control Protocol/Internet Protocol
[11]Open Systems Interconnection
[12]Remote Procedure Call
[13]Distributed File System

- **Fácil administração** - o gerenciamento do sistema pode ser feito sem a necessidade do conhecimento interno do mesmo.
- **Segurança** - provida através de *software* especializado (Kerberos).
- **Desempenho** - aumenta a produtividade dos usuários, e reduz o processamento necessário para uso do sistema, através de rápidas respostas e alta vazão de informações pela rede.
- **Disponibilidade** - está sempre disponível para qualquer aplicação.
- **Escalabilidade** - possibilita que usuários comecem com uma configuração apropriada as suas necessidades, podendo aumentar para grandes configurações sem afetar o desempenho, confiabilidade e flexibilidade.

• O DCE também apresenta algumas desvantagens:

- Não suporta padrões ISO/OSI, com exceção do X-500.
- Não suporta replicações de objetos de aplicação.
- Não suporta transações.

Arquitetura

Como já citado, o DCE é um conjunto de serviços integrados que suporta o uso, desenvolvimento e manutenção de aplicações distribuídas. A disponibilidade uniforme desse conjunto de serviços, em qualquer lugar da rede, possibilita que aplicações utilizem todos os recursos disponíveis na mesma.

Para alcançar esses objetivos propostos, a OSF integrou um conjunto de oito tecnologias dentro de um sistema coerentemente estruturado, provendo uma camada lógica que mascara a complexidade física do ambiente distribuído. Essa arquitetura é baseada em camadas, numa abordagem *bottom-up*, desde os serviços mais básicos (sistema operacional) para os consumidores desses serviços (aplicações). A figura 3.2 apresenta a arquitetura DCE[OSF90b, OSF90c].

Serviços DCE

Os serviços oferecidos pela DCE podem ser usados individualmente ou em conjunto. Eles são divididos em duas categorias:

• **Serviços Distribuídos Fundamentais** ⇒ formam um conjunto base de serviços e ferramentas que podem ser usados por programadores para construir ambientes distribuídos e aplicações para usuários finais. São eles:

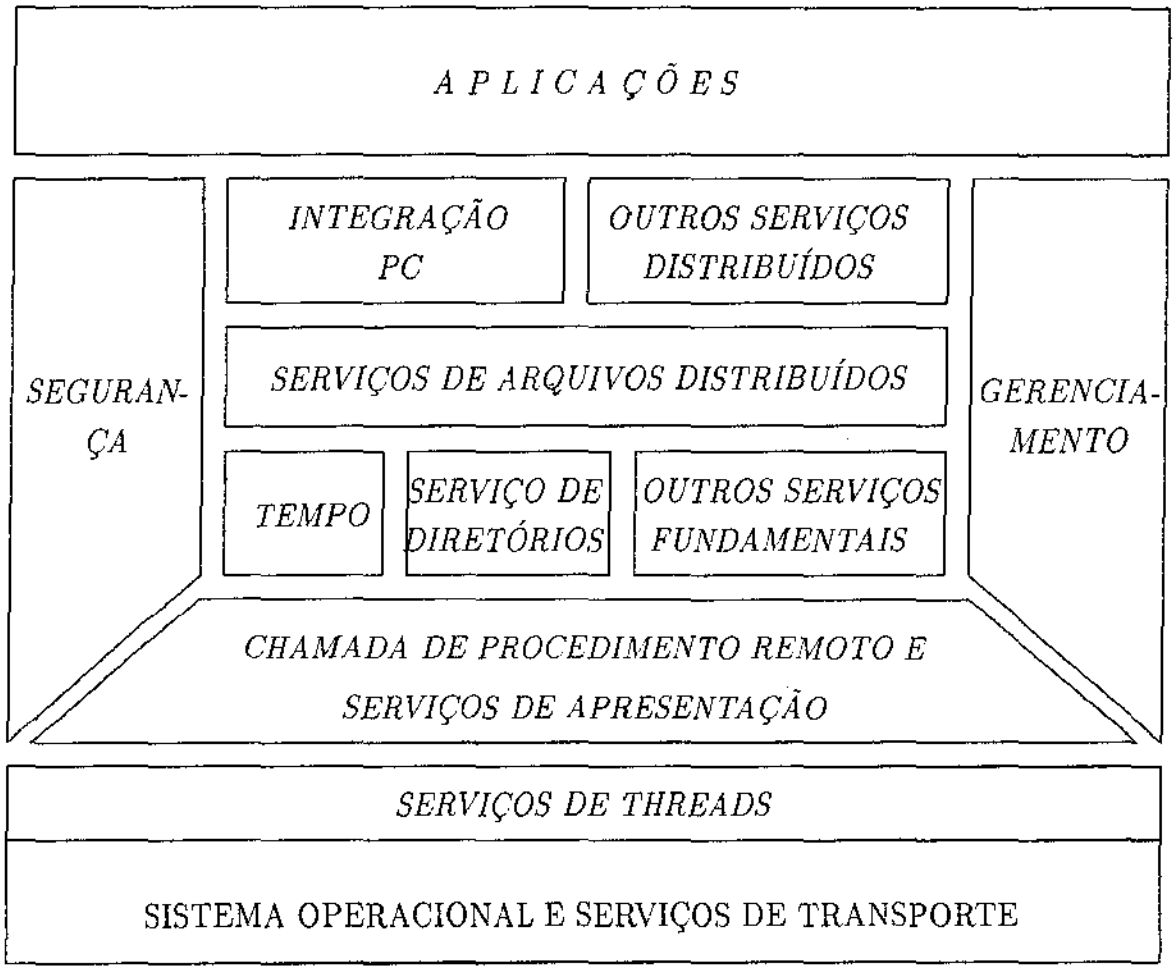


Figura 3.2: Arquitetura DCE

- **Chamada de Procedimento Remoto (RPC)** → possibilita que procedimentos de uma aplicação possam ser executados em qualquer lugar na rede, baseado no paradigma cliente/ servidor[OSF90e].
- **Serviço de Diretórios Local (Cell Directory Service)** → administra nomes e os mapeia em endereços de rede, possibilitando que usuários, através de nomes, identifiquem recursos como servidores, arquivos, discos, filas de impressão, etc [OSF90a].
- **Serviço de Temporização** → possibilita uma referência de tempo para programar e determinar a duração e a seqüência de um evento. Realiza também a sincronização de cada computador em relação a um padrão.
- **Serviço de threads** → provê facilidades que suportam a programação concorrente, admitindo que a aplicação execute algumas ações simultaneamente.
- **Serviço de Segurança** → provê a rede com três serviços convencionais; autenticação, autorização e gerenciamento de contas de usuários. Essas facilidades estão disponíveis através de meios seguros de comunicação que garantem tanto a integridade quanto a privacidade do sistema.

• **Serviços de Compartilhamento de Dados** ⇒ construídos sobre os serviços fundamentais, são integrados somente com o sistema operacional, provendo aos programadores e usuários finais o compartilhamento de informações sem a necessidade de uma programação adicional.

- **Sistema de Arquivos Distribuídos (DFS)** → é o componente chave do compartilhamento de informações,utilizado na administração de arquivos remotos. Através da função de sistemas de arquivos locais com uma interface consistente, o sistema de arquivos faz também o acesso global. Utiliza-se do modelo cliente/servidor para acessar outros sistemas de arquivos distribuídos[OSF90d].
- **Suporte a Diskless** → possibilita a utilização de estações *diskless* e provê protocolos de propósito geral bem definidos para essa função.
- **Serviço de Integração de PCs** → possibilita que usuários de minicomputadores, mainframes e PCs possam compartilhar arquivos, periféricos e aplicações em um ambiente distribuído.'
- **Serviço de Diretórios Global** → Utiliza o serviço de diretórios da ISO (X-500 da ITU-T), possibilitando que serviços de diretórios locais (Cell Directory Services) sejam incluídos em grandes sistemas[OSF90a].

Se formos analizar o DCE sob o ponto de vista ODP, veremos diversas discordâncias entre eles, tendo em vista que o DCE é orientado a processos e o RM-ODP é orientado

a objetos. Porém, pode-se dizer que o DCE possui alguns conceitos ODP, como por exemplo o conceito de interface computacional, apesar de aceitar uma única terminação por interface. Apesar de o DCE prover um ambiente distribuído, ele não é um ambiente de suporte tão aberto como é previsto pelo RM-ODP. Com isso pode-se concluir que não será uma tarefa simples a idéia de realizar uma concordância entre DCE e os conceitos previstos no RM-ODP.

3.3.2 Sistema ANSA

O sistema ANSA[14] é o precursor do RM-ODP, isto é, ele apresenta um subconjunto do RM-ODP. A ANSA é uma arquitetura para construção de sistemas distribuídos que opera como um todo. O fato da distribuição é transparente para usuários e programadores de aplicação. A ANSA produz um sistema que pode ser gerenciado como subsistemas coordenados apropriados para empresas[Lim89, Lim91b, Lim91a].

Arquitetura

A meta do projeto ANSA tem sido desenvolver uma arquitetura, provendo um conjunto simples de conceitos necessários para a construção de sistemas distribuídos.

A descrição da ANSA é baseada em um conjunto de projeções da arquitetura em cima de modelos que representam os cinco pontos de vista, como no RM-ODP (empresa, informação, computação, engenharia, tecnológico). Esses modelos são descritos da seguinte forma:

- Modelo Empresa → tem o propósito de prover um suporte para esclarecer e justificar a função do sistema de processamento da informação dentro de uma organização. A descrição da empresa é baseada em todos os objetos do sistema em relação a funções, ações, metas e políticas. Esse modelo especifica as atividades localizadas dentro da organização utilizando-se do sistema, as funções executadas por pessoas, as interações entre a organização, o sistema e o ambiente em que estão posicionados.

- Modelo de Informação → Provê um suporte para descrever as necessidades de informação do sistema. A descrição do sistema é feita através da estrutura dos elementos de informação, das regras de relacionamento entre eles no sistema e de suas restrições.

- Modelo Computacional → provê um suporte para modelar as operações de transferência, recuperação, transformação e gerenciamento da informação necessárias à automatização do processamento da informação. Os mecanismos necessários para suportar o modelo computacional são especificados na projeção do sistema.

[14]Advanced Networked Systems Architecture

Durante a montagem do sistema, maiores detalhes do fluxo das decisões do projeto são visíveis no modelo computacional. A sua filosofia é baseada em encontrar um menor número de conceitos necessários para descrever a computação distribuída, com uma formulação declarativa para cada um desses conceitos.

O modelo computacional concentra-se nos problemas e oportunidades apresentados pela execução de aplicações em alguns sistemas de computação acoplados. O modelo provê conceitos necessários para definir linguagens de programação de aplicação distribuída futuras para escrita de programas de aplicação em ambientes distribuídos.

Para o processamento da informação distribuída, o modelo computacional necessariamente habilita a modelagem de componentes de aplicação, que podem ser executados em paralelo, falhar independentemente ou ainda estabelecer *bindings* dinamicamente.

- Modelo de Engenharia → provê o suporte para descrever como operacionalizar uma definição de aplicação identificada, utilizando o modelo computacional. Esse suporte incluirá a definição da distribuição física para realizar o particionamento definido na projeção computacional. Ele é um suporte do compilador e dos componentes do sistema operacional para a realização da comunicação e computação em ambientes distribuídos.

  O modelo de engenharia dá ao projetista de sistema uma visão dos recursos de engenharia disponíveis, provendo o mecanismo para suportar uma determinada função definida no modelo computacional. Esse modelo dá ao implementador do sistema ferramentas para a construção de ambientes apropriados aos serviços que desejam ser executados.

- Modelo de Tecnologia → provê um suporte para descrever os componentes que formam o sistema distribuído. O modelo mostra como é o *hardware* e o *software* do sistema local, o esquema de entrada e saída, e os pontos de acesso a comunicação são mapeados dentro de mecanismos identificados no modelo de engenharia.

A estrutura geral dos componentes ANSA é mostrada através das figuras que se seguem. A figura 3.3 mostra uma federação (um conjunto de elementos (clientes, servidores e *trader*) que procuram alcançar um objetivo comum) de sistemas ANSA, em que cada sistema está executando várias aplicações ligadas ao gerente de configuração e ao *trader* (entidade que executa a negociação entre servidores que exportam interfaces e clientes que importam interfaces).

A figura 3.4 inclui o componente núcleo, que a partir dos recursos locais disponíveis, constrói um ambiente de computação distribuída básica para cada máquina.

Na ANSA várias transparências de distribuição estão disponíveis. Essas transparências determinam a extensão com que os programadores devem se preocupar no controle e

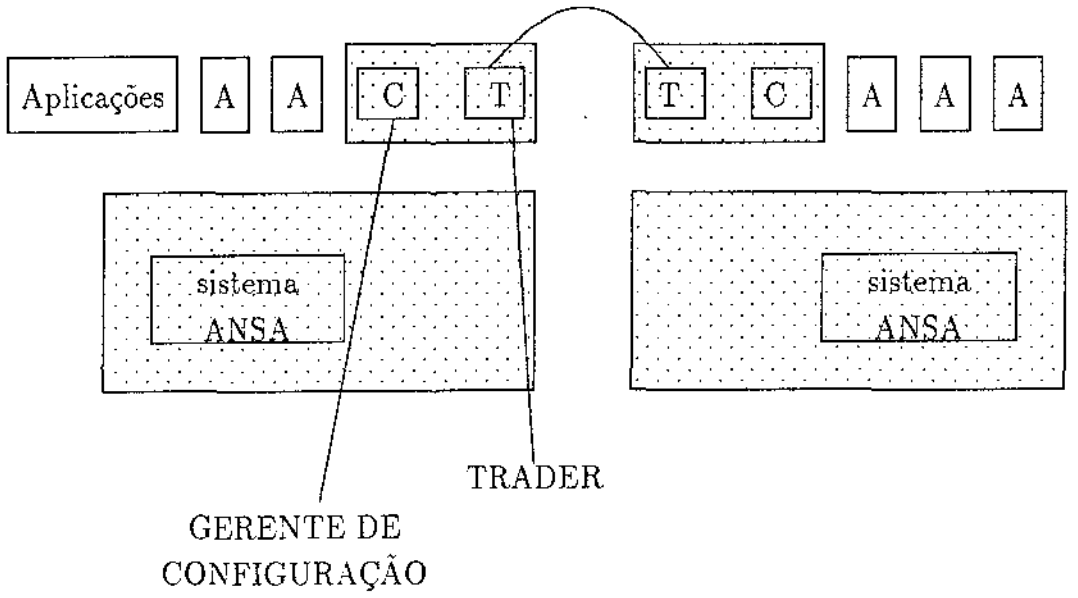


Figura 3.3: Federação de Sistemas ANSA

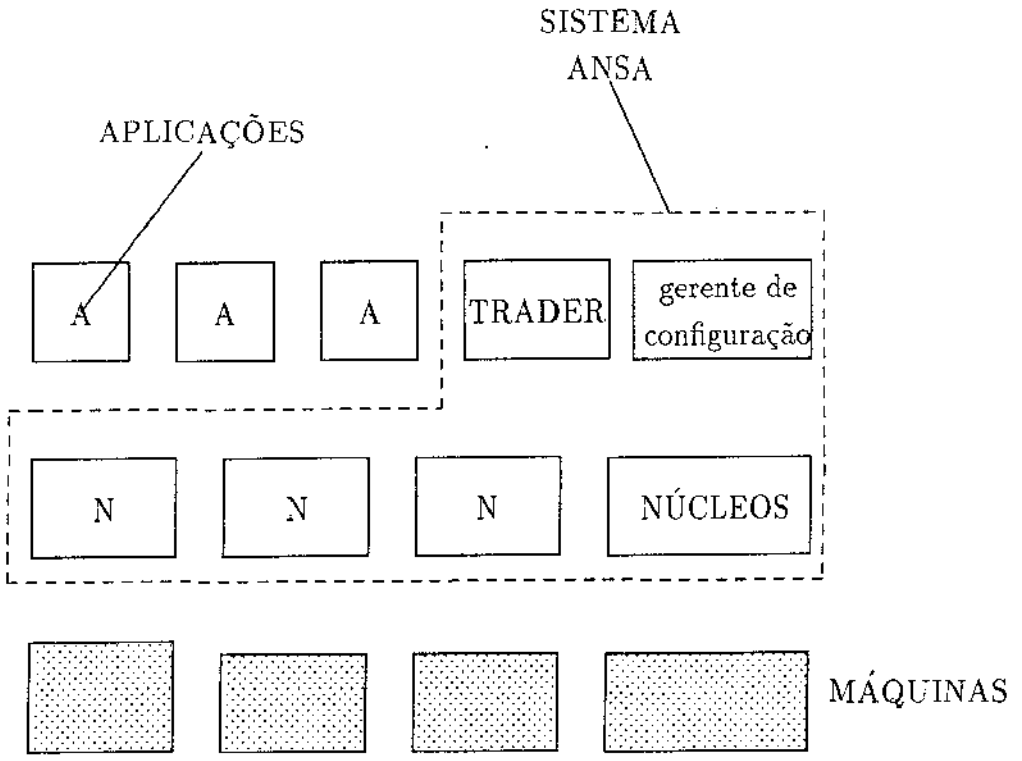


Figura 3.4: Sistema ANSA

na integração de partes distribuídas de programas de aplicação. A ANSA dispõem das transparências de acesso e localização, e parcialmente as de concorrência e falha.

Serviços ANSA

O sistema oferece módulos de serviço que estão disponíveis para o cliente. São eles[Lim89]:

- **Gerenciamento de threads** → possibilita a programação concorrente, caso não seja provida pela máquina. Ela será necessária se servidores necessitarem responder a vários clientes em paralelo, ou para que esses possam executar serviços em paralelo ou executar serviços remotos em paralelo.
- **Gerenciamento de espaço de endereço** → sua função é complementar a função do gerenciamento de *threads*. Possui facilidades que gerenciam pilhas múltiplas e *buffers* de comunicação dentro de um único espaço de endereço.
- **Trader** → realiza o serviço de diretórios e facilita a oferta de serviços para componentes de aplicações distribuídas.
- **Gerente de configuração** → possibilita a inclusão ou não de componentes de aplicação sobre o sistema.
- **Interpretador** → provê um interface padrão independente da implementação para que possam ser realizadas interações entre *threads* em diferentes localizações.

3.4 MID/API

A MID/API é uma interface para programas de aplicação baseada na metodologia orientada a objetos , com a finalidade de prover portabilidade às aplicações do sistema, bem como oferecer um conjunto de serviços de forma homogênea e transparente para o desenvolvimento, execução e gerenciamento de aplicações para usuários finais[Fle92a, Fle92b, Shn87]. Essa transparência tem como objetivo não expor ao cliente a complexidade da plataforma que será acionada. Para alguns serviços em certas estações o cliente nem precisaria saber qual plataforma será acionada para a execução do mesmo. Isso só acontecerá em casos específicos, pois na maioria das vezes existirá somente uma plataforma *midware* por estação. Esses serviços são providos pelas plataformas *midware* como ANSA e DCE. Essa interface é bidirecional, baseada no paradigma cliente/servidor, sua estrutrura é em blocos, dividida em duas camadas, uma para o cliente e a outra provedora de serviços. A metodologia orientada a objetos[TLX90, BGHS91] propicia que seus objetos sejam modelados de forma geral para que aceitem a implementação em qualquer linguagem e englobem propriedades básicas como interação, abstração, encapsulamento, entre outras.

A idéia de utilizarmos uma API para estabelecer a interação entre cliente e servidor ocorre em função de algumas características inerentes a sistemas distribuídos. Imaginemos o caso onde um cliente deseje abrir uma conta, ou realizar um saque, ou verificar um saldo. Para que isso seja possível são utilizadas interfaces operacionais (interfaces baseadas no paradigma cliente/servidor), nas quais serviços são oferecidos ao cliente (vide figura 3.5). De forma semelhante, definimos a MID/API como uma interface operacional, em que os serviços providos pelas plataformas *midware* são oferecidos.

Para que as aplicações distribuídas possam dispor da melhor forma possível dos serviços oferecidos, alguns objetivos da nossa especificação são:

- **Portabilidade** - total portabilidade, onde propomos uma total independência do sistema operacional e *hardware*.
- **Bindings** - Estabelecer bindings com diversas linguagens, reduzindo a complexidade no desenvolvimento de *software*.
- **Portabilidade de Midware** - independe da plataforma *midware*, isto é, pode operar sobre qualquer tipo de sistemas comerciais.

Os serviços oferecidos pela MID/API podem ser obtidos de duas formas:

- A primeira forma é quando o cliente invoca um serviço, especificando qual a plataforma *midware* que será utilizada na realização do serviço, caso exista mais de uma plataforma *midware* em uma mesma estação de trabalho. Nessa situação, a MID/API é conhecedora de qual sistema deverá interagir para realizar o serviço seja ele de alto ou baixo nível. É o caso comum.
- A segunda forma é quando o cliente invoca um determinado serviço, e esse não sabe qual a plataforma *midware* será acionada para a execução do serviço. O serviço solicitado seja ele de alto ou baixo nível, é encaminhado via API até a plataforma de execução. Normalmente, uma estação de trabalho possui uma única plataforma *midware*, mas caso haja mais de uma plataforma em uma mesma estação que execute o mesmo serviço, a MID/API decidirá qual destas irá executá-lo.

3.4.1 Estrutura

Como já foi definido, a MID/API é composta por duas camadas lógicas, a do cliente e a provedora de serviços. Esta estrutura da MID/API é baseada no modelo de API para sistemas de comunicação especificado em [EM90], adaptado para o processamento distribuído aberto. A camada do cliente ou biblioteca de funções é composta por módulos

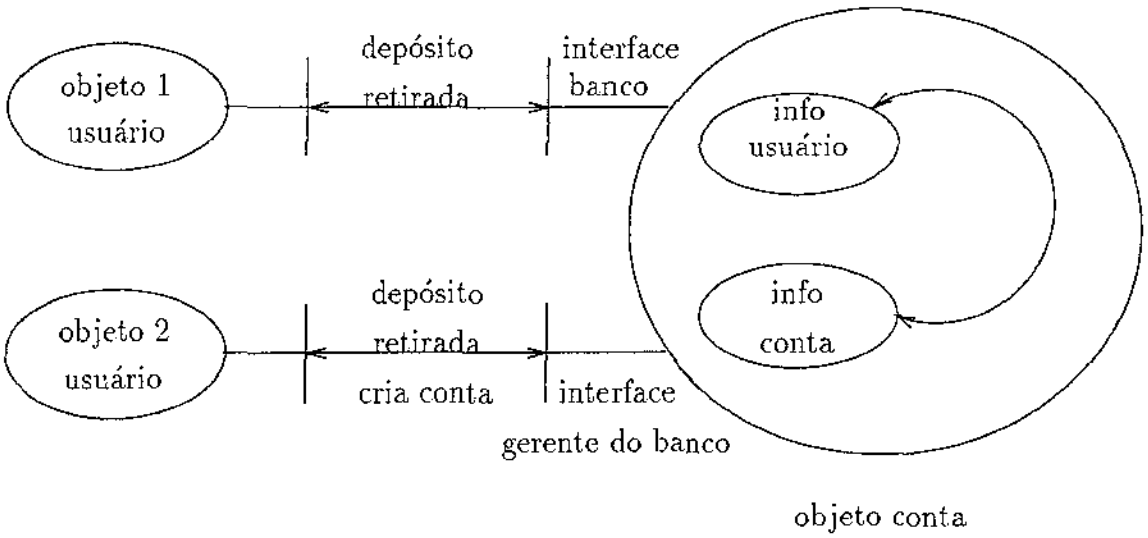


Figura 3.5: Interfaces Operacionais

com funções de alto e baixo níveis. A camada provedora de serviços é responsável em fornecer serviços para a camada do cliente, como também tornar possível a chegada da solicitação do cliente para a execução do serviço na estação remota. A arquitetura proposta para a MID/API é baseada em idéias apresentadas em [MAP88, RMM92], utilizadas em sistemas de comunicação, porém a reestruturamos de forma que seja utilizada para o processamento distribuído aberto. A figura 3.6 mostra a estrutura da arquitetura da MID/API.

Camada do Cliente

Esta camada é responsável por todo o processamento necessário à definição das solicitações do cliente, através de suas interfaces junto a camada provedora de serviços. Algumas funções são executadas por essa camada:

- Garantir espaço suficiente no *buffer* para a passagem de dados e recebimento de resultados, utilizando-se de uma política de alocação de memória.
- Informar ao cliente de que a sua solicitação foi aceita.
- Informar ao cliente qualquer problema que haja para a execução do serviço solicitado.
- Garantir que o resultado do serviço solicitado pelo cliente retorne ao mesmo.
- Ordenar todas as solicitações de serviços feitas pelo cliente.

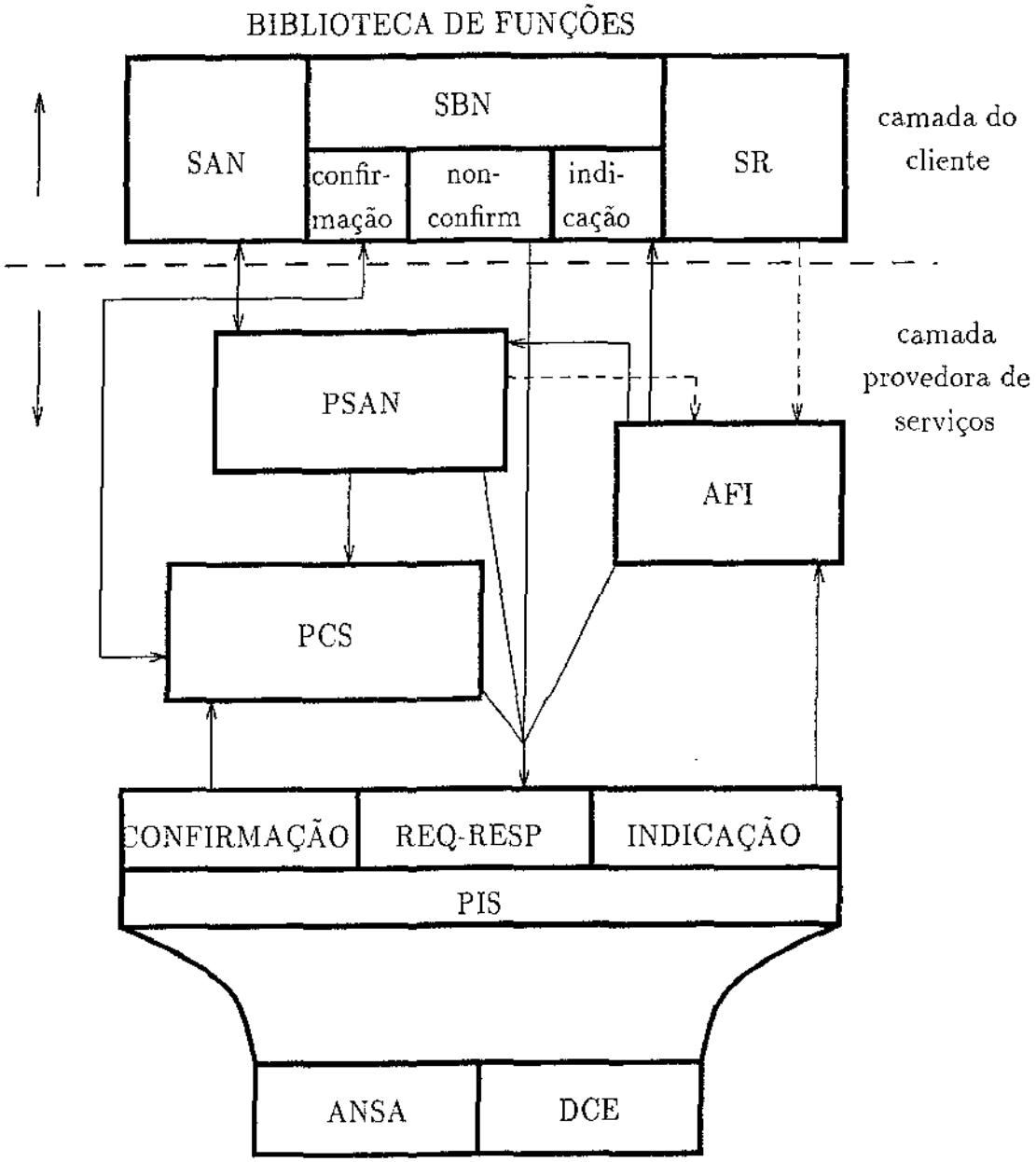


Figura 3.6: Arquitetura MID/API

- É responsável por decidir qual plataforma *midware* deverá ser acionada para a execução de um determinado serviço em certos casos muito particulares.

A camada cliente ou biblioteca de funções é composta por três módulos de funções:

- **Serviços de Alto Nível (SAN)** → contém um conjunto de serviços de alto nível composto por serviços oferecidos pelas plataformas *midware*. O cliente tem acesso a esses serviços como se estivesse acessando uma biblioteca de funções.

- **Serviços de Baixo Nível (SBN)** → contém um conjunto de serviços de baixo nível que auxiliarão na execução de um serviço alto nível. Proporcionam ao cliente um controle completo de todas as interações com as funções solicitadas. O cliente também pode acessar esses serviços como se estivesse acessando uma biblioteca de funções.

- **Serviço Respondedor (SR)** → esse módulo possui um conjunto de funções respondedoras que irão preparar a AFI para receber as solicitações feitas por serviços remotos durante a execução de um serviço solicitado pelo cliente. Durante o estabelecimento da conexão são definidos os parâmetros de filtragem, e estes são específicos a uma conexão. Os métodos relacionados com a filtragem de solicitações são ativados quando solicitados pelos módulos SR e PSAN. Como exemplo pode-se citar as funções que responderão as solicitações feita por servidores quando existe a necessidade da abertura de um arquivo para leitura. A filtragem realizada nesta função só foi possível devido a programação prévia feita por funções respondedoras. Essa solicitação retorna até a camada do cliente, passando pelo AFI, sendo então invocado o serviço necessário à conclusão da solicitação do cliente.

Camada Provedora de Serviços

Os serviços disponíveis na camada provedora de serviços são visualizados, inicialmente, como serviços que não fazem parte da aplicação, onde cada bloco de funções é uma entidade própria.

A camada provedora de serviços é organizada em módulos funcionais sendo que os serviços destes módulos são utilizados por outros módulos. Estes módulos determinam de forma funcional o que acontece a um serviço quando solicitado pelo cliente. Os módulos provêem serviços entre si, como também para a camada do cliente. É através dessa camada que é possível a chegada de solicitações dos clientes aos sistemas ANSA e DCE.

A camada provedora de serviços é composta dos seguintes módulos:

- **Provedor de Serviços de Alto Nível (PSAN)** → este módulo possui funções que irão interagir com os módulos de confirmação de serviços ou com o árbitro e filtro

de indicações, de forma que a solicitação feita pelo cliente chegue até ao módulo provedor de interfaces de serviço. Na verdade, ele quebra os serviços de alto nível em serviços de baixo nível.

- **Provedor de Confirmação de Serviços (PCS)** → esse módulo possui funções que realizam o casamento das requisições com as confirmações.
- **Arbitrador e Filtro de Indicação (AFI)** → esse módulo possui funções que filtram as solicitações remotas que podem ser realizadas das que não podem segundo a programação prévia das aplicações. O AFI comporta-se da seguinte forma:
  - Tanto o respondedor (SR) quanto o PSAN podem ativar alguns métodos (filtros), programados por eles no AFI, de forma que seja feita a filtragem das indicações vindas pelo PSI. Não existem problemas quanto ao uso pelo SR e pelo PSAN de um mesmo filtro do AFI durante a realização de uma determinada função. Caso haja conflito, é interrompida a intervenção pelo SR ou PSAN, e analisado esse conflito.
  - Caso uma solicitação seja gerenciada para passar por todos os filtros, essa pode ser captada tanto pelo SBN ou pelo PSAN.
  - Deve ser capaz de ordenar todas as solicitações feitas durante a execução de uma função, de forma que sejam liberadas de acordo com a necessidade do cliente.
- **Provedor de Interfaces de Serviço (PIS)** → esse módulo possui funções que interagem com as plataformas ANSA e DCE enviando as solicitações dos clientes e recebendo respostas das mesmas. Ele também possibilita que as indicações de solicitação alcancem o AFI, assim como as confirmações de serviço sejam enviadas até a camada do cliente. É esse módulo o responsável direto pela verificação junto as plataformas da disponibilidade para a execução dos serviços solicitados.

  O processo de receber solicitações ou enviar respostas é feito um a cada vez, obedecendo o sistema de fila. Porém pode receber simultaneamente as indicações ou confirmações para o PCS ou AFI.

3.4.2 Estratégia de Alocação de Memória

Durante a invocação de uma solicitação, existe a necessidade da passagem de informações da camada do cliente para a camada provedora de serviços (ou vice-versa), para a execução do serviço. Essas informações podem vir através de dados, por exemplo, *strings* com tamanhos variados. Para que isso seja possível reserva-se uma área para dados ou um *buffer*

controlada pela camada do cliente, onde essas informações são armazenadas e posteriormente passadas para a camada provedora de serviços.

Para a execução de algumas funções não há condições de se saber o espaço necessário para o armazenamento das informações, porém existem casos em que isso pode ser previsto. A fim de acomodar essa flexibilidade, duas formas de alocação são apresentadas:

- O próprio cliente (programa) aloca espaço suficiente no *buffer* para a passagem de seus dados.
- A segunda forma é a alocação dinâmica, realizada pela camada do cliente. Esse mecanismo usado na alocação automática do *buffer* de entrada ou de saída, é específico ao mapeamento de linguagem. Para cada especificação de uma função será definida se o *buffer* é de entrada de saída, ou ambos, e serão alocados automaticamente.

Podería-se fazer uso de uma política de alocação em tempo de compilação. Essa, porém, não é viável tendo em vista que pode haver um gasto excessivo de memória.

3.4.3 Modelagem de Serviços

Nessa seção são definidas as interfaces com os seus serviços. São determinados na camada do cliente todos aqueles serviços oferecidos pelos sistemas ANSA e DCE, como também serviços de alto e baixo níveis. São definidos também as interfaces com os serviços da camada provedora de serviços.

Funções Síncronas e Assíncronas

A MID/API suporta funções síncronas e assíncronas. As funções síncronas recebem a definição de um evento de espera (nome) associado com a emissão da função. Isso proporciona um tempo de latência necessário até a liberação dos resultados. A chamada de uma função assíncrona cria uma mensagem para ser enviada ao PSAN correspondente. Tão logo essa mensagem seja enviada, o cliente recupera o controle e continua a executar outra função.

Camada do Cliente

Essa camada possui todos os serviços de alto e baixo níveis, como também os serviços respondedores. Seus blocos estão estruturados hierarquicamente em módulos e funções, como pode ser visto na figura 3.7.

1. **Serviços de Alto Nível** - Funções alto nível são aquelas ricas em funcionalidade. A chamada de uma dessas funções corresponde a execução de uma ou mais funções de baixo nível. Estas, ajudam na execução de um serviço de alto nível. A MID/API

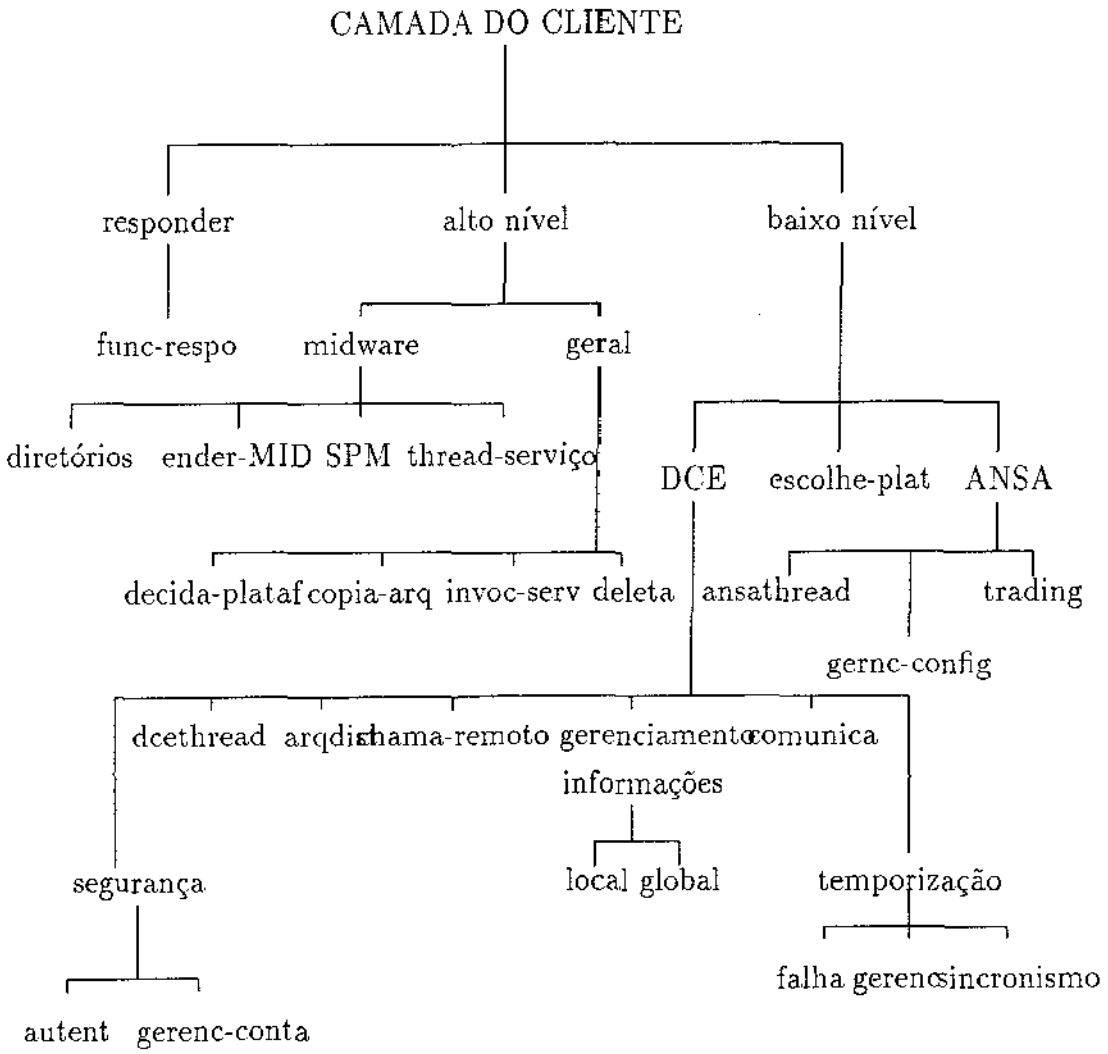


Figura 3.7: Estrutura hierárquica da camada do cliente

oferece vários serviços dos sistemas ANSA e DCE para o cliente. Esses serviços estão disponíveis em forma de objetos e divididos em unidades funcionais, podendo ser acessados como funções.

- **Módulo Midware** - esse módulo contém serviços referentes à utilização direta de serviços comuns às plataformas *midware*:
  - **Serviço de Diretórios** → executado local ou globalmente, pode ser realizado por ambas plataformas *midware*. Fazendo uso da plataforma DCE, pode-se dizer que o serviço de diretórios é distribuído, onde os objetos são definidos pelos seus nomes. Serviços e aplicações acessam objetos através de seus nomes, independente de mudanças de suas posições ou características.
  - **ender-MID** → Possibilita a retirada ou adição de novos endereços dentro do ambiente de computação. O serviço de *trading* da ANSA ou o serviço de diretórios da DCE verifica a existência ou não daquele objeto e caso constate a sua inexistência dentro do sistema, é realizada a operação de alocação de espaço de endereço.
  - **MPS (Serviço de Passagem de Mensagens)** → Tem como propósito de gerenciar a conecção e desconecção, como também a transmissão e recepção de mensagens entre nós da rede. Esse serviço é utilizado para estabelecimento de *bindings* e pelo serviço de *trading* na plataforma ANSA, ou ainda pela plataforma DCE para estabelecer comunicação entre interfaces, através de RPC. Pode ser transparente para o cliente qual dessas plataformas *midware* serão acionadas para a realização do serviço.
  - **Thread-serviço** → Esse serviço tem por finalidade acionar os serviços de *threads*, tanto da ANSA quanto da DCE, para a realização de serviços que possuem capacidade de processamento, paralelo. Com isso ele possibilita a realização de serviços que necessitem de programação concorrente.
- **Módulo GERAL** - Esse módulo contém serviços que executam funções de âmbito geral. Esses serviços fazem uso das plataformas *midware* para serem concluídos.
  - **copia-arq** → Esse serviço possibilita o acesso local ou remoto a um arquivo da rede para sua leitura ou cópia. Para que esse serviço seja executado, algumas funcionalidades das plataformas *midware* serão utilizadas. Precisa-se ter acesso ao sistema, obtido pelo serviço de segurança e posteriormente, estabelecer uma conexão, abrir o arquivo e executar o serviço desejado no mesmo.
  - **Invoc-serv** → Esse serviço tem a finalidade de invocar algum serviço disponível na rede. Para isso é necessária a utilização de algumas das funci-

onilidades disponíveis pelas plataformas *midware*, como RPC, serviços de diretórios e o *trader* na ANSA. Essa função é modelada como um serviço independente do programa do usuário e pode enviar a solicitação até a camada provedora de serviços, que será a responsável pela ruptura da requisição em serviços de baixo nível.

- **deleta** → é uma operação executada logo após a função ter sido executada. Automaticamente libera o espaço de memória alocada para a execução daquele serviço, se chamado assincronamente.
- **decida-plataf** → Tem a finalidade de decidir qual plataforma *midware* irá executar o serviço solicitado em casos específicos. Essa função recebe da camada provedora de serviços a informação de quais plataformas têm condição de executar o serviço solcitado pelo cliente. Essa decisão será baseada em um padrão previsto durante a implementação da API.

2. **Serviços Baixo Nível** -São serviços que são utilizados no auxílio da execução de funções alto nível, bem como na utilização direta na execução de um serviço.

   - **escolhe-plat** → Tem a finalidade de decidir qual plataforma *midware* irá executar o serviço solicitado em casos específicos. Essa função recebe da camada provedora de serviços a informação de quais plataformas têm condição de executar o serviço solcitado pelo cliente. Essa decisão será baseada em um padrão previsto durante a implementação da API.
   - **Módulo ANSA** - esse módulo possui algumas das funções que são oferecidas pela plataforma *midware* ANSA.
     - **gerenc-config** → O serviço de configuração provido pela plataforma ANSA, controla a forma em que uma coleção de objetos é configurada dentro de um sistema. O gerenciador de configuração negocia essa configuração, provendo meios para instanciar os componentes da aplicação sobre a plataforma.
     - **trading** → Realiza o emparelhamento entre as interfaces oferecidas e os respectivos *requests*. Estabelece o *biding* entre cliente e servidor através de suas interfaces importadoras e exportadoras. Durante essa fase, operações de empacotamento de informações sobre as interfaces importadora e exportadora são armazenadas pelo objeto *binder*, onde são passadas para o cliente ou servidor. O sucesso dessa associação não garante que as interações entre objetos ocorram especificamente através dessa associação. Possibilita o acesso a estrutura de diretórios, que pode ser buscada pelo *path name*, valores de propriedade ou pela combinação de ambos.

- **ansathread** → O serviço de *thread* provido pelo sistema ANSA, sincroniza uma execução em paralelo de vários *threads* durante as operações de invocação em um objeto computacional.

- **Módulo DCE** - esse módulo possui alguns dos principais serviços que são oferecidos pela plataforma *midware* DCE.
  - **comunica** → Possibilita o trabalho com vários conjuntos de caracteres, dando o suporte aos serviços de comunicação no estabelecimento de *bidings*.
  - **Módulo de Temporização** →Tem como propósito proporcionar às aplicações uma referência no tempo para que estas executem suas atividades. Regula o sistema de *clocks* na rede possibilitando um sincronismo adequado para a realização das aplicações distribuídas.
    - **falha** → Identifica quais os servidores que possuem seus *clocks* falhos, para que seja realizada a sincronização sem levá-los em consideração.
    - **gerenc** → Exerce o controle e o monitoramento do serviço de sincronização, de tempo em tempo.
    - **sincronismo** → Realiza a sincronização de todos os *clocks* da rede. Esse serviço pode ser realizado ao início do sistema ou quando for necessário.
  - **Módulo de Segurança** → é um conjunto de serviços que possibilita tanto a autenticação quanto a autorização do usuário no acesso ao sistema. Provê à rede também o serviço de gerenciamento de contas.
    - **autent** → É um serviço baseado no sistema *Kerberos* que valida a identidade do usuário ou serviço no sistema. Com a identificação positiva, é dada ao usuário ou serviço a autorização de utilização dos recursos disponíveis.
    - **gerenc-conta** → Resolve o problema de gerenciamento de contas do sistema, assegurando que o usuário registrado possua um nome e uma senha de uso exclusivo do mesmo.
  - **Módulo Gerenciamento de Informações** → é o módulo que possui um conjunto de serviços para manipular informações dentro do sistema.
    - **local** → Acoplado ao X.500, realiza a busca dentro do seu domínio administrativo. Por exemplo, caso um cliente deseje uma informação dentro do ambiente administrativo de um país, envia uma solicitação ao serviço de diretórios local do país. Caso satisfaça o *request*, a resposta é enviada ao cliente. Caso seja verificado que a solicitação feita não é do domínio administrativo daquele país, é feita uma busca a nível de diretório global, executada pelo sistema DCE.

* **global** → Acoplado também ao X.500, executa a função de busca quando a solicitação feita não está dentro do domínio administrativo a solicitação. Como exemplo, pode-se citar uma solicitação feita pelo Brasil ao EUA. Essa solicitação é semelhante à anterior, porém, ao ser verificado que essa não é de domínio local é repassada automaticamente ao domínio solicitado. Como o serviço local, esse também é um serviço executado pela DCE.

- **chama-remoto** → O serviço de RPC fornecido pela plataforma DCE é baseado exclusivamente na proposição de tornar procedimentos individuais de aplicações serem executadas em qualquer parte da rede. O RPC beneficia o paradigma cliente/servidor, utilizando-se do serviço de conversão de dados que, além de simplificar o desenvolvimento de aplicações distribuídas através da produção de um código origem portável ainda admite que chamadas de procedimento comportem-se como chamadas de procedimento local. Sua facilidade de ativação em tempo de execução possibilita que aplicações possam ser executadas em sistemas múltiplos e heterogêneos.
- **arqdist** → Possibilita que clientes compartilhem de informações em larga escala dentro de um ambiente de computação. Essas informações fluem de qualquer lugar onde estejam armazenadas para qualquer lugar desejável dentro da rede. Esse serviço possibilita: a consistência de dados ao acessar um arquivo em outro nó na rede; o acesso uniforme a arquivos; segurança, pois somente os usuários com permissão terão acesso aos mesmos; confiabilidade e disponibilidade, mantidas através da replicação e de sistemas administrativos para desempenhar rotinas de manutenção dos servidores (*hardware*,*software* e arquivos); melhora do desempenho.
- **dcethread** → O serviço de *threads* do sistema DCE provê facilidades de portabilidade que suporta a programação concorrente, admitindo que uma aplicação execute algumas ações simultaneamente. Esse serviço inclui ainda operações que criam e controlam múltiplos *threads* de execução em um processo, e sincroniza o acesso a dados globais dentro da aplicação. Esse serviço é usado por vários serviços da DCE, como RPC, DFS e outros.

3. **Serviço Responder**

func-respo → aciona o AFI para uma solicitação de uma função durante a execução de outra função. Essa função programa o AFI com todas as funcionalidades que poderão ser invocadas durante a realização de um serviço.

Camada Provedora de Serviços

Esta seção especifica todas as interfaces dos blocos com a suas respectivas operações. Todos esses blocos estão dentro do módulo Provedor_Serviço. Uma de suas funções é o recebimento dos serviços alto nível, para que sejam desmembrados em serviços de baixo nível para a execução.

- **PSAN**
  - **envia** → essa operação tem a finalidade de receber solicitações de serviços alto nível, quebrá-las em serviços baixo nível, e invocar o serviço solicitado.
  - **resposta** → os resultados provenientes do provedor de interfaces de serviços são retornados ao cliente através dessa função, que resgata os resultados e os envia à camada do cliente.
- **PCS**
  - **envia-confirma** → essa função tem a finalidade de enviar a confirmação do serviço à camada do cliente. Inicia-se a operação de acordo com a solicitação proveniente do PSAN. Esta confirmação é obtida através do casamento das requisições com a confirmação proveniente da camada provedora de interfaces de serviços. No caso da estação tiver mais de uma plataforma *midware* que execute o mesmo serviço, além da confirmação, também é obtida quais plataformas que podem executar este serviço.
- **AFI**
  - **filtra** → esta função tem por finalidade filtrar todas as indicações de serviços solicitados pelas plataformas *midware*. As indicações são estabelecidas durante a realização de uma conexão, sendo sua validade também em função do tempo de vida dessa conexão.
  - **indica** → esta função tem a finalidade de enviar respostas remotas aos serviços solicitados ao bloco provedor de serviços de alto nível ou indicações ao módulo de serviços de baixo nível.
- **PSI**
  - **confirma-plat** → esta função tem como objetivo interagir com as plataformas *midware* para verificar a disponibilidade das mesmas para a realização de um determinado serviço solicitado pela camada do cliente. Isto só acontece no caso particular de existir mais de uma plataforma na mesma estação, onde essa

funcionalidade é executada através do envio de solicitações de realização de um serviço, aguardando a confirmação da disponibilidade das plataformas. Caso haja só uma plataforma (caso geral), ela tem como objetivo somente verificar se foi estabelecida a conexão com o sistema que irá executar o serviço, como também enviar a resposta para o PCS.

- **resposta** → determina se o request foi completo, isto é, se a operação foi executada. Obtém assim as respostas dos sistemas ANSA e DCE, enviando-as para o PSAN e para o serviço baixo nível non_confirmed.
- **invoca** → essa função interage com a plataforma *midware*, que realizará o serviço, para que seja invocada a função solicitada pela camada do cliente.
- **indicação** → recebe das plataformas *midware* indicações de funções solicitadas pela estação remota. Essa indicação é enviada ao AFI.

3.4.4 Seqüência de uma Chamada

Essa seção explica como um cliente executa uma solicitação a um dos sistemas acoplados. Isso é feito através de uma seqüência de chamada.

- O cliente chama uma função da biblioteca relacionada com o serviço a ser executado. Caso o serviço seja de alto nível a seqüência será a seguinte:
  - A requisição é realizada, criando-se um request para a MID/API, podendo inclusive não estabelecer qual a plataforma *midware* que irá realizar o serviço.
  - A API recebe essa solicitação e a camada do cliente interage com a camada provedora de serviços enviando todos os dados necessários à conclusão do serviço.
  - A solicitação ao chegar no bloco PSAN que a quebra em serviços baixo nível.
  - A camada provedora de interfaces interage com esta plataforma para a realização do serviço.
  - O PCS realiza o casamento da confirmação com a requisição no estabelecimento da conexão, e envia à camada do cliente a confirmação do serviço a ser executado.
  - O serviço é executado e retornado o resultado à camada do cliente, através da bloco provedor de serviços de alto nível.
- Caso a solicitação do cliente seja de um serviço baixo nível, a seguinte seqüência será realizada:

- A solicitação da realização de um serviço baixo nível é realizada diretamente da camada do cliente para o bloco PIS. Esses serviços estão diretamente relacionados com uma determinada plataforma *midware*.
- O procedimento do bloco PIS junto as plataformas *midware* é o mesmo feito pelos serviços alto nível.
- A confirmação do estabelecimento da conexão também é informada diretamente à camada do cliente.

A figura 3.8 mostra o caminho percorrido pela solicitação desde o cliente até os sistemas ANSA e DCE e também o retorno do resultado até o solicitante.

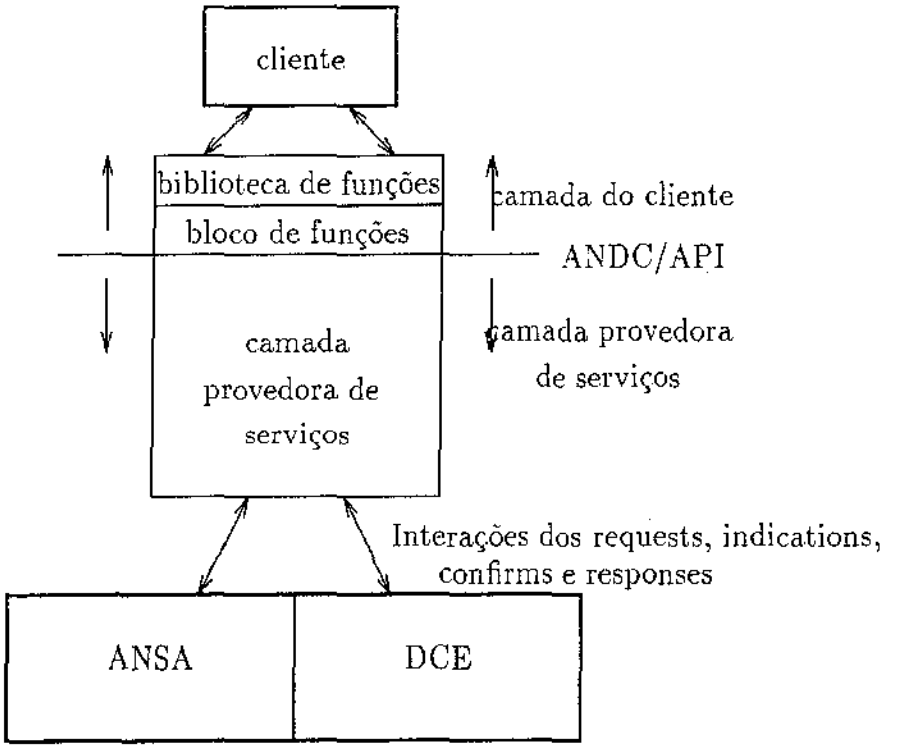


Figura 3.8: Caminhos Percorridos pelos Requests, Indications, Responses e Confirms

Capítulo 4

CORBA

4.1 Introdução

Da mesma forma que ao analisarmos as APIs no capítulo 3, onde vimos as plataformas *midware* ANSA[1] e DCE[2], veremos neste capítulo a **CORBA**[3], que é uma das soluções que pode ser utilizada para resolver os problemas de comunicação e distribuição em relação a sistemas distribuídos apresentados no capítulo 1.

A CORBA, desenvolvida no fim dos anos 80 e colocada para conhecimento público no ano de 1991, é uma plataforma para ambientes de serviços abertos que provê mecanismos, através dos quais, objetos podem comunicar-se de forma transparente por meio de solicitações e respostas. Sua arquitetura define uma estrutura de suporte à comunicação de objetos em sistemas distribuídos heterogêneos, proporcionando a integração entre uma grande variedade de sistemas de objetos. Um sistema de objetos é um conjunto de objetos que isola o cliente do servidor. [CCODIC93]

A CORBA baseia-se no **modelo de referência OMA**[4] da OMG[5], que identifica e caracteriza componentes, interfaces e protocolos que o compõe. Esse modelo é composto por quatro partes (veja figura4.1)[Cen92a, Cen92b, Cen93b]:

- **Objetos de Aplicação** - são objetos específicos de aplicações para usários finais.
- **Facilidades Comuns** - é uma coleção de serviços provendo capacidades, de propósito geral, que podem ser utilizadas por algumas aplicações.
- **Serviço de Objetos (Object Services)** - é uma coleção de serviços (interfaces e objetos) que suportam funções básicas para a utilização e implementação de objetos.

[1]Advanced Networked Systems Architecture
[2]Distributed Computing Environment
[3]Common Object Request Broker Architecture
[4]Object Management Architecture
[5]Object Management Group

- **ORB (Object Request Broker)** - possibilita que objetos enviem e recebam solicitações e respostas de uma forma transparente em um ambiente distribuído.

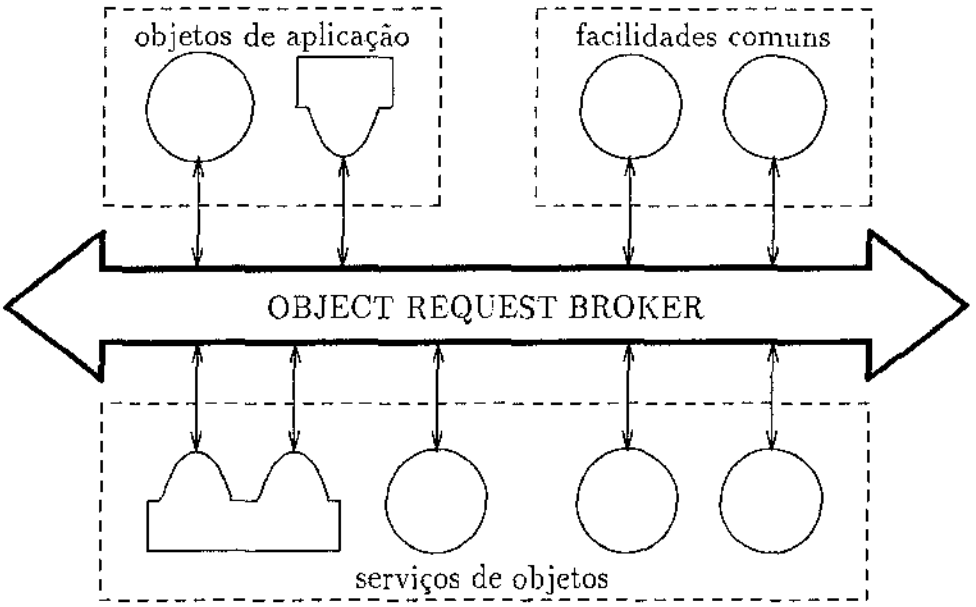


Figura 4.1: RM-OMA

Os serviços de objetos citados anteriormente, como o *trader* [dPLJ94], podem ser acessados por APIs citadas no capítulo 3, porém nesse capítulo os serviços de objetos podem ser acessados sem a utilização dessas APIs, sendo utlizado o ORB para isso.

A especificação CORBA é bastante flexível, pois possibilita que projetistas implementem diferentes ORBs, como também os objetos que o compõe, de acordo com as necessidades dos seus sistemas, utilizando-se de mecanismos para melhorar o desempenho e a funcionalidade da mesma (independente de quaisquer arquitetura de *hardware* e ambientes de *software*). Toda essa flexibilidade é válida desde que o princípio fundamental da CORBA (o ORB deve prover somente o mecanismo básico para o estabelecimento de solicitações e o recebimento de respostas entre objetos, deixando outros serviços para outros objetos) seja mantido, porém essa flexibilidade ocasiona problemas como a interoperabilidade, onde protocolos de comunicação e padrões para solicitações são definidos para cada tipo de implementação. Logo, a CORBA passa a ser uma boa solução para a comunicação entre objetos, apesar da possibilidade de haver vários tipos de implementação e a dificuldade de interoperar esses sistemas.

A escolha da CORBA como a plataforma inicial no desenvolvimento da camada *Middleware* da plataforma *Multiware* baseia-se em sua simplicidade de arquitetura, por possuir

os conceitos básicos da especificação ODP e também por admitir a adição de novos objetos (serviços de objetos) sem afetar as características do sistema.

4.2 Modelo de Objetos da CORBA

O modelo de objetos provê uma apresentação organizada dos conceitos e uma terminologia dos objetos, definindo um modelo parcial para computação que engloba todas as características principais dos objetos. A CORBA utiliza um modelo clássico, onde um cliente envia uma mensagem para um objeto, esse a recebe, identifica, interpreta, retira os seus parâmetros e excuta a operação retornando a resposta à origem[CCODIC93].

4.2.1 Semântica dos Objetos

Os objetos da CORBA são identificados através de referências de objeto. Essas referências são únicas para cada objeto, sendo que a sua criação ou destruição é uma conseqüência das requisições feitas pelo cliente. Os objetos são acessados através de interfaces bem definidas, compostas por operações e tipos de dados utilizados por essas operações. Essa semântica é descrita através dos seguintes objetos:

- **cliente** → cliente de um serviço é uma entidade capaz de requisitar um serviço.
- **implementação de objeto** → é uma entidade identificavél que provê um ou mais serviços, que podem ser solicitados pelo cliente.
- **requisição** → é uma interação entre um cliente e um objeto de implementação. É composta por uma operação, um objeto alvo e zero ou mais parâmetros.
- **referência de objeto** → identifica o objeto dentro do ORB.
- **tipo** → é uma entidade definida que restringe um determinado parâmetro ou caracteriza um possível resultado.
- **interface** → descrição de um conjunto de possíveis operações que um cliente pode requisitar a um objeto.

Todos os parâmetros de uma requisição são valores, definidos por tipos compostos ou básicos. A figura 4.2 mostra a árvore de tipos definidos pela especificação CORBA.

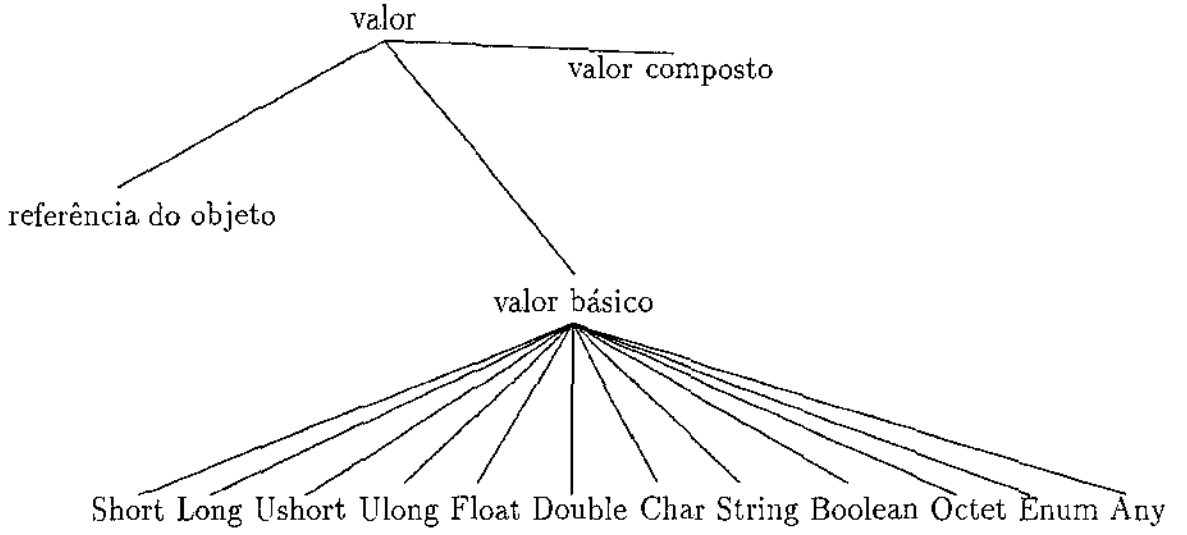


Figura 4.2: Árvore de tipos

4.3 Arquitetura

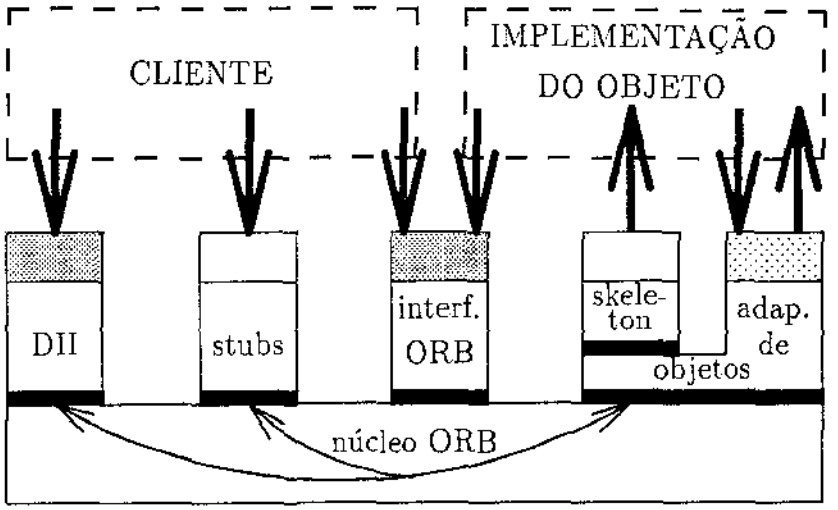
A arquitetura é baseada no paradgima cliente/servidor, onde uma solicitação feita por um cliente a um objeto, é executada independente do local do objeto. A finalidade da arquitetura é prover meios para direcionar e sincronizar a comunicação entre cliente e servidor, possibilitando que clientes com diferentes propósitos acessem a uma mesma implementação do objeto (execução concorrente)[CCODIC93].

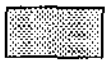
A figura 4.3 mostra a arquitetura geral da CORBA com os seus componentes básicos, como também o caminho percorrido pela solicitação desde o cliente até a implementação do objeto.

Para um melhor compreendimento do funcionamento dessa arquitetura, vamos percorrer o caminho feito por uma requisição de um cliente, desde a sua origem até a sua execução, retornando resultados quando houverem.

- **Invocação** → o cliente pode invocar um serviço de duas formas:
  - **estaticamente** - através de *stubs* específicos de uma interface. Cada interface possui um conjunto de *stubs* correspondentes, gerados por um compilador IDL[6], que ao receber uma invocação do cliente, empacota os parâmetros da operação, e os transmite ao núcleo do ORB. Os resultados recebidos são desempacotados e entregues ao cliente.

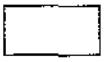
[6]Interface Definition Language




interface idêntica para todas as impementações


interface dependente do ORB


podem haver muitos adaptadores de objetos


há um stub e um skeleton para cada tipo de objeto


interface de chamada ascendente


interface de chamada comum

Figura 4.3: Seqüência de Invocação

- **dinamicamente** - essa é a forma utilizada quando não se conhece a interface em tempo de compilação, utilizando a interface de invocação dinâmica (DII) para realizar a invocação de uma requisição de serviços. A semântica utilizada nesse tipo de invocação é a mesma utilizada numa invocação estática.

A figura 4.3 indica as possibilidades da realização de uma requisição, tanto estática quanto dinâmica através das respectivas interfaces.

- **execução** → no servidor, a requisição chega através do adaptador de objetos e *skeletons*, que repassam a chamada para a implementação do objeto (servidor), que executa o serviço, repassando seus resultados aos *skeletons*, que os enviam ao cliente através do adaptador de objetos. O adaptador de objetos tem a finalidade de prover serviços de suporte à implementação do objeto, através de uma interface para o ORB. Os *skeletons* correspondem aos *stubs* do lado do servidor. Eles recebem os parâmetros e os passam a implementação do objeto, como também recebem os resultados dessa impementação e os mandam para o cliente. A figura 4.3 mostra a chamada da solicitação ao servidor, e o repasse dos resultados.

4.4 Componentes do Sistema

Nessa seção são descritos todos os componentes da CORBA, como o ORB e suas interfaces, o cliente, etc. Esses componentes são objetos que executam funções específicas dentro desse contexto, colaborando para um melhor desempenho da CORBA[CCODIC93].

4.4.1 Cliente

Cliente de um objeto é aquele que através de uma referência do objeto (informação necessária para especificar o objeto dentro da ORB) pode invocar operações estática ou dinamicamente. O cliente tem conhecimento lógico da estrutura do objeto a ser invocado de acordo com sua interface, como também conhece o seu comportamento através das invocações. Os clientes vêem os objetos e as interfaces ORB através de um mapeamento de linguagem[CCODIC93, Cen93a, Inc93], possibilitando que conversem entre si numa mesma linguagem.

Os clientes são totalmente portáveis, isto é, são capazes de trabalhar independente da implementação ORB que estiverem operando. Sua estrutura (vide figura 4.4) pode incluir um ou mais conjuntos de *stubs* e a interface de invocação dinâmica. A invocação inicia quando o cliente acessa um *stub* específico de um objeto (como se fosse acessar uma rotina de biblioteca de programa) ou a interface de invocação dinâmica (DII), e passa a referência do objeto para as rotinas *stubs* ou para a DII. Esses objetos interagem com o

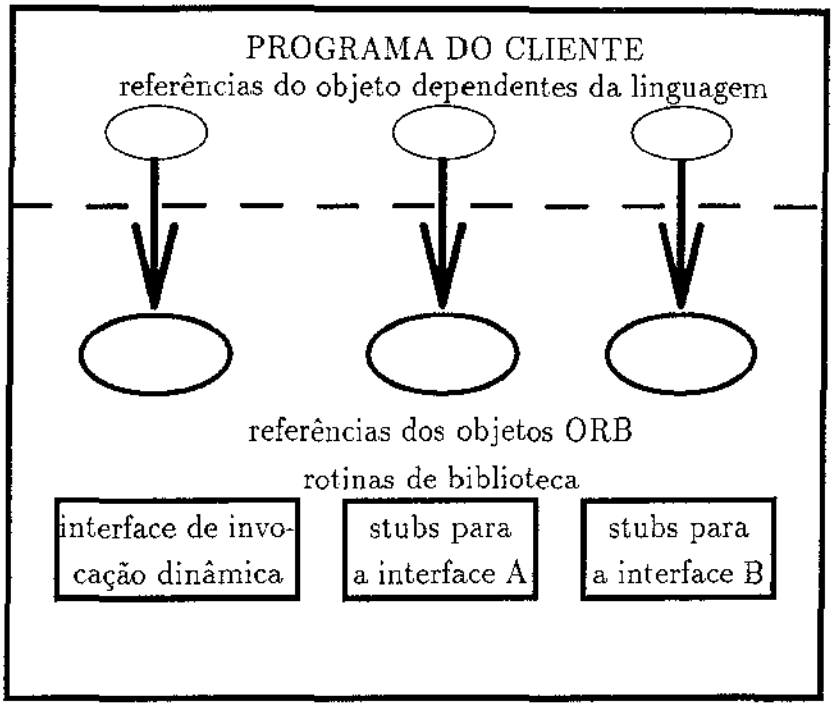


Figura 4.4: Estrutura do Cliente

ORB utilizando a referência do objeto na composição da requisição. A figura 4.5 mostra como as informações referentes a interfaces e *stubs* estão disponíveis para o cliente.

4.4.2 Implementação de Objetos

A implementação do objeto provê o estado atual e o comportamento de um objeto. Sua estrutura pode ser organizada de várias formas, com diferentes métodos e dados.

A implementação do objeto interage diversas vêzes com o ORB para obter a sua identidade, criar novos objetos ou obter serviços dependentes da ORB. Ao estabelecer-se uma invocação, o núcleo do ORB, o adaptador de objetos e o *skeleton* interagem de forma que a chamada seja feita ao método correto. Esses métodos, chamados de forma ascendente pelos *skeletons*, utilizam-se de rotinas para realizarem seus serviços. A implementação do objeto, como o cliente, independem da implementação do ORB, e também da forma como o cliente a invoca (estática ou dinâmica).Todo esse mecanismo é o responsável pela execução local da implementação. A figura 4.6 mostra a estrutura típica da impementação do objeto.

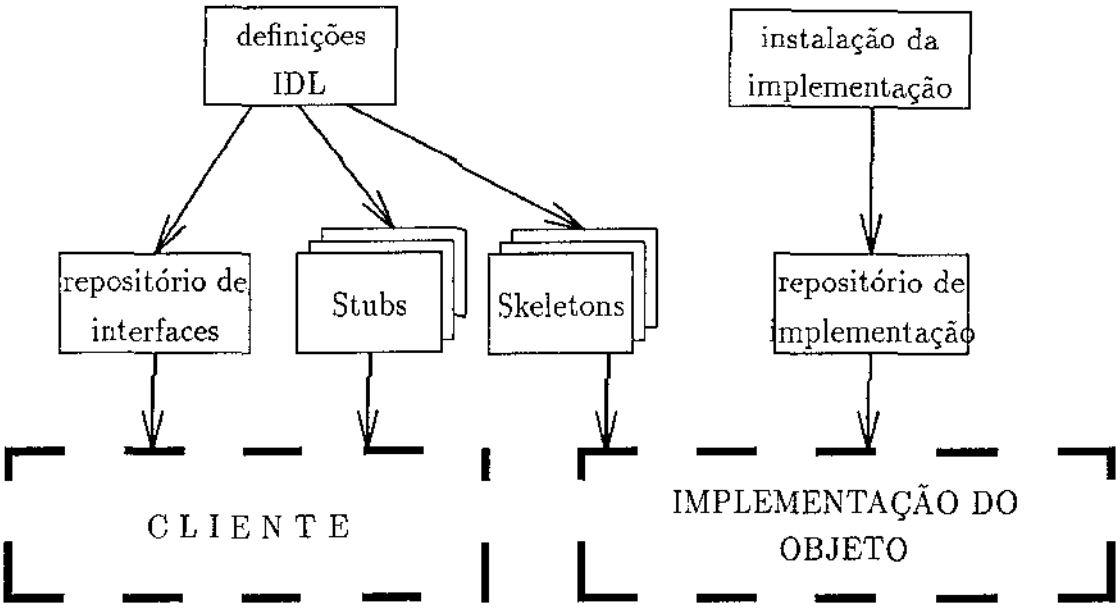


Figura 4.5: Repositórios de Implementação e Interfaces

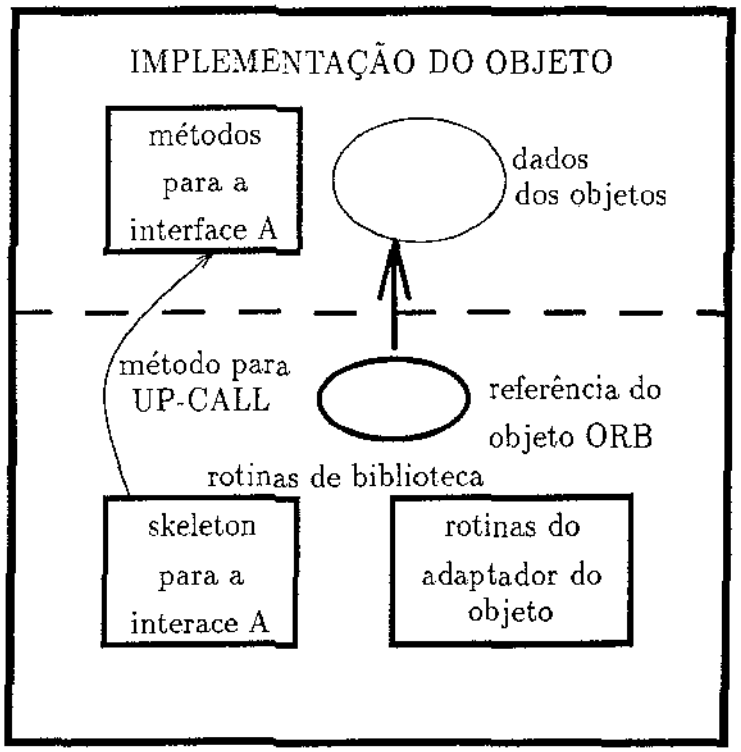


Figura 4.6: Estrutura da Implementação do Objeto

4.4.3 Repositório de Implementações

O repositório de implementações possui informações que possibiltam o ORB localizar e ativar as implementações de objetos. A instalação de implementações e o controle de políticas relativas a ativação e a execução de implementações de objetos é feita através de operações no repositório de implementações.

Embora a maioria das informações contidas no repositório de implementação serem relativas ao ORB e ao ambiente operacional, ele também possibilita o registro de informações adicionais relativas aos objetos do ORB, controle administrativo, ou a alocação de serviços.

4.4.4 Object Request Broker

O ORB, na arquitetura, não pode ser pensado e implementado como um único componente, mas definido através de suas várias interfaces. Uma interface ORB pode conter operações que são válidas para diversas implementações ORBs, ou específicas a um tipo particular de objetos, ou ainda específicos a estilos de implementação de objetos. Com isso, diferentes ORBs podem escolher diferentes formas de implementação, e junto com os compiladores IDL, repositórios e vários adaptadores de objetos, provêem um conjunto de serviços para clientes e implementações de objetos.

O núcleo do ORB provê a base de comunicação que possibilita o encaminhamento das solicitações, como também o retorno dos resultados. Essa complexidade de comunicação dentro do núcleo, ou entre diferentes núcleos, torna-se transparente aos olhos do sistema através das interfaces providas pelos componentes do ORB.

De uma forma geral, as interfaces que definem o ORB são as seguintes:

- **Stub Client** → *stub* é um procedimento local que corresponde a uma única operação. O *stub* depende da interface do objeto alvo, isto é, existe um *stub* para cada *skeleton* do lado do servidor.

  As operações especificadas nos *stubs* são definidas pela IDL. Com isso, existe a necessidade de se realizar um mapeamento para uma linguagem compatível com a do cliente. Esse procedimento deve ser feito, pois a CORBA independe da linguagem de programação usada na construção de clientes e implementações de objetos. O mapeamento é o mesmo para todas as implementações da CORBA, e inclui uma estrutura da interface *stub client*, da interface de invocação dinâmica, do *skeleton*, da interface ORB e dos adaptadores de objetos, possibilitando assim que os programadores acessem as funcionalidades oferecidas pelo ORB através de suas particulares linguagens.

  Feito o mapeamento, os *stubs* podem ser invocados estaticamente pelos clientes, e assim realizarem suas chamadas ao ORB através de interfaces privadas ao núcleo ORB.

- **Interface de Invocação Dinâmica** → essa interface tem a finalidade de criar e invocar dinamicamente solicitações de serviços a serem executados em determinados objetos. Ela possibilita o acesso a objetos cujas interfaces são desconhecidas em tempo de compilação. Durante sua operação, várias chamadas ao ORB são realizadas, de forma que todas as informações necessárias (ex:buscar no repositório de interfaces (seção 4.4.5) a definição de interfaces desejadas) para compor a requisição do cliente sejam obtidas. Pronta a requisição, a interface de invocação dinâmica realizará a invocação do serviço junto a implementação do objeto.

- **Interface ORB** → é uma interface para as funções ORBs que independem do adaptador de objetos para serem chamadas. É uma interface que todo ORB, independente de sua implementação, deve possuir. Possui pequena funcionalidade, provendo apenas algumas operações, como converter referências de objetos em *strings* e vice-versa (para facilitar o armazenamento das referências). Essas operações são comuns para qualquer tipo de objeto, podendo ser utilizados tanto por clientes quanto por servidores (implementações de objetos).

- **Skeleton** → é uma interface up-call, posicionada entre o núcleo do ORB e os métodos que implementam cada tipo de objeto. É utilizada na recepção das requisições de clientes e nas chamadas dos correspondentes métodos. Existe uma concordânncia entre o *skeleton* e a implementação do objeto, de forma que possam ser invocados os métodos referentes à implementação desejada. Toda essa funcionalidade recebe a ajuda do adaptador de objetos, que ativa antes o objeto onde será realizada a implementação. O *skeleton* depende das definições IDL, tendo para cada *stub* um *skeleton* correspondente. Porém, nem para todo *skeleton* existirá um *stub*, pois existe a possibilidade de uma invocação dinâmica por parte do cliente.

- **Adaptador de Objetos** → é uma interface que provê o acesso a um conjunto de funções necessárias a uma implementação do objeto, de forma que possibilite o seu correto funcionamento. Essas funções são:

  - Geração e interpretação de referências de objetos.
  - Autenticação do cliente.
  - Ativação e desativação de objetos individuais.
  - Invocação de métodos através de *skeletons*.

  Para que haja a interação entre as implementações de objetos, *skeletons* e adaptadores de objeto, deve-se inicialmente ativar a implementação referente a requisição feita. Executada as rotinas de inicialização, a implementação é registrada junto ao adaptador de objejtos. Esse adaptador pede que a implementação do objeto que

realizará o serviço seja ativado. O *skeleton* invoca o método na implementação e o adaptador de objetos pode acessar alguns dos seus serviços para executar o método invocado. A figura 4.7 mostra a seqüência do funcionamento desse mecanismo.

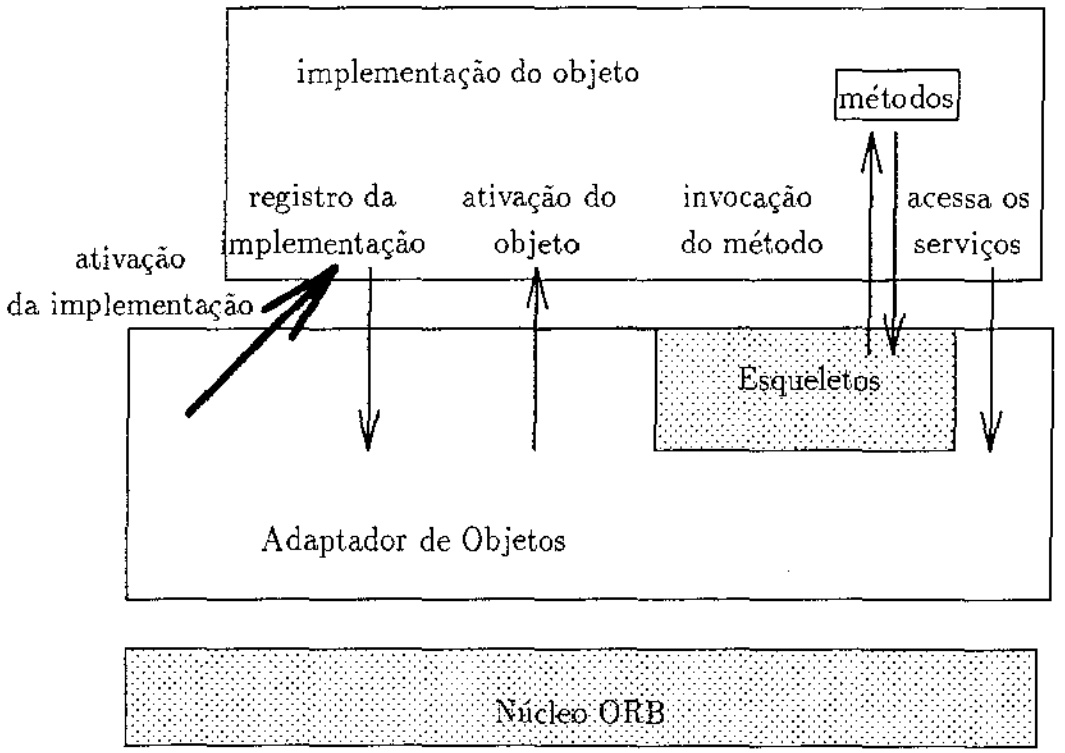


Figura 4.7: Estrutura e operacionabilidade do adaptador de objetos

4.4.5 Repositório de Interfaces

O repositório de interfaces é um serviço que provê o armazenamento consistente de definições de interfaces. A informação do repositório pode ser utilizada pelo ORB para a composição das requisições, além disso possibilita ao programa encontrar o objeto cuja interface não foi conhecida em tempo de compilação. Também como o repositório de implementações, ele armazena informações adicionais associadas com interfaces para objetos ORB (biblioteca de *stubs* ou *skeletons*, rotinas de formato para alguns tipos de objetos,etc).

4.4.6 IDL

A linguagem IDL (Interface Definition Language) é usada para descrever as interfaces que estão disponíveis para os objetos clientes, como também as implementações dos objetos

que podem ser utilizadas. Uma definição de interface escrita na linguagem IDL define a interface por completo, especificando totalmente cada parâmetro das operações.

A interface IDL fornece informações necessárias ao objeto cliente para que esse possa invocar as operações especificadas pelas interfaces.

- IDL obedece as mesmas regras léxicas da linguagem C++, embora algumas palavras chaves sejam introduzidas para suportar o conceito de distribuição.
- Provê suporte total para as características de pré-processamento C++.
- Sua gramática é um subconjunto do padrão ANSI C++, com construtores adicionais para suportar os mecanismos de invocação de operações.
- É uma linguagem declarativa, suporta a sintaxe C++ para constantes, tipos e declarações de operações.
- Possui recursos que possibilitam a criação de novos tipos de dados a partir de tipos básicos e compostos já definidos.
- Define também constantes, módulos, interfaces, operações e exceções.
- Os módulos podem englobar outras definições e são declarados especificando seu nome e uma palavra chave. É utilizado para definir um escopo de identificadores IDL.
- As interfaces são declaradas da mesma forma que os módulos. A IDL suporta a herança múltipla de interfaces, desde que não haja operações iguais com parâmetros diferentes nas interfaces herdadas.
- A declaração de constantes faz uso de outras constantes para realizar operações com valores literais especificados.
- Suporta operações com números inteiros e ponto flutuante.
- Atributos das operações são do tipo entrada ou saída.

Para concluirmos, a IDL é o meio pelo qual uma implementação de um determinado objeto oferece serviços aos clientes através das operações disponíveis e também informa como elas serão invocadas.

4.5 Interoperabilidade

A interoperabilidade entre diferentes sistemas de objetos e ORBs é um meta explícita da CORBA. Devida à grande diversidade de técnicas de implementação de ORBs essa meta de interoperabilidade torna-se bastante difícil. Para sanar esse problema, estudos substanciais baseados em experiênncias de produtos já comercializados na área de conexão de redes (que utilizam diferentes protocolos) estão sendo realizados.

No geral não existem protocolos que possam satisfazer a necessidade de realizar a interoperabilidade. Existem sim alguns ambientes em que múltiplos protocolos coexistem. Outras sugestões são oferecidas em [HOL+93].

A interoperabilidade entre sistemas de objetos com diferentes adaptadores de objetos pode ser obtida através do ORB. Ele possibilita que as requisições cheguem corretamente ao adaptador do objeto (caso ele esteja cadastrado), independente do número ou o tipo de adaptadores existentes no ORB.

A figura 4.8 mostra os três possíveis cenários em que múltiplos ORBs coexistem.

- ORB 1 é implementado nas máquinas A e B. Ambas implementações usam a mesma referência de objeto e o mecanismo de comunicação. Com isso a referência do objeto pode ser livremente passada da máquina A para a B. Esse caso é considerado o mais simples. Um único ORB implementado em duas máquinas.

- Na máquina A o mesmo cliente pode ter alguns objetos implementados pelo ORB 1 e pelo ORB 2. Ao ser realizada um invocação, a referência do objeto é passado como parâmetro de um ORB para o outro.

- Entre as máquinas A e C, não existem ORBs comuns. Nesse caso existe a necessidade de construir talvez um *gateway* para realizar a passagem entre referências de objetos. Outras sugestões são dadas na [HOL+93]. Quanto a este item, está sendo desenvolvido um trabalho [Zuq95], com término previsto para o ano de 1995.

4.6 Análise Comparativa

Dentro dessa seção realiza-se uma análise comparativa entre a utilização da arquitetura CORBA e a utlização da MID/API, como duas abordagens para acesso a objetos em ambientes distribuídos. São analisadas as suas funcionalidades, levando-se em consideração todas as vantagens e desvantagens da utilização ou não de uma ou outra forma de acessar os objetos distribuídos dentro de um sistema. Essa análise também utiliza-se desses parâmetros para uma avaliação da sua utilização separadamente ou dentro da plataforma *Multiware*.

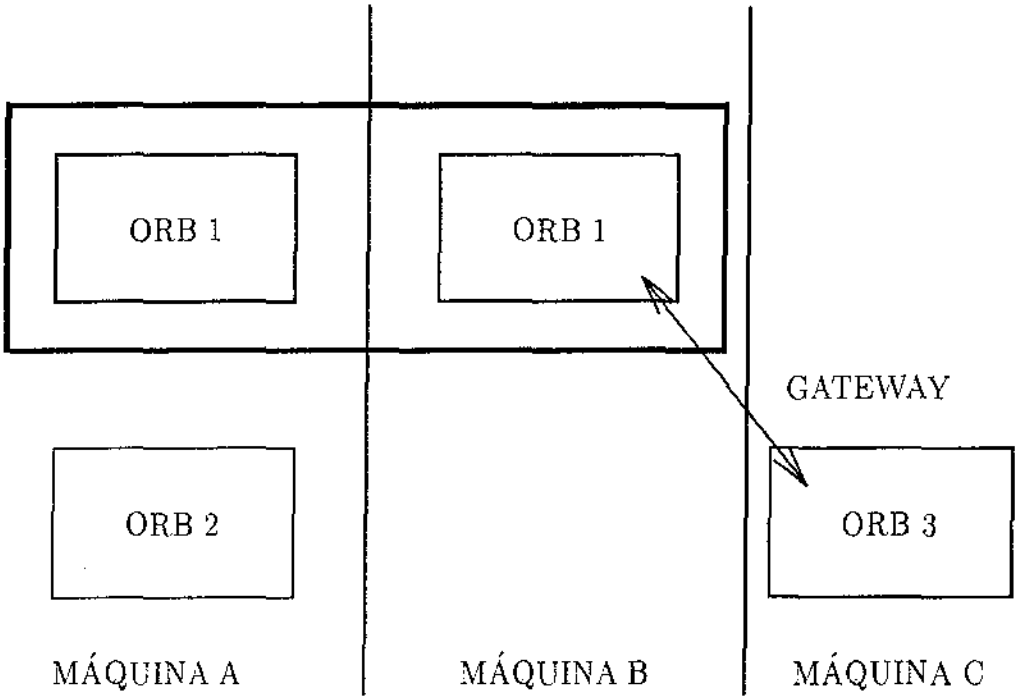


Figura 4.8: Múltiplos ORBs

Durante essa avaliação alguns aspectos são levados em consideração a fim de que se possa realizar uma comparação, buscando-se observar qual abordagem mais vantajosa dentro de um sistema de objetos. Esses aspectos juntamente com suas respectivas avaliações são as seguintes:

- **Portabilidade** → Quanto ao item portabilidade, a MID/API possui maior independência das plataformas *midware* com que opera, realizando algumas restrições quanto ao uso de diferentes e múltiplas plataformas. Essas plataformas por conseguinte independem, das plataformas (sistemas operacionais e *hardware*) que dão suporte.

  No caso da CORBA essa portabilidade é conseguida através das interfaces da ORB, possibilitando tanto aos clientes e implementações de objetos quanto a outros objetos da arquitetura a independência da forma como o ORB foi implementado. Essa implementação do ORB varia de acordo com as necessidades de seu projetista, porém, independe também das plataformas (sistema operacional e *hardware*) que o suporta.

  Pode-se concluir então que ambas abordagens possuem características semelhantes quanto à portabilidade.

- **Disponibilidade de Serviços** → Quanto à disponibilidade de serviços, a MID/API

restringe-se aos serviços oferecidos pelas plataformas comerciais, isto é, restringe-se ao seu ambiente de computação.

Quanto a CORBA, essa disponibilidade de serviços fica restrita inicialmente aos serviços disponíveis pelo sistema, cuja interfaces são definidas em tempo de compilação no repositório de interfaces. A vantagem da CORBA sobre a MID/API, vem em função da possibilidade do incremento de serviços, através da criação de serviços ou até mesmo a utilização de outros serviços de outros sistemas na conclusão de uma solicitação (interoperabilidade), sem que haja a dependência da inclusão de uma outra plataforma comercial como acontece na MID/API.

- **Invocação de Operações** → Quanto ao ítem invocação da operação a MID/API necessita de estabelecimento de *bindings* de linguagem para a realização de invocações por clientes. Caso o cliente esteja na mesma linguagem da interface, essa invocação é direta, através do estabelecimento de uma requisição. Um fator que favorece a utilização da MID/API, é a possibilidade de um cliente especificar em qual sistema um determinado serviço poderá ser executado.

  A CORBA por si possui duas formas de estabelecer essa invocação, estaticamente ou dinamicamente. Em ambos os casos o processo é relativamente simples, pois esses *requests* são estabelecidos de uma forma concisa através de *stubs* ou da interface de invocação dinâmica, a qual utiliza-se das definições de interfaces disponíveis no repositório de interfaces. Como na MID/API, mapeamentos de linguagens também são utilizados, onde a CORBA já possui dois mapeamentos para linguagens de programação C e C++.

  A análise desse ítem torna-se um quanto similar, pois em ambos os casos a forma de invocar operações são bem semelhantes. Dá-se um maior valor a CORBA devido a flexibilidade que o cliente terá para invocar a sua operação.

- **Arquitetura** → As arquiteturas das duas abordagens são expressivamente simples. Os módulos que compõe a MID/API são em menor número que a CORBA, porém a complexidade de execução das operações de cada um desses módulos é mais complexa, tendo em vista que a MID/API acumula um número maior de funções a serem executadas.

  Um exemplo dessa maior complexidade pode ser verificada quando da confecção do *request* pela API. O módulo provedor de serviços da API tem como função a negociação com as funções de baixo nível, para realizar o serviço desejado. No caso da CORBA isso fica bem mais simples, pois após o *request* ter sido feito, ele simplesmente será encaminhado para a sua execução via o ORB, posteriormente retornando seus resultados, caso existam, para o cliente.

- **Desenvolvimento** → Quanto ao desenvolvimento dos projetos, pode-se dizer que a MID/API trabalha sobre produtos já comercializados, como a DCE, que é orientada a processos, é uma ferramenta bastante difundida, enquanto produtos orientados a objetos ainda estão em desenvolvimento.

Uma outra forma de se comparar essas duas abordagens, foi através de uma avaliação de seus comportamentos perante ao projeto da camada *Middleware* da plataforma *Multiware*, citando algumas das vantagens da utilização da CORBA e não da MID/API.

- **Simplicidade** - A idéia de se utilizar a CORBA como um componente do projeto vem em função de sua simplicidade na implementação e também na utilização pelos clientes, em relação a MID/API.

- **Broker** - o seu *broker* é de fácil implementação e utilização, possibilitando o envio rápido e seguro das solicitações do cliente, como também suas respostas.

- Ela é baseada em **orientação a objetos**, possibilitando assim uma maior visualização de todo o sistema, e possibilitando também o acréscimo (implementação) de novos objetos a sua estrutura, sem que afete o seu desempenho.

- A **CORBA** basicamente acabou com a necessidade de utilizações de APIs para a realização de invocação de operações, pois o seu ORB realiza essa funcionalidade.

- A possibilidade de interconectar e interoperar sistemas.

A utilização da CORBA na camada *middleware* torna-se desfavorável quando deseja-se realizar um serviço específico de uma plataforma de serviço. A CORBA não possui a capacidade oferecida pela MID/API que possibilita o cliente especificar no seu *request* a plataforma de serviço que a aplicação deseja realizar o seu serviço (embora normalmente tenta-se para cada estação uma plataforma).

Capítulo 5

Esquema de Acesso aos Repositórios de Interfaces e Implementações

5.1 Introdução

Nesse capítulo, estamos apresentando um esquema de acesso aos repositórios de interfaces e implementação, componentes da ORB, parte integrante da arquitetura CORBA [CCODIC93]. Esta arquitetura é um dos componentes da camada *middleware* da plataforma *multiware*, e foi escolhida para o desenvolvimento inicial da plataforma em função de possuir uma estrutura simples e coesa, e de fácil adaptação devido a algumas características ODP que contém.

A especificação CORBA prescreve vários objetos que necessitam ser analisados e talvez restruturados para que assim possam servir da melhor forma possível aos propósitos de sistemas distribuídos em geral, como por exemplo a plataforma *multiware*. Um desses objetos é o repositório de interfaces, um serviço de objeto (*object service*) da ORB que tem a finalidade de prover um armazenamento consistente de definições de interfaces. Um outro serviço dc objetos com que trabalhamos é o repositório de implementações, que armazena os endereços necessários ao ORB para localizar e ativar as implementações do objeto. Baseado nesse princípio, criamos um esquema de acesso que tem a finalidade de tornar o acesso aos repositórios de interfaces e implementações mais simples e eficiente. Este esquema é basicamente composto por quatro objetos: o objeto repositório, o objeto de acesso, e os repositórios de interfaces e implementações. O objeto repositório possui um conjunto comum de operações oferecidas em sua interface, que possibilita o gerenciamento e o acesso aos objetos dentro do repositório de interfaces. O objeto de acesso é aquele que tem por finalidade receber do ORB uma solicitação de algum serviço no objeto repositório, direcionando essa solicitação ao objeto repositório específico onde será realizado o serviço. O objeto acesso, através do objeto repositório, é responsável também pelo acesso a outros

repositórios dentro ou fora do ambiente em que está posicionado. A figura 5.1 apresenta um modelo geral de como o ORB acessa o repositório de interfaces e implementações através do esquema de acesso.

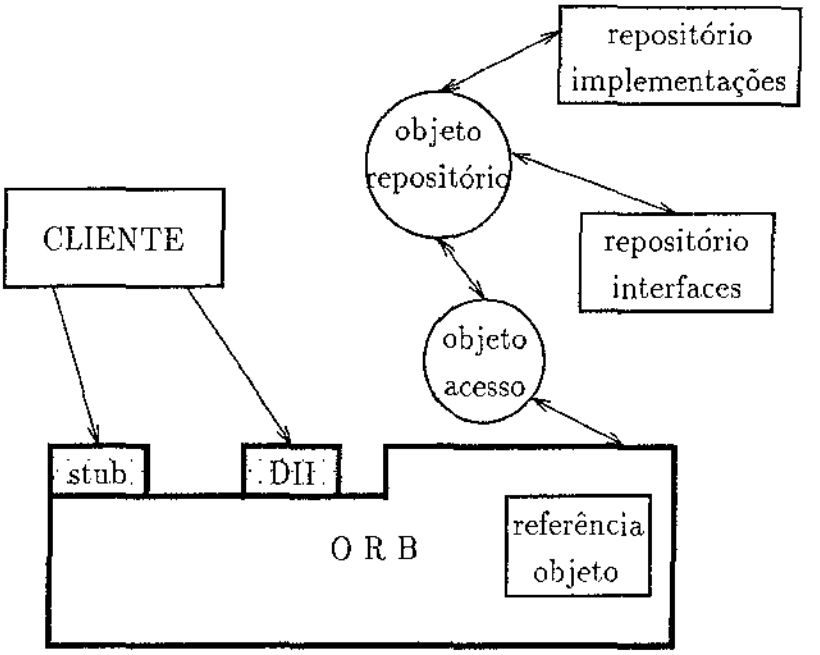


Figura 5.1: Estrutura de interações entre o ORB e o repositório de interfaces

5.2 Estrutura

A estrutura do esquema de acesso possui além dos objetos citados na introdução, também uma base de dados (arquivo de acesso) e um *cache* de memória. O arquivo de acesso é local ao ambiente ORB e ligado a um objeto de acesso, fornecendo a este informações pertinentes ao tipo da interface e da operação, como também a localização do repositório de interfaces a qual pertencem. Esse arquivo é utilizado somente quando existe um repositório de interfaces em cada nó do sistema.Esse arquivo é utilizado somente quando existe um repositório de interfaces em cada nó do sistema. O *cache* de memória une-se a um repositório de interfaces, como também a um objeto repositório, que tem por finalidade armazenar temporariamente informações pertinentes aos escopo da máquina que estiver utilizando um determinado repositório. A idéia de se colocar o repositório de implementações junto ao objeto repositório, é exclusivamente para se aproveitar as funções existentes neste objeto na obtenção das informações existentes no repositório de implemantações. A figura 5.2 apresenta a arquitetura do esquema de acesso ao repositório de

interfaces.

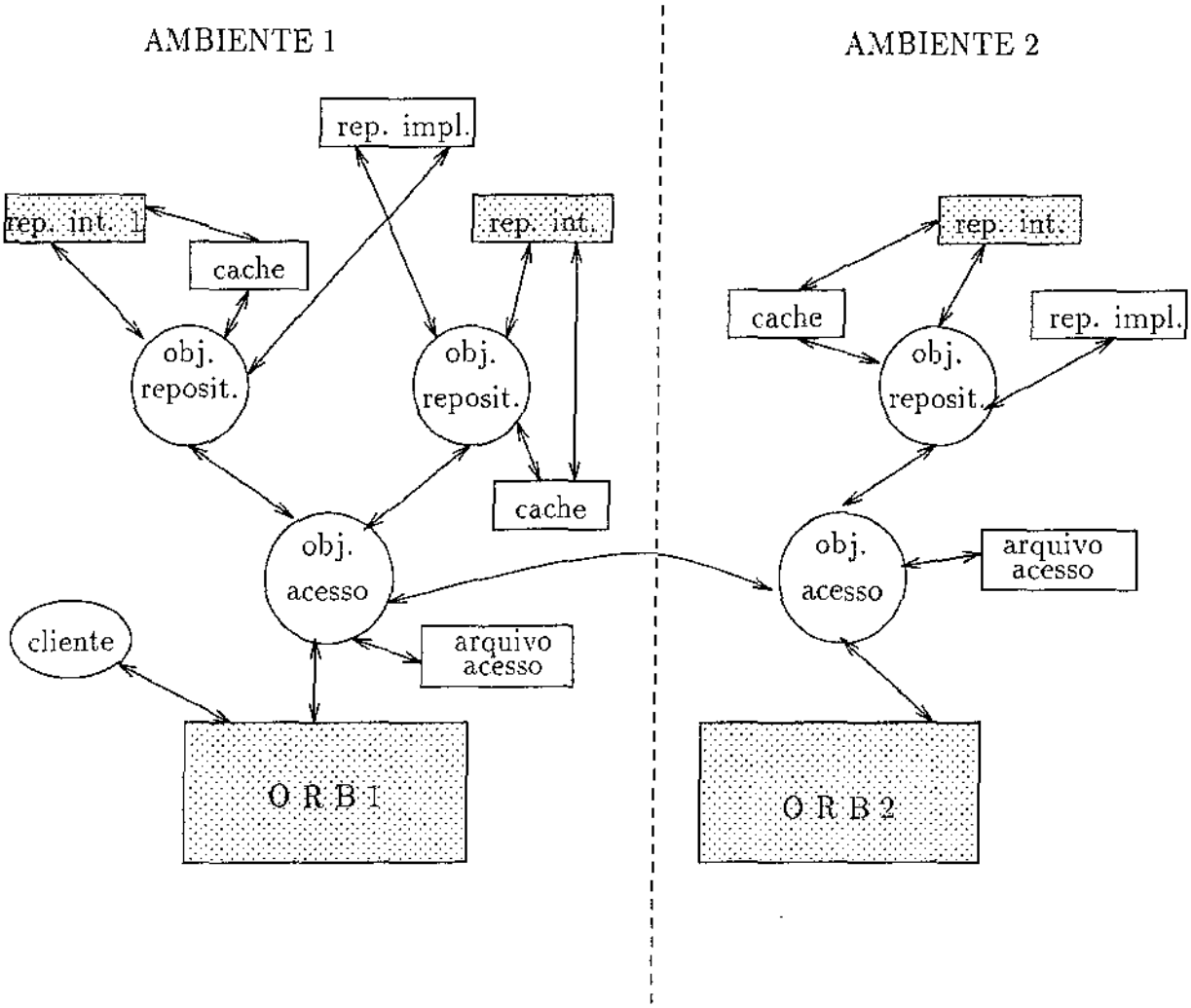


Figura 5.2: Arquitetura do Esquema de Acesso

Para que o esquema pudesse satisfazer a todas ou quase todas as necessidades de gerenciamento e acesso ao repositório de interfaces, levamos em consideração alguns fatores que serão citados a seguir:

- **Tipos e formas de armazenamento** → as informações podem ser armazenadas em banco de dados[ABD+89] ou em sistemas de arquivos, e são estruturadas hierarquicamente em objetos.
- **Tipos de informação** → cada repositório de interfaces possui um determinado tipo de informação armazenada. Esses variam de acordo com o escopo da máquina.

- **Funcionalidade** → a fim de prover uma melhor funcionalidade para CORBA, o esquema possibilita a interoperabilidade entre repositórios de interfaces em diferentes ambientes.

Baseado nesses fatores, o esquema de acesso foi projetado para atuar em um ambiente ORB como um todo. Logo, existe um único esquema de acesso para cada ambiente ORB do sistema, onde existe um único objeto de acesso com um arquivo de acesso, e um ou mais objetos repositórios, dependendo da complexidade do sistema. Cada objeto repositório é ligado a um repositório de interfaces, e todos esses objetos ligados ao único repositório de implementações do sistema.

A vantagem da utilização de um objeto de acesso por esquema, vem em função de um melhor encaminhamento de todas as solicitações feitas ao repositório de interfaces que chegam pela ORB, facilitando assim a aquisição das informações sobre os repositórios em um ambiente, utilizando-se o arquivo de acesso. Com isso otimizamos e reduzimos o custo operacional do sistema (implementação, gerenciamento de solicitações de serviços). Porém, como desvantagem pode-se citar a centralização da passagem de todos essas solicitações por um único objeto em uma única máquina, reduzindo a confiabilidade do sistema. Esse problema pode ser resolvido através da existência de cópias desse objeto em outras máquinas.

Ao se utilizar o objeto repositório (com todas as suas funções) na ORB, tem-se uma maior autonomia na obtenção das informações nos repositórios de interface, tendo em vista que o objeto de acesso direciona a solicitação de um serviço ao objeto repositório específico do repositório de interfaces que contém a informação desejada. Este fato pode ser observado tanto na utilização de um sistema de arquivos como na utilização de um banco de dados distribuído, onde em ambos os casos essas informações são obtidas diretamente pelo objeto repositório.

A questão da interoperabilidade entre repositórios de interface em diferentes ambientes é um tanto complexa. Vários fatores devem ser considerados, tais como a consistência de informações e os conflitos entre identificadores de objetos e conteúdos de informações. Tudo deve ser levado em consideração quando se deseja obter determinadas informações nos repositórios de interface localizados em diferentes ambientes [Zuq95].

Uma forma de solucionar o problema citado no parágrafo acima é a utilização de identificadores padrões de tipos de interface e identificadores padrões de repositórios de interface (localizados no arquivo de acesso), de forma que as informações possam manter-se consistentes, independente do ORB a que estão ligadas. Desta maneira os identificadores de interface seriam valores que representariam os tipos de informação armazenados. Os identificadores de repositórios dariam nome aos repositórios que identificariam exclusivamente módulos, interfaces, constantes, operações, etc. A interoperabilidade entre repositórios num mesmo ambiente é bem mais simples. A cada nova solcitação feita, o próprio ob-

jeto de acesso busca em que repositório encontra-se essa informação, e invoca o respectivo objeto repositório.

Para um melhor entendimento do funcionamento geral do Esquema de Acesso, explicaremos como uma solicitação, feita por um cliente, ao chegar no esquema de acesso irá se comportar. A solicitação feita pelo cliente estaticamente, através de stubs (ex:realização de um *typecheking*), ou dinamicamente (ex:busca de uma interface), através da DII (interface de invocação dinâmica), chega ao objeto de acesso pelo ORB. O objeto acesso verifica em que repositório estará contida a informação desejada pelo cliente. Tal fato é possível devido a versatibilidade do objeto, que verificará em seu arquivo a existência e localização do identificador solicitado. No caso da informação desejada não estar contida no ambiente no qual realiza-se a solicitação, o objeto de acesso pode buscá-la em outros ambientes através da interoperabilidade entre esquemas. Já o objeto repositório pode buscar a informação no repositório, desde que o conheça e, obtida a informação solicitada pelo cliente, pode enviá-la a este através do ORB. A solicitação feita pela implementação do objeto chega ao esquema de acesso através do ORB. Com o tipo da interface e o identificador da operação a ser invocada, o objeto de acesso interage com o objeto repositório, e esse com o repositório de implementações obtendo o endereço correspondente a operação a ser invocada pelo *skeleton*.

5.3 Repositório de Interfaces

Como já foi visto, o repositório de interfaces é um serviço componente da ORB, que tem por finalidade prover um armazenamento consistente de definições de interfaces. O ORB possibilita o acesso distribuído a essas definições através de interfaces conhecidas publicamente[CCODIC93].

O acesso às definições do repositório é necessário para que o ORB funcione corretamente, e seja capaz de processar devidamente todas as requisições. Essas definições podem estar disponíveis para o ORB de duas formas:

- **Estaticamente** → através do acesso às rotinas *stubs*. O cliente sabe especificamente que objeto deseja invocar. O ORB utiliza-se do repositório de interfaces exclusivamente para realizar uma verificação (*typechecking*) com a interface do cliente.
- **Dinamicamente** → através da interface de invocação dinâmica (DII). O ORB acessa o repositório de interfaces para buscar a informação necessária para a conclusão do seu serviço.

As definições de interfaces são mantidas no seu repositório como um conjunto de objetos, acessadas e gerenciadas através do objeto repositório. Essas definições contêm a des-

crição das operações que a interface suporta, incluindo os tipos de parâmetros, exceções, e informações referentes ao contexto da operação.

O repositório também armazena constantes que podem ser utilizadas em outras definições de interfaces, ou simplesmente definidas para o uso conveniente do programador. Os *Typecodes*, que são valores utilizados para descrever tipos em termos estruturais, também podem ser armazenados pelo repositório, que utilizar-se-á de módulos como uma forma de agrupar logicamente as interfaces. Com isso, pode-se navegar através desses módulos, facilitando a busca de informações dentro do repositório. Os módulos podem conter constantes, definições de tipos, exceções, definições de interfaces e outros módulos. Dependendo da necessidade, os módulos podem corresponder à organização das definições IDL, ou podem representar estruturas definidas para administração ou outros propósitos, caso de não exista qualquer padrão de especificação do tipo que constitua um grupo de interfaces em um determinado módulo.

5.3.1 Tipos de Armazenamento

O repositório de interfaces armazena objetos de uma forma consistente. Normalmente, o tipo utilizado nesse armazenamento determina como as definições de interfaces serão distribuídas ou replicadas através do domínio da rede. Dois tipos podem ser utilizados:

- **Sistema de arquivos** → o repositório que utiliza essa forma de armazenamento em sua implementação, pode ter armazenado todas as definições de interfaces referentes a uma única máquina nesta máquina. Com isso, para cada máquina existente no domínio da rede pode existir um repositório de interfaces com as definições referentes ao escopo daquela máquina. O acesso a um repositório por outra máquina torna-se viável conhecendo-se o identificador do repositório desejado. Isso é possível através do arquivo de acesso que contêm esse tipo de informação.

- **BDOO**[1] → o repositório que utiliza essa forma de armazenamento em sua implementação, pode ter múltiplas cópias das definições de interfaces distribuídas em várias máquinas através da rede. Com isso, pode-se ter um único objeto de armazenamento contendo todas as definições de interfaces, independente do escopo das máquinas . Para cada máquina pode existir um *cache* local que conterá algumas definições referentes ao escopo daquela máquina, reduzindo assim o tempo de acesso às definições pertinentes àquele ambiente. O tempo de latência desse *cache* poderá variar de acordo com a vida útil do *request*.

O tipo do objeto utilizado no armazenamento pode determinar o escopo das definições de interfaces providas pela implementação do repositório de interfaces. Por exemplo, o

[1]Banco de Dados Orientado a Objetos

escopo das informações pode determinar se cada usuário tem uma cópia local de um conjunto de interfaces, ou se existe uma cópia por grupo de usuários. O objeto utilizado no armazenamento também determina quais são os clientes que tem acesso a um determinado grupo de interfaces, e por quanto tempo esse grupo estará disponível.

A implementação do repositório de interfaces depende também do dispositivo de segurança utilizado. O mecanismo de segurança (normalmente opera de acordo com o tipo utilizado no armazenamento), determina a natureza e a granularidade dos controles de acesso disponíveis para restringir o acesso aos objetos no repositório.

5.3.2 Estrutura do Repositório de Interfaces

O objeto repositório é do tipo *repository*, podendo ser composto pelos objetos do tipo moduleDef (grupamento lógico de interfaces), interfaceDef (definição de uma interface), constantDef (definição do nome da constante), typeDef (definição do tipo, que não é uma interface) e exceptionDef (definição de uma exceção). O objeto moduleDef pode ser composto pelos objetos do tipo moduleDef, exceptionDef, typeDef, interfaceDef e constantDef. O objeto interfaceDef pode ser composto pelos objetos constantDef, operationDef (definição de uma operação na interface), exceptionDef, attributeDef e typeDef. Com exceção dos objetos repository. interfaceDef e moduleDef, já citados anteriormente, todos os outros são objetos simples.A figura 5.3 possibilita-nos a visão da estrutura do repositório de interfaces[CCODIC93].

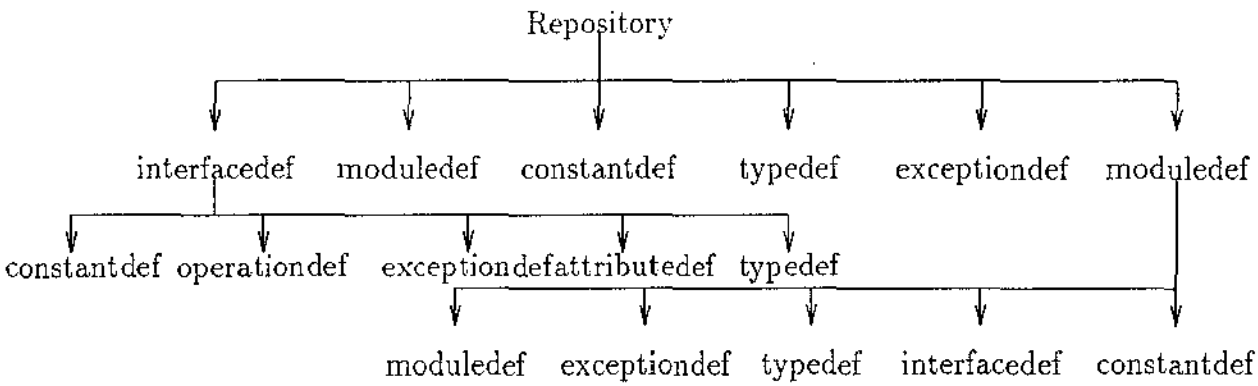


Figura 5.3: Estrutura Hierárquica do Repositório de Interfaces

Algumas vantagens na utilização dessa estrutura hierárquica podem ser citadas:

- Facilidade de localização dos objetos dentro do repositório de interfaces.

- Redução do custo operacional na manutenção (ex: realização de *updates*) dos objetos através da reutilização dos seus componentes.
- Confiabilidade, devido ao alto nível de integração entre os objetos.

Os objetos armazenados no repositório de interfaces estão especificados através de definições de interfaces. Estes objetos são definidos como subclasses[2] de duas classes[3] abstratas chamadas de *container* e *contained*. A interface *contained* representa a interface *header* para todas as interfaces do objeto repositório, com exceção da *repository*. A interface *container* é usada para localizar os objetos que estão contidos por outros objetos.

Tanto a interface *contained* quanto a interface *container* possuem operações comuns aos objetos do repositório de interfaces, que são descritos no apêndice A desta dissertação. A interface *contained* possui as operações *within* e *describe*. A operação *within* tem a finalidade de retornar uma lista de objetos que contêm um determinado objeto. A operação *describe* retorna uma estrutura, com o seu respectivo nome, que contêm todos os atributos definidos para uma interface. A interface *container* possui as operações *contents*, *lookup_name* e *describe_contents*. A operação *contents* retorna uma lista de objetos contido por um objeto. A operação *lookup_name* localiza um objeto dentro de outro através de seu nome. A operação *describe_contents* é uma combinação das operações *contents* e *describe*, no qual cada objeto retorna junto com a sua descrição.

Os objetos do repositório de interfaces têm as suas definições de interfaces compostas através da concatenação dos elementos das interfaces *container* e *contained* e dos elementos da sua interface. A definição de interface do objeto repositório além de herdar os elementos da interface *container*, possui a operação *lookup_id*. Esta operação procura um objeto no repositório de interfaces dado o tipo do objeto. As definições de interfaces dos objetos ModuleDef e InterfaceDef herdam os elementos das interfaces *container* e *contained*, sendo que o objeto InterfaceDef possui ainda a operação *describe_contents*. Esta operação obtém a descrição de todas as operações e atributos definidos na interface. As demais definições de interfaces dos objetos contidos no repositório de interfaces (OperationDef, AttributeDef, etc) herdam os elementos da interface *contained*.

Além dessas interfaces, são definidos os *TypeCodes*, que são valores que representam a invocação de tipos de argumentos e atributos. Eles têm uma variedade de utilizações. São utilizados pelas interfaces de invocação dinâmica para indicar os tipos de argumentos, e pelo repositório de interfaces na especificação de algumas interfaces.

A figura 5.4 que é apresentada em [dM95] mostra como os objetos do repositório de interface são especificados através das interfaces *container* e *contained*.

[2]uma subclasse caracteriza o comportamento de um conjunto de objetos que herdam algumas características da classe pai, mas possuem também características não compartilhadas pela mesma

[3]uma classe descreve o comportamento de um tipo de dado abstrato básico, definindo a interface para todas as operações que podem ser realizadas.

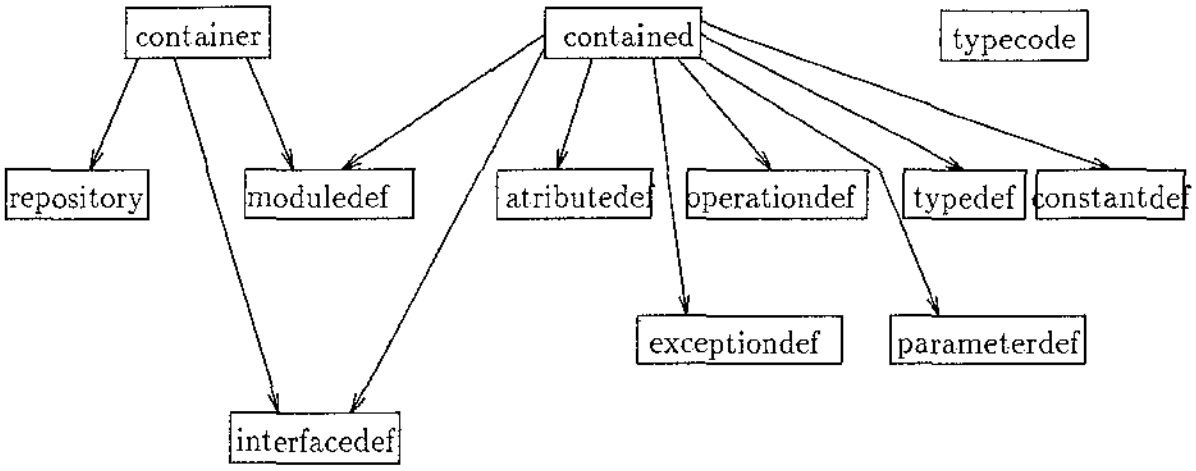


Figura 5.4: Herança de Interfaces do Repositório de Interfaces

5.4 Repositório de Implementações

O repositório de implementações possui as informações referentes à localização e ativação de objetos. Essas informações são utilizadas pelo ORB, assim, o objeto adaptador junto com o *skeleton* podem ativar e invocar a correta implementação, referente à solicitação feita pelo cliente, tanto estática quanto dinâmica.

5.5 Objeto Repositório

O **objeto repositório** tem a finalidade de auxiliar tanto no acesso e gerenciamento dos objetos armazenados no repositório de interfaces, quanto na busca de informações referentes a implementações no seu repositório. Ele auxilia no acesso e gerenciamento, porque possui funções que fornecerão os serviços adequados com segurança, para a obtenção das informações necessárias à conclusão de serviços solicitados pelo cliente.

Os métodos do objeto repositório são invocados pelo objeto de acesso de acordo com a necessidade da solicitação do cliente. O objeto repositório trabalha basicamente como um mediador entre o objeto de acesso e o repositório de interfaces. Exerce a função de mediador, porque recebe as invocações de serviços do objeto de acesso e interage com o repositório de interfaces, sempre utilizando-se de seus métodos na busca da realização de seus propósitos. Esse objeto, além de interagir com o repositório de interfaces, também interage com o repositório de implementações. A interação do objeto repositório com o repositório de implementações ocorre quando a invocação dinâmica ou estática da solicitação do cliente necessita obter informações referentes à implementação do objeto, informações estas utilizadas para ativar e localizar corretamente o objeto necessário à conclusão do

serviço.

As funções pertinentes ao objeto foram criadas visando as necessidades impostas pelo cliente numa invocação estática ou dinâmica. As funções referentes à busca de informações no repositório de interfaces foram estruturadas de forma a aproveitarem sempre os métodos oferecidos pelos objetos armazenados no mesmo.

Um bom exemplo da utilização do objeto repositório é quando existe a necessidade da realização de um *Typechecking* da interface solicitada pelo cliente. A solicitação feita estaticamente pelo cliente, através de *stubs*, tem conhecimento tanto do tipo da interface quanto do seu nome. De posse dessas informações, o ORB utiliza-se do repositório de interfaces única e exclusivamente para realizar uma verificação entre essas informações e a definição da interface existente dentro do repositório. Logo, essa funcionalidade oferecida pelo objeto repositório possibilita uma maior segurança e confiabilidade do cliente, no serviço solicitado pelo mesmo.

Uma outra forma de se utilizar o objeto repositório é durante a invocação dinâmica de uma requisição pela interface de invocação dinâmica. O ORB envia o tipo da interface junto com a operação desejada pelo cliente ao objeto repositório. Este, interage com o repositório de interfaces e através de suas operações, retorna a especificação completa da interface, que irá compor a referência do objeto para posteriormente criar a requisição do serviço. Através destes dois exemplos citados anteriormente, pode-se constatar a importância do objeto repositório para a obtenção das informações no repositório de interfaces.

Caso o repositório de interfaces tenha sido implementado por um banco de dados distribuído orientado a objetos, no qual estarão contidas todas as informações referentes às interfaces do sistema, independente do escopo da máquina, o nosso esquema de acesso utilizar-se-á de um *cache* de memória junto do objeto repositório. Esse *cache* possibilitará a otimização do tempo de busca das informações solicitadas pelo ORB. Isso é possível porque nele estarão contidas todas as informações referentes ao escopo das máquinas as quais estiver ligado. A busca, assim, fica reduzida a uma quantidade menor de informações, facilitando o funcionamento do esquema de acesso. O objeto repositório possui uma funcionalidade que possibilita a busca das informações referentes ao escopo de algumas máquinas e as armazena dentro desse *cache*.

O objeto repositório está definido através da interface repositório:

```
interface repositório{
    boolean store(InterfaceName);
    InterfaceDef dynamic_invoc(Repositoryld, OperationName);
    InterfaceDef suply(Repositoryld);
    boolean lookup(Repositoryld, search_name);
    boolean include(Repositoryld, search_name, RepositoryName);
```

```
    string search_address(OperationName);
    boolean del_int(Repositoyld, search_name);
}
```

As funções prescritas na interface repositório são especificadas da seguinte forma:

- **include** → essa função tem por finalidade incluir um objeto dentro do repositório de interfaces e tem como parâmetros o nome do repositório onde será incluído o identificador do tipo e algum atributo referente ao objeto. O objeto repositório interage com o repositório através da operação *lookup_name* com a finalidade de verificar se já existe este objeto no repositório. Caso exista o objeto, a operação não será concluída. Caso não exista, invoca-se a operação *contents*, navegando-se através do repositório, para obter a posição a ser ocupada. Identificada a sua posição, a operação **include** pode incluir o objeto dentro do repositório. A função **include** possibilita que interfaces não definidas durante a instalação do sistema possam ser incluídas posteriormente.

- **lookup** → esta função tem a finalidade de realizar um *Typechecking* entre interfaces. Ela recebe do objeto de acesso o identificador do tipo da interface e algum atributo do objeto a ser comparado, e interage com o repositório de interfaces, invocando a operação **lookup_id** a fim de buscar a interface com o nome especificado pelo ORB. Caso a função **lookup_id** ache o objeto, pode-se constatar a existência desse objeto dentro do repositório. Logo, a função **lookup** retornará um valor booleano dependendo do resultado apresentado pela função **lookup_id**. A função **lookup** oferece uma checagem adicional à solicitação feita pelo cliente.

- **dynamic_invoc** → esta operação tem por finalidade retornar o objeto **InterfaceDef**, isto é, a especificação completa da interface, componente necessário para compor a referência do objeto (informação utilizada para especificar um objeto dentro do ORB), que será utilizado no *request* feito dinamicamente pela interface de invocação dinâmica. Ela recebe o **Repositoryld**, bem como o nome da operação do ORB e interage com o repositório de interfaces invocando a operação **lookup_id** junto com a operação *describe_interface* para que possa obter toda a descrição da interface.

- **suply** → esta função tem por finalidade suprir as necessidades solicitadas durante a realização de um determinado serviço. Possui uma funcionalidade semelhante a da função **dynamic_invoc**. A função suply recebe do objeto de acesso o **Repositoryld** do ORB, e este interage com o objeto repositório, invocando a operação **lookup_id** e retornando o objeto desejado.

- **store** → esta função tem a finalidade de buscar todas as informações inerentes ao escopo de uma máquina a qual ela está ligada e armazená-la no *cache*. Ela poderá ser utilizada quando a implementação do repositório de interfaces utilizar-se de um banco de dados distribuídos orientado a objetos. O objeto repositório interage com repositório de interfaces através da função *within* para retornar todos os objetos que contém o objeto a ser armazenado. A partir desta lista, utiliza-se a operação *include* para armazenar estas informações no *cache* de uma máquina do sistema.

- **del_int** → a função *del_int* tem a finalidade de deletar um objeto contido no repositório de interfaces. O objeto repositório interage com o repositório de interfaces invocando a operação *lookup_id* ou *lookup_name* para procurar o objeto a ser deletado. Caso o objeto não tenha sido achado, a operação não será realizada. Caso o objeto seja achado e sua definição de interface herde somente elementos da interface *contained*, a operação *del_int* realiza uma simples deleção deste objeto. Caso o objeto que será deletado tenha a sua definição de interface herdado elementos da interface *container*, utiliza-se da operação *contents* para se obter a lista dos objetos que são contidos por ele. São deletados todos estes objetos da lista, mais o objeto que originou esta lista.

- **search_address** → essa função tem a finalidade de interagir com o repositório de implementações à fim de buscar informações necessárias ao objeto adaptador e ao *skeleton* para que possa ser invocada a correta implementação. Ela recebe o tipo da operação a ser buscada, e com isso obtém as informações referentes à localização desse objeto e a forma como está armazenado.

5.6 Objeto Acesso

O **objeto de acesso** é um componente importante para o esquema de acesso proposto aos repositórios de interfaces e implementações. É através dele que descobrimos em qual repositório de interfaces está localizada uma determinada interface. O objeto de acesso também possibilita a interoperabilidade entre repositórios de interfaces dentro ou fora de um ambiente. Sua funcionalidade é muito simples. O objeto possui algumas funções que ajudam na localização do repositório de interfaces para buscar determinada informação que auxiliará na realização de algum serviço solicitado pelo cliente. Este recebe do ORB todas as informações necessárias à realização de algum serviço no repositório de interfaces. Analisa essas informações e direciona as solicitações para o objeto repositório que correspondente ao repositório onde será realizado o serviço. Quanto a sua funcionalidade em relação ao repositório de implementações, recebe o tipo da interface junto com a operação correspondente, seleciona o objeto repositório e invoca-o. Essa busca do objeto repositório

pode não estar relacionada diretamente com o tipo da interface, tendo em vista que temos um único repositório de implementações contendo informações de todas as funções do sistema e todos os objetos repositórios podem interagir com o mesmo.

Para dar início a viabilização dessas funcionalidades, o objeto de acesso conta com a ajuda de um elemento fundamental, o arquivo de acesso. É através deste arquivo que obtemos as informações referentes ao tipo da interface, o endereço e o identificador do repositório de interfaces onde ela se encontra. Uma forma de implementar o arquivo de acesso é através de uma tabela **hash**, que facilita a inserção, busca e deleção dinâmica das informações armazenadas no arquivo de acesso. O arquivo de acesso é utilizado somente quando existe um objeto repositório de interfaces em estações distintas do sistema, e quando o arquivo de acesso está em uma estação distinta do objeto de acesso.

A interoperabilidade é tratada pelo objeto de acesso de duas formas:

- A primeira quando ocorre **dentro do ambiente ORB**, isto é, quando deseja-se uma determinada informação fora do escopo de um repositório. Todavia, esta informação desejada pertence ao escopo de outro repositório que por sua vez, pertence ao mesmo ambiente ORB. Esta é a forma mais simples, pois é suficiente para o objeto de acesso receber a solicitação e interagir com o arquivo de acesso para buscar o repositório onde encontra-se a informação desejada.

- A segunda forma ocorre quando a interoperabilidade é realizada entre repositórios em ambientes diferentes. Esse caso é considerado mais difícil devido a fatores como a consistência de informações entre repositórios. Uma forma de ter essa funcionalidade é manter diferentes identificadores de repositórios, de forma que ao solicitar alguma informação em outro ambiente ORB possa ser utilizado esse identificador junto com o tipo da interface que se deseja obter. A função **search_ORB** pode executar essa funcionalidade, caso tenha esses parâmetros.

À fim de possibilitar um melhor entendimento do funcionamento do objeto de acesso, vamos observar um exemplo de como ele trabalha:

Suponha a realização de uma verificação de tipos. O objeto de acesso recebe do ORB a informação referente a uma interface e interage com o arquivo de acesso. O objeto de acesso obtém o identificador do repositório de interfaces como também o tipo da mesma. Estas informações são utilizadas para invocar funções no objeto repositório, a fim de realizar o serviço solicitado.

A figura 5.5 demonstra o esquema de como as solicitações oriundas do ORB são direcionadas ao objeto repositório. As linhas cheias dessa figura representam que o objeto de acesso está ativo, interagindo com os objetos repositórios e o arquivo de acesso.

As linhas tracejadas do desenho significam que existem réplicas do objeto de acesso em outras máquinas do ambiente, podendo ser acionadas caso haja necessidade. Com

isto, estabelecemos um maior grau de confiabilidade na utilização do esquema de acesso proposto.

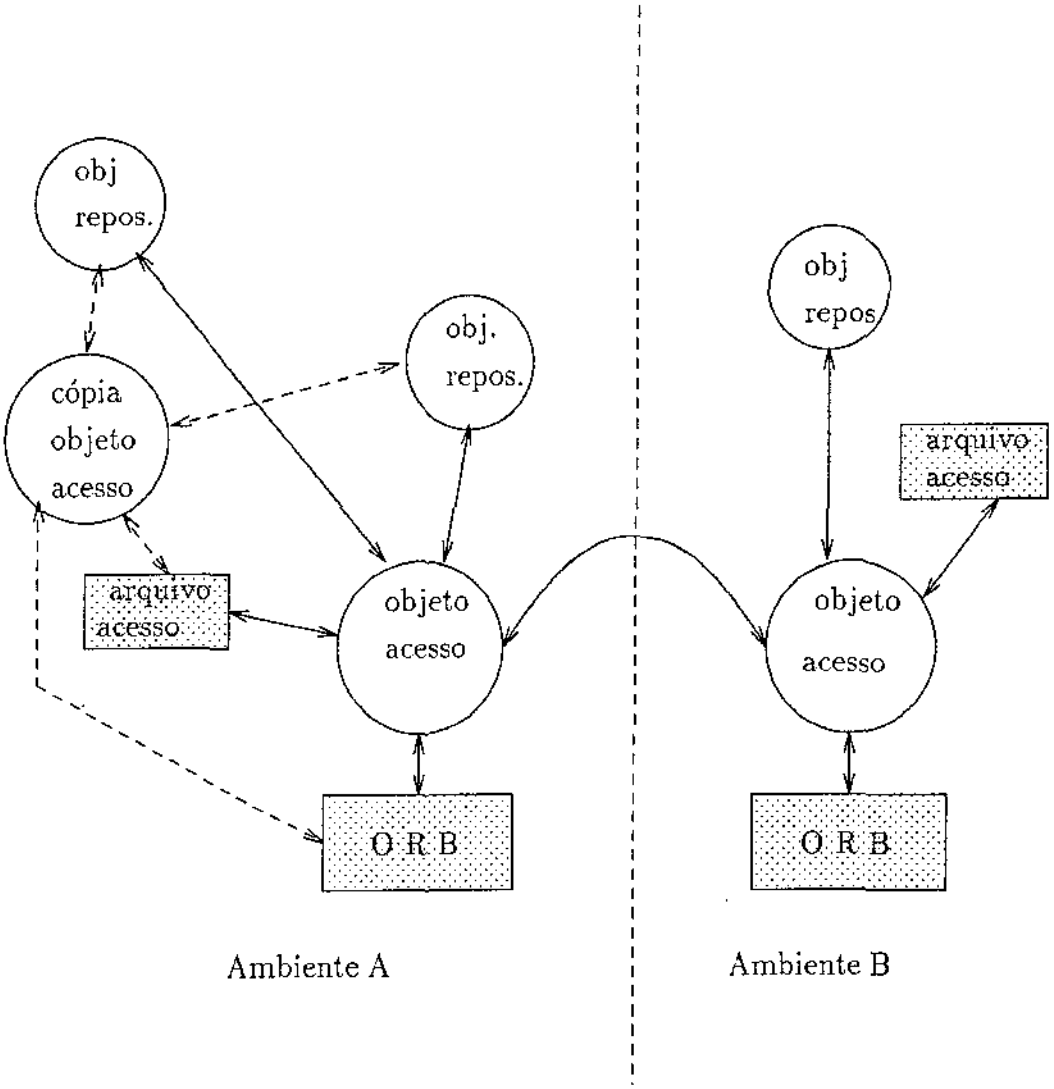


Figura 5.5: Esquema de Obtenção das Informações pelo Objeto de Acesso

O objeto de acesso é definido através da interface de acesso:

```
interface acesso {
   void search_rep(Repositoryld);
   void include_rep(Repositoryld, RepositoryName);
   void search_ORB(Repositoryld, RepositoryName);
   void del_rep(Repositoryld, RepositoryName);
```

}

As funções prescritas na interface de acesso são especificadas abaixo:

- **search_rep** → esta função tem por finalidade buscar o nome e o identificador do repositório de interfaces, dado o tipo da interface. Sua funcionalidade é muito simples: ela recebe o tipo da interface, interage com o arquivo de acesso e busca o nome da interface e o identificador do repositório de interfaces no qual esta encontra-se.

- **include_rep** → esta função, ao contrário da função **search_rep**, realiza uma atualização do arquivo de acesso através da inclusão de uma nova interface nesse arquivo. Esta também é uma função relativamente simples. Tem como parâmetros o tipo, o nome da interface e o identificador do repositório de onde está esta interface.

- **search_ORB** → como citado anteriormente, a interoperabilidade entre repositórios de interfaces em diferentes ambientes ORBs é uma questão um tanto quanto complexa. A forma sugerida é utilizar como parâmetros o identificador de repositórios e o tipo da interface. Um exemplo prático da utilização dessa função é durante a realização de um *Typechecking*. A função **search_ORB**, de posse desses parâmetros, aciona outros arquivos de acesso buscando esse tipo de informação no repositório de interfaces especificado pelo identificador, até que a informação desejada seja encontrada. Esquemas mais genéricos são encontrados em [Zuq95].

- **del_rep** → esta função, bem como a função *inclui_rep*, realizam atualizações do arquivo de acesso. Esta função é invocada quando for realizada alguma deleção de tipos de interfaces no repsitório de interfaces.

5.7 Esquema de Acesso X ODP

Nesta seção estamos realizando uma análise do desenvolvimento do esquema de acesso sob os conceitos existentes no modelo de referência ODP.

Como citado no capítulo 2, o modelo de referência ODP oferece subsídios para uma estruturação dos ambientes de processamento distribuído aberto. Devido às suas características, projetistas de sistemas utilizam-no como um guia na construção de seus sistemas abertos. A CORBA não foge a esse contexto, sendo uma plataforma para ambientes de serviços abertos, segue também algumas das funcionalidades definidas no RM-ODP[CCODIC93, K.R93, ISO94a].

Como no modelo de referência ODP, a CORBA também fornece uma estrutura consistente e bem definida para o processamento distribuído aberto. Essa estrutura engloba a heterogeneidade dos ambientes, a distribuição física e lógica dos equipamentos e também a portabilidade das aplicações entre os diversos ambientes. Podemos considerar a CORBA

como um padrão específico para o ODP, cobrindo determinados conceitos e funções comuns referentes a uma empresa, no que tange a especificação da comunicação e da coordenação das informações distribuídas. Baseado na concordância de conceitos entre as duas especificações, estruturamos o esquema de acesso utilizando das idéias existentes no RM-ODP.

5.7.1 Modelagem de Objetos

A metodologia orientada a objetos utilizada pelo modelo de referência ODP, também é utilizada na CORBA. Os objetos do esquema de acesso, de uma forma geral, pode ser aplicado em várias situações e implementado em várias linguagens de programação.Seus objetos obedecem a algumas propriedades:

- **Abstração** - ocorre quando a descrição de seu serviço abstrai-se da forma como é implementado. Todos os objetos do esquema oferecem serviços que indempendem da forma como são implementados.
- **Interação** - os serviços são oferecidos através de interfaces. Os serviços dos objetos do esquema, bem como os objetos dos repositórios, estão a disposição através de interfaces.
- **Encapsulamento** - todas as informações dos objetos do esquema de acesso são obtidas através de invocações.
- **Hierarquia** - os objetos do repositório de interfaces são classificadas de acordo com o seu serviço.
- **Herança** - essa propriedade é provida pela através da IDL, durante a definição das interfaces dos objetos contidos nos repositórios.
- **Distribuição** - as implementações dos objetos podem ser relocadas, sem afetar os clientes. Essas mudanças de posicionamento são atualizadas no repositório de implementações.

5.7.2 Pontos de Vista ODP

Durante o desenvolvimento do esquema de acesso, procuramos visualizar o sistema sob os cinco pontos de vista estabelecidos pelo ODP.

- **Empresa** → Nesse ponto de vista são estabelecidos os objetivos que devem ser alcançados pelo esquema. Foram definidas as políticas de armazenamento das informações nos repositórios de interfaces e implementações. São definidas as políticas de gerenciamento das informações dos repositórios (a forma como serão acessados,

atualizados e utilizados), como também o escopo dessas informações. Estabeleceu-se que cada objeto do esquema seria um agente (objeto que desempenha uma ação), e os repositórios junto com o arquivo de acesso seriam artefatos (objetos que não iniciam uma ação).

- **Informação** → Durante esse ponto de vista, foi definida a semântica das requisições. Definiu-se a linguagens a ser utilizada na implementação do esquema, bem como as estruturas e quais as funções que seriam desempenhadas por cada um dos objetos do esquema.

- **Computacional** → Sobre esse ponto de vista, visualizou-se o esquema como um grande objeto computacional que iria interagir com os outros objetos da CORBA.

  A interação entre os objetos do esquema é feita de forma operacional, no qual as interfaces dos objetos são operacionais, provendo um modelo cliente/servidor. Essas interfaces possuem operações com parâmetros, terminações, requisições e resultados. Foram especificados os algorítimos que corresponderiam as funções dos objetos.

  A invocação ao esquema segue o princípio do modelo computacional, que utiliza-se de duas ações atômicas para a sua conclusão. A primeira é a *invocation submit*, quando o cliente submete a invocação ao ORB, e a segunda *invocation deliver*, quando o adaptador de objetos chama a implementação para receber a invocação. Para a conclusão do serviço, a CORBA também segue o conceito do RM-ODP, que utiliza-se de duas ações atômicas para a sua realização. Inicialemente, a *termination submit*, quando o *skeleton* invoca o método correto da implementação, e a *termination deliver*, quando o cliente recebe seus resultados e encerra o serviço. Todas as operações realizadas na CORBA são **interrogações**, onde qualquer invocação é seguida por uma terminação de um serviço.

  Outro conceito que pode ser abordado nesse ponto de vista é o de *trader* (onde são negociadas as interfaces entre o cliente e o servidor), levando-se em consideração a negociação entre o ORB, o repositório de interfaces e o cliente na composição de uma requisição, onde são feitas várias interações entre esses objetos. O *trader* com todas as suas funcionalidades é um serviço de objetos.

- **Engenharia** → dentro do ponto de vista de engenharia foi estabelecida a estrutura distribuída do sistema, isto é, foram descritos os apectos de distribuição do esquema, definindo-se a infraestrutura do modelo. O esquema como um todo é um objeto da infraestrutura de distribuição do sistema. Poderia visuá-lo como uma cápsula no sistema, ocupando um espaço de memória provendo recursos para o sistema concluir um determinado serviço.

  O esquema de acesso provê algumas transparências:

- **Falha** - A transparência de falha pode ser observada parcialmente, pois caso haja queda no sistema em função da máquina em que se encontra o objeto de acesso, outra máquina pode ser acionada de forma que mascare os erros de comunicação do esquema, proporcionando uma maior confiabilidade e disponibilidade do sistema.
- **Federação** - Esta transparência pode ser verificada parcialmente, através do compartilahmento de recursos providos pela integração dos repositórios de interfaces.
- **Localização** - Podemos citar também a transparência de localização, no qual o esquema proporciona para o ORB um acesso as informações nos repositórios de um forma uniforme e consistente, independente da localização das mesmas.

5.7.3 Funções

A arquitetura CORBA identifica algumas funções que asseguram a execução eficaz e sem problemas dos serviços solicitados pelos clientes. Estamos descrevendo algumas destas funções citadas no RM-ODP como funções repositório:

- **Armazenamento** - esta função pode ser verificada tanto no repositório de interfaces quanto no repositório de implementações. O gerenciamento destes é feito através de interfaces que possibilitam a realização de modificações, deleções e obtenções de informações
- **Repositório de Tipos** - esta função é verificada através do repositório de interfaces, onde são armazenados os tipos de interface, mantendo-se a hierarquia de tipos e as relações. Esta função possibilita a verificação de tipos durante a invocação dinâmica.
- **Relocador** - a função relocador pode ser observada através do arquivo de acesso, no qual este mantém o posicionamento dos tipos de interfaces, mesmo quando existe qualquer alteração desse posicionamento.

5.8 ORBeline

O ORBeline é uma implementação completa da especificação da arquitetura CORBA, isto é, ele é um representante das plataformas ORB comerciais existentes hoje [Inc94]. Ele possui todos os objetos citados na especificação, porém, nesta seção abordaremos apenas a forma como os seus repositórios são acessados, realizando uma breve comparação com o nosso esquema de acesso.

A implementação ORBeline provê um repositório de interfaces dinâmico e um repositório de implementações. O repositório de interfaces é implementado como um banco

de dados distribuído contendo todas as definições de interfaces do sistema, onde múltiplas instâncias podem estar sendo executadas e acessadas por clientes. Esses repositórios são identificados através de nomes, e acessados através de solicitações do cliente. Como no nosso esquema de acesso, o repositório de interfaces é gerenciado e manipulado através de suas funções existentes dentro do repositório.

O ORBeline não especifica funções de verificação de tipos como proporcionado em nosso esquema. Para estabelecer uma ligação do cliente com um específico repositório de interfaces, o cliente deve especificar o nome a ser chamado. Na invocação dinâmica, tanto o ORBeline quanto o esquema de acesso recebem o tipo da interface como também o nome da operação para obter o repositório onde se encontra a especificação completa da interface. O ORBeline não prevê a interoperabilidade entre repositórios de interfaces tanto no mesmo quanto em diferentes ambientes ORBs. Concluindo, pode-se dizer que ambas as abordagens se preocupam com a eficiência em relação à obtenção das informações contidas no repositório de interfaces.

Capítulo 6

Aspectos de Implementação

6.1 Introdução

Como vimos no capítulo anterior, o modelo conceitual do nosso **esquema de acesso ao repositório de interfaces** tem a finalidade de facilitar e otimizar o acesso ao repositório de interfaces da **CORBA**. Neste capítulo, estaremos comentando os aspectos de como foi desenvolvida a implementação do **protótipo** do referido esquema. Esse protótipo tem como objetivo validar os conceitos principais desse esquema durante a obtenção de informações no repositório de interfaces necessárias aos serviços solicitados pelo cliente, podendo ser implementado na linguagem de programação **C ou C++**.

As idéias de implementação da plataforma *Multiware* estão inicialmente baseadas na arquitetura **CORBA**. Contudo, a plataforma *Multiware* está em fase final de especificação e no início de sua implementação. Com isso, a ORB está em fase de aquisição, a qual seria ideal para o total desenvolvimento do projeto. Em função disto, o protótipo do esquema de acesso foi desenvolvido sobre o **RPC**[1] **da SUN**[Mic90], prática comum em projetos desta área. O **RPC da SUN** exerceu a funcionalidade do ORB e estabeleceu uma ligação entre o cliente e o esquema de acesso, capaz de enviar e receber mensagens entre eles e viabilizar a comunicação entre os objetos dentro do esquema. Utilizamos também o **NFS**[2], implementado sobre o RPC da SUN, que teve como objetivo de auxiliar na localização dos repositórios de interfaces e implementação.

A estrutura do protótipo é composta por três *stubs clients* (programas) responsáveis pelo envio das solicitações de serviços ao esquema de acesso, através da invocação de funções remotas no objeto de acesso, pelo RPC. Outro componente do esquema é o objeto de acesso que, uma vez composto por programas, tem a finalidade de buscar no arquivo de acesso informações necessárias à invocação de funções remotas, e comunicar com o objeto

[1]Remote Procedure Call
[2]Network System File

repositório utilizando o RPC. O outro objeto do esquema é o objeto repositório, composto também por programas, onde executam-se funções de busca, inserção e verificação de tipos no repositório de interfaces. O esquema possui ainda três bases de dados representadas por arquivos, repositório de interfaces, repositório de implementações e o arquivo de acesso. A figura 6.1 demonstra a arquitetura do protótipo criado.

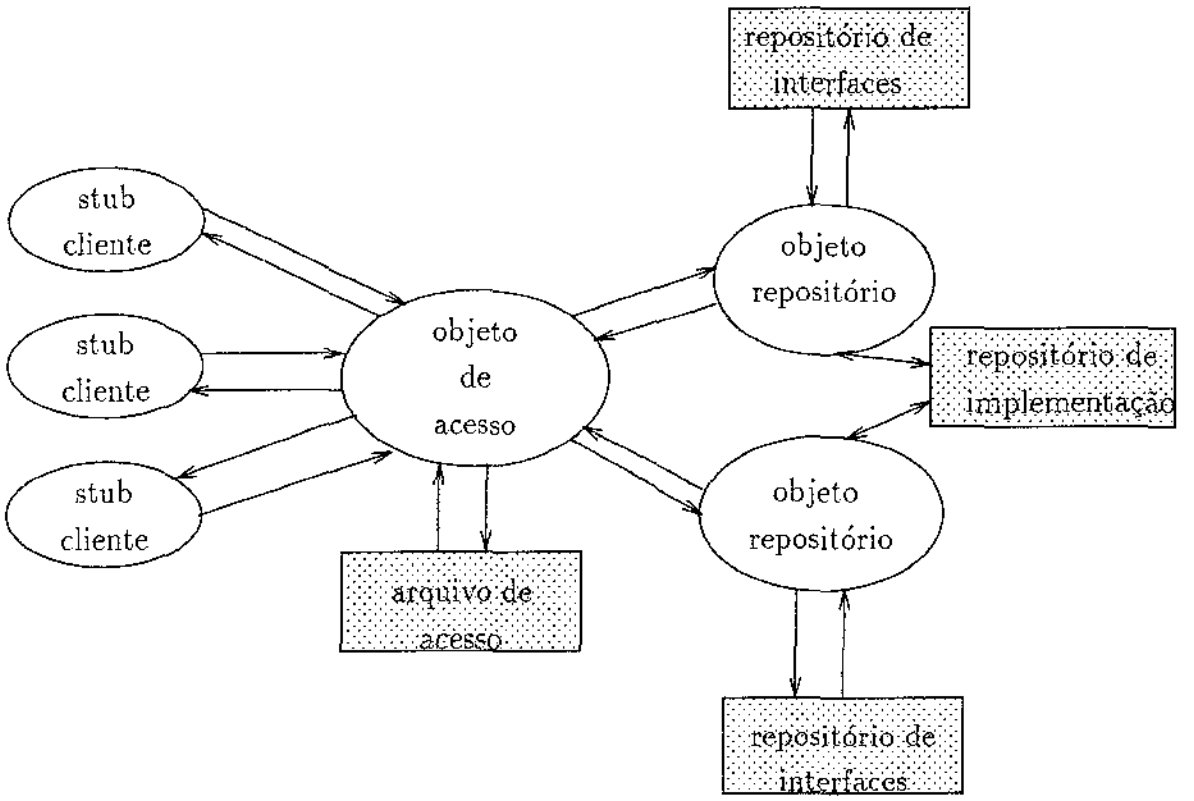


Figura 6.1: Arquitetura do Protótipo

6.2 RPC

O **RPC** é um protocolo de comunicação de alto nível criado pela SUN, que possibilita o desenvolvimento de aplicações através de chamadas de procedimento, mascarando detalhes básicos dos mecanismos da rede. Esse protocolo usa um padrão chamado **XDR**[3] e é apoiado por **Portmappers** (local ao servidor), que tem a finalidade de realizar ligações entre um cliente remoto e um servidor local. O protocolo RPC da SUN consiste em uma camada colocada em cima do XDR e de um protocolo de transporte, podendo ser o

[3]External Data Representation

TCP (Transmission Control Protocol) ou **UDP** (User Datagram Protocol). Logo, o RPC implementa um sistema de comunicações lógico baseado no paradigma cliente/servidor no qual clientes executam chamadas de procedimentos remotamente, enviando ao servidor solicitações de serviço para que sejam executadas, retornando suas respostas[Mic90].

No RPC, a chamada de procedimento remoto é similar ao modelo de chamada de procedimento local. Numa chamada local, a função chamadora coloca os argumentos para o procedimento em alguma posição pré-determinada. Transfere-se o controle para o procedimento, posteriormente retornando-o à origem. No procedimento remoto o cliente envia uma mensagem para o servidor, contendo os parâmetros do procedimento e fica aguardando o retorno da resposta. O servidor permanece inativo até a chegada da chamada e só então, retira os parâmetros, computa os resultados e os envia como resposta para o cliente que extrai os resultados e encerra o serviço.

A figura 6.2 mostra os componentes do protótipo desde o *stub client* até o repositório de interfaces. Pode-se ter uma noção geral do caminho percorrido por uma solicitação.

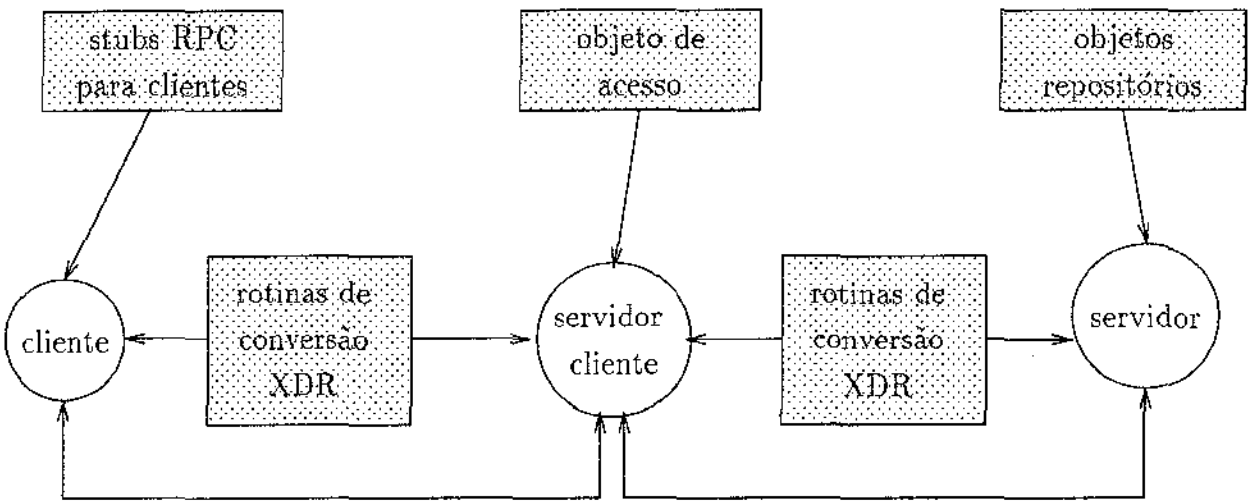


Figura 6.2: Estrutura RPC

Com o objetivo de melhor esclarecer o funcionamento do nosso esquema, utilizando-se do RPC, vamos seguir uma chamada de procedimento remoto executada por um *stub* cliente:

- **O stub cliente** (cliente) envia uma solicitação de serviço para o esquema de acesso, através de uma invocação de uma função remota no objeto de acesso, utilizando-se do RPC, passando como parâmetro uma estrutura de dados ou uma *string*.

- O **objeto de acesso** (servidor) recebe a chamada e tem sua função ativada. Assim, através dos dados referentes ao stub cliente passados pela rede, busca no arquivo de acesso as informações necessárias à realização de uma chamada remota ao objeto repositório. Realiza a chamada de uma função remotamente no objeto repositório, passando como parâmetro uma estrutura de dados, através da rede, necessária à execução do serviço.
- O **objeto repositório** (servidor) é ativado ao receber a chamada, executa o serviço solicitado e retorna a resposta ao objeto de acesso.
- O **objeto de acesso** dá por encerrado o serviço por ele solicitado e retorna a resposta ao *stub* cliente.
- O **stub cliente** recebe a resposta e dá por encerrado o serviço solicitado.

A figura 6.3 mostra de uma forma mais detalhada a seqüência citada acima.

6.3 NFS

O NFS (Network File System) da SUN é uma extensão do sistema operacional UNIX, que provê um serviço de arquivos distribuído, utilizando-se do sistema de redes implantado. Este sistema já vem incluído no software que é fornecido junto com as estações de trabalho da SUN. Esse sistema de arquivos possibilita o acesso remoto aos arquivos que estão armazenados de forma distribuída em várias localições da rede facilitando, assim, o acesso as informações que são buscadas na conclusão dos serviços solicitados pelo clientes RPC. O NFS dentro do protótipo possibilita os objetos de acesso e repositório acessarem o arquivo de acesso e os repositórios de interfaces, respectivamente,no sistema.

6.4 Disponibilidade dos Objetos

Nesta seção estamos especificando como os objetos e suas funções estão localizados em relação à disponibilidade de máquinas do sistema. Como pode-se observar, a figura 6.2 mostra a estrutura para cada tipo de serviço que o sistema deseja realizar, baseada no RPC.

O nosso protótipo propõe a realização de três tipos de serviço. Baseando-se no parágrafo anterior, teremos três estruturas diferentes representando cada tipo de serviço. Foram utilizadas diferentes rotinas de conversão de dados para cada relação cliente/servidor. Levando-se em consideração esse princípio, podemos ter diferentes máquinas na realização da invocação do serviço, isto é, cada máquina faz o papel do *stub client*. As funções executadas pelo objeto de acesso são carregadas numa mesma máquina e podem centralizar

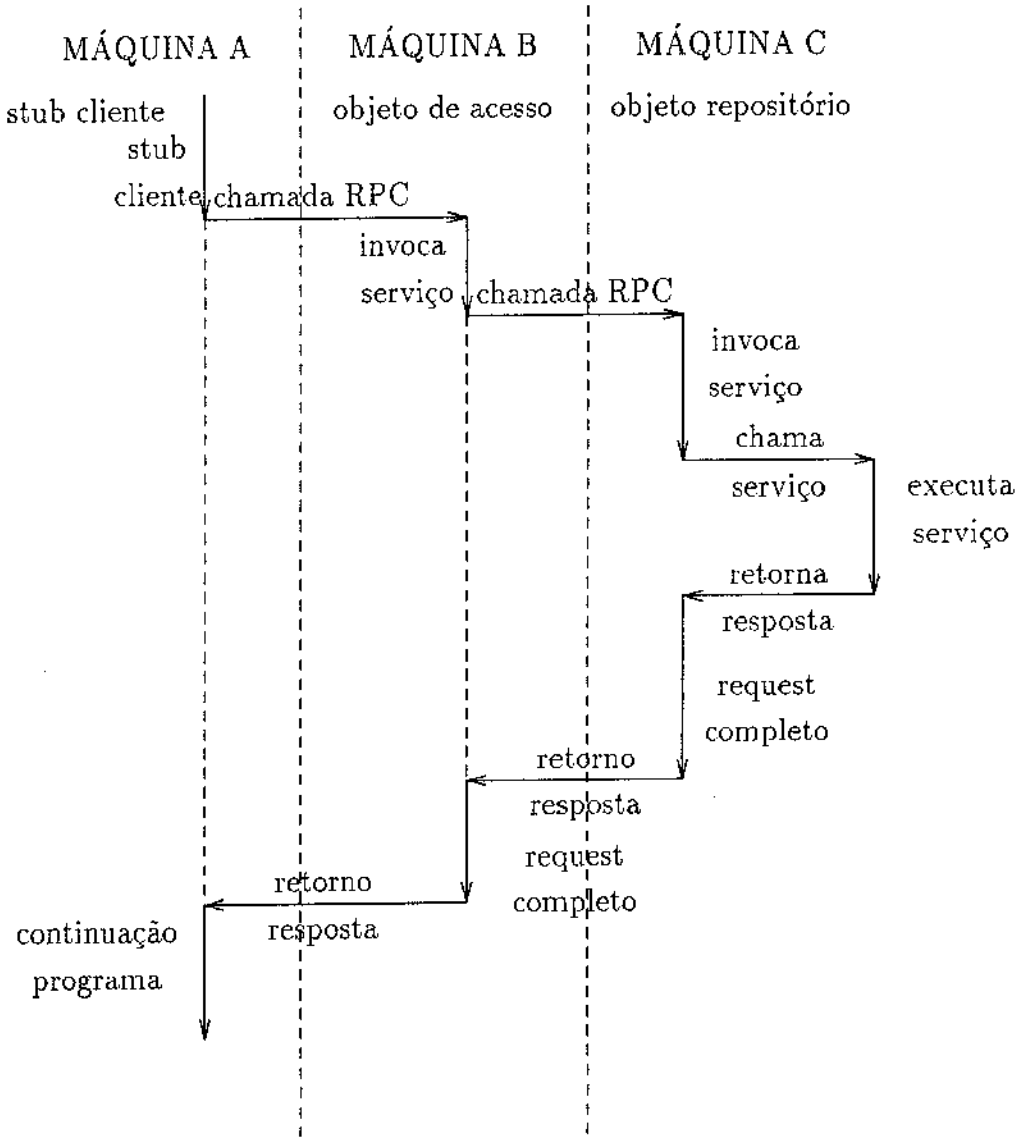


Figura 6.3: Seqüência de Chamada

todas as requisições, bem como direcioná-las para o objeto repositório. As funções de cada objeto repositório são carregadas em máquinas diferentes, sendo que em cada uma destas (objeto repositório) exercem o controle sobre um repositório de interfaces. A figura 6.1 possibilita uma melhor visualização dessa estrutura.

6.5 Stub Cliente

Dentro do nosso esquema de implementação, os *stubs clients* são programas responsáveis pela invocação de funções localizadas no objeto de acesso. O sistema implementado possui três *stubs client*, sendo que o primeiro tem a finalidade de solicitar uma inclusão de um novo tipo de interface (caso ela não exista) junto com o tipo da operação que ela executa. Esse *stub* realiza uma chamada remota ao objeto de acesso, passando como parâmetro uma estrutura de dados que contém o tipo da interface a ser incluída, a operação que ela executa e o repositório onde será armazenado.

O segundo *stub client* tem a finalidade de solicitar uma verificação de tipos de interface junto ao objeto de acesso e verificar o tipo de operação a ser executada pela interface junto ao repositório. Esse *stub* realiza uma chamada remota ao objeto de acesso, passando como parâmetro uma estrutura de dados que contém o tipo de interface e o tipo de operação executada.

O terceiro e último *stub client* tem a finalidade de buscar dinamicamente o endereço referente à uma operação pertencente a uma interface específica. São passados o tipo da operação e da interface como parâmetros. O terceiro e último *stub client* tem a finalidade de buscar dinamicamente o endereço referente à uma operação pertencente a uma interface específica. São passados o tipo da operação e da interface como parâmetros.

6.6 Objeto de Acesso

O **objeto de acesso**, dentro do nosso sistema, funciona tanto como um cliente quanto um servidor do RPC da SUN. Como servidor, o objeto de acesso recebe um tipo de dados através da rede e busca alguma informação no arquivo de acesso necessária à invocação de um serviço no objeto repositório. Como cliente, basicamente realiza uma chamada RPC do objeto repositório, passando as informações necessárias à realização do seviço desejado.

Foram implementadas, parcialmente, as funções *search_rep* e *include_rep*, especificadas no modelo conceitual, de forma compatível com o RPC da SUN. A função *search_rep*, no protótipo, tem a finalidade de interagir com o arquivo de acesso, e através do identificador do tipo de interfaces buscar qual o repositório que se encontra a interface, bem como a máquina correspondente ao repsitório. A função *include_rep* tem a finalidade de incluir um tipo de interface junto com uma operação num repositório de interfaces. Tanto em

uma como na outra função, após a realização de sua funcionalidade inicial, realiza uma invocação remota de uma função no objeto repositório correspondente ao repositório de interfaces que está localizado o tipo da interface que foi passada como parâmetro inicial. A identificação das máquinas no arquivo de acesso é importante devido à necessidade do RPC ter conhecimento destas para realizar uma chamada remota.

Baseada nas funções acima citadas, vejamos como fica a seqüência de ações no objeto de acesso para que seja realizada uma verificação de tipos, uma inclusão de uma interface ou uma invocação dinâmica de uma solicitação.

Durante a realização de uma verificação de tipos, a função *search_rep* é invocada realizando as seguintes ações:

- A função *search_rep* do **objeto de acesso** é invocada através da chamada do *stub* cliente. Recebe o identificador de interfaces e o identificador da operação que serão verificados.
- **Arquivo de Acesso** é aberto para leitura.
- Realiza-se a verificação do tipo da interface dentro do **arquivo de acesso.**
- Caso o **tipo da interface** a ser buscada não seja encontrado, o objeto repositório não é invocado e retorna para o *stub client.* um parâmetro indicando a falha da operação.
- Caso o **tipo** tenha sido encontrado, obtém-se do **arquivo de acesso** a máquina no qual se encontra o repositório de interfaces e o seu identificador.
- O **nome da máquina** é utilizado na realização da chamada remota do procedimento ao objeto repositório.
- O **objeto de acesso** faz a chamada remota do procedimento passando como parâmetros uma estrutura de dados contendo o tipo de interfaces , o identificador da operação e o repositório onde se encontra essa interface e aguarda a resposta do seu serviço.
- De posse do resultado, o objeto de acesso retorna-o ao *stub cliente*, um parâmetro indicando o sucesso da operação.

Durante a realização de uma inclusão de uma interface, junto com uma operação, a função *include_rep* realiza as seguinters ações:

- A função *include_rep* do **objeto de acesso** é invocada através da chamada do *stub client* e recebe o tipo da interface, o identificador da operação e o identificador do repositório, onde serão incluídos esses dados.

- O **arquivo de acesso** é aberto para leitura e escrita do tipo da interface.
- Realiza-se a busca no arquivo de acesso da máquina onde está localizado o repositório.
- O **nome da máquina** é utilizado na realização da chamada remota do procedimento ao objeto repositório.
- Verifica-se a existência do tipo da interface a ser incluída no repositório de interfaces. Caso já exista, o arquivo de acesso não é atualizado, pois nesse caso o protótipo só possui tipo de interfaces iguais em diferentes repositórios.
- Caso esse tipo de interface não conste em nenhum dos repositórios, o **arquivo de acesso** é atualizado com uma *string* que contém o tipo da interface, o identificador do repositório e a máquina correspondente a esse repositório.
- O **objeto de acesso** realiza uma chamada remota do procedimento que realizará a inclusão da interface e/ou da operação no repositório de interfaces, e fica aguardando a resposta. Passa como parâmetros uma estrutura de dados contendo o identificador do tipo da interface e da operação a serem incluídas e o repositório onde será feita a inclusão.
- Recebe a resposta e a envia para o **stub cliente**.

Durante a realização de uma invocação dinâmica, a função *search_rep* executa as seguintes ações:

- A função *search_rep* do **objeto acesso** é invocada através da chamada do *stub cient* e recebe como parâmetro o tipo da interface através da rede.
- O arquivo de acesso é aberto para leitura.
- É verificado junto ao arquivo de acesso se esse tipo de interface existe. Caso não exista, o objeto repositório não é invocado e retorna-se ao cliente um parâmetro indicando que esse tipo de interface não é especificado no arquivo de acesso.
- Caso o tipo da interface exista, a função busca no arquivo de acesso a máquina onde está localizada o repositório referente à interface.
- O nome da máquina é utilizado para realizar a chamada do procedimento remoto.
- O objeto acesso realiza uma chamada remota do procedimento, passando como parâmetros uma estrutura de dados que contém o identificador do tipo de interfaces e o nome do repositório.

- O objeto de acesso recebe o nome das operações referentes ao tipo da interface específica, junto com o seu endereço.

6.7 Objeto Repositório

O objeto repositório no nosso protótipo é um servidor do RPC da SUN. O esquema possui dois objetos repositórios representados por máquinas sendo que cada um desses objetos interage somente com o seu respectivo repositório de interfaces, implementados no nosso protótipo como simples arquivos. Foram implementadas, parcialmente, as funções *include*, *lookup* e *search_address*, especificadas no modelo conceitual, e compatíveis com o RPC da SUN. A função *include* tem a finalidade de incluir uma interface junto com a sua operação. Esta função também atualiza o repositório de implementações caso seja necessário. A função *search_address* tem a finalidade de interagir com o repositório de implementações para buscar o endereço e a forma pela qual a operação está armazenada. A função *lookup* procura no repositório de interfaces um determinado tipo de interface, retornando um valor booleano referente ao sucesso ou não da busca.

Baseada nestas funções, vejamos como fica a seqüência de ações no objeto repositório para realizar uma verificação de tipos, inclusão de uma interface ou a invocação dinâmica de uma solicitação.

Durante a realização de uma verificação de tipos, a função *lookup* é invocada realizando as seguintes ações:

- A função *lookup* do **objeto repositório** é invocada por uma chamada do objeto de acesso. Recebe como parâmetros uma estrutura de dados contendo os identificadores do tipo da interface, da operação, e do repositório de interfaces.
- O **repositório de interfaces**, representado por uma árvore B, é aberto para leitura.
- Realiza-se uma busca dentro do repositório de interfaces do tipo da interface desejada, retornando o sucesso ou não da operação. Caso não se obtenha sucesso na operação, conclui-se que o conteúdo do repositório de interfaces não está compatível com o arquivo de acesso.
- Caso a **interface** seja encontrada, abre-se o arquivo referente ao tipo da interface desejada para leitura.
- Lê-se o **conteúdo do arquivo** realizando uma comparação com o nome da função solicitada pelo cliente.
- Caso a operação **não seja encontrada**, encerra-se a execução da função *lookup*, e o objeto de acesso é informado da incompatibilidade da operação.

- Caso a operação seja encontrada, o sucesso da função *lookup* é retornado ao objeto de acesso.

Durante a realização de uma inclusão de uma interface junto com uma operação, a função *include* realiza as seguintes ações:

- A função *include* do **objeto repositório** é invocada por uma chamada do objeto de acesso e recebe como parâmetro uma estrutura de dados contendo o identificador do tipo da interface, da operação e do repositório de interfaces onde será realizada a inclusão.
- O **repositório de interfaces**, representado por uma árvore B, é aberto para leitura e escrita.
- Realiza-se uma busca dentro do repositório de interfaces para verificar a existência ou não do tipo da interface a ser inserida.
- Caso o **tipo já exista**, é aberto o arquivo referente ao tipo da interface para leitura e escrita.
- Verifica-se a existência ou não dessa operação na interface desejada.
- Caso a **interface** já possua essa operação, a inserção não é feita e retorna-se ao objeto de acesso um parâmetro informando a existência dessa função.
- Caso a **interface** não possua esse tipo de operação, realiza-se a sua inserção no arquivo referente à interface, e retorna-se ao objeto de acesso um parâmetro indicando o sucesso da função *include*.
- Caso o **tipo da interface** não exista, é inserido o tipo de interface no repositório de interfaces (árvore B) e posteriormente cria-se um arquivo com o nome da interface a ser inserida com o tipo da operação.
- Abre-se o repositório de implementações para leitura e escrita.
- Verifica se a operação a ser inserida no repositório de interfaces existe no repositório de implementações. Caso não exista, insere-se esta operação no repostório com o seu endereço e sua forma de armazenamento. Esta funcionalidade tem a finalidade de comprovar que o esquema de acesso pode inserir novas operações no repositório de implementações criadas pelo sistema.

Durante a realização de uma invocação dinâmica as operações *lookup* e *search_address* são invocadas e realizam as seguintes ações:

- A função *lookup* do **objeto repositório** é invocada por uma chamada do objeto de acesso. Recebe como parâmetros o identificador do tipo de interfaces e da operação.
- O **repositório de interfaces**, representado por uma árvore B, é aberto para leitura.
- A função do objeto repositório realiza uma busca visando descobrir o tipo de interface enviada pelo objeto de acesso, no repositório de interfaces.
- Achada a interface, invoca-se a operação *search_address*.
- Abre-se o arquivo referente a interface e lê-se todas as operações lá existentes e verifica se esta operação lá existe.
- Caso exista, abre-se o repositório de implementações para buscar os endereço e a forma de armazenamento da operação referente à interface.
- Com o nome da operação, obtém-se as informações acima citadas. Essa funcionalidade é implementada, para demonstrar que o esquema de acesso tem possibilidades de fornecer ao adaptador e ao *skeleton* as informações necessárias à invocação correta da operação.
- Essas informações retornam ao objeto de acesso.

6.8 Armazenamento de Informações

Essa seção aborda as quatro bases de dados do esquema. O **arquivo de acesso**, ligado ao objeto de acesso, é um arquivo implementado sob a forma de *strings*. Cada um desses *strings* contém o tipo da interface, o identificador da máquina e o identificador do repositório. Com isso, esse arquivo possui todas as informações referentes as interfaces contidas nos dois repositórios de interface.

O **repositório de implementações**, também ligado ao objeto repositório, é um arquivo implementado com *strings*. Cada uma dessas *strings* contém o endereço (localização), o identificador e a forma como está armazenada uma operação. O identificador da operação é usado como uma entrada na obtenção de outros dados fornecidos por esse arquivo.

Os dois **repositórios de interfaces** estão estruturados em árvores do tipo B, que são de busca balanceada e trabalham bem em esquemas de armazenamento secundário com acesso direto. Cada nó dessa árvore possui quatro chaves, isto é, pode conter até quatro tipos de interface por nó. Cada chave possui um apontador para um arquivo com o nome da respectiva interface. Cada arquivo possui todas as operações correspondentes à interface que tem o nome do arquivo, possibilitando que cada tipo de interface possa ter mais de uma

operação. Para esse ambiente ORB nenhum repositório aceitará um tipo de interface que já tenha sido especifcado em outro repositório. Todas as operações definidas nos arquivos das interfaces, são também especificados no repositório de implementações[FZ92, CLR90].

6.9 Análise do Protótipo do Esquema de Acesso

Nesta seção é abordado alguns fatores suportados pelo protótipo, tentando demonstrar o quanto é vantajosa a utilização desse esquema de acesso dentro do contexto da CORBA:

- A implementação não faz restrições quanto ao aumento da quantidade de informações a disposição do cliente em tempo de execução. Esse aumento pode ser proveniente da inclusão de um novo tipo de interface com suas operações no repositório de interfaces, bem como a inclusão de uma nova operação no repositório de implementações. Toda e qualquer inserção ou deleção de tipos de interfaces no repositório de interfaces, o arquivo de acesso é atualizado.

- O acréscimo e a diversificação das informações através do aumento do número de repositórios em tempo de compilação possível, uma vez que a implementação do protótipo suporta quantos repositórios forem desejados, de forma que cada um esteja agregado a uma máquina. O repositório de interfaces não aceita a inserção de tipos de interfaces já existentes e nem a inserção de operações já existentes em uma determinada interface.

- Outro parâmetro fundamental que valida o nosso modelo é a funcionalidade apresentada pelo objeto de acesso dentro da implementação.

  - **Direcionamento**→ pode-se observar que o objeto de acesso na implementação recebe uma solicitação e envia esta ao objeto repositório, pois tem a sua disposição o arquivo de acesso que possui as informações necessárias para a realização dessa funcionalidade.
  - **Suporte a Falhas**→ o problema referente a queda de um nó ou da máquina onde está localizado o objeto de acesso. Isso pode ser sanado através da manutenção de cópias desse objeto em outro nó ou máquina no sistema resolvendo, assim, esse problema.

- A obtenção das informações pelo objeto repositório, através de suas operações a serem realizadas no repositório de interfaces, passa a ser mais rápida e precisa. A rapidez e a precisão vêm em função do objeto repositório já receber do objeto de acesso a operação a ser realizada junto com todas as informações necessárias a execução da mesma.

Capítulo 7

Conclusão

Nesta dissertação, apresentamos duas formas de acessar objetos em ambientes distribuídos, com o objetivo de otimizar e facilitar aos clientes a utilização de serviços e obtenção de informações em sistemas distribuídos. Para desenvolvermos este trabalho consideramos as arquiteturas dos sistemas distribuídos existentes, as especificações para Processamento Distribuído e para Interfaces para Programas de Aplicação e plataformas distribuídas comerciais.

Inicialmente especificamos uma API portável, que recebeu o nome de MID/API, com a finalidade de prover portabilidade às aplicações, bem como para possibilitar a execução, desenvolvimento e gerenciamento também das aplicações através de serviços providos por plataformas comerciais. A composição dos serviços é genérica, mas para a especificação foram consideradas as plataformas ANSA e DCE. Com essa API, esperamos que as aplicações possam ter acesso a diversos serviços, fornecidos por várias destas plataformas, por elas suportadas, deixando transparente a complexidade das mesmas, e em alguns casos a aplicação não tem a obrigação de conhecer qual plataforma invocar. Desta forma facilita-se a execução de solicitações de serviços. A MID/API provê serviços de alto e baixo níveis. Os serviços de alto nível foram divididos em módulos *midware*, com serviços referentes às plataformas *midware*, e *geral*, com serviços de âmbito geral (ex: deletar arquivo).

Especificamos um esquema de acesso aos repositórios de interfaces e implementações, para possibilitar a obtenção das informações disponíveis nestes repositórios de forma mais rápida e segura no ambiente da CORBA. Este esquema também apresenta soluções para resolver a interoperabilidade entre repositórios de interfaces, pertencentes ou não ao mesmo ambiente ORB, através do seu objeto de acesso. Com isso, esperamos que a camada *Middleware* possa oferecer serviços de processamento distribuído de forma mais rápida e segura às aplicações, em especial da plataforma *Multiware*.

Com a finalidade de validar as idéias deste esquema, implementamos um protótipo com as funções de verifcação de tipos, inclusão de interfaces e funções, e invocação dinâmica

de operações. Devido à disponibilidade de equipamentos, utilizamos plataformas SUN, no qual o RPC realizou a funcionalidade primária do ORB, e possibilitou a comunicação entre os objetos do esquema. O NFS do RPC da SUN foi o sistema de arquivos utilizado pelos repositórios no armazenamento de suas informações. Neste protótipo validamos o esquema de acesso, pois a proposta de se obter informações rápidas e seguras foi obtida devido à centralização da busca destas em torno do objeto e do arquivo de acesso, acionando assim o repositório correspondente à informação desejada. Este esquema foi projetado para atuar em um ambiente ORB como um todo, permitindo vários repositórios por ORB, utlizando-se de um arquivo de acesso para direcionar a solicitação da obtenção das informações ao repositório correspondente. Este esquema ainda possibilita a obtenção, no repositório de implementações, dos endereços e a forma como a operação estão armazenadas necessárias ao adaptador de objetos e *skeleton* para a correta inmvocação da operação. Por fim o esquema também possibilita tanto a inclusão de novas interfaces no repositório de interfaces quanto a inclusão de novas operações no repositório de implementações.

Esta dissertação também apresentou uma visualização da estruturação do esquema de acesso sob o modelo de referência (capítulo 5), possibilitando uma verificação de pontos compatíveis entre os modelos. Outro estudo realizado foi uma análise comparativa entre a CORBA e a MID/API (capítulo 4) sobre várias funcionalidades, bem como a verificação das vantagens e desvantagens de se utilizar uma ou outra na plataforma *Multiware*, dando ao projetista uma opção a mais de escolha para a camada *Middleware*. Como vantagens da utilização da CORBA citou-se por exemplo a sua simplicidade na utilização pelo cliente e a utilização do *Broker*. Como vantagem da MID/API citou-se por exemplo a possibilidade do cliente chamar serviços de alto nível.

Com a finalidade de dar continuidade ao trabalho por nós desenvolvido, alguns fatores devem ainda serem analisados e desenvolvidos para que os objetivos iniciais propostos por esta dissertação possam ser mantidos bem como expandidos.

Em relação à MID/API, estudos devem ser aprofundados para que novas plataformas possam ser suportadas, dando assim uma maior disponibilidade de serviços a serem oferecidos às aplicações de um sistema, como por exemplo, a introdução da ORB quando do aparecimento de novos serviços de objetos. Implementar todas as suas funcionalidades, tendo como princípio a metodologia orientada a objetos e o objetivo de manter sempre a transparência ao cliente da complexidade da plataforma e em alguns casos a transparência de plataforma (a aplicação não precisa conhecer qual plataforma atenderá a sua solicitação).

Quanto ao esquema de acesso, prosseguirão os estudos para estabelecer uma quantidade maior de funcionalidades além das propostas, procurando a melhor forma de implementar todas as funções propostas no modelo conceitual. No protótipo verificou-se a funcionalidade dos objetos do esquema, constatando que as idéias apresentadas no modelo conceitual foram validadas através deste protótipo, porém existe ainda a necessidade de se testar a

funcionalidade das outras funções propostas pelo modelo conceitual.

Outro fator a ser analizado dentro do esquema de acesso é a interoperabilidade entre esquemas em ambientes diferentes, estruturando de uma forma melhor o objeto de acesso para possibilitar a implementação do mesmo, bem como a realização de testes entre repositórios em vários ambientes ORB.

A proposta da API avança no sentido de que antes tínhamos APIs para sistemas de comunicação, e agora temos APIs para plataformas distribuídas (ex: ANSA e DCE). O próximo passo seria a possibilidade de se ter suporte à Aplicação Distribuída (ex: TMN[1]), no qual teríamos APIs para oferecer os serviços de Gerenciamento de Telecomunicações, verificando-se a tendência de se ter APIs num nível mais alto.

Por último, análises baseadas em testes deverão ser realizadas para que seja verificada a adequação da CORBA ou ANDC/API à utilização na camada *Middleware* da plataforma *Multiware*, bem como ao oferecimento de serviços às aplicações em ambientes distribuídos.

[1]Telecommunication Management Network

Apêndice A

Especificação dos Objetos do Repositório de Interfaces

Este apêndice possui a descrição de todas as definições de interfaces dos objetos contidos no repositório de interfaces. Estas definições de interfaces são descritas a partir de interfaces abstratas conhecidas por interfaces *container* e *contained*. A descrição destas interfaces é feita da seguinte forma:

typedef string **identifier**;
//identifica módulos, interfaces, constantes, typedefs, exceções, atributos e operações.
typedef string **RepositoryId**;
//usado pelo ORB para identificar módulo, interface, constante, typedef, exceção, atributo ou operação.
typedef identifier **InterfaceName**;
//é um string que contém o nome de um dos objetos do repositório de interfaces.
interface Container;
interface Contained;
typedef sequence<container>; **ContainerSeq**;
//define um template que retorna todos os objetos que ele contém.
typedef sequence<contained>; **ContainedSeq**;
//define uma lista que retorna todos os objetos que o contém

A estrutura da interface Contained é a seguinte:
Interface Contained{

struct Description{
identifier name;//identifica o objeto contido por outros objetos.

any value;//qualquer valor que deseja retornar na interface.
};
attribute identifier name;//identifica o objeto contido por outros objetos.
attribute Repositoryld id;//identifica também o objeto contido por outros objetos.
attribute Repositoryld defined_in;//identifica a interface do objeto do qual ele é derivado.
ContainerSeq within();
Description describe();
};

As operações do objeto repositório referentes à interface Contained são:

- **within** → essa operação tem por finalidade retornar a lista dos objetos que o contêm. Caso seja uma interface ou módulo, o objeto só poderá ser contido por objetos que o definem.
- **describe** → essa operação tem por finalidade retornar a estrutura contendo todos os atributos definidos para a interface. Para cada definição de interface existe um estrutura com um nome, no qual este nome retorna junto com a específica estrutura. Por exemplo, se a operação describe é invocada no objeto atributo, o campo *name* contém *AttributeDescription*, e o campo *value* contém o *any* que contém a estrutura *AttributeDescription*.

A interface Container é usada para localizar objetos que são contidos por outros objetos. Sua estrutura é a seguinte :

Module CORBA{

Interface Container{

struct Container{
Contained contained_object;
identifier name;//identifica o objeto contido por outros objetos. any value;//qualquer valor que deseje retornar na interface.
};
typedef sequence<Description> DescriptionSeq;
//define um template que retorna uma estrutura do tipo Description.

ContainedSeq contents(
in InterfaceName limit_type,

```
//se limit_type for all, os objetos de todos os tipos de interfaces são retornados. Caso
assuma uma interface específica, são retornados os objetos específicos a essa interface.
        in boolean    exclude_inherited,
//se exclude_inherited for igual a true objetos herdados não são retornados. Caso seja
false todos os objetos contidos são retornados.
    );
    ContainedSeq   lookup_name(
      in identifier      search_name,//especifica que nome será buscado.
      in long            levels_to_search,//controla o nível de busca no objeto desejado.
      in InterfaceName   limit_type,
      in boolean         exclude_inherited
    );
    DescriptionSeq   describe_contents(
      in InterfaceName   limit_type,
      in boolean           exclude_inherited,
      in long            max_returned_objs
//limita o número de objetos que podem ser retornados numa invocação.
    );
  };
};
```

As operações do objeto repositório referentes a interface Container são:

- **contents** → esta operação tem por finalidade retornar uma lista de objetos contidos pelo objeto. Ela é utilizada para navegar através da hierarquia dos objetos, começando pelo objeto repository.
- **lookup_name** → essa operação tem por finalidade localizar um objeto pelo nome dentro de outro, ou dentro de objetos contidos por ele.
- **describe_contents** → esta operação é uma combinação das operações contents e describe. Para cada objeto retornado pela operação contents, a descrição do objeto é retornada.

Além dessas interfaces, são definidos os *TypeCodes*, que são valores que representam a invocação de tipos de argumentos e atributos. Eles têm uma variedade de utilizações. São utilizados pelas interfaces de invocação dinâmica para indicar os tipos de argumentos, e pelo repositório de interfaces na especificação de algumas interfaces.

Sua interface é definida da seguinte forma:

Interface TypeCode{

```
    enum Kind{
        tk_null,tk_void,tk_short,tk_long,tk_ushort,tk_ulong,tk_float,
        tk_double,tk_boolean,tk_char,tk_array,tk_string,tk_enum,tk_any,
        tk_sequence,tk_objref
};
    long param_count();//número de parâmetros para esse TypeCode.
}
```

Objetos do Repositório de Interfaces

Nesta seção estão definidos todos os objetos que compõem o repositório de interfaces, com suas respectivas operações.

- **Objeto Repository** → é um objeto que provê o acesso global ao repositório de interfaces.

  A estrutura da interface Repository é a seguinte:

```
Interface Repository : Container{
    contained lookup_id(
     in RepositoryIdsearch_id//especifica qual o objeto a ser buscado);
};
struct RepositoryDescription{
    identifier      name;
    RepositoryId    id;
    RepositoryId    defined_in;
};//essa estrutura é definida para ser usada exclusivamente pela operação describe_contents.
```

  As operações do objeto Repository são:

  - **lookup_id** → esta operação é usada para buscar um objeto no repositório de interfaces, dado o seu RepositoryId.
  - **contents** → definida pela interface *container*, pode ser usada para listar as constantes, definições de tipos, exceções, interfaces e módulos contidos pelo objeto repositório.
  - **lookup_name** → definida pela interface *container*, pode ser usada para achar instâncias de um particular nome dentro do repositório de interfaces.

- **Objeto ModuleDef** → este objeto tem por finalidade representar os objetos constantes, definições de tipos, exceções, interfaces e outros módulos.

  A estrutura da interface ModuleDef é a seguinte:

  Interface ModuleDef : Container, Contained{};

  struct ModuleDescription{

  identifier name;

  RepositoryId id;

  RepositoryId defined_in;

  };//esta estrutura é definida para ser usada pela operação describe.

  Este objeto faz uso da operação **lookup_name**, a qual é usada para localizar constante, atributo ou a operação definida nesse módulo.

- **Objeto AttributeDef** → tem por objetivo representar a informação que define um atributo.

  A interface AttributeMode é especificada da seguinte forma:

  enum AttributeMode {Normal, Readonly};

  Interface AttributeDef : Contained{

  attribute TypeCode type;//especifica o TypeCode desse atributo.

  attribute AttributeMode mode;

  //especifica se o atributo é somente para leitura ou para leitura e escrita.

  };

  struct AttributeDescription{

  Identifier name;

  RepositoryId id;

  RepositoryId defined_in;

  Typecode type;

  AttributeMode mode;

  };//essa estrutura é definida para uso pela operação describe.

- **Objeto InterfaceDef** → tem por objetivo representar a definição de interface.

A estrutura da InterfaceDef é a seguinte:

Interface InterfaceDef : Container, Contained{

struct FullInterfaceDescription{

identifier name;

RepositoryId id;

RepositoryId defined_in;

OpDescriptionSeq operations;//descreve todas as operações da interface.

AttrDescriptionSeq attributes;//descreve todos os atributos da interface.

};

attribute RepositoryIdSeq base_interfaces;

//esse atributo representa a lista de todas as interfaces do qual uma interface herda.

FullInterfaceDescription describe_interface();

//esta operação possibilita a obtenção da descrição de todas as operações e atributos definidos na interface através da estrutura FullInterfaceDescription.

};

struct InterfaceDescription{

identifier name;

RepositoryId id;

RepositoryId define_in;

};//esta estrutura é definida para uso da operação describe.

As operações referentes ao objeto interface são as seguintes:

- **contents** → definida pela interface *container*, retorna a lista de constantes, definições de tipos e exceções definidas nesse objeto, como também a lista de atributos e operações definidas ou herdadas por esse objeto.
- **lookup_name** → definida pela interface *container*, é usada para localizar a constante, typedef, exceção, atributo, parâmetro ou operação definido nesse objeto pelo nome.

- **describe_contents** → definida pela interface *container*, provê uma completa descrição dessa interface.
- **describe_interface** → provê uma conveniente forma de obter a descrição de todas as operações e atributos definidos na interface, através do retorno da estrutura FullInterfaceDescription.

- **Objeto OperationDef** → tem por objetivo representar a informação para definir a operação.

A estrutura da interface OperationDef é a seguinte:

enum OperationMode {OP_NORMAL, ATTR_READONLY};

Interface OperationDef : Contained{

attribute TypeCode result;

//este atributo especifica o TypeCode do valor retornado por essa operação.

attribute OperationMode mode;

//este atributo especifica o modo da operação. Essa operação retornará ou não um resultado.

attribute IdentifierSeq contexts;

//este atributo especifica a lista de identificadores de contexto que se aplica a operação em questão.

};

struct OperationDescription{

identifier name;

repositoryId id;

repositoryId defined_in;

TypeCode result;

OperationMode mode;

StringSeq contexts;

//especifica a lista de identificadores de contexto que se aplica a essa operação.

ParDescriptionSeq parameters;//define a lista de parâmetros da operação.

ExcDescriptionSeq exceptions;//define a lista de exceções da operação.

};//esta estrutura é definida para ser utilizada pela operação describe.

- **Objeto ParamaterDef** → tem por objetivo representar a informação necessária para definir o argumento da operação.

A estrutura da interface ParameterDef é a seguinte:

enum ParameterMode {IN, OUT, INOUT};//enumera se o parâmetro é de entrada, saída, ou entrada e saída.

Interface ParameterDef : Contained{

attribute TypeCode type;//este atributo especifica o TypeCode desse argumento

attribute ParameterMode mode;//este atributo especifica se o argumento é usado como entrada, saída ou entrada e saída

};

struct ParameterDescription{

identifier name;

RepositoryId id;

RepositoryId defined_in;

TypeCode type;

ParameterMode mode;

};//esta estrutura é utilizada pela operação describe.

- **Objeto TypedefDef** → são criados somente para tipos com nomes, não para tipos anônimos.

A interface TypedefDef é estruturada da seguinte forma:

Interface TypedefDef : Contained {

attribute TypeCode type;

//este atributo define o TypeCode para essa definição de tipo.

};

struct TypeDescription{

identifier name;

RepositoryId id;

Repositorylpd define_in;

TypeCode type;

};//esta estrutura é utilizada pela operação describe.

- **Objeto ConstantDef** → tem por objetivo definir o nome da constante.

A estrutura da interface ConstantDef é a seguinte:

Interface ConstantDef : Contained {

attribute TypeCode type;

//este atributo especifica o TypeCode para essa constante de tipo simples (long, short, float, etc).

attribute any value;//contêm o valor da constante.

};

struct ConstantDescription{

identifier name;

Repositoryld id;

Repositoryld defined_in;

TypeCode type;

any value;

};//esta estrutura é usada pela operação describe.

- **Objeto ExceptionDef** → tem por objetivo representar a definição da exceção.

A estrutura da interface ExceptionDef é a seguinte:

Interface ExceptionDef : Contained {

attribute TypeCode type;//este atributo especifica o tipo de parâmetro de exceção.

};

struct ExceptionDescription{

identifier name;

Repositoryld id;

Repositoryld defined_in;

```
    TypeCode    type;
};
```

Bibliografia

[ABD+89] Malcon Atkinson, François Bancilkon, David Diltrich, David Maier, and Stanley Zdonik. The object-oriented database system - manifesto. *Report-Technical*, (30), Outubro 1989.

[Ber93] Philip A. Bernstein. Middleware an architeture for distrubuted system services. Março 1993.

[BGHS91] Gordon Blair, John Gallagher, David Hutchison, and Dong Skepherd. *Object-Oriented Languages, Systems and Applications.* Pitman Publishing, 1991.

[Bla89] U.D. Black. *Data Networks.* Prentice-Hall, 1989.

[CCODIC93] Digital Equipment Corporation, NCR Corporation, Hyper Desk Corporation Object Design Inc, SunSoft Inc, and Hewlett-Packard Company. The common object request broker: Architecture and specification. *OMG*, 1.2(93.xx.yy), Dezembro 1993.

[CD88] George F. Coulouris and Jean Dollimore. *Distributed Systems Concepts and Desing.* Addison-Wesley, 1988.

[Cen92a] Framingham Corporate Center. Object services architecture. *OMG*, 6(92-8-4), Agosto 1992.

[Cen92b] Framingham Corporate Center. Object services roadmap. *OMG*, (92-8-5), Agosto 1992.

[Cen93a] Framingham Corporate Center. C++ language mapping request for proposals. *OMG*, (92-12-11), Abril 1993.

[Cen93b] Framingham Corporate Center. Object services requets for proposal. *OMG*, (92-8-6), Fevereiro 1993.

[CLR90] Thomas H. Cormem, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The Mit Press and Mc Graw-Hell Book Company, 1990. .

[dM95] Alexandre M. Vaz de Mello. *Um Serviço de Comunicação entre Objetos Remotos.* Dissertação de Mestrado em andamento do DCA da UNICAMP, 1995.

[dPLJ94] Luiz Augusto de Paula Lima Júnior. *Um Modelo para a Implementação de Federação de Traders.* Dissertação de Mestrado - Departamento da Ciência da Computação da UNICAMP, 1994.

[EM90] E.R.M.Madeira and M.J.Mendes. An application interface model for communication software. *IEEE Telecommunications Conference - GLOBECOM'90*, Dezembro 1990.

[EWM94] E.R.M.Madeira, W.P.D.C.Loyolla, and M.J.Mendes. Multiware platform: Some issues about the middleware and groupware layers. *Submetido ao ICODP 95*, 1994.

[FC93] F.Vogt and C.Andrae. Middleware for distributed applications support:odp and/or corba. *International Conference on Open Distributed Processing*, 1:405–410, Setembro 1993.

[Fle92a] Eric Fleischman. A user guide to data communications apis. *The Open Systems Newsletter*, 6(5), 1992.

[Fle92b] Eric Fleischman. A user guide to data communications apis , part 2. *The Open Systems Newsletter*, 6(6), 1992.

[FZ92] Michael J. Folk and Bill Zoellick. *File Structures.* Addison-Wesley Publishing Company Inc., 1992.

[HOL+93] Andrew Herbert, Dave Otway, Rob Vander Linden, Andrew Watson, and Ian Domville. Ansa - orb interoperability. *Architecture Projects Management Limited*, (OMG-93-5-9), Maio 1993.

[Inc93] Hewlett-Packard Company IONA Technologies Ltd-Sus Soft Inc. Idl c++ language mapping specification. *OMG*, (93-4-4), Abril 1993.

[Inc94] Post Modern Computing Technologies Inc. *ORBeline User's Guide.* Post Modern Computing Technologies Inc., 1994.

[ISO93] ISO/IEC. *Basic Model of ODP-Part 2 : Descritive Model.* ISO, 1993.

[ISO94a] ISO/IEC. *Basic Model of ODP-Part 1 : Overview and Guide to use.* ISO, 1994.

[ISO94b] ISO/IEC. *Basic Model of ODP-Part 3 : Prescritive Model.* ISO, 1994.

[JJP+93] J.Slonim, J.W.Hong, P.J.Finnigan, N.Coburn, D.L.Erickson, and M.A.Bauer. Does middleware provide an adequate distributed application environment. *International Conference on Open Distributed Processing*, 1:34–46, Setembro 1993.

[K.R93] K.Raymond. Reference model of open distributed processing: a tutorial. *International Conference on Open Distributed Processing*, 1:3–14, Setembro 1993.

[LEM+94] W.P.D.C. Loyolla, E.R.M.Madeira, M.J.Mendes, E.Cardozo, and M.F.Magalh aes. Multiware platform: an open distributed environment for multimedia cooperative applications. *COMPSAC*, 1994.

[Lim89] Architeture Projects Management Limited. *ANSA : An Engineer's Introducion.* Architeture Projects Management Limited, 1989.

[Lim91a] Architeture Projects Management Limited. *ANSAware 3.0 Implementation Manual.* Architeture Projects Management Limited, 1991.

[Lim91b] Architeture Projects Management Limited. *ANSAware 3.0 Release Notes.* Architeture Projects Management Limited, 1991.

[MAP88] MAP_Attachment. Application interface model and specification requirements. *Interface Model and Especification Requirements*, 1988.

[Mic90] SUM Microsystems. *RPC Manual.* SUN microsystems, 1990.

[MM94] M.J.Mendes and E.R.M. Madeira. Plataforma multiware: Projeto e desenvolvimento da camada middleware. *SBRC*, (12), Fevereiro 1994.

[Mul89] Sape Mullender. *Distributed Systems.* ACM PRESS, 1989.

[MWE93] M.J.Mendes, W.P.D.C.Loyolla, and E.R.M.Madeira. Demos: A distributed decision - making open support system. *4th IEEE Workshop on Future Trends of Distributed Computing System - Lisboa*, Setembro 1993.

[Mye92] Brad A. Myers. Demonstrational interfaces : A step beyond direct manipulation. *Computer*, 25, August 1992.

[New92] Owen Newman. Se-osi : A prototype suport envirommment for open system interconnection. *Computer Communication Review*, 22(2), Abril 1992.

[OSF90a] OSF. *Directory Services for a Distributed Computing Environment.* OSF, 1990.

[OSF90b] OSF. *Distributed Computing Environment Overview.* OSF, 1990.

[OSF90c] OSF. *Distributed Computing Environment Rationale.* OSF, 1990.

[OSF90d] OSF. *File Systems in a Distributed Computing Environment.* OSF, 1990.

[OSF90e] OSF. *Remote Procedure Call in a Distributed Computing Environment.* OSF, 1990.

[PS85] James L. Peterson and Abraham Silberschatz. *Operating System Concepts.* Addison-Wesley, 1985.

[PZTT92] R. Popescu-Zeletin, V. Tschammer, and M. Tschichholz. A service plataform for distributed applications. 1992.

[RMM92] Edmundo R.M, Madeira, and Manoel J. Mendes. Programming environments for application software. 1992.

[Shn87] Ben Shneiderman. *Designing the User Interface : Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1987.

[SLC93] M.Van Sinderen, L.F.Pires, and C.A.Vissers. Design concepts for open distributed systems. *International Conference on Open Distributed Processing*, 1:369–374, Setembro 1993.

[Tan89] Andrew Tanenbaum. *Computer Networks.* Prentice-Hall, 1989.

[Tan92] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice-Hall, 1992.

[TLX90] Tadao Takakashi, Hans K.E. Liesenberg, and Daniel Tavares Xavier. *Programação Orientada a Objetos - Uma Visão Integrada do Paradigma de Objetos.* USP, 1990.

[VMW+93] V.Tcshammer, M.J.Mendes, W.L.Souza, E.R.M.Madeira, and W.P.Loyolla. Processamento distribuído aberto e o modelo rm-odp/iso. *Anais do Simpósio Brasileiro de Redes de Computadores*, 1(11), Maio 1993.

[WJJJ89] W.F.Giozza, J.F.M.Araújo, J.A.B.Moura, and J.P.Sauvé. *Redes Locais de Computadores.* Mc. Grown, 1989.

[Zuq95] Nuccio Marcel Scott Zuquello. *Um Esquema para Possibiltar a Interoperabilidade entre Plataformas ORBs baseado em Proxies.* Dissertação de Mestrado em andamento do Departamento da Ciência da Computação da UNICAMP, 1995.