UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

UM PROTÓTIPO DE NEGOCIADOR DE REQUISIÇÕES DE OBJETOS

por: Alexandre Magalhães Vaz de Mello

orientador: Prof. Dr. Manuel de Jesus Mendes

Tese apresentada à Faculdade de Engenharia Elétrica FEE - UNICAMP como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA

Este exemplar corresponde à redação final da tese defendida por Alexandre M. V. de Mello e aprovada pela Comissão Julgadora em ____

Orientador

Março/1995

UNICAMP
BIBLIOTECA CENTRAL

AGRADECIMENTOS E DEDICATÓRIA

Aos meu pais, Conceição e Fernando, sem cuja colaboração, amor e carinho, nada teria se concretizado e a minha irmã Luciana pelo carinho dispensado.

A Cristiane por todo o incentivo e apoio para a conclusão do trabalho.

Ao CNPq, pelo apoio financeiro.

À Universidade Estadual de Campinas pelo ambiente acadêmico e pela infra-estrutura de pesquisa.

Ao Prof. Dr. Manuel de Jesus Mendes pela orientação ao longo do desenvolvimento do trabalho.

A todos os colegas do LCA, em especial Maria Inês Valderrama Restovic e Plínio Roberto de Souza Vilela, cujos incentivos e contribuições foram fundamentais para a conclusão do trabalho.

Ao amigo José Guilherme, pelas discussões intermináveis. Aos amigos Alfredo, Élcio, Gustavo e Leonardo pelas visitas na FENASOFT. A Evelyn pelo apoio e incentivo para fazer o mestrado.

A todos os professores e colegas do projeto MULTIWARE, em especial ao Otávio, Cláudio e Marcos Euzébio pelas contribuições advindas de nossas discussões.

A Andersen Consulting pela disponibilidade de tempo concedida para a conclusão do trabalho.

Para meus pais, Conceição e Fernando.

"Até onde sabemos, nosso computador nunca teve um erro não detectável."

Weisert

RESUMO

O presente trabalho faz uma análise da padronização de sistemas distribuídos orientados a objeto no que concerne ao modelo de objetos e à infra-estrutura de comunicação através de uma comparação das abordagens das duas organizações mais importantes nesse processo: ISO (*International Standards Organization*) e OMG (*Object Management Group*). Também foi implementado um protótipo de negociador de requisições de objetos em ambiente UNIX baseado em RPC (*Remote Procedure Call*) e nos documentos de padronização das organizações já citadas.

ABSTRACT

This work makes an analysis of the standardization of object oriented distributed systems concerning the object model and the communication framework by means of a comparison of the approaches of the two major organizations in this process: ISO (International Standards Organization) and OMG (Object Management Group). A prototype of an ORB (Object Request Broker) was also implemented in a UNIX environment using RPC (Remote Procedure Call) and the documents of standardization published by the organizations mentioned.

SUMÁRIO

Página

AGRADECIMENTOS E DEDICATÓRIA i

RESUMO iv

ABSTRACT v

SUMÁRIO vi

LISTA DE FIGURAS x

LISTA DE TABELAS xiii

LISTA DE ABREVIATURAS xiv

CAPÍTULO 1. INTRODUÇÃO 1

CAPÍTULO 2. MODELOS DE OBJETO 6

2.1 Modelo de Objeto para RM-ODP 6

2.1.1 Modelamento de um Objeto 6

2.1.2 Características dos Objetos 11

2.1.3 Conceitos de Comportamento 13

2.1.4 Criação e *Binding* de Interfaces 15

2.2 O modelo de objeto da OMG 17

2.2.1 O Core Object Model 19

2.3 Comparação entre os dois Modelos 23

2.3.1 Objetos Clientes e Clientes 23

2.3.2 Composição e Decomposição de Objetos 23

2.3.3 Comportamentos e Ações 23

2.3.4 Templates 24

2.3.5 Instanciação e Instâncias 24

2.3.6 Criação de Objetos 24

2.3.7 O Conceito de Tipo 25

2.3.8 Referências de Objeto e Referências de Interface 25

CAPÍTULO 3. INFRA-ESTRUTURA E SERVIÇOS AUXILIARES 27

3.1 ODP 27

3.1.1 A infra-estrutura definida no ponto de vista de engenharia do RM-ODP 27

3.1.2 Funções definidas por ODP 33

3.1.3 Transparências de Distribuição Propostas pelo RM-ODP 37

3.2 OMG 38

3.2.1 A Infra-estrutura proposta por CORBA (Common Objetc Request Broker Architecture) 39

3.2.1.1 Modelo de Objeto Utilizado pela Arquitetura 40

3.2.1.2 Visão Global da Arquitetura 41

3.2.1.3 Os Componentes do Sistema 42

3.2.1.4 O Funcionamento no Cliente 43

3.2.1.5 O Funcionamento no lado da Implementação de Objeto 44

3.2.1.6 Stubs, Esqueletos e Repositórios 45

3.2.1.7 A Estrutura de um Cliente 46

3.2.1.8 A Estrutura de uma Implementação de Objeto 47

3.2.1.9 Uma Implementação que suporta várias Interfaces 48

3.2.2 A Linguagem de Definição de Interface proposta por CORBA 49

3.2.3 Os Serviços de Objeto propostos pelo OMG 52

3.3 Análise Comparativa 56

3.3.1 Infra-estrutura para Comunicação entre Objetos 56

3.3.2 Serviços Auxiliares 58

3.3.2.1 Funções de Gerenciamento 58

3.3.2.2 Funções de Coordenação 60

3.3.2.3 Funções de Repositório 61

3.3.2.4 Funções de Segurança 61

CAPÍTULO 4. UM PROTÓTIPO PARA INVOCAÇÃO DE OPERAÇÕES EM OBJETOS REMOTOS 62

4.1 Requisitos do Protótipo 62

4.2 A Solução RPC 62

4.2.1 O sistema RPC da SUN 63

4.2.1.1 O rpcgen 64

4.2.2 O padrão XDR 66

4.2.3 O Protocolo RPC da SUN 68

4.3 A Abordagem de Implementação Usando RPC 70

4.4 O Protótipo como foi Implementado 71

4.4.1 Restrições do sistema em relação ao padrão CORBA 72

4.4.2 Fluxo de Informação no Sistema 73

4.4.3 O Funcionamento no Cliente 75

4.4.4 O Funcionamento na Implementação do Objeto 76

4.4.5 A Referência de Objeto 77

4.4.6 O Mecanismo de Objeto Cliente e Objeto Servidor 78

4.4.7 Plataforma e Ferramentas de Desenvolvimento Utilizadas 80

4.5 Especificação do Mapeamento IDL para Stubs e Esqueletos 81

4.5.1 Mapeamento para Stubs 81

4.5.2 Mapeamento para Esqueletos 81

4.5.3 Mapeamento para Tipos de Dados 82

4.6 Análise do Sistema Implementado Segundo o Modelo ODP 82

CAPÍTULO 5. CONCLUSÕES E TRABALHOS FUTUROS 83

5 .1 Sugestões para Trabalhos Futuros 84

BIBLIOGRAFIA 86

LISTA DE FIGURAS

Figura Página

Figura 1-1. Pontos de Vista ODP 4

Figura 1-2. Modelo de Referência OMG 5

Figura 2-1. Composição de Objetos 8

Figura 2-2. Decomposição de Objetos 8

Figura 2-3. A Instanciação de um Template pode produzir um Objeto, uma Interface ou a ocorrência de uma Ação 10

Figura 2-4. Templates, suas Instanciações e Instâncias 10

Figura 2-5. Uma Hierarquia de Classes para Formas Geométricas 11

Figura 2-6. Contratos, Liaison e Comportamentos 13

Figura 2-7. No Estado Inicial existe um Objeto I que será o Importador 15

Figura 2-8. É Instanciado um Objeto E que será o Exportador 15

Figura 2-9. O Objeto E cria uma Interface do tipo P1. Essa Interface tem o papel de servidor 15

Figura 2-10. O Objeto E anuncia a existência da Interface X 16

Figura 2-11. Criação da interface Y do tipo T2 no Objeto I como Interface Cliente 16

Figura 2-12. O Objeto I decide falar com uma Interface do tipo T 17

Figura 2-13. O Objeto I descobre por algum meio que "X" é uma Referência de Interface para uma Interface do tipo T 17

Figura 2-14. É realizado o Binding entre as Interfaces: I e E passam a interagir. ... 17

Figura 3-1. Exemplo da Estrutura de um Nó 28

Figura 3-2. Exemplo da Estrutura de uma Cápsula 29

Figura 3-3. Exemplo da Estrutura que apoia um Objeto Básico de Engenharia 30

Figura 3-4. Um Exemplo de Canal Cliente Servidor Ponto a Ponto 31

Figura 3-5. Um Exemplo de Canal Multiponto 32

Figura 3-6. Tradução da Computação para a Engenharia 33

Figura 3-7. Valores Legais 41

Figura 3-8. Uma Requisição enviada através do ORB 42

Figura 3-9. A estrutura da Interface do ORB 43

Figura 3-10. Um Cliente usando uma Stub ou a Interface de Invocação Dinâmica. 44

Figura 3-11. Uma Implementação de Objeto Recebendo uma Requisição 45

Figura 3-12. Repositórios de Interface e de Implementação 46

Figura 3-13. A Estrutura de um Cliente Típico 47

Figura 3-14. A Estrutura de uma Implementação de Objeto Típica 48

Figura 3-15. A Estrutura de um Adaptador de Objetos Típico 49

Figura 3-16. Exemplo de Herança Múltipla Legal 50

Figura 4-1. Estrutura do Sistema RPC da SUN 64

Figura 4-2. Arquivos e Processos para Geração de um Sistema RPC utilizando o rpcgen 66

Figura 4-3. Módulos para Geração dos Executáveis do Cliente e do Servidor no Protótipo 72

Figura 4-4. A Arquitetura do Sistema 74

Figura 4-5. Funcionamento no Cliente 76

Figura 4-6. Funcionamento na Implementação de Objeto 77

Figura 4-7. O Mecanismo do Objeto Cliente/Objeto Servidor 79

Figura 4-8. Múltiplos Objetos Clientes Referenciando o Mesmo Objeto Servidor.. 80

LISTA DE TABELAS

Tabela Página

Tabela 3-1. Funções de Gerenciamento 34

Tabela 3-2. Funções de Coordenação 35

Tabela 3-3. Funções de Repositório 36

Tabela 3-4. Funções de Segurança 37

Tabela 3-5. Transparências de Distribuição 38

Tabela 3-6. Serviços de Objeto 56

Tabela 3-7. Comparação de Infra-estrutura 58

Tabela 3-8. Comparação entre Funções de Gerenciamento ODP e conceitos OMG60

Tabela 3-9. Comparação entre Funções de Coordenação ODP e Conceitos OMG. 61

Tabela 3-10. Comparação de Funções de Repositório ODP com conceitos OMG . 61

Tabela 4-1. Mapeamento dos Tipos de Dados definidos na CORBA para os disponíveis em XDR 68

LISTA DE ABREVIATURAS

CORBA	Common Object Request Broker Architecture
DES	Data Encryption Standards
DIS	Draft International Standard
IDL	Interface Definition Language
ISO	International Standards Organisation
ODP	Open Distributed Processing
OMG	Object Management Group
ORB	Object Request Broker
PDU	Protocol Data Unit
RFP	Request For Proposals
RM-ODP	Reference Model for Open Distributed Processing
RM-OSI	Reference Model for Open Systems Interconnection
RPC	Remote Procedure Call
SNA	System Network Architecture
TCP	Transport Control Protocol
TCP/IP	Transport Control Protocol / Internetworking Protocol
UDP	User Data Protocol
XDR	eXternal Data Representation

CAPÍTULO 1

INTRODUÇÃO

Rapidamente os computadores têm sido interligados formando redes, como uma consequência das facilidades criadas pelo desenvolvimento da tecnologia empregada em telecomunicações. Devido à grande disponibilidade de equipamentos a custo baixo para interconexão de computadores uma grande parte deles foi interligada e apareceram problemas nas suas comunicações. Esses problemas, consequência de deficiências e interferências no *hardware*, geram informações errôneas ou sua perda. Para assegurar que a transmissão se efetue mesmo com esses problemas de comunicação foram criados sistemas de *software* que têm mecanismos para resolver esses e outros problemas, como conversão de dados e roteamento de informação. Esses sistemas de *software* usam protocolos de comunicação para garantir a transmissão correta de dados. Esses sistemas foram criados por vários fornecedores, cada um deles usando seu protocolo próprio, impedindo a comunicação entre dois equipamentos que não usassem o mesmo protocolo.

As necessidades dos protocolos de comunicação foram analisadas pela ISO (*International Standards Organization* ou Organização Internacional de Padrões) e estes foram padronizados através do RM-OSI (*Reference Model for Open Systems Interconnection* ou Modelo de Referência para Interconexão de Sistemas Abertos). Neste padrão, os protocolos de comunicação foram estruturados em sete níveis, cada um com uma função específica. Esse é o padrão *de juri* para interligação de equipamentos de processamento de dados. Simultaneamente à criação do RM-OSI, produtos para interligação muito usados evoluiram em padrões de fato. Entre eles podem ser citados o SNA (*System Network Architecture* ou Arquitetura de Sistema de Rede) da IBM para seus *mainframes*, o NETWARE da NOVELL para computadores pessoais padrão IBM e o

TCP/IP (*Transport Control Protocol/ Internetworking Protocol* ou Protocolo de Controle de Transporte/ Protocolo Inter-rede) para equipamentos que usam sistema operacional UNIX. A estrutura padronizada pela ISO facilita a definição dos protocolos, o projeto e a implementação de *software* que os atendam. Com esta padronização foi possível a conexão em rede de um grande número de computadores. Como exemplo pode ser citada a rede do departamento de defesa dos EUA que abrangia todo o país e começou a ser usada também pelas universidades daquele país. Gradualmente universidades e outras instituições estrangeiras também se juntaram a essa rede que hoje está com centenas de milhares de nós e milhões de usuários.

Foi criado por esta rede um mercado onde serviços começaram a ser oferecidos aos usuários. Um serviço pode então ter sua execução solicitada a qualquer equipamento que o ofereça. Os problemas criados com estas solicitações e execuções de serviços são referentes à definição e à caracterização dos serviços prestados por equipamentos de processamento de dados. Surgiram dúvidas para as quais não se tinha ainda soluções definidas. Sobre estas dúvidas apareceram algumas perguntas, entre as quais:

Quais os tipos de serviço existentes?

Qual a semântica de cada um deles?

O que precisa ser especificado para o serviço ser prestado?

Qual a qualidade do serviço prestado?

O que deve ser esperado da informação de retorno do prestador de serviço?

Qual a semântica dessa informação?

Quais prestadores de serviço podem me atender?

Qual o de melhor qualidade?

Qual o de melhor preço?

Qual o de melhor relação custo/benefício?

Como o serviço vai ser pago?

A metodologia mais empregada para responder a esses problemas é a denominada orientação a objeto. Nela o serviço é considerado encapsulado em um objeto que possui uma interface bem definida. Nessa interface estão descritas as requisições que podem ser feitas ao objeto. Todas as interações do objeto com o mundo exterior são feitas através das requisições descritas nessa interface. Como exemplo de aplicação de serviços encapsulados em objeto tem-se num sistema de processamento de texto a inclusão pelo usuário de planilhas de cálculo, gráficos, desenhos e até mesmo o projeto de um equipamento no seu documento. A inclusão de outros tipos de documentos fica bem mais simples quando eles estão encapsulados em objetos.

Duas organizações internacionais estão analisando os problemas acima descritos utilizando essa metodologia de orientação a objeto. Nessa análise estão sendo criados padrões que deverão ser seguidos por fornecedores de *software* que forem desenvolver aplicações que envolvam troca de serviços em ambientes distribuídos.

Esses problemas e alguns outros a eles relacionados estão sendo analisados pela ISO como problemas ODP (*Open Distributed Processing* ou Processamento Distribuído Aberto)[ODP94]. Nesta análise estão sendo gerados documentos sob o título RM-ODP (*Reference Model for Open Distributed Processing* ou Modelo de Referência para Processamento Distribuído Aberto), onde o problema do processamento distribuído aberto é analisado sob cinco pontos de vista: empreendimento, informação, computação, engenharia e tecnologia. O relacionamentos do sistema com a organização a qual ele serve é objeto de estudo do empreendimento. A informação do mundo real armazenada no sistema é o objeto da informação. O objeto da computação são os objetos do mundo real, seus relacionamentos e os processos nos quais eles tomam parte. Para a engenharia o objeto é a infra-estrutura sobre a qual ele será implementado, o que está descrito pela

computação empregando a tecnologia disponível. O ponto de vista de tecnologia descreve a tecnologia na qual o sistema está disponível. A Figura 1-1 relaciona como são vistas as várias funções desempenhadas e a fase do projeto do sistema em que são tratadas.

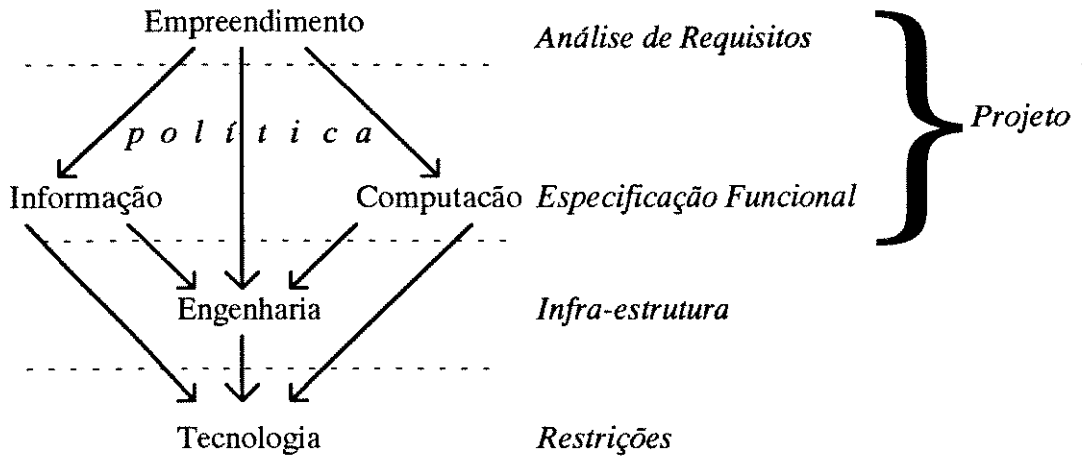


Figura 1-1. Pontos de Vista ODP

O OMG (*Object Management Group* ou Grupo de Gerenciamento de Objetos) é outra instituição que está analisando estes problemas. Seus documentos incluem um modelo de objeto[1] e um modelo de referência [OMA92], mostrado na Figura 1-2, que definem e relacionam entre si um ORB (*Object Request Broker* ou Negociador de Requisições de Objetos) [ORB91], um conjunto de serviços de objeto (*Object Services*) [OSA92] e um conjunto de facilidades comuns a vários tipos de aplicações (*Common Facilities*). O padrão CORBA (*Common Object Request Broker Architecture* ou Arquitetura para um Negociador de Requisições de Objetos) já foi aceito por grande parte dos grandes fornecedores de *software*.

[1]Modelo de objeto é uma descrição da semântica usada pelos objetos do sistema

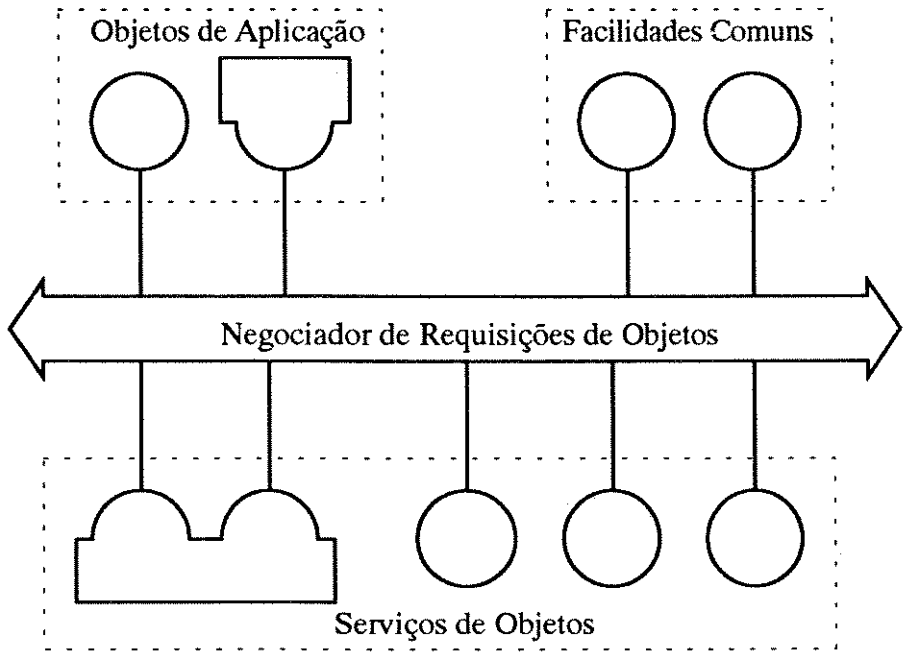


Figura 1-2. Modelo de Referência OMG

O presente trabalho pretende estudar os documentos gerados por essas duas organizações sobre sistemas distribuídos. No Capítulo 2 será feita uma descrição sucinta dos modelos de objeto das duas abordagens seguida de uma análise comparativa entre elas. O Capítulo 3 segue a estrutura do Capítulo 2 mas seu objeto de estudo é a infra-estrutura de comunicação dos objetos e os serviços de apoio ao sistema. No capítulo 4 é descrita uma implementação simples de infra-estrutura de comunicação entre objetos usando o sistema de RPC (*Remote Procedure Call* ou chamada de procedimento remoto) da SUN e a linguagem C++. O capítulo 5 contém as conclusões e trabalhos futuros.

CAPÍTULO 2

MODELOS DE OBJETO

O primeiro aspecto a ser analisado no processo de padronização de sistemas de objetos distribuídos[1] deve ser o protocolo através do qual o objeto se comunica, ou seja, suas características externamente visíveis. Essas características são denominadas pelo OMG de modelo de objeto. Os conceitos de modelamento do RM-ODP serão colocados aqui representando o seu modelo de objeto, pois descrevem essas características. O modelo de objeto descrito no OMG *guide* também será apresentado, além de uma comparação entre os dois modelos.

2.1 Modelo de Objeto para RM-ODP

A ISO parte de um sistema de processamento distribuído num ambiente aberto. Toda entidade do sistema que tem capacidade de comunicação é chamada de objeto e essa comunicação é efetuada através de suas interfaces. Cada um dos objetos tem um número de interfaces que podem ser de tipos diferentes. Cada interface pode ter papel de servidor, quando ela recebe invocações ou de cliente, quando ela faz requisições. Para haver comunicação entre duas interfaces uma deve ser cliente e a outra servidora. A ISO define uma série de conceitos relevantes para caracterização dos objetos.

2.1.1 Modelamento de um Objeto

Os objetos do modelo de referência são modelos das entidades do mundo real. Eles são caracterizados pelo seu comportamento e pelo seu estado. Seu comportamento define quais as sequências de ações possíveis e impõe restrições sobre o instante em que elas

[1]Denomina-se sistema de objetos distribuídos a um sistema de objetos que funcione em ambiente distribuído.

podem ocorrer. Seu estado, por sua vez, é a sua condição que em determinado instante de tempo e juntamente com o seu comportamento determina o conjunto de todas as sequências de ações nas quais ele pode tomar parte.

Os evento de interesse que ocorrem no sistema são chamados de ações. Toda ação está associada a pelo menos um objeto. Essas ações podem ser uma interação quando o objeto interage com seu ambiente durante a ação ou uma ação interna no caso de não haver interação.

Os objetos são encapsulados implicando em que qualquer mudança no estado de um objeto só pode acontecer através de uma ação interna ou de uma interação (quando há mais de um objeto envolvido). Todas as interações de um objeto com o seu ambiente são feitas em seus pontos de interação. O rastro de um objeto é o conjunto das interações de que ele participou desde o seu estado inicial até outro estado.

As possíveis interações de um objeto são particionadas em interfaces. Cada interface é, portanto, um subconjunto das possíveis interações daquele objeto e está identificada através de um identificador único denominado referência de interface. O fato de um objeto ter várias interfaces e delas poderem participar de interações simultaneamente pode causar indeterminismo para um observador que só tem acesso ao objeto através de uma de suas interfaces. A assinatura de uma interface é o conjunto de *templates*[2] das ações associadas com as suas interações.

Dois ou mais objetos podem ser combinados dando origem a um objeto composto. As características desse objeto composto são determinadas pelas características dos objetos componentes bem como pela forma de composição. O processo de composição também pode ser aplicado da mesma forma a comportamentos. Num objeto composto,

---

[2]*Template* é definido no RM-ODP como a especificação das características comuns de um grupo de ítens.

cada objeto componente pode ter um identificador de comportamento que é o seu papel. Uma decomposição consiste na representação de um objeto como uma composição de dois ou mais objetos. Na Figura 2-1 observa-se um exemplo de composição de objetos enquanto que na Figura 2-2 observa-se como uma decomposição pode facilitar a descrição do comportamento de um objeto mostrando a sua estrutura interna.

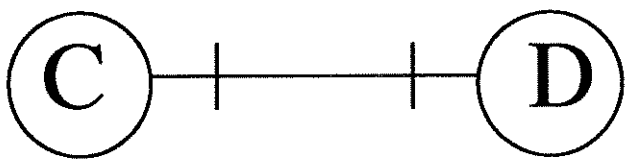
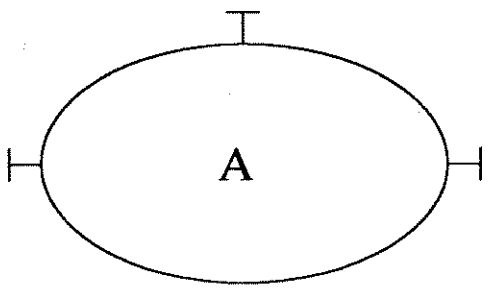


Figura 2-1. Composição de Objetos



Decomposto como:

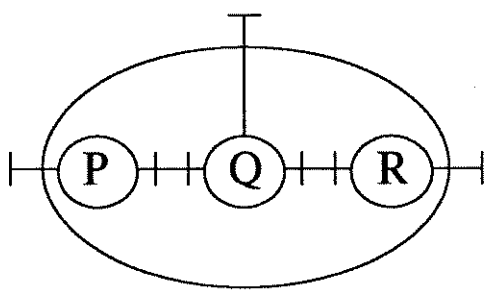


Figura 2-2. Decomposição de Objetos

A compatibilidade comportamental define até quando um objeto pode substituir outro em determinado ambiente sujeita a restrições. Caso todo o comportamento do objeto original seja exercitado nesse ambiente pelo objeto substituto essa compatibilidade comportamental é chamada de extensão. Se a compatibilidade comportamental for intencional, ela é chamada coerciva, caso contrário, ela é chamada de natural.

Refinamento é o processo pelo qual se aumenta o nível de detalhe de uma especificação. Uma especificação e o seu refinamento raramente conviverão na mesma descrição do sistema, devido a diferenças nos seus níveis de abstração.

Objetos, interfaces e ações satisfazem tipos. Tipo é um conceito bastante geral que está associado a um predicado. Basta um predicado ser satisfeito para se pertencer ao tipo a ele associado e como vários predicados podem ser satisfeitos simultaneamente, pode-se pertencer a vários tipos simultaneamente. X é do tipo ou satisfaz ao tipo se satisfaz ao seu predicado. A cada tipo corresponde uma classe, que é o conjunto de todos os Xs que pertencem ao tipo. Entre tipos existem as seguintes relações de sub-tipo e super-tipo. A é super-tipo de B se todo X que satisfaz B também satisfaz A. A é sub-tipo de B se todo X que satisfaz A também satisfaz B. As relações de sub-classe e super-classe relacionam as classes da mesma forma como as relações de sub-tipo e super-tipo relacionam os tipos. Sub-tipos e super-tipos definem uma hierarquia de tipos igual à hierarquia de classes definida pelas relações de sub-classe.

Objetos, interfaces e ações são instanciados a partir de *templates*. *Templates* especificam as características comuns de um conjunto de objetos e, além disso, podem ter parâmetros para a definição de características específicas. Existem um tipo e uma classe associados a cada *template*. Um objeto satisfaz a um tipo de *template* se ele é uma instância desse *template*. A classe do *template* é o conjunto de todas as suas instâncias. O processo de instanciação de objetos, interfaces e ações a partir de seus respectivos *templates* pode ser visto na Figura 2-3.

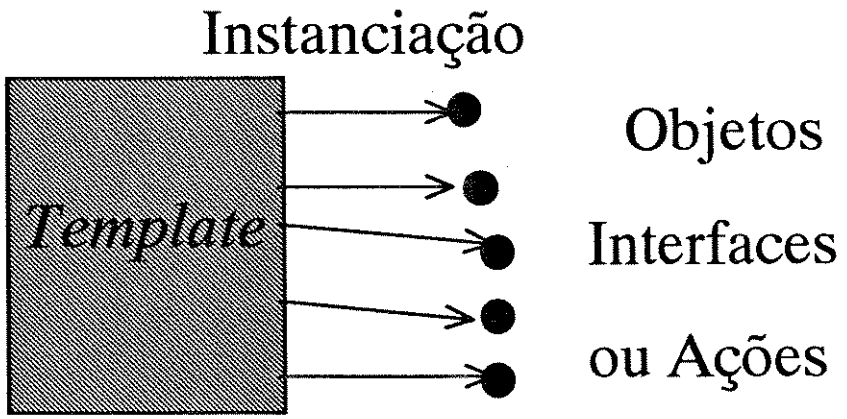


Figura 2-3. A Instanciação de um *Template* pode produzir um Objeto, uma Interface ou a ocorrência de uma Ação

Objetos e interfaces podem ser criados e eliminados enquanto ações ocorrem. Objetos também podem ser introduzidos no sistema, quando seu mecanismo de criação não fizer parte do modelo.

Como pode ser observado na Figura 2-4, os termos instanciação e instância têm significados diferentes. Instanciação é a denominação dada ao processo de criação de um objeto a partir de um *template* bem como a esse objeto em relação ao *template* a partir do qual ele foi criado. Instância de um *template* é um conceito mais geral porque associa um objeto a todos os *templates* do mesmo tipo do *template* que o instanciou.

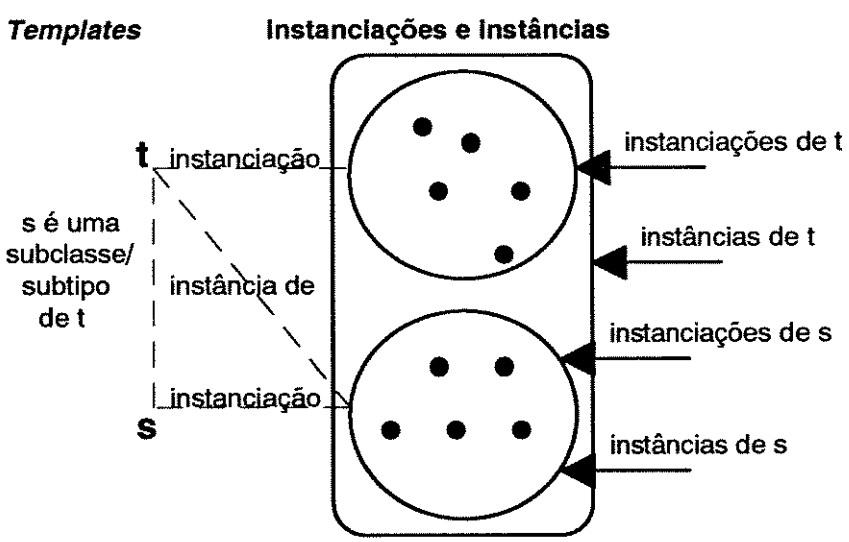


Figura 2-4. *Templates*, suas Instanciações e Instâncias

Classes derivadas são aquelas cujo *template* é resultado de uma modificação incremental de outro *template* de uma classe dita classe base. Pode ser permitido que uma classe derivada tenha mais de uma classe base, nesse caso diz-se que há herança múltipla. Essas relações de classe base/classe derivada definem uma hierarquia de classes que pode e na maioria dos casos será distinta da hierarquia de sub-classes/super-classes. A Figura 2-5 mostra um exemplo de hierarquia de classes onde hierarquia de herança é coincidente com a hierarquia de sub-classe/ super-classe.

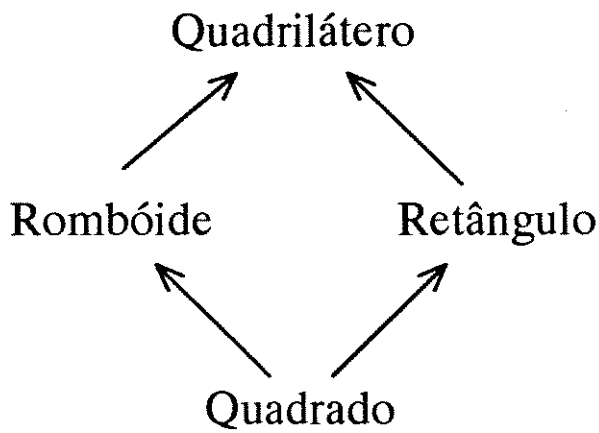


Figura 2-5. Uma Hierarquia de Classes para Formas Geométricas

2.1.2 Características dos Objetos

Como já foi citado, o modelo de referência utiliza-se da orientação a objeto pelas vantagens que ela oferece em aplicações de sistemas distribuídos. Essas vantagens que são características da orientação a objeto, são uma consequência do modelo de objeto acima apresentado e serão aqui enumeradas:

• abstração: o objeto executa um determinado serviço que independe da tecnologia que o está implementando. O serviço é representado pelo comportamento do objeto;

• Interação: toda interação com o objeto é feita através de operações. Cada objeto aceita um número de interfaces que, por sua vez, aceitam várias operações. As operações aceitas por um determinado objeto podem ser invocadas por outros objetos;

• Encapsulamento: essa propriedade garante que toda interação do objeto com o mundo exterior ocorre através de interfaces. O estado do objeto só pode ser acessado ou modificado fazendo-se invocações às suas interfaces.

• Distribuição: objetos podem ser relocados. Não existem diferenças para seus clientes se as interações são locais ou remotas;

• Falha: uma falha ocorre num objeto como um todo ou não ocorre, inexistindo objeto parcialmente falho. Essa propriedade permite a degradação suave de sistemas.

• Replicação: para melhorar a disponibilidade e a tolerância a falhas podem ser feitas cópias de um certo objeto que prestam o mesmo serviço que o original.

• Segurança: o encapsulamento permite que se determinem regras para o acesso aos objetos.

• Hierarquia de Tipo: as relações de super-tipo e sub-tipo definem uma hierarquia que relaciona todos os tipos do sistema. Se houver um tipo no sistema que é super-tipo de todos os outros tipos, ele será a raiz dessa hierarquia. Quanto mais geral o tipo, mais alta será sua posição hierárquica, quanto mais específico mais baixa será sua posição hierárquica.

• Hierarquia de Herança: as relações de classe base e classe derivada dão origem a uma hierarquia mostrando as relações de herança entre as classes do sistema. Essa hierarquia é distinta da hierarquia de tipos. Essa distinção ocorre porque uma classe derivada não é necessariamente um sub-tipo da classe base.

• Restrições de Ambiente: essas restrições são impostas aos objetos na execução de operações. Podem haver restrições de tempo, resolução, carga, confiabilidade, etc...

2.1.3 Conceitos de Comportamento

O modelo de referência define alguns termos usados na descrição de comportamentos de objetos e nas forma de interação destes.

Sequências de ações onde a ocorrência da ação predecessora causou a ocorrência de sua sucessora são chamadas de cadeias de ações. Uma *thread* é uma cadeia de ações onde pelo menos um objeto participou de todas as ações. Existem ações que fundem cadeias bem como ações que dividem cadeias.

Contratos podem ser estabelecidos entre objetos onde um objeto prestador de serviço se compromete a manter uma certa qualidade de serviço (tempo, preço, etc...) para o seu objeto cliente durante um certo período de tempo.

Na Figura 2-6 observam-se os conceitos relativos a contratos entre objetos. Contratos são estabelecidos entre objetos durante o seu tempo de vida. Durante o estabelecimento de um contrato o objeto apresenta o que se chama de comportamento estabelecedor. Durante o período de validade de um contrato, os objetos que dele participam têm o seu comportamento restringido pelo contrato. Esse comportamento restrito é chamado de comportamento habilitado. O conhecimento que os objetos têm da validade de um determinado contrato é denominado contexto contratual e o período pelo qual esse contexto contratual existe é denominado período de *liaison*. O fim do período de validade de contrato acontece com um comportamento finalizador.

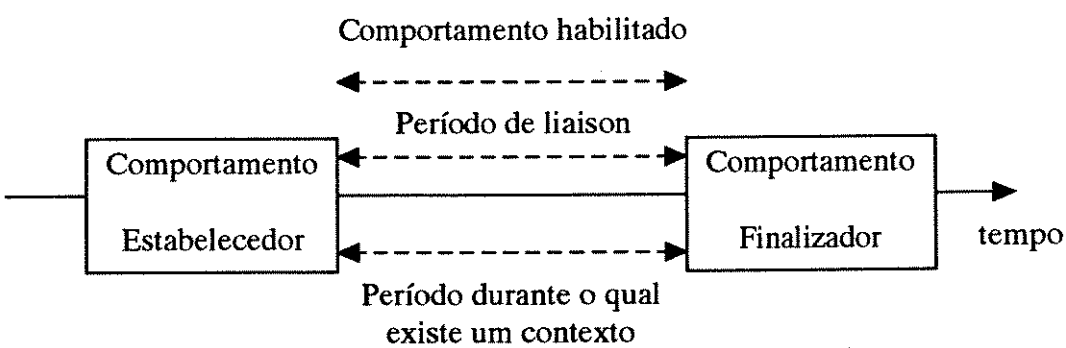


Figura 2-6. Contratos, *Liaison* e Comportamentos

Existem várias formas de comportamentos estabelecedores. Comportamento de *binding* é um comportamento estabelecedor feito entre duas ou mais interfaces. O contexto contratual de um contrato atingido através de um comportamento de *binding* é denominado *binding* propriamente dito. Um comportamento de *binding* pode depender de condições para a sua execução, as quais são denominadas pré-condições de *binding*. O comportamento finalizador de um *binding* é denominado *unbinding*. *Trading* é a denominação que se dá à troca de informação sobre contratos novos e potenciais. O *trading* é realizado através de um *trader* para o qual pode-se importar ou exportar ofertas de potenciais contratos.

Violações de contratos são denominadas falhas, estas são classificadas em modos de falha. A parte do estado de um objeto que pode levar a uma falha é denominada erro, este pode desaparecer sem ter causado falhas ou mesmo sem ter sido detectado. A falta, por sua vez, é a denominação dada para uma situação que pode levar a erros, faltas podem estar latentes ou ativas. Quando estão ativas causam erros e podem ser detectadas pela detecção desses erros. Se não há possibilidade de um objeto apresentar determinado modo de falhas o modelo diz que esse objeto apresenta estabilidade em relação a esse modo de falha.

Os objetos são classificados pelo modelo de referência em:

- objeto iniciador, quando são os causadores de uma comunicação;
- objeto retrucador, um objeto que toma parte de uma comunicação da qual ele não foi o objeto iniciador;
- objeto produtor, um objeto que é fonte de informação numa determinada comunicação;
- objeto consumidor, um objeto que é o sorvedouro de informação numa determinada comunicação;

• objeto cliente, um objeto que requisita serviços de um outro objeto;

• objeto servidor, um objeto que presta serviços a outros objetos

2.1.4 Criação e *Binding* de Interfaces

O processo de criação e *binding* de interfaces pelos objetos, com o objetivo de estabelecimento de comunicação entre eles, pode ser visualizado em passos nas Figuras de 2-7 até 2-14.

Na Figura 2-7, observa-se que apenas um objeto I existe no sistema no estado inicial. Esse objeto será um importador de serviços, ou seja, um cliente.

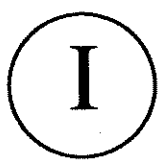


Figura 2-7. No Estado Inicial existe um Objeto I que será o Importador

O objeto E que está instanciado na Figura 2-8 será o exportador de serviços ou servidor na interação.

Figura 2-8. É Instanciado um Objeto E que será o Exportador

Na Figura 2-9, o objeto E (exportador) cria uma interface X do tipo T1 através da qual seus serviços podem ser acessados, ou seja, X é uma interface do tipo servidor. Essa interface poderia ter sido criada como parte do processo de instanciação do objeto.

Figura 2-9. O Objeto E cria uma Interface do tipo P1. Essa Interface tem o papel de servidor.

Na Figura 2-10, a existência da interface X do tipo T1 é anunciada ao resto do sistema. A sua existência pode então ser conhecida por outros objetos do sistema que poderão vir a ser clientes dos serviços por ela oferecido.

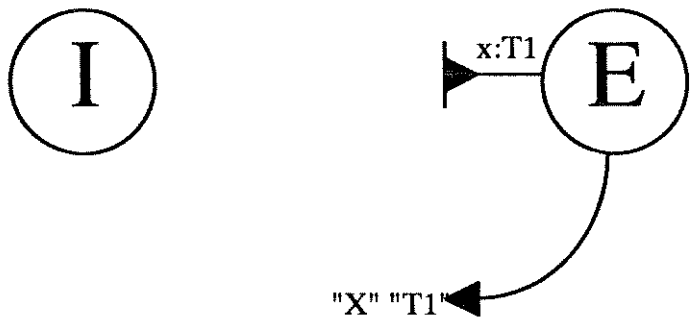


Figura 2-10. O Objeto E anuncia a existência da Interface X

Na Figura 2-11 observa-se a criação da interface Y do tipo T2 pelo objeto I como uma interface do tipo cliente. Essa interface também poderia ter sido criada como parte do processo de instanciação do objeto I. Tanto T1 (tipo da interface X de E) quanto T2 (tipo da interface Y de I) são sub-tipos de T.

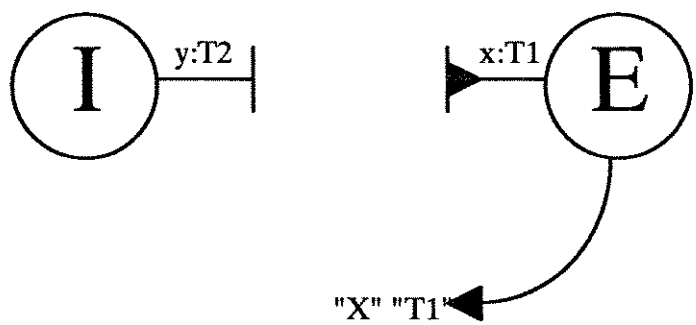


Figura 2-11. Criação da interface Y do tipo T2 no Objeto I como Interface Cliente

Na Figura 2-12 observa-se o objeto I procurando uma interface de servidor do tipo T. I está procurando algum objeto que seja servidor de um serviço oferecido através de uma interface do tipo T.

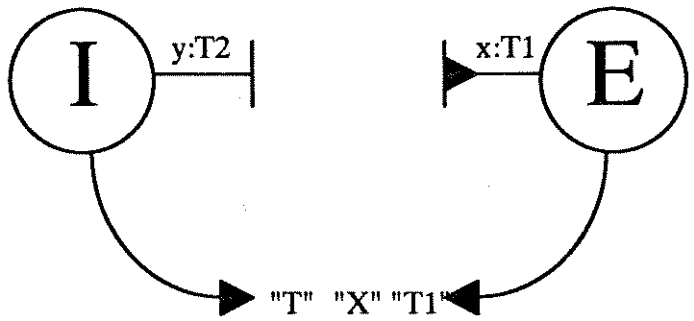


Figura 2-12. O Objeto I decide falar com uma Interface do tipo T

Na Figura 2-13 o objeto I descobre que existe uma interface de servidor X do tipo T pertencente ao objeto E. Essa "descoberta" pode ser feita através de um terceiro objeto, que seria um *trader*, ou seja, um negociador de serviços.

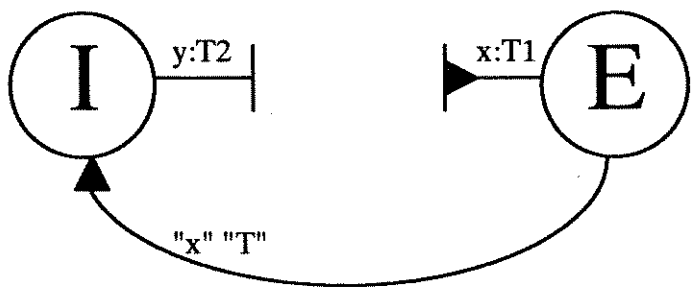


Figura 2-13. O Objeto I descobre por algum meio que "X" é uma Referência de Interface para uma Interface do tipo T

Observa-se na Figura 2-14 a realização de um *binding* entre a interface Y do objeto I e a interface X do objeto E. O objeto I pode ter realizado uma invocação no objeto E e este está se preparando para mandar uma terminação para o objeto I. Para o *binding* estar estabelecido os dois devem possuir o identificador de *binding*.

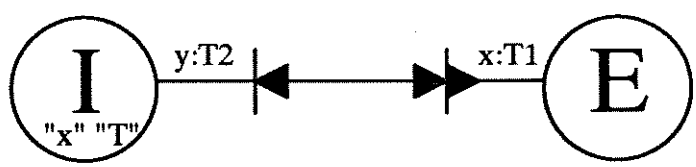


Figura 2-14. É realizado o *Binding* entre as Interfaces: I e E passam a interagir.

2.2 O modelo de objeto da OMG

O modelo de objeto da OMG [OMA92] define uma semântica básica para objetos que representa um protocolo para comunicação com ou entre essas entidades. Como será

visto ele é compacto definindo apenas um conjunto mínimo de características para aplicações orientadas a objeto.

O OMG define um *Core Object Model* (modelo de objeto nuclear) onde todas as capacidades básicas para orientação a objetos estão presentes. Esse modelo deve atender aos requisitos mínimos de todas as áreas e pode ser estendido através de *components* (componentes). Cada área terá um conjunto de *components* desejáveis, que será chamado de *profile*. As áreas que estão hoje sendo consideradas como candidatas para terem *profiles* específicos são:

- negociadores de requisições de objetos (*object request broker*),
- bancos de dados orientados a objetos,
- requisitos e análise,
- interface com o usuário.

O objetivo deste modelo de objetos é alcançar portabilidade, tanto entre tecnologias diferentes, como entre mesmas tecnologias provenientes de vários fornecedores. Essa portabilidade será alcançada se todos os produtos ou tecnologias usarem o mesmo modelo de objetos e será apenas no projeto, não no código fonte nem no executável, onde se depende de uma padronização num nível mais baixo que o modelo de objetos (Essa padronização do código fonte, é provida por CORBA [ORB91] para a linguagem C). A padronização no modelo de objeto define a semântica com a qual os objetos devem se comunicar.

Os *profiles* são alvo dos fornecedores de tecnologia, pois é de interesse de qualquer fornecedor que seus produtos atendam a um ou outro *profile*. Os produtos poderão atender aos *profiles* de duas formas diferentes: atendimento parcial ou atendimento total. Um produto atende a um *profile* de forma parcial se ele não tem nenhuma característica incompatível com o *profile* mas ele não implementa toda a funcionalidade proposta por

ele. Quando o atendimento é total, além de não ter nenhuma incompatibilidade toda a funcionalidade proposta pelo *profile* deve estar disponível. É importante salientar que um *profile* é um conjunto de *components* e que nenhum *component* pode criar incompatibilidades com o *Core Object Model.*

2.2.1 O *Core Object Model*

Este modelo de objeto define uma semântica de objetos comum a todas as áreas. Os conceitos usados nesse modelo são objetos, tipos, operações e sub-tipos. Um objeto é encarado de forma equivalente a orientação a objeto clássica ou seja um modelo de entidade que tem estado e comportamento. Operações são requisições ou mensagens que podem ser enviadas para os objetos; uma operação pode ou não alterar o estado do objeto a que foi aplicada. Os objetos são criados como instâncias de tipos. O tipo de um objeto define o conjunto das operações que pode ser a ele requisitado. Os tipos podem ter uma hierarquia através de relações de sub-tipo e super-tipo.

Cada objeto tem uma identidade única que é imutável durante a vida do objeto . As características e o estado do objeto podem variar com o tempo mas a sua identidade não. A especificação dos campos e qual o tamanho da identidade do objeto está além do modelo de objeto, podendo ser dependente de implementação.

Cada objeto aceita um conjunto de operações que caracteriza o seu comportamento. Esse conjunto de operações é descrito pelo tipo do objeto. Objetos são instâncias de um tipo imutável durante sua vida. Como no RM-ODP, cada operação tem uma assinatura e o conjunto das assinaturas das operações de um certo tipo de objeto constitui-se na sua interface. Os tipos formam uma hierarquia através das relações de super-tipo e sub-tipo que é um grafo acíclico, essa hierarquia tem uma raiz que é o tipo *Object.* Todos os tipos de objeto novos devem ser derivados desse tipo. Essa raiz única permite que um argumento de uma operação seja um objeto de qualquer tipo declarando-o como sendo do tipo *Object.* Um tipo é distinto de sua interface na medida em que ela pode variar com o

tempo sem mudar a identidade do tipo. Esse modelo de objeto não provê objetos que representem tipos e operações (conhecidos como meta-objetos) mas isso poderá ser provido através de um *component*.

Objetos não são os únicos possuidores de tipos nesse modelo. Entidades muito simples às quais não se deseja ter o *overhead* de serem tratadas como objetos são chamadas de não objetos. Tipos de não objetos não são sub-tipos do tipo *Object* e regras de herança e sub-tipo não se aplicam a eles. O conjunto de tipos de não objetos não é definido nesse modelo, mas pode ser visto como exemplo o conjunto total de tipos, que inclui o tipo referência de objeto do padrão CORBA no próximo capítulo, Seção 3.2.1.1. As *profiles* definem o que será objeto e o que será não objeto. Uma *profile* que se aplique a *smalltalk*, por exemplo, teria um conjunto de tipos de não objetos nulo porque nessa linguagem todas as entidades são objetos. Não existe um super-tipo geral que agrupe tanto objetos quanto não objetos.

Toda a dinâmica do sistema de objetos ocorre através de operações. A invocação de uma operação, também chamada de requisição, é um evento que é enviado ao objeto com possíveis parâmetros colocados por um requisitante (cliente). As consequências de uma operação podem ser:

um conjunto de resultados retornados imediatamente,

efeitos colaterais na forma da mudança de estado do objeto,

exceções (atualmente elas não fazem parte do *Core Object Model*, podendo ser introduzidas como um *component*), indicando a ocorrência de algum evento inesperado.

Cada operação tem uma assinatura, que inclui o nome da operação, a lista de parâmetros e a lista dos valores de retorno, se houver. Não é especificado pelo modelo uma linguagem na qual as assinaturas das operações serão especificadas. Cada operação é definida pelo seu nome, pelo conjunto ordenado dos tipos de seus parâmetros e pelo

conjunto ordenado do tipo de seus valores de retorno. Um e apenas um dos parâmetros da operação é chamado de parâmetro de controle e se refere ao objeto ao qual a operação será aplicada (essa restrição de um e apenas um parâmetro de controle vem da orientação a objeto clássica). O *Core Object Model* não especifica a ordem de execução de operações (sequencialmente ou concorrentemente) ou a atomicidade das mesmas nem requer uma descrição formal da sua semântica, mas não impede que uma implementação seja feita com essas características.

A definição de sub-tipo para o modelo se baseia no fato que para A ser um sub-tipo de B, todas as operações presentes em B devem estar presentes em A com o mesmo nome, o mesmo número e mesmo tipo de parâmetros e o mesmo número e mesmo tipo de valores de retorno. A única exceção feita é ao parâmetro de controle na medida em que ele sempre será do mesmo tipo da interface.

Herança é tratada pelo modelo no que diz respeito à interface dos tipos. O modelo define que quando um tipo S herda um tipo T todas as operações definidas para T estão também definidas para S. O modelo não especifica se a semântica de operações herdadas deve ou não ser mantida. O modelo relaciona sub-tipos e herança da seguinte forma, se S é um sub-tipo declarado de T então S herda as operações de T. O modelo aceita herança múltipla, permitindo que um tipo tenha vários super-tipos. O problema de resolução de conflitos em nomes de operações e redefinições de assinaturas de operações em sub-tipos é deixado para um *component.*

A requisição de uma operação implica nela ser despachada. O critério para o despacho da mensagem é baseado em seu parâmetro de controle (ele representa o objeto ao qual a operação será aplicada), podendo ser feito em tempo de compilação ou execução.

Os valores dos parâmetros passados para uma operação devem ser todos de um super-tipo do tipo especificado para o parâmetro correspondente na assinatura da

operação. É importante notar que caso os tipos dos valores de retorno sejam tipos de objetos serão retornados sempre identificadores de objetos e nunca uma cópia dos mesmos. A semântica de passagem de parâmetros é a de passagem por valor, excetuando-se os casos de objetos, nesse caso o identificador do objeto é retornado.

Nesse modelo um tipo exporta todas as operações definidas para ele, não podendo haver múltiplas interfaces com um conjunto de operações menor que todas as operações definidas para o tipo. Atributos de visibilidade das operações tais como *private, public, sub_type visible* e *friend* também não são aceitos. Essas características podem ser definidas por algum *component.* Também não é possível se restringir o acesso de um programa a apenas um subconjunto dos tipos definidos no sistema, esse tipo de característica também poderá ser definido em um *component.*

Como já foi visto, o modelo só considera relevante a semântica dos objetos, nada especificando sobre a implementação. Um tipo pode ou não ter várias implementações, como um grupo de tipos relacionados pode ou não compartilhar de uma mesma implementação. Para cada tipo podem estar associadas zero ou mais implementações, sendo que cada implementação define o conjunto de estruturas de dados que representa o estado do objeto bem como o código que será executado para cada operação. Os detalhes da implementação não são visíveis, apenas as operações que ela disponibiliza. O conjunto de operações disponibilizado deve ser o conjunto total de operações definido pelo tipo. Duas implementações diferentes do mesmo tipo podem usar estruturas de dados diferentes e código diferente para executar cada operação, mas com o mesmo resultado, ou seja, a execução daquele conjunto de operações. A combinação de uma especificação de um tipo com uma de suas implementações é denominada classe. Um objeto é uma instância de uma determinada classe em um certo momento mas nada impede que um mesmo objeto troque de implementação durante a sua vida.

2.3 Comparação entre os dois Modelos

Será apresentada agora uma análise comparativa entre os dois modelos de objeto. Já podem ser vistas algumas das diferenças dos dois modelos que serão analisadas a seguir.

2.3.1 Objetos Clientes e Clientes

No RM-ODP toda a comunicação ocorre entre objetos, e todas as entidades do sistema são colocadas como objetos. No OMG, embora o prestador de serviço deva ser um objeto, a entidade que está contratando um serviço não precisa ser um objeto, como fica claro no CORBA. Essa característica evidencia que o aspecto de maior relevância para o OMG é a padronização da interface do serviço.

2.3.2 Composição e Decomposição de Objetos

Composição e decomposição de objetos são recursos comummente usados em aplicações orientadas a objeto e analisados pelo RM-ODP não sendo mencionados nos documentos OMG. Como a semântica do OMG permite a implementação de uma composição de objetos de uma maneira bastante simples, bastando-se colocar um objeto composto como cliente do objeto componente, esse caso não mereceu sua atenção. É importante ressaltar que ao se ignorar a presença de composições de objetos no seu sistema torna-se mais complexo a implementação de tolerância a falhas, transações e outras facilidades normalmente usadas em sistemas distribuídos.

2.3.3 Comportamentos e Ações

RM-ODP trata extensivamente o comportamento dos objetos, conceituando ações, atividades, comportamentos e rastros de objetos. Também é conceituada a compatibilidade comportamental entre dois objetos. O OMG não define conceitos nem tem qualquer especificação a respeito de comportamento de objetos. Num ambiente OMG pode haver dois objetos com interfaces idênticas e comportamentos totalmente diferentes,

sem o conhecimento do sistema. Um cliente requisitando serviços a esses dois objetos não tem como distinguí-los.

2.3.4 *Templates*

Templates são tratados detalhadamente pelo RM-ODP nem sendo citados pelos documentos do OMG. A falta de *templates* (classes de objetos parametrizáveis) é considerada uma falha grave do modelo OMG pois classes parametrizáveis têm sido vastamente utilizadas provendo facilidades relevantes no desenvolvimento de aplicativos orientados a objeto.

2.3.5 Instanciação e Instâncias

RM-ODP faz uma distinção entre o conceito de instanciação e instância. Instanciação, além de denominar a ação de criação de um objeto, também denomina esse objeto em relação ao *template* a partir do qual ele foi criado. Se o objeto A foi criado a partir do *template* X ele é uma instanciação desse *template*. Instância é um conceito mais geral por estar associada a super-tipos. Um objeto A, instanciação de um *template* X, é *instância* de qualquer *template* super-tipo de X. O termo instância é usado pelo OMG com o mesmo significado do RM-ODP. A mudança de nomenclatura existente é que o objeto chamado de instanciação pelo RM-ODP é chamado de instância direta pelo OMG. Outro aspecto relevante é que todas essa relações se fazem com tipos para o OMG, visto que essa organização não definiu *templates*.

2.3.6 Criação de Objetos

A forma como os objetos são inseridos no sistema é analisada no RM-ODP. Um objeto é criado quando ele é produto da instanciação de um *template*. Um objeto pode também, ao invés de ser criado, ser introduzido no sistema. Esse outro mecanismo é usado quando o processo de criação do objeto não é conhecido pelo sistema. Os documentos OMG não fazem uma análise do processo de criação de objetos. Uma conclusão que pode

ser tirada é que todos os objetos OMG são, em terminologia RM-ODP, introduzidos no sistema.

2.3.7 O Conceito de Tipo

O conceito de tipo para o RM-ODP é bastante geral, na medida em que tipo é definido em função de um predicado, que pode referenciar qualquer propriedade da entidade. Tipo de objeto para a OMG representa o conjunto de operações que o objeto aceita, e somente ele. Essa diferença na conceituação de tipo implica em um conceito diferente de super-tipo e sub-tipo que por sua vez alterará o grafo que representa a hierarquia de tipos para os dois modelos. O exemplo seguinte mostrará isso de forma mais explícita.

Supondo que temos três tipos de objetos: navio, avião e automóvel. Todos os três tipos têm uma mesma operação `velocidade` que nos retorna o seu valor. A operação `velocidade`, entretanto, retorna um número de valores diferentes para os três objetos diferentes. Para o automóvel o valor de retorno único é a sua velocidade escalar. Para o navio, são retornadas duas componentes da velocidade, uma na direção norte-sul e a outra na direção leste-oeste. No avião, além das duas componentes retornadas para o navio é também retornada uma terceira componente vertical deste, totalizando três valores de retorno. Segundo o conceito de tipo proposto pelo RM-ODP, todos esse tipos seriam sub-tipos de um tipo associado ao predicado "tem a operação `velocidade`". Segundo o OMG, entretanto, não haveria nenhuma relação entre esse três tipos porque não há nenhuma operação em comum entre os três com a mesma assinatura pelo fato de elas diferirem no número de valores de retorno.

2.3.8 Referências de Objeto e Referências de Interface.

O RM-ODP não atribui um identificador a seus objetos, apenas às interfaces desses objetos. O OMG especifica em [OMA92] e em [ORB91] que seus objetos têm

identificadores (referências de objeto), nunca falando em identificadores de interface. Os documentos OMG citados não deixam claro se as operações providas por um objeto podem ser particionadas em múltiplas interfaces, conforme descrito no modelo ODP.

Essas afirmativas permitem concluir que o que é uma referência de interface para o RM-ODP corresponde a uma referência de objeto para o OMG. Para o OMG, o objeto só é relevante na medida em que ele possua uma interface. O OMG não vê diferenças entre dois objetos diferentes acessados por interfaces diferentes ou um único objeto com duas interfaces diferentes, o que significa para ele dois objetos diferentes, mas essa funcionalidade pode ser dada através de artifícios de implementação. Como o OMG só considera a interface do objeto, esses artifícios de implementação usados para transpor as suas limitações lhe serão transparentes, não sendo sentidos por ele. Um caso de um objeto com mais de uma interface é citado no RM-ODP como um objeto com uma interface de gerenciamento e uma interface operacional sendo que as duas devem ter permissões de acesso diferentes em relação a clientes diferentes.

Uma forma simples de se implementar o objeto com duas interfaces citado acima usando o modelo OMG é declarar que uma implementação aceita as duas interfaces como se pertencessem a dois objetos diferentes. Na realidade isso significa que temos operações de objetos diferentes afetando o estado do mesmo objeto. Como artifício de implementação isso cobre a lacuna de não se admitir um objeto com mais de uma interface e reforça a conclusão de que para cada referência de interface no modelo RM-ODP haverá uma referência de objeto no modelo OMG.

CAPÍTULO 3

INFRA-ESTRUTURA E SERVIÇOS AUXILIARES

Nesse capítulo serão analisadas as infra-estruturas de apoio propostas pelas duas abordagens e os conjuntos de serviços auxiliares por elas definidos. Conforme a estrutura do capítulo anterior será feita uma descrição da abordagem ODP, uma descrição da abordagem OMG e uma comparação entre as duas.

3.1 ODP

Nos documentos da ISO [ODP94] é descrita uma infra-estrutura de suporte a processamento distribuído orientado a objeto bem como um conjunto de funções auxiliares a essa infra-estrutura.

3.1.1 A infra-estrutura definida no ponto de vista de engenharia do RM-ODP.

Conforme a estrutura de pontos de vista definida por ODP e descrita no Capítulo 1, analisaremos a infra-estrutura de suporte definida no ponto de vista de engenharia. Esse ponto de vista é um dos que está em contato mais íntimo com o *hardware*. Essa infra-estrutura funciona como uma interface entre os três pontos de vista mais abstratos (empreendimento, informação e computação) com o *hardware* e o *software* básico do sistema. Os componentes da infra-estrutura são hierarquizados em nós, cápsulas, *clusters* e objetos básicos de engenharia, conceitos que serão analisados a seguir.

Nessa infra-estrutura o equipamento de processamento de dados é denominado nó. O nó é a unidade de provimento de recursos de processamento, armazenamento e comunicação, além de representar uma localização no espaço, ou seja, um endereço de rede. Ele é composto por seu núcleo e um conjunto de cápsulas. O núcleo incorpora as funções de gerenciamento do nó. A estrutura de um nó está representada na Figura 3-1.

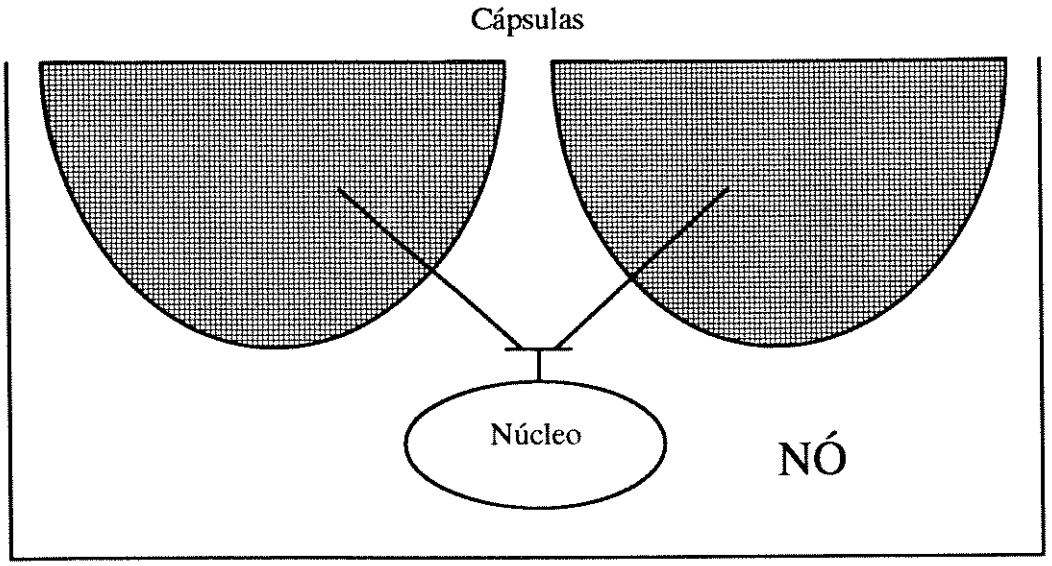


Figura 3-1. Exemplo da Estrutura de um Nó

Um processo ou espaço de endereçamento é denominado pelo modelo de cápsula que é a unidade de processamento e armazenamento. A cápsula consiste em seu gerente, seus *clusters* com os respectivos gerentes e seus canais, uma estrutura de comunicação que será analisada mais tarde. O gerente da cápsula tem um *binding* com cada gerente dos seus *clusters* para encapsular suas políticas de gerenciamento. A estrutura de uma cápsula típica está representada na Figura 3-2.

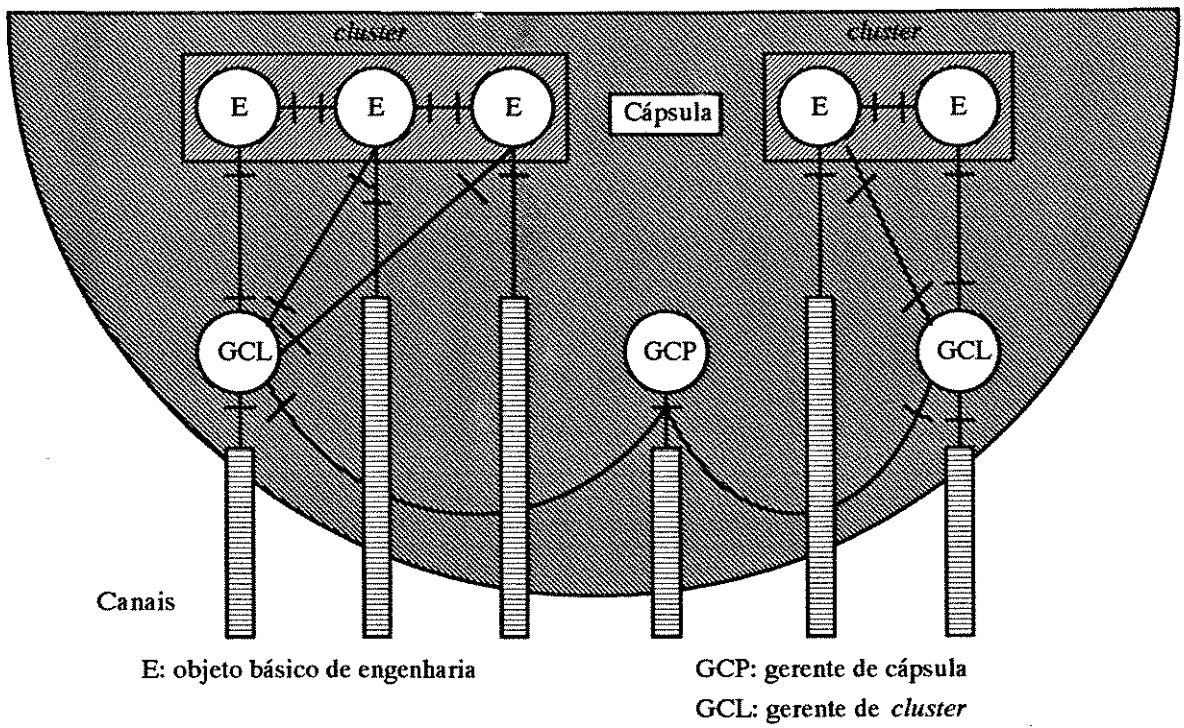


Figura 3-2. Exemplo da Estrutura de uma Cápsula

O *cluster* corresponde à noção de um segmento de memória virtual que agrupa objetos. O propósito desse agrupamento é definir uma unidade que pode ser desativada, reativada, recuperada ou migrada e para a qual possam ser criados *checkpoints*. Cada objeto de engenharia pode ter uma interface de gerenciamento que está ligada ao gerente do *cluster*, este encapsula a política de gerenciamento dos objetos do *cluster* e tem um *binding* com o gerente da cápsula. O *cluster* é responsável pela sua segurança e pelo gerenciamento de referências de interface, podendo fazer uso de funções de apoio para realizar essas tarefas. Todos os objetos de engenharia no *cluster* tem um *binding* com o núcleo e com o gerente de *cluster*. Todas as outras interfaces do objeto têm *binding* ou com outro objeto de engenharia no mesmo *cluster* ou com um canal.

Na Figura 3-3 está esquematicamente representada a infra-estrutura de suporte a objetos de engenharia no *cluster*. O gerente do *cluster*, por sua vez, tem um *binding* com o gerente de cápsula. Um aspecto relevante a ser destacado é que dois objetos dentro de

um *cluster* podem ter um *binding* direto enquanto que para haver *binding* com objetos fora do *cluster* é necessário um canal.

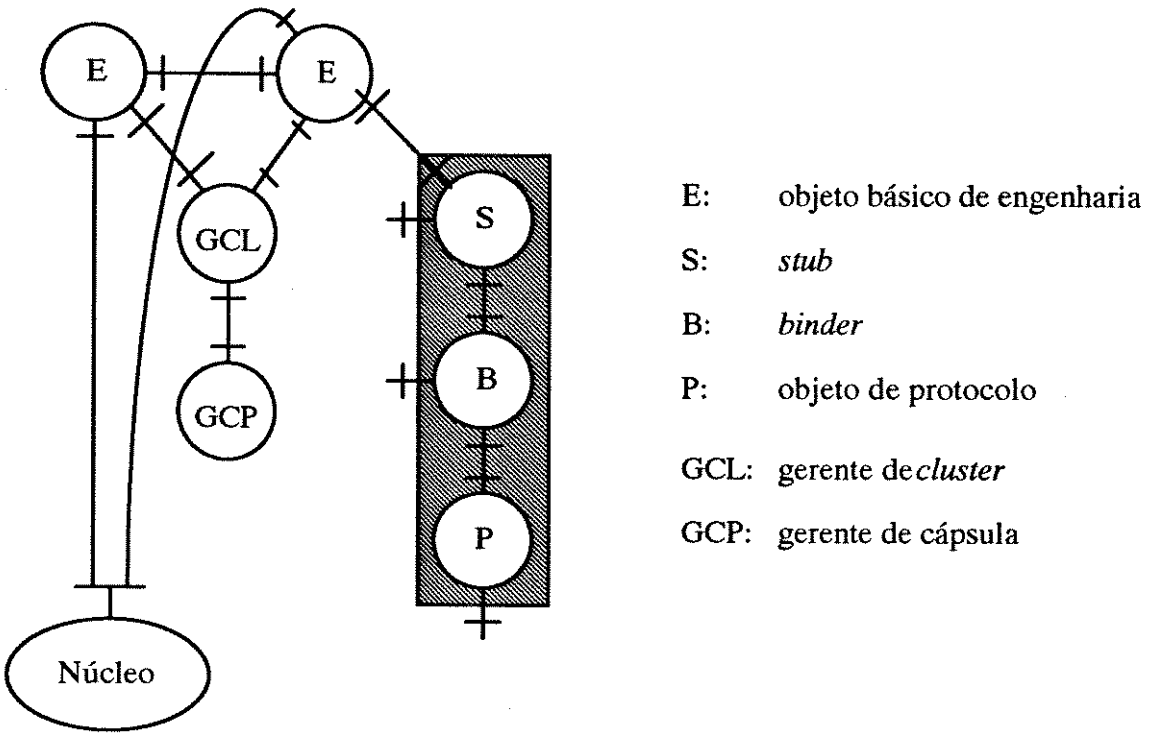


Figura 3-3. Exemplo da Estrutura que apoia um Objeto Básico de Engenharia

Nesse ponto de vista a entidade mais simples é o objeto básico de engenharia. Os objetos básicos de engenharia estão contidos em *clusters*. Cada objeto tem suas interfaces através das quais suas interações ocorrem. Para se ter acesso à interface de um objeto deve se ter uma referência de interface de engenharia que é na realidade um identificador para a interface.

Para a comunicação entre objetos localizados em clusters diferentes é necessário um canal, ele é composto de *stubs*, *binders*, objetos de protocolo e interceptadores. *Stubs* fazem a conversão dos dados utilizados na comunicação. *Binders* garantem a comunicação entre as extremidades do canal e a qualidade do serviço. Objetos de protocolo fazem a comunicação propriamente dita. No trajeto normal os dados saem de um objeto de engenharia, passando por um *stub*, por um *binder*, por um objeto de

protocolo, por um interceptador se necessário e por um objeto de protocolo, por um *binder* e por um *stub* novamente, chegando finalmente ao outro objeto de engenharia.

A Figura 3-4 representa esquematicamente um canal onde apenas dois objetos estão se comunicando. O canal representado é um canal para comunicação entre dois domínios que não têm um protocolo comum, o interceptador fazendo a tradução de protocolo. Caso haja um protocolo de comunicação em comum entre os dois domínios o interceptador não é necessário.

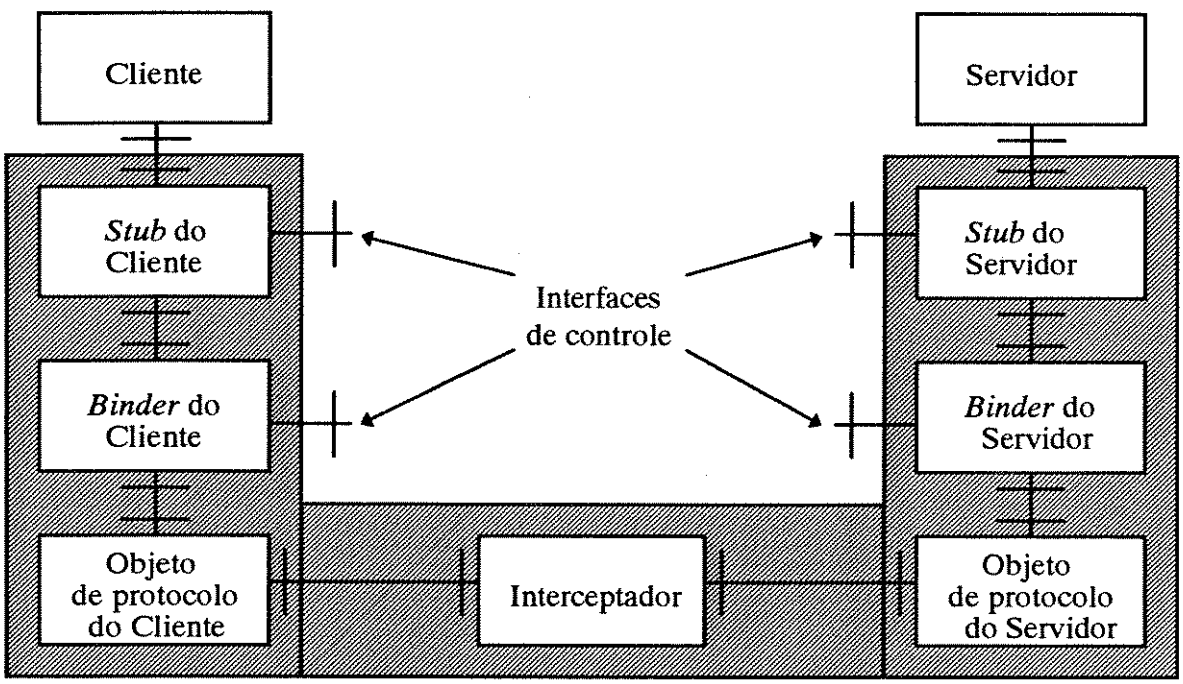


Figura 3-4. Um Exemplo de Canal Cliente Servidor Ponto a Ponto

A Figura 3-5 mostra um canal multiponto e multiprotocolo. Está representado um objeto de um domínio[1] que aceita os protocolos P1 e P2 se comunicando com três outros objetos em domínios diferentes. Os outro objetos estão em domínios que aceitam tipos de protocolos diferentes. A comunicação com os domínios que aceitam os protocolos P1 e P2 não apresenta problema, visto que o domínio do primeiro objeto aceita esses

[1]domínio é um conjunto de nós que aceita o mesmo protocolo de comunicação

protocolos. A comunicação com o domínio do objeto que aceita apenas o protocolo P3, entretanto, pedirá um interceptador para ser estabelecida, pois os outros domínios não aceitam esse protocolo.

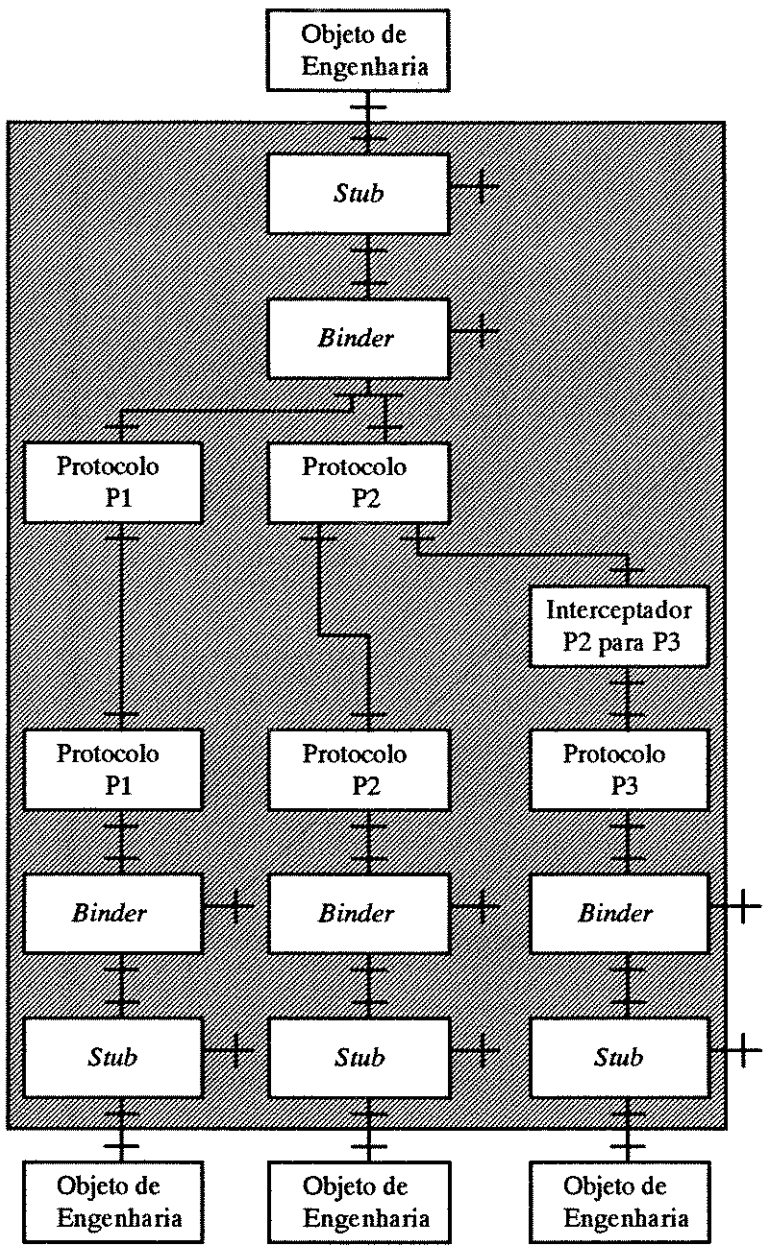


Figura 3-5. Um Exemplo de Canal Multiponto

A Figura 3-6 representa de forma simples como a engenharia dá suporte à computação. Os objetos de transparência são objetos que envolvem os objetos básicos de engenharia para acrescentar funcionalidades na forma de transparências de distribuição,

como descrito na Seção 3.1.3. O objeto básico de engenharia pode ser a implementação ou o código objeto do objeto computacional. O objeto computacional deverá, na maioria dos casos, ser mapeado para uma configuração de objetos de engenharia. Essa configuração fará as funções alocadas a esse objeto computacional nesse ponto de vista.

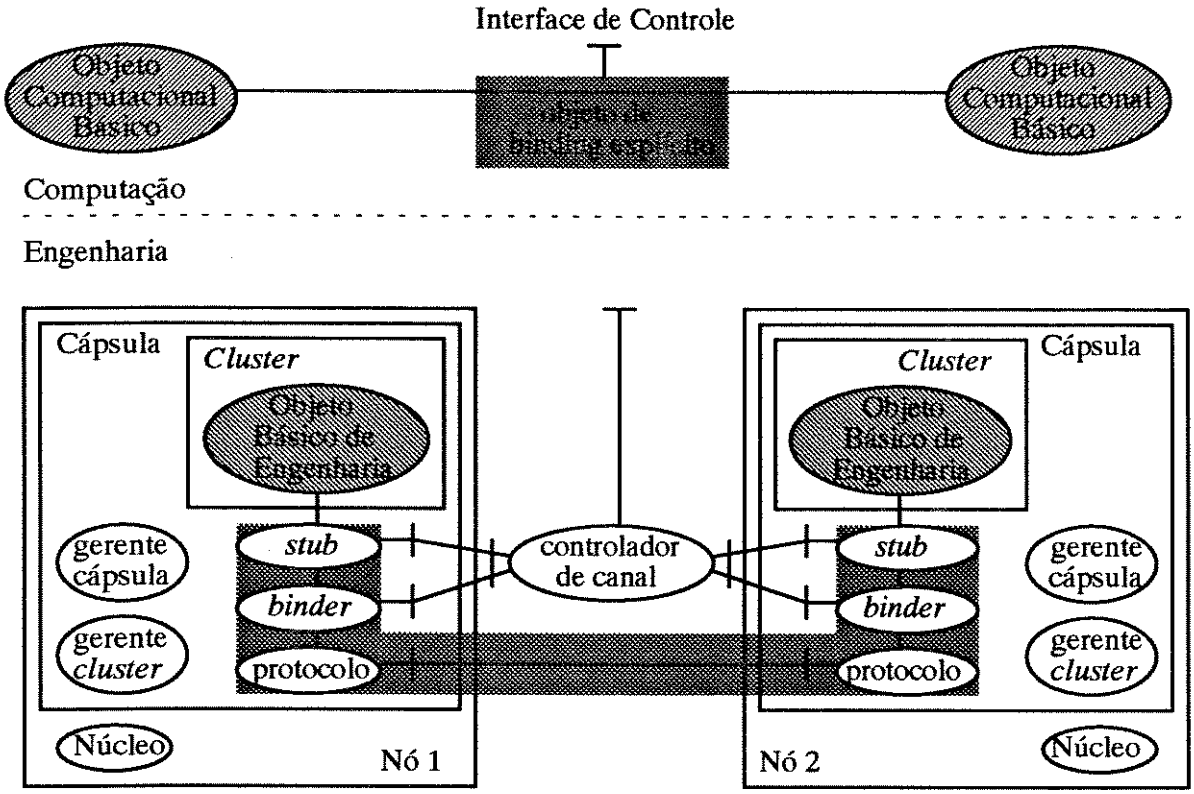


Figura 3-6. Tradução da Computação para a Engenharia

3.1.2 Funções definidas por ODP

Para tornar a infra-estrutura descrita na seção anterior operacional são necessárias funções ou serviços auxiliares. Esses serviços são descritos no modelo de referência divididos em quatro grupos: gerenciamento, coordenação, repositório e segurança.

As funções de gerenciamento, indicadas na Tabela 3-1, existem para cada um dos elementos básicos do ponto de vista de engenharia (nó ou núcleo, cápsula, *cluster* e objeto básico de engenharia) além de uma função para o gerenciamento das referências de interface do sistema.

FUNÇÃO DE GERENCIAMENTO DE	DESCRIÇÃO
Núcleo	Responsável pelo controle do processamento, armazenamento e comunicação em um nó.
Objetos	Necessária para a eliminação e *checkpoint* de objetos básicos de engenharia de *clusters* que serão desativados, farão *checkpoint* ou migrarão.
Cluster	Responsável pela realização das seguintes operações no *cluster*: modificação da sua política de gerenciamento, desativação do *cluster*, *checkpoint* do *cluster*, migração do *cluster* e eliminação do *cluster*.
Cápsula	Responsável pela realização das seguintes operações na cápsula: instanciação de um *cluster*, desativação da cápsula, *checkpoint* da cápsula e eliminação da cápsula.
Referência de Interface	Mantém um registro dos portadores de referências de interface.

Tabela 3-1. Funções de Gerenciamento

As funções listadas na Tabela 3-2 têm como objetivo auxiliar a coordenação do sistema. Elas permitem que se faça o *checkpoint* e a sua correspondente recuperação, desativação ou reativação de *clusters* e notificação de eventos entre objetos. Existem também funções para coordenação de grupo, migração, replicação e transação.

FUNÇÕES DE COORDENAÇÃO	DESCRIÇÃO
Checkpoint e recuperação	Responsável pela criação de *checkpoints* de *clusters* e recuperação destes em caso de falhas.
Desativação e reativação	Responsável pela desativação e reativação de *clusters* para efeito de migração.
Notificação de eventos	Responsável pela notificação a objetos de eventos considerados por eles relevantes.
Grupo	Responsável pelo gerenciamento de *bindings* entre objetos pertencentes a um grupo.
Migração	Responsável pela migração de *clusters*.
Replicação	Replicação é um caso especial de grupo em que seus membros são compatíveis comportamentalmente.
Transação	Responsável pelo monitoramento das ações para evitar conflitos e garantir consistência.

Tabela 3-2. Funções de Coordenação

As funções listadas na Tabela 3-3 têm como objetivo auxiliar o armazenamento de dados do sistema. Existem funções para armazenamento de dados, relacionamentos, localizações de interfaces (relocação), tipos e serviços (*trading*).

FUNÇÕES DE REPOSITÓRIO	DESCRIÇÃO
Relacionamentos	Responsável pelo gerenciamento de repositórios de relacionamentos entre objetos e interfaces.
Relocação	Responsável pelo gerenciamento de repositórios de localizações de interfaces.
Armazenamento	Responsável pela criação de interfaces para armazenamento de dados.
Trading	Responsável por achar ofertas de serviços no sistema.
Tipos	Responsável pelo gerenciamento de especificações de tipos e os seus relacionamentos.

Tabela 3-3. Funções de Repositório

As funções descritas na Tabela 3-4 ajudam a garantir a segurança no sistema. Existem funções para controle de acesso, auditoria de segurança, autenticação, integridade, confidencialidade, impossibilidade de repúdio e gerenciamento de chaves.

FUNÇÕES DE SEGURANÇA	DESCRIÇÃO
Controle de acesso	Responsável pela intermediação de interações na presença de requisitos de segurança.
Auditoria de segurança	Responsável pelo monitoramento, coleta e análise de informações relevantes para a segurança do sistema.
Autenticação	Responsável pela autenticação da identidade de objetos do sistema.

Integridade	Responsável pela detecção e prevenção da alteração ou eliminação não autorizada de dados.
Confidência	Responsável pela prevenção da exposição não autorizada de informação.
Impossibilidade de Repúdio	Responsável por garantir a impossibilidade de um objeto negar ter participado de uma interação da qual ele participou.
Gerenciamento de chave	Responsável por criar facilidades para gerar, armazenar, distribuir, arquivar, eliminar, rescindir, registrar e desregistrar chaves criptográficas.

Tabela 3-4. Funções de Segurança

3.1.3 Transparências de Distribuição Propostas pelo RM-ODP

O objetivo da infra-estrutura e das funções descritas é prover transparências de distribuição. Essas transparências permitem que os pontos de vista mais abstratos do que o de engenharia tenham a visão que teriam para um sistema centralizado. Essas transparência podem ser vistas na Tabela 3-5.

Transparências de Distribuição	Descrição
Acesso	Essa transparência especifica que o acesso a recursos remotos deve ser feita da mesma forma que a recursos locais.
Falha	Essa transparência especifica que a tolerância a falhas deve ser mascaradas dos objetos a que ela se aplica.

Localização	A localização das interfaces deve ser mascarada de seus clientes. O cliente de um objeto não precisa nem deve conhecer a localização do objeto. A referência de interface não pode ter informação de localização acessível ao cliente.
Migração	Essa transparência mascara de um objeto a possibilidade dele ser migrado durante o seu tempo de vida.
Persistência	Essa transparência mascara do objeto a possibilidade dele ser desativado e reativado durante o seu tempo de vida para economizar recursos de processamento, armazenamento e comunicação.
Relocação	Essa transparência mascara a relocação de uma interface de outras interfaces a ela ligadas.
Replicação	Essa transparência mascara o uso de um grupo de objetos compatíveis comportamentalmente para atender a uma única interface.
Transação	Essa transparência mascara de uma configuração de objetos o fato das suas atividades estarem sendo coordenadas para manter consistência.

Tabela 3-5. Transparências de Distribuição

3.2 OMG

A infra-estrutura para comunicação entre objetos definida pelo OMG foi chamada arquitetura comum para um negociador de requisições de objetos (CORBA) [ORB91]. O OMG também definiu um conjunto de serviços de objetos básicos (*Object Services*). CORBA representa a infra-estrutura usada para se fazer a comunicação entre os objetos enquanto que os *Object Services* formam num conjunto de serviços auxiliares para apoiar essa comunicação.

3.2.1 A Infra-estrutura proposta por CORBA (*Common Objetc Request Broker Architecture*)

A especificação [ORB91] define uma arquitetura para uma infra-estrutura que permita a um cliente requisitar um serviço a um objeto que pode ser remoto. *Object Request Broker* é a terminologia para o sistema que toma as providências para que uma requisição endereçada chegue ao objeto destino. *Common* é a propriedade do sistema de poder ser utilizada em e entre quaisquer arquiteturas de *hardware* ou ambientes de *software*.

Essa especificação tem vários pontos em aberto, definindo apenas os componentes básicos do sistema, suas funções e dependências. Ela permite toda a liberdade ao projetista do sistema para colocar mecanismos que melhorem o desempenho e a funcionalidade do sistema. Por esses motivos essa arquitetura tem hoje o suporte da maioria dos fornecedores de *software* e *hardware*.

O fato da especificação ter pontos em aberto pode também trazer problemas de interoperabilidade. Não é definido um protocolo de comunicação para as invocações nem um padrão para transferência de dados. Em cada implementação, escolhas diferentes podem ser feitas o que pode causar problemas quando se tiver necessidade de estabelecer comunicação entre duas implementações diferentes.

Para se comunicar com um objeto é necessário conhecer a sua interface, o que ele pode fazer pelos seus clientes e como os serviços devem ser solicitados. A interface do objeto é na realidade como ele é visto pelos clientes. A especificação [ORB91] inclui uma linguagem para definição de interfaces chamada IDL (*Interface Definition Language*) semelhante ao C++. Essa linguagem tem recursos para a descrição de interfaces, as operações que elas podem executar e os tipos de dados usados nessas operações.

Essa especificação é uma solução minimalista pois só uma infra-estrutura mínima é mandatória para a comunicação entre objetos. Os pontos em aberto, como já foi visto, criam uma grande liberdade de implementação, dando margem a diferentes implementações com diferentes características de desempenho e funcionalidade fornecidas por empresas diferentes. A desvantagem dessa liberdade são os potenciais problemas de interoperabilidade entre sistemas de fornecedores diferentes.

3.2.1.1 Modelo de Objeto Utilizado pela Arquitetura

O modelo de objeto utilizado por CORBA é derivado do modelo de objeto definido pelo OMG. O modelo de objeto OMG é um modelo abstrato enquanto que o modelo de objeto CORBA é um modelo concreto, pronto para ser utilizado. Os objetos, nesse modelo, são identificados através de referências de objeto. Cada referência de objeto identifica univocamente um objeto. Deve haver uma referência de objeto, diferente de todas as outras, que significa a inexistência de objeto (correspondente a um valor *NULL* de ponteiro da lingugaem C). A criação e destruição de objetos são consequências de requisições. Os clientes não têm acesso a mecanismos específicos para executar essas funções.

Esses objetos são acessados através de interfaces bem definidas. Todas as interações entre os objetos e seus clientes se dão através dessas interfaces. As interfaces são uma forma de encapsular os objetos e dar mais liberdade à implementação.

As interfaces dos objetos são compostas de operações e tipos de dados utilizados nessas operações. Cada operação tem um número de parâmetros, que podem ser de entrada, saída, ou ambos. Outras características das operações são o contexto e as exceções que elas podem retornar. O contexto especifica valores que podem alterar o desempenho da operação e as exceções são possíveis condições anormais no retorno das operações que têm valores de retorno específicos.

Uma requisição é um evento composto de uma operação, um objeto alvo e zero ou mais parâmetros. As requisições são as interações existentes entre clientes e objetos.

Todos os parâmetros de uma requisição são valores. Cada valor tem um tipo dentre os vários definidos pela especificação. A especificação define tipos para referências de objetos, tipos compostos e tipos básicos. A árvore de tipos definidos pela especificação pode ser vista na Figura 3-7.

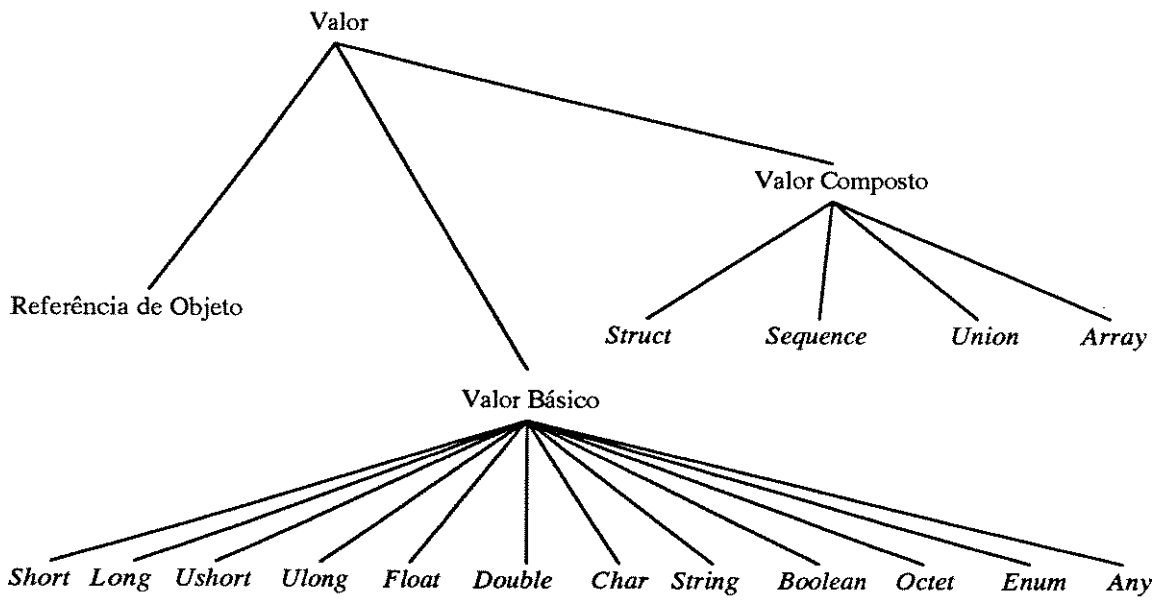


Figura 3-7. Valores Legais

3.2.1.2 Visão Global da Arquitetura

A arquitetura se baseia no fato de haver um cliente precisando de serviços que podem ser providos por um servidor, que na especificação é chamado de implementação de objeto. Esse problema básico de comunicação está esquematizado na Figura 3-8.

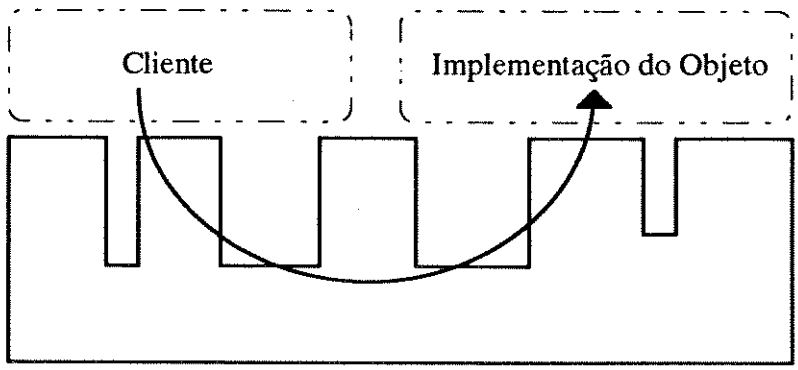


Figura 3-8. Uma Requisição enviada através do ORB

3.2.1.3 Os Componentes do Sistema

A proposta da arquitetura é regulamentar essa comunicação, definindo um conjunto mínimo de módulos para estabelecê-la e para a interação entre cliente e implementação de objeto. O objetivo almejado com essa regulamentação é um alto grau de interoperabilidade entre diferentes plataformas, tanto de *software* quanto de *hardware*, colocadas num mesmo sistema. No caso de implementações de objeto, a flexibilidade pode ser ainda maior, pois pode-se ter diferenças quanto às operações suportadas, desde que a interface requisitada pelo cliente seja um subconjunto da interface oferecida pela implementação do objeto.

Os componentes mínimos para um sistema que atenda a [ORB91] estão mostrados na Figura 3-9. A flexibilidade do lado do cliente é garantida através de uma interface de invocação dinâmica, que permite a clientes acessarem objetos cuja interface veio a ser conhecida em tempo de execução. No lado servidor, existe um adaptador de objetos que presta serviços às implementações de objetos, e que pode variar de acordo com o tipo de serviço requerido pela implementação.

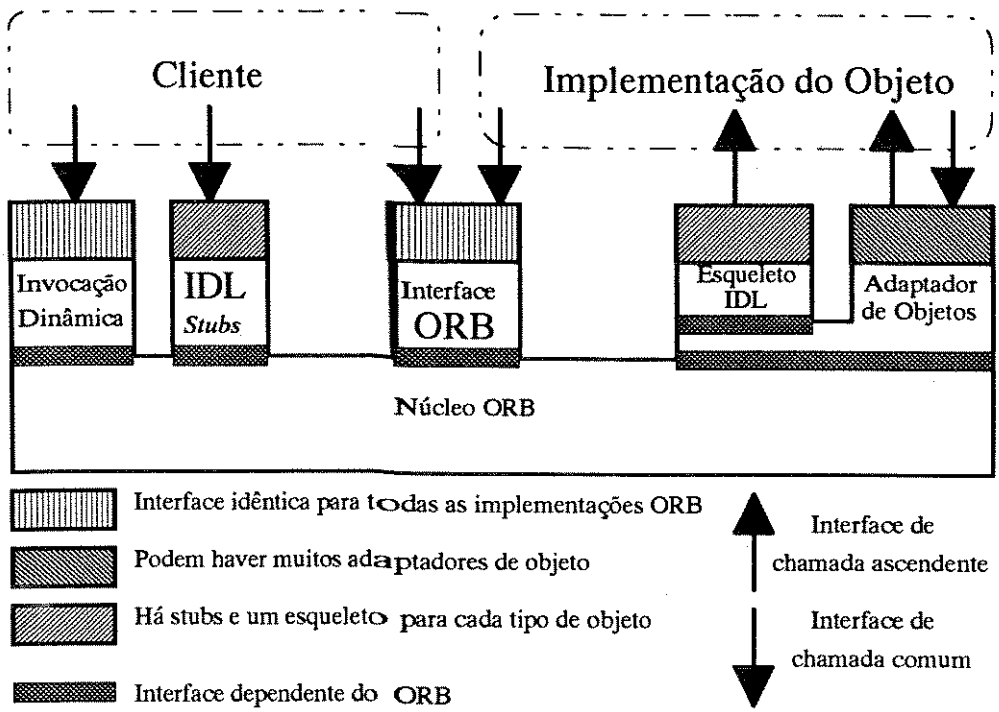


Figura 3-9. A estrutura da Interface do ORB

3.2.1.4 O Funcionamento no Cliente

No lado do cliente, como se observa na Figura 3-10, as requisições de serviços podem ser feitas através de *stubs* específicas de interfaces conhecidas ou através de uma interface de invocação dinâmica. Essas duas formas de uso são equivalentes e a implementação do objeto não tem como diferenciar como está sendo feita a invocação.

Cada interface tem um conjunto de *stubs* correspondente. Esse conjunto é gerado por um compilador IDL (um compilador cuja entrada é código IDL), e é específico de cada interface. O conjunto de *stubs* contém código para receber do cliente e empacotar os parâmetros da operação, transmitindo-os ao núcleo ORB e receber os resultados da operação, desempacotando-os e entregando-os ao cliente.

Caso o cliente não conheça a interface do objeto em tempo de compilação, ele pode usar a interface de invocação dinâmica, onde as invocações às operações são construídas em tempo de execução. Essa interface faz o processo de invocação da mesma forma que

as *stubs*. A forma deve ser idêntica para a implementação de objeto não ter como distinguir qual dos dois processos foi usado na invocação.

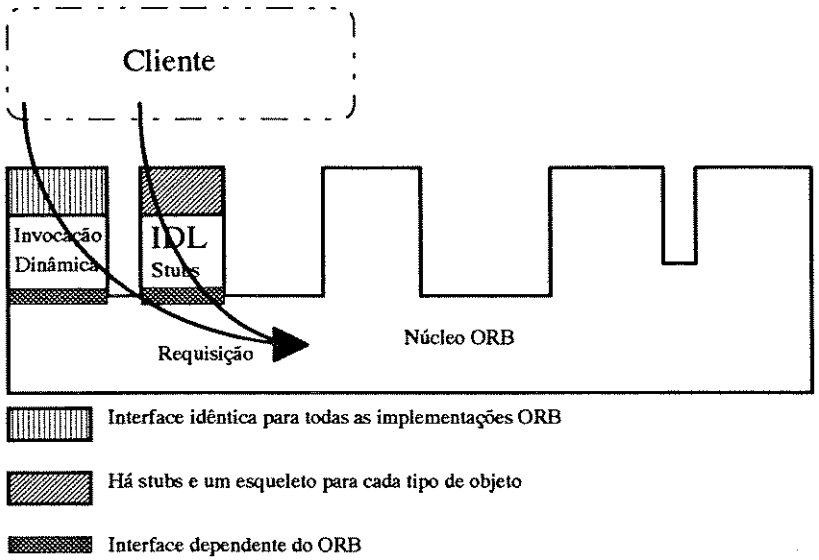


Figura 3-10. Um Cliente usando uma *Stub* ou a Interface de Invocação Dinâmica

3.2.1.5 O Funcionamento no lado da Implementação de Objeto

No lado da implementação de objeto, na Figura 3-11, a requisição chega através do adaptador de objetos e esqueletos que repassam a chamada à implementação de objeto. A implementação executa o serviço e repassa os resultados aos esqueletos que, utilizando o adaptador de objetos, respondem ao cliente.

O adaptador de objetos presta serviços de suporte à implementação de objeto. A qualidade de serviços prestados pelo adaptador depende do fim para o qual ele foi feito. Para aplicações simples ter-se-ía um adaptador de objetos simples com uma funcionalidade mínima, em aplicações mais complexas, como um banco de dados distribuídos orientado a objeto, ter-se-ía um adaptador complexo dando suporte coerente com esse tipo de aplicação.

Os esqueletos são o correspondente às *stubs* do lado da implementação de objeto. Eles cuidam de receber os parâmetros e passá-los à implementação de objetos, fazendo a

invocação e também a recepção dos resultados dessa implementação e seu envio ao cliente.

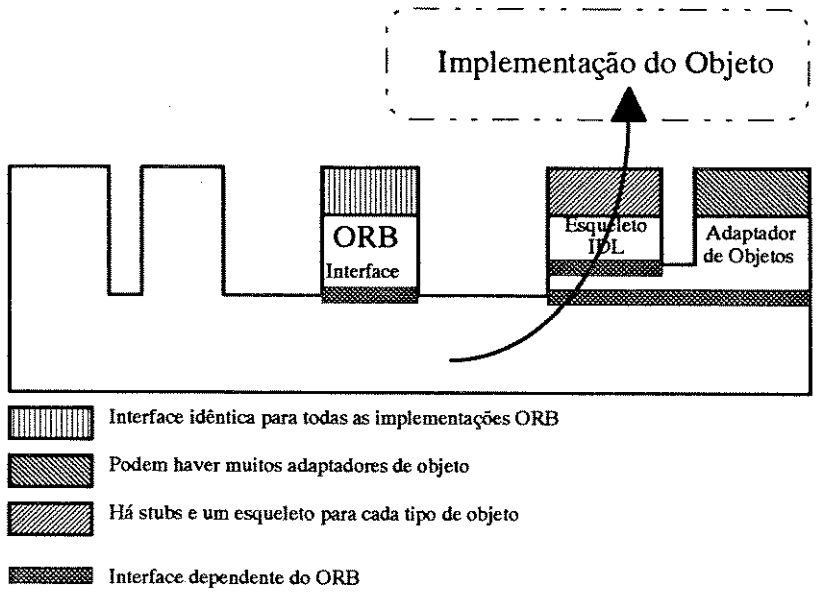


Figura 3-11. Uma Implementação de Objeto Recebendo uma Requisição

3.2.1.6 *Stubs*, Esqueletos e Repositórios

A Figura 3-12 indica a relação entre as definições IDL e as implementações de objeto com os repositórios, *stubs* e esqueletos. A partir das definições IDL geram-se os *stubs*, os esqueletos e as informações pertinentes às interfaces para serem adicionados ao repositório de interfaces. Quando é instalada a implementação é registrada no repositório de implementações.

O repositório de interfaces é uma base de dados que guarda todas as informações de interesse com respeito a interfaces. Essas informações incluem as operações aceitas, seus parâmetros, resultados e exceções. Além desses dados, a árvore de compatibilidade de interfaces é uma das estruturas de dados que faz parte desse repositório e através da qual é possível encontrar interfaces compatíveis com a que é procurada, no caso de não existir para ela uma implementação específica.

O repositório de implementações é outra base de dados relativa às implementações. Quando uma implementação é instalada, ela deve ser registrada nesse repositório informando seus atributos relevantes, que incluem quais as interfaces por ela aceitas, sua forma de invocação e ativação, etc. Essas informações serão usadas pelo adaptador de objetos quando este receber requisições endereçadas a essas implementações.

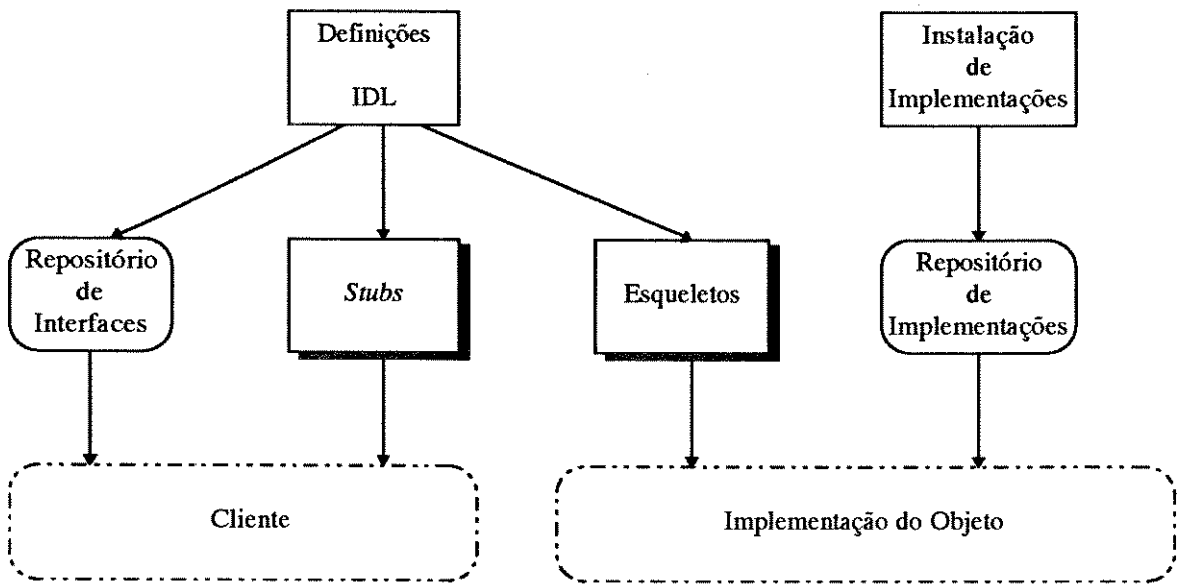


Figura 3-12. Repositórios de Interface e de Implementação

3.2.1.7 A Estrutura de um Cliente

A estrutura de um cliente, indicada na Figura 3-13, pode incluir um ou mais conjuntos de *stubs* e a interface de invocação dinâmica. Tanto os *stubs* quanto a interface de invocação dinâmica fazem uso de rotinas contidas em bibliotecas de uso comum para a execução do serviço.

As referências de objeto manipuladas pelo cliente estão em formato dependente da linguagem e precisam ser traduzidas para o formato ORB. Essa tradução pode ser feita através das rotinas de biblioteca.

Toda a arquitetura acima descrita possibilita uma grande funcionalidade. Do lado cliente, essa funcionalidade se concretiza nas transparências de distribuição, como

definidas em [ODP94] e já descritas na Seção 3.2.1.7. No lado do servidor, a construção de implementações de objeto fica facilitada pela existência do adaptador de objetos.

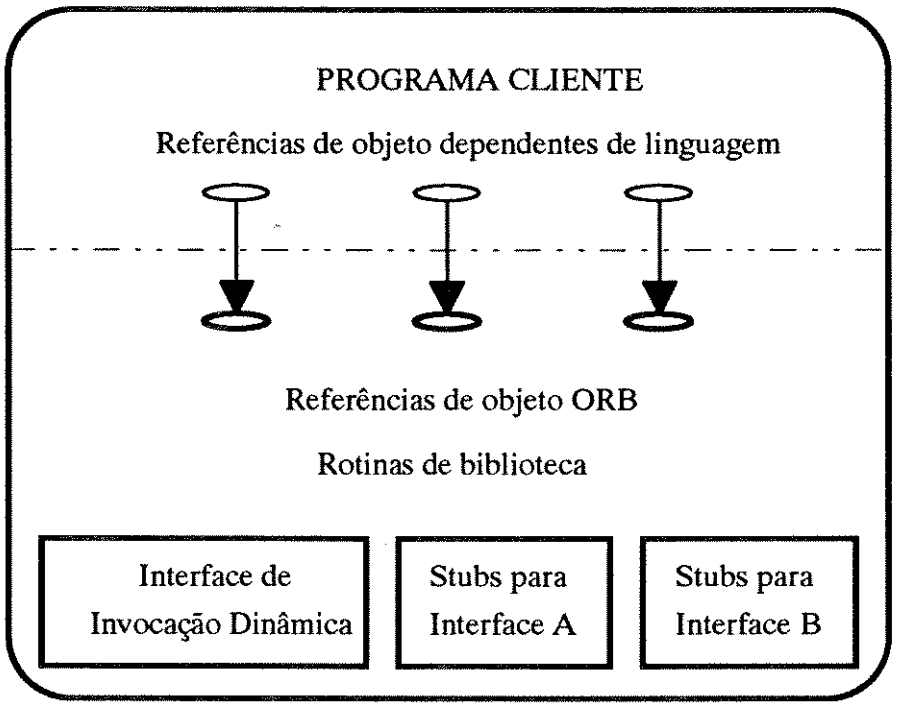


Figura 3-13. A Estrutura de um Cliente Típico

3.2.1.8 A Estrutura de uma Implementação de Objeto

Do lado da implementação do objeto, a dinâmica é diferente do lado do cliente. A implementação de objeto contém o código objeto dos métodos das classes por ela implementadas e armazena os dados relativos a cada objeto.

Esses métodos são executados de forma ascendente pelos esqueletos que foram gerados pelo compilador IDL. O código dos métodos utiliza rotinas de biblioteca e do adaptador de objetos para cumprir sua tarefa. As rotinas de biblioteca fazem tarefas de apoio usadas pelas implementações de objeto. As tarefas realizadas pelas rotinas do adaptador de objetos são relativas ao tipo de implementação a que ela foi direcionada. Essas tarefas podem ser o grau de complexidade adequado para satisfazer a qualquer aplicação.

Todo esse mecanismo é responsável pela execução no local onde se encontra a implementação do método requisitado pelo cliente. Na Figura 3-14 a dinâmica nesses módulos do sistema está esquematizada.

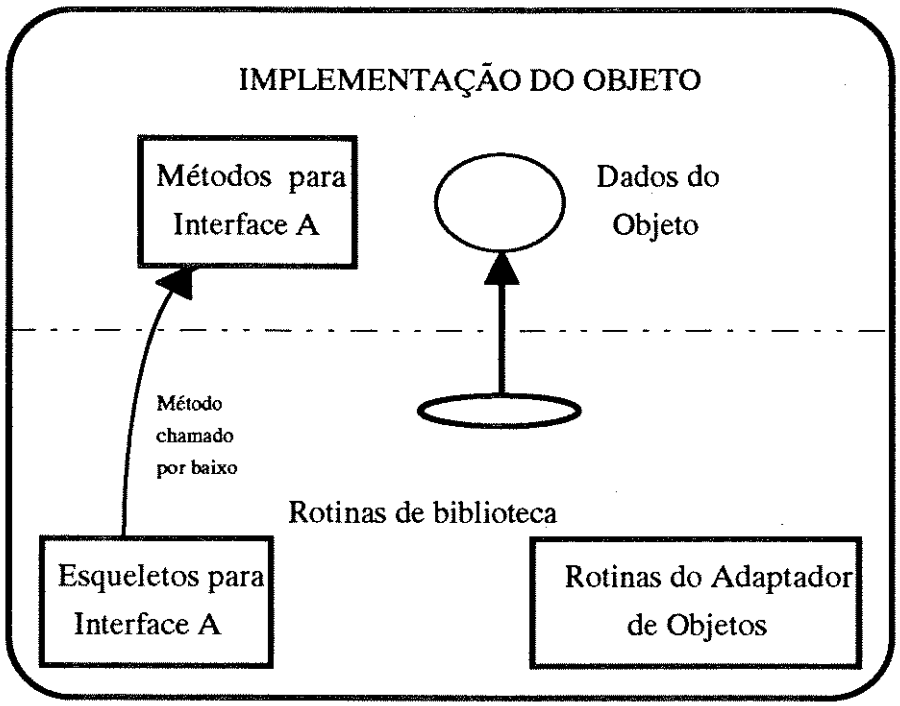


Figura 3-14. A Estrutura de uma Implementação de Objeto Típica

3.2.1.9 Uma Implementação que suporta várias Interfaces

Uma mesma implementação pode ter métodos para mais de uma interface. Nesse caso o adaptador de objetos seleciona que método de que interface será utilizado para executar o serviço.

No caso de haver várias interfaces, tem-se um conjunto de esqueletos e um conjunto de métodos para cada interface implementada. Esse esquema ajuda a manter essa separação entre interfaces diferentes na mesma implementação, tornando o sistema mais robusto. Os dois conjuntos de esqueletos são chamados pelo adaptador de objetos, que fornece serviços para a implementação da interface.

Um exemplo dos módulos que se teria numa implementação de objetos com duas interfaces pode ser observado na Figura 3-15.

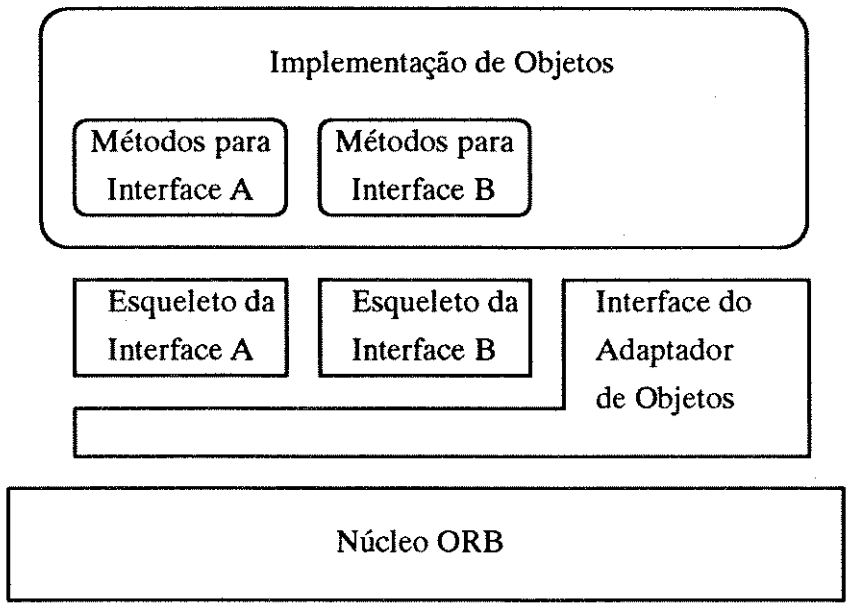


Figura 3-15. A Estrutura de um Adaptador de Objetos Típico

3.2.2 A Linguagem de Definição de Interface proposta por CORBA

A base para a geração de *stubs* e esqueletos é a linguagem de definição de interfaces (IDL). A interface do objeto, isto é, suas características externas são definidas nessa linguagem, que é originária do C++. Como todo o contato que o cliente tem com o objeto é através dessa interface, qualquer implementação que atenda a essa interface atenderá às invocações de qualquer cliente desse objeto.

Essa linguagem dispõe de recursos para se criar novos tipos de dados a partir dos tipos básicos e compostos nela definidos. Além de tipos, podem ser definidos constantes, módulos, interfaces, operações e exceções. Os módulos e as interfaces podem englobar as outras definições. Operações definem a forma como as requisições devem ser feitas, seus valores de entrada e saída bem como suas possíveis exceções. Os formatos das exceções de usuário também são definidas com seus valores de retorno.

Módulos são declarados através da especificação do seu nome e uma palavra chave. Sua utilidade é definir um escopo de identificadores IDL. Quaisquer declarações IDL são válidas em um módulo, inclusive a declaração de outro módulo.

Interfaces são declaradas da mesma forma que módulos, com a vantagem de se poder especificar interfaces base, que são aquelas das quais a interface original herdará definições. Dentro de uma interface podem ser declaradas constantes, tipos, exceções, atributos e operações.

A herança múltipla de interfaces é permitida, mas desde que não haja operações iguais com parâmetros diferentes no conjunto união das operações das interfaces herdadas. O grafo de herança é um grafo convencional de herança múltipla, ou seja, um grafo acíclico. Um exemplo de herança múltipla legal está representado na Figura 3-16.

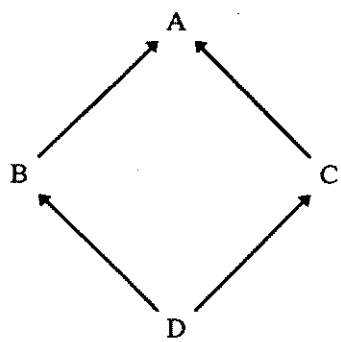


Figura 3-16. Exemplo de Herança Múltipla Legal

A declaração de constantes pode fazer uso de constantes já declaradas e fazer operações com valores literais especificados. As operações aritméticas e lógicas básicas são válidas para inteiros e as aritméticas para ponto flutuante.

Há formas de se declarar uma infinidade de tipos de dados. Podem ser feitas composições de tipos usando os tipos compostos. Os tipos básicos são: inteiro com e sem sinal, curto ou longo, ponto flutuante, simples ou duplo, caráter, booleano, octeto e qualquer (*any*). Octetos são usados para valores que não devem sofrer nenhum tipo de conversão. O tipo *any* pode significar virtualmente qualquer tipo que possa ser declarado em IDL. O tipo do valor de um dado do tipo *any* é codificado como parte do próprio dado.

Os tipos compostos são estruturas, uniões, enumerações, sequências, vetores e *strings*. Estrutura é uma *struct* comum em linguagem C. Uniões devem discriminar o tipo do dado armazenado para possibilitar a conversão dos dados. Uma enumeração em si não é um tipo composto, é apenas um conjunto de valores definido pelo usuário. Sequências podem ser de qualquer tipo e têm tamanho variável, enquanto vetores têm tamanho fixo. *Strings* são um tipo especial de sequência de caracteres.

Para a declaração de operações, devem ser especificados seus atributos, tipo do valor de retorno, identificador, parâmetros, possíveis exceções e contexto. O único atributo existente para operações sem valor de retorno ou parâmetros de saída é o ***oneway***. Quando esse atributo é especificado a semântica da operação se torna *best-effort*. Caso ele não seja especificado a semântica é *at-most-once* com exceções e *exactly-once* sem exceções. Os parâmetros devem ter seu tipo e nome especificados, bem como se são de entrada, saída ou ambos. As exceções de usuário que a operação pode gerar devem ser citadas, bem como os valores do contexto do cliente que a operação vai necessitar.

Uma interface também pode ter atributos. Atributos são valores da interface que podem sempre ser lidos e às vezes escritos pelo cliente. A declaração de um atributo em IDL especifica se o cliente tem direito a escrevê-lo ou apenas a lê-lo. A declaração de um atributo é equivalente à declaração de operações para lê-lo e escrevê-lo com nomes especiais.

O escopo dos identificadores é uma característica importante da IDL. A não ser pela diferença dos módulos que criam um novo escopo, o funcionamento é semelhante ao da linguagem C++.

As definições seguintes criam novos escopos: módulo, interface, estrutura, união, operação e exceção. Essas definições criam escopo para os seguintes tipos de identificadores: tipos, constantes, valores de enumeração, exceções, interfaces, atributos e operações.

A IDL, por estar baseada em C++, aceita o mesmo pre-processador que esta linguagem. As restrições de IDL em relação a C++ são as seguintes:

- o tipo de valor de retorno de uma função é mandatório;
- um nome deve ser especificado com cada parâmetro formal numa declaração de operação;
- uma lista de parâmetros consistindo do *token* único ***void*** não é considerada sinônimo para uma lista vazia;
- *tags* são necessárias para estruturas, uniões discriminadas e enumerações;
- tipos inteiros não podem ser definidos simplesmente como ***int*** ou ***unsigned***; eles devem ser declarados explicitamente como ***short*** ou ***long***;
- ***char*** não pode ser qualificado pelas palavras chaves ***signed*** ou ***unsigned***.

3.2.3 Os Serviços de Objeto propostos pelo OMG

A arquitetura CORBA já descrita fornece as funcionalidades básicas. Para aplicações reais são necessárias funcionalidades mais complexas. Para auxiliar no provimento tanto da funcionalidade básica do modelo CORBA como de outras funcionalidades o OMG definiu os serviços de objeto. Esses serviços provêm operações básicas relacionadas como o modelamento lógico ou o armazenamento físico de objetos. As operações são consideradas básicas na medida em que são independentes do domínio da aplicação e estão mais próximas do ORB do que do usuário final. Para cada serviço pode ser definido um conjunto de operações intrínsecas que todas as classes devem herdar e implementar. Esses serviços são construídos sobre o CORBA, tendo como base das suas especificações as definições de interface em IDL.

A arquitetura de serviços de objeto tem os seguintes requisitos:

- As interfaces dos serviços de objeto devem ser orientadas a objeto e devem ser descritas em IDL.

- Extensões propostas para a IDL, a CORBA ou o modelo de objeto da OMG devem ser identificadas.

- O sequenciamento de operações deve ser especificado quando for o caso.

- Especificações de serviços de objeto não devem conter descrições de implementação.

- Especificações devem ser completas.

Esses requisitos visam que a arquitetura cumpra os seguintes objetivos:

- Independência e modularidade.

- Minimização da duplicação de funcionalidade.

- Inexistência de interfaces escondidas entre serviços de objeto.

- Consistência entre serviços de objeto diferentes.

- Extensibilidade de serviços de objeto individuais.

- Extensibilidade da coleção de serviços de objeto.

- Configurabilidade.

- Descrições precisas dos serviços de objeto.

- Integridade, confiabilidade e proteção dos serviços de objeto.

- Desempenho.

- Possibilidade de crescimento.

- Portabilidade.

No documento [OSA92] da OMG está declarada a possibilidade, e mesmo a necessidade, de um objeto ter mais de uma interface. Os seguintes tipos de interface são colocados como relevantes para serviços de objeto:

- interface funcional: a interface pela qual um usuário de um serviço de objeto lhe faz invocações;

- interface gerencial: interface para se gerenciamento ou controle de um determinado serviço;

- interface construtiva: interface que contém operações para a comunicação entre o núcleo do serviço de objeto e outros objetos envolvidos na execução desse serviços. Essas operações são definidas pelo serviço e herdadas e implementadas pelos objetos participantes do serviço.

Os serviços de objeto propostos pela OMG estão listados na Tabela 3-6:

Serviço	Função
Arquivamento	Apoiar o mapeamento entre objetos ativos e *backups*.
Backup/Restore	Apoiar o *backup* e a recuperação de objetos
Gerenciamento de mudanças	Apoiar a identificação e evolução consistente de objetos incluindo gerenciamento de versão e configuração.
Controle de concorrência	Apoia o acesso concorrente a um ou mais objetos por um ou mais objetos.
Troca de dados	Apoia a troca de parte ou de todo o estado visível de um objeto entre dois objetos.
Notificação de eventos.	Apoia a notificação de eventos a objetos interessados
Externalização	Apoia a externalização e internalização de objetos.

Repositório de implementação	Apoia o gerenciamento de implementações de objeto.
Instalação e ativação	Provê mecanismos para distribuir, ativar, desativar e relocar "objetos gerenciados".
Repositório de interfaces	Apoia o gerenciamento de definições de interfaces de objetos.
Licenciamento	Apoia o gerenciamento de licenças.
Ciclo de vida	Apoia a criação, eliminação, cópia e equivalência de objetos.
Nomeamento	Apoia o mapeamento entre nomes e objetos
Controle Operacional	Apoia o controle do comportamento dinâmico de "objetos gerenciados".
Persistência	Apoia a persistência de um objeto independente do tempo de vida tanto da aplicação que acessa o objeto e da implementação que executa os métodos dos objetos.
Propriedades	Provê atributos dinâmicos nomeados associados com um objeto
Consulta	Apoia operações em estruturas com objetos de grão fino permitindo a indexação desses objetos.
Relacionamento	Apoia operações em associações entre dois ou mais objetos.
Replicação	Apoia a replicação explícita de objetos num ambiente distribuído e o gerenciamento de consistência das cópias.

Segurança	Apoia o controle de acesso nos objetos.
Inicialização	Apoia a inicialização e terminação de serviços de objetos.
Threads	Apoia o controle de múltiplas *threads* de execução.
Tempo	Apoia a sincronização de relógios num sistema distribuído.
Trading	Apoia o cliente na procura a um serviço que atenda a suas exigências.
Transações	Apoia a execução atômica de uma ou mais operações.

Tabela 3-6. Serviços de Objeto

Os serviços de objeto são interdependentes na medida em que para um deles funcionar pode ser necessário que outros já estejam funcionando. Como exemplo temos o serviço de arquivamento que depende do serviço de externalização para armazenar objetos.

3.3 Análise Comparativa

Serão aqui feitas algumas considerações que permitirão analisar e discutir as semelhanças e diferenças das abordagens propostas pela OMG e pela ISO nos aspectos referentes à estrutura e aos serviços de suporte do sistema.

3.3.1 Infra-estrutura para Comunicação entre Objetos

A infra-estrutura proposta pela OMG no documento [ORB91] (CORBA) é bem mais simples que a proposta contida em [ODP94] elaborada pela ISO. A ISO propõe uma infra-estrutura que atende à grande maioria dos problemas em ambientes distribuídos abertos, pensando em longo prazo. A OMG, pelo contrário, elaborou uma solução analisando produtos comerciais existentes e tecnologias existentes. Essa abordagem deve levar a uma solução que esteja operacional mas também obsoleta num prazo mais curto que a abordada pela ISO, porque não foram analisados a fundo todos os potenciais futuros

problemas em processamento distribuído orientado a objeto. Esse fato é consequência do padrão CORBA [ORB91] ser produto de um consórcio de fornecedoras de *hardware*, *software* e serviços atuantes no mercado, que necessitam de um padrão para seus interesses em sistemas distribuídos orientados a objetos.

A infra-estrutura proposta pela ISO cobre todo o *software* do sistema incluindo o que é denominado sistema operacional e o *software* básico. Essas funções são exercidas principalmente pelo gerente de núcleo, que é responsável pela alocação de recursos para as cápsulas. O OMG considera que toda essa infra-estrutura proposta pelo sistema operacional está disponível, consequentemente conceitos como nó, cápsula e *cluster* não se aplicam nos seus documentos.

A Tabela 3-7 faz uma análise comparativa entre os elementos usados nas duas abordagens para a comunicação entre objetos.

Tarefa	Elemento ODP que a executa	Elemento CORBA que a executa	Comentários
Colocação de parâmetros em pacotes e conversão de dados	*Stub*	No cliente, *stub* IDL e na implementação de objeto, Esqueleto IDL	A conversão de dados é necessária quando as convenções de armazenamento de dados são diferentes entre equipamentos.
Binding entre objeto cliente e objeto servidor	*Binder*	ORB *Core*, usando *Object Services*.	O *binding* consiste em garantir a comunicação fim a fim entre cliente e servidor.

Comunicação	Objeto de protocolo		O CORBA não coloca um elemento responsável pela comunicação, apenas assume que esse elemento existe.
Conversão de protocolo	Interceptador	*Gateway*?	A questão da interoperabilidade está sendo revista no padrão CORBA.

Tabela 3-7. Comparação de Infra-estrutura

Quando o OMG necessita de uma solução padrão para problemas tecnológicos e que deverá ser incorporada a seus padrões é feita uma RFP (*Request For Proposals* ou requisição para propostas). Esse documento é uma requisição pública da OMG à platéia interessada na elaboração de soluções.

Por meio da RFP 93-9-15 o OMG requisita soluções para interoperabilidade e inicialização. O aspecto importante nesse documento é que o OMG exige que as respostas sigam o modelo de referência ODP proposto pela ISO. Esse fato confirma que o modelo de referência ODP será obedecido na implementação de sistemas de objetos distribuídos, se dentro do escopo do sistema implementado.

3.3.2 Serviços Auxiliares

A maioria de funções definidas no RM-ODP encontra uma correspondência no conjunto de serviços de objeto definido pelo OMG. Essa correspondência será analisada a seguir tomando-se como base da comparação as funções ODP.

3.3.2.1 Funções de Gerenciamento

As funções de gerenciamento RM-ODP estão relacionadas aos conceitos OMG nos serviços de objeto. A Tabela 3-8 faz uma breve análise dessas funções sob a visão OMG.

• FUNÇÃO ODP	CORRESPONDENTE NO OMG
Núcleo	A maioria das atribuições dessa função não são cobertas pelo modelo OMG, sendo providas normalmente pelo sistema operacional. Excecões são feitas ao serviço de *threads* e hora que sendo atribuição do gerente de núcleo são serviços de objeto definidos pelo OMG. Outra exceção é feita na geração de referências de interfaces (referências de objeto no OMG) que é realizada pelo adaptador de objeto.
Objetos	Essa função encontra correspondência no serviço de ciclo de vida de objetos.
Cluster	Essa função não se aplica diretamente por não existirem conceitos no nível de abstração do *cluster* nos documentos OMG.
Cápsula	Essa função não se aplica diretamente porque não existem conceitos no nível de abstração da cápsula nos documentos OMG. Se a cápsula for considerada um processo, essa função será provida por uma biblioteca de gerenciamento do processo, comummente chamada de biblioteca de *runtime*.
Referência de Interface	Essa função não encontra correspondência direta em nenhum conceito OMG mas tem relação com o serviço de objeto de ciclo de vida porque ele deve garantir a integridade referencial no sistema.

Tabela 3-8. Comparação entre Funções de Gerenciamento ODP e conceitos OMG

3.3.2.2 Funções de Coordenação

As funções de coordenação RM-ODP têm correspondentes principalmente nos serviços de objeto definidos pela OMG. A Tabela 3-9 faz uma breve análise comparativa entre essas funções

FUNÇÕES ODP	CORRESPONDENTE NO OMG
Checkpoint e recuperação	Os serviços de objeto correspondentes a essa função são os de arquivamento, *backup/ restore* e persistência.
Desativação e reativação	Essa função não encontra correspondente nos documentos OMG porque essa organização não prevê suporte a migração.
Notificação de eventos	Essa função encontra um correspondente direto no serviço de objeto de mesmo nome.
Grupo	Essa função não encontra correspondência nos documento OMG porque essa organização não prevê suporte a grupo.
Migração	Essa função não encontra correspondente nos documentos OMG porque essa organização não prevê suporte a migração.
Replicação	Essa função encontra um correspondente direto no serviço de objeto de mesmo nome lembrando-se que não há suporte a grupo no OMG. Isso impede que a replicação seja implementada como um caso especial de grupo.
Transação	Essa função encontra um correspondente direto no serviço de objeto de mesmo nome.

Tabela 3-9. Comparação entre Funções de Coordenação ODP e Conceitos OMG.

3.3.2.3 Funções de Repositório

As funções de repositório RM-ODP têm correspondentes principalmente nos serviços de objeto definidos pela OMG. A Tabela 3-10 faz uma breve análise comparativa entre as funções para as duas organizações.

FUNÇÕES ODP	CORRESPONDENTE NO OMG
Relacionamentos	Essa função tem como conceito OMG correspondente o serviço de objeto de mesmo nome e os serviços de repositório de implementação .
Relocação	Essa função não encontra conceito OMG correspondente pois essa organização não prevê relocação de objetos ou interfaces.
Armazenamento	Essa função encontra correspondência no serviço de consulta, que permite armazenamento de objetos de grão fino.
Trading	Essa função encontra correspondência direta no serviço de objeto de mesmo nome. A especificação do serviço de *trader* OMG referencia como padrão o modelo prescritivo ODP bem como o documento de especificação do *trader*.
Tipos	Essa função encontra um corresponde direto, embora menos abrangente no repositório de interfaces OMG.

Tabela 3-10. Comparação de Funções de Repositório ODP com conceitos OMG

3.3.2.4 Funções de Segurança

As funções de segurança descritas no RM-ODP têm correspondência num único serviço de objeto. O serviço de objeto de segurança provê controle de acesso a objetos, encriptação de dados e auditoria.

CAPÍTULO 4

UM PROTÓTIPO PARA INVOCAÇÃO DE OPERAÇÕES EM OBJETOS REMOTOS

Foi implementado em linguagem C++ um protótipo para invocação de operações em objetos remotos tendo como base o sistema de RPC (*Remote Procedure Call* ou chamada de procedimento remoto) da SUN como exemplo de encapsulação de serviços em objetos. O protótipo provê transparências de acesso e de localização tanto para o cliente quanto para a implementação de objeto, nem o cliente do objeto nem a sua própria implementação precisam considerar que o objeto possa estar sendo acessado remotamente.

4.1 Requisitos do Protótipo

O protótipo foi proposto como um sistema que permita o encapsulamento de serviços em objetos e o uso destes objetos por clientes localizados em ambientes de rede heterogêneos. Outro requisito importante é que as transparências de acesso e de localização citadas na Tabela 3-5 devem ser providas pelo protótipo. Esses requisitos correspondem, segundo os conceitos [ORB91], a um subconjunto da funcionalidade de um *object request broker.*

Para atingir as transparências citadas através do encapsulamento dos serviços em objetos é importante que se tenha uma linguagem para definição da interface do objeto. Outro requisito do sistema é que a linguagem IDL descrita em [ORB91] seja empregada para essa finalidade.

4.2 A Solução RPC

Sistemas de RPC disponíveis no mercado funcionam em ambientes de rede heterogêneos. Foi decidido construir o protótipo empregando o sistema de RPC da SUN,

um dos mais difundidos no mercado e o único disponível na instalação onde o protótipo foi implementado. Essa abordagem evita os problemas de comunicação interprocesso existentes num ambiente de rede heterogêneo (RPC efetua a comunicação), faltando apenas o encapsulamento dos objetos e prover as transparências de acesso e localização para conclusão do protótipo. A cada invocação de operação num objeto corresponde uma chamada RPC com os mesmos parâmetros da invocação adicionada a referência do referido objeto.

4.2.1 O sistema RPC da SUN

A SUN criou um sistema próprio de RPC para o seu sistema operacional de rede que permite a clientes acessarem serviços remotos desde que o endereço de rede do equipamento onde o prestador de serviço está instalado seja conhecido. Esse sistema é utilizado de forma extensiva pelo *software* da SUN para prover serviços como sistemas de arquivos distribuídos, sistemas de gerenciamento de *lock* distribuído e sistemas de informação de rede.

A cada chamada provida pelo serviço, corresponderá uma *stub* no cliente e outra no servidor que utilizam a biblioteca RPC. Pode ser observado que a terminologia empregada pela SUN é de *stub* do cliente e *stub* do servidor enquanto que em [ORB91] se utiliza *stub* no cliente e esqueleto no servidor. Rotinas XDR (*eXternal Data Representation* ou representação externa de dados) são rotinas de conversão para o padrão de transmissão de dados XDR.

O processo *portmapper* tem como função manter um cadastro dos servidores ativos no equipamento e fornecer essa informação aos clientes que a solicitarem. Há um processo *portmapper* por equipamento e todos os servidores devem ser nele registrados ao se tornarem ativos e fazer o processo inverso ao se desativarem. Na terminologia ODP, o *portmapper* é responsável pelo *binding* entre cliente e servidor.

O sistema RPC está disponível para o programador de aplicação através de chamadas diretas às bibliotecas RPC e XDR ou, de forma mais simples, através de um compilador de serviços de RPC provido pela SUN denominado *rpcgen*. A codificação do sistema pode ser executada inteiramente pelo *rpcgen*, inteiramente pelo programado ou por ambos. A codificação manual é mais complexa mas permite maior flexibilidade ao programador. Muitas dessas flexibilidades vêm sendo incorporadas ao *rpcgen*, permitindo que seu emprego seja cada vez mais difundido. Está representada na Figura 4-1 a estrutura típica de um sistema gerado com o uso do *rpcgen*.

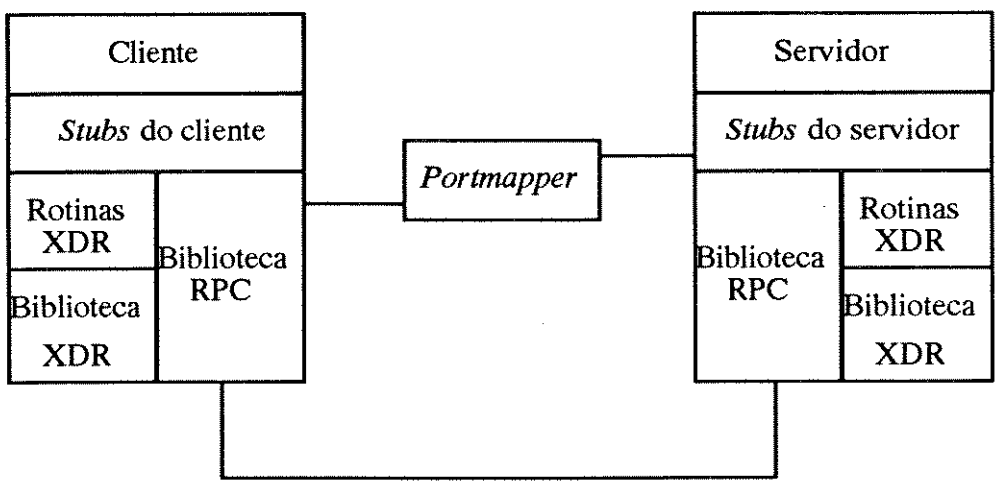


Figura 4-1. Estrutura do Sistema RPC da SUN

4.2.1.1 O *rpcgen*

Para a utilização do compilador *rpcgen* a interface do serviço deve ser descrita numa linguagem usada pela SUN. Esse compilador, tendo a descrição da interface como dado de entrada, gerará o código correspondente às *stubs* do cliente e do servidor e aquele correspondente às rotinas de conversão XDR. O código gerado é responsável pela comunicação entre cliente e servidor numa rede heterogênea. O processo para geração de um sistema distribuído empregando o *rpcgen* consta das seguintes etapas:

1. Criação do arquivo fonte com a definição da interface do sistema.

2. O arquivo fonte criado na etapa 1 é compilado pelo *rpcgen* gerando quatro arquivos em C: um arquivo com o código para conversão dos dados segundo o padrão XDR, um com as *stubs* do cliente e um com as *stubs* do servidor além de um arquivo *header* referenciado pelos anteriores.
3. Codificação do programa implementando as chamadas do servidor.
4. Compilação e *linking* do programa codificado na etapa 3 com os *stubs* do servidor e as rotinas de conversão de dados, gerando o executável do servidor.
5. Codificação do programa cliente.
6. Compilação e *linking* do programa codificado na etapa 5 com os *stubs* do cliente e as rotinas de conversão de dados, gerando o executável do cliente.

Essas etapas devem ser obedecidas para se obter um sistema distribuído procedural completo. A adição de algumas outras ao sistema é necessária para se obter a orientação a objeto.

O fluxo de dados entre arquivos no processo de criação do sistema está representado na Figura 4-2. Como indicado na figura, há vários arquivos envolvidos no processo de criação de um sistema usando o *rcpgen*. A denominação dos arquivos que entram no processo são: "sistema.x" que contém a descrição da interface sob a qual se dará a comunicação, "servidor.c" que contém o código a ser executado pelo servidor para as chamadas descritas na interface e "cliente.c" que contém o código do cliente que faz uso das chamadas descritas na interface. Os arquivos gerados pelo *rpcgen* têm as seguintes denominações: "sistema_xdr.c" contém rotinas de conversão de dados segundo o padrão XDR para cada tipo de dado usado na interface, "sistema_svc.c" contém todo o código das *stubs* do servidor, "sistema_clnt.c" contém o código das *stubs* do cliente e

"sistema.h" contém definições de tipos de dados e rotinas da interface do sistema e é incluído por todos os outros arquivos em C citados.

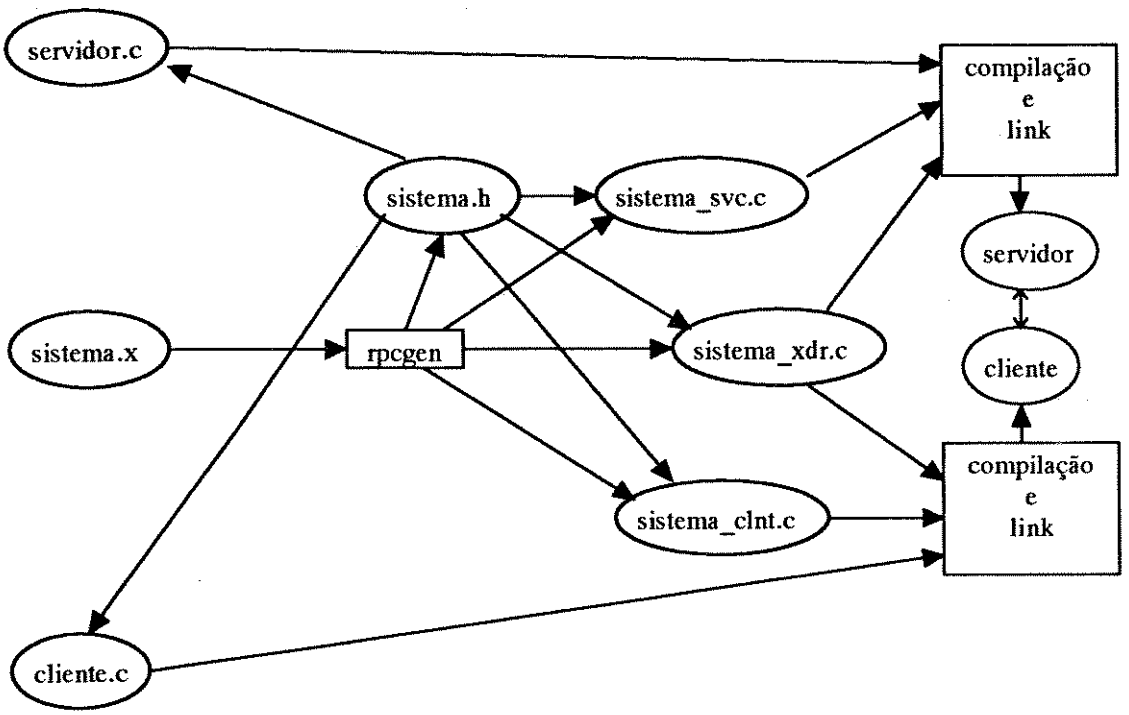


Figura 4-2. Arquivos e Processos para Geração de um Sistema RPC utilizando o *rpcgen*

4.2.2 O padrão XDR

O padrão XDR consiste em um conjunto de convenções para representação de uma série de tipos de dados elementares e de suas composições. Ele não transmite informação especificando que tipo de dado está sendo enviado ou qual a representação desses dados naquela máquina fonte, mas apenas encaminha os dados originais seguindo suas convenções. O tamanho de todos os tipos de dados é arredondado para múltiplos de quatro *bytes* para facilitar os processos de conversão e desconversão.

O padrão XDR é da convenção *big-endian* que é aquela utilizada pela maior parte do *hardware* vendido pela SUN.

Os tipos de dados aceitos pelo padrão são:

- inteiro com ou sem sinal(4 *bytes*),

• enumeração (inteiro),

• lógico (enumeração { FALSE=0,TRUE=1}),

• hiper-inteiro com ou sem sinal (8 *bytes*),

• ponto flutuante (4 *bytes* IEEE),

• ponto flutuante com dupla precisão (8 *bytes* IEEE),

• dados opacos com tamanho fixo ou variável,

• *string* (vetor de caracteres de tamanho variável),

• vetor de tamanho fixo ou variável,

• estrutura,

• união discriminada e

• vazio.

No padrão XDR podem ser definidas constantes e criados novos tipos de dados empregando um mecanismo semelhante ao *typedef* em C. Podem ser declarados dados opcionais, que embora possam ser codificados como uma união discriminada, possuem uma declaração específica.

A especificação de tipos de dados é feita através da linguagem de especificação XDR. Essa linguagem se assemelha ao subconjunto de C empregado para declaração de variáveis e permite o uso de todos os tipos de dados descritos acima. O mapeamento entre tipos de dados XDR e tipos de dados CORBA é direto e sem conflitos e pode ser visto na Tabela 4-1.

Tipo de Dado CORBA	Tipo de Dado XDR
short	*integer*

long	*integer*
ushort	*unsigned integer*
ulong	*unsigned integer*
float	*floating point*
double	*double-precision floating point*
char	*opaque*
string	*string*
boolean	*boolean*
octet	*opaque*
enum	*enum*
any	*discriminated union*
struct	*structure*
sequence	*variable-length array*
union	*discriminated union*
array	*fixed-length array*

Tabela 4-1. Mapeamento dos Tipos de Dados definidos na CORBA para os disponíveis em XDR

4.2.3 O Protocolo RPC da SUN

Esse protocolo é constituído por uma camada colocada sobre o XDR e de um protocolo de transporte. A semântica que pode ser inferida de operações realizadas utilizando o RPC é dependente do protocolo de transporte e não do RPC. Caso seja utilizado como protocolo de transporte o TCP (*Transport Control Protocol*) e se tenha uma resposta da requisição, sabe-se que a operação foi executada exatamente uma vez,

caso não se tenha resposta, no máximo uma vez. Caso se utilize o UDP (*User Datagram Protocol*), se não houver resposta, nada pode ser afirmado, se houver resposta, a operação foi executada pelo menos uma vez.

O RPC da SUN se baseia em programas que têm versões e procedimentos. Os identificadores de programa e procedimento são representados nas mensagens como inteiros sem sinal. Os números de programas são divididos em oito faixas, a primeira é definida pela SUN, a segunda pelo usuário, a terceira é para programas transientes e as outras são reservadas. A segunda faixa, definida pelo usuário, foi utilizada. nesse trabalho. Cada programa tem uma série de procedimentos que são numerados e que ele está apto a executar, os programas estão relacionados com as versões do protocolo que eles implementam. Cada servidor implementa uma faixa de versões do protocolo. Quando uma chamada vai ser executada devem ser indicados, além dos parâmetros de entrada da operação, o programa, o procedimento e a versão a serem empregados. O servidor verificará se pode executar aquela operação para aquela versão do protocolo.

O RPC provê serviços também que não são padrão em RPC. Os serviços presentes fora dessa categoria são o processamento de requisições em lote (uma série de requisições são transmitidas de uma só vez para processamento em lote) e *broadcast* de requisição, onde após a transmissão de uma requisição, ela será multiplicada pelos vários nós da rede e poderá dar origem a numerosas respostas.

O PDU (*Protocol Data Unit*) básico do RPC consiste num identificador de transação, no seu tipo (chamada ou resposta), e o PDU da chamada ou resposta. O PDU de chamada consiste no identificador da versão do protocolo RPC (atualmente na Versão 2), no identificador do programa, no identificador da versão, no identificador do procedimento, nas credenciais de autenticação do cliente , na verificação da autenticação e nos parâmetros específicos do procedimento. Existem diversos tipos de PDU de resposta, pois várias situações anormais podem ocorrer na resposta, entre elas:

• Programa errado.

• Versão do protocolo do programa incompatível.

• Problema na autenticação.

Caso o resultado da chamada seja bem sucedido, apenas os seus resultados são retornados.

Como estratégia de autenticação, o RPC deixa três opções.

Não usar nenhuma, ou autenticação nula, quando não se está preocupado com clientes não autorizados.

Autenticação modelo UNIX. Nessa opção se trabalha com os dados de autenticação do sistema UNIX (*uid* e *gids*). A autenticação UNIX pode ser facilmente falsificada pois não há nenhuma encriptação envolvida então uma terceira opção é colocada à disposição.

A autenticação DES (*Data Encryption Standards*) trabalha com um identificador de usuário geral, no sentido de ser independente do sistema operacional usado e além disso, único na rede. Além desse identificador, é transmitido também um verificador, que é um *timestamp* encriptado. Se o servidor decriptar o verificador e chegar a um valor próximo ao tempo real (hora atual), então a encriptação feita pelo cliente deve estar correta. O verificador é encriptado empregando uma chave de conversação. A chave da conversação é criada pelo cliente e transmitida para o servidor encriptada por um esquema de chave pública. O cliente tem que estar sincronizado com o servidor pois, se isso não acontecer, o verificador decriptado não estará próximo ao tempo real.

4.3 A Abordagem de Implementação Usando RPC

Na utilização do RPC para implementação de sistemas distribuídos orientados a objeto a opção mais natural é o emprego de uma chamada RPC para cada método de objeto. Todas as chamadas RPC teriam um parâmetro *default* que seria uma referência ao objeto sobre o qual o método será aplicado. Esta abordagem soluciona o problema da

chamada de métodos de objetos remotos, ficando a cargo do RPC o processo de *binding*. Ele estabelece o *binding* entre cliente e servidor, empacota os parâmetros, transmite, espera a resposta, empacota a resposta e a recebe. Caso haja qualquer exceção nesse processo um código que a identifique é retornado.

A colocação de transparências na forma descrita por [ODP94] faz o sistema mais cooperativo, tornando essa infra-estrutura aberta . Algumas transparências são mais diretas na implementação como acesso e localização, outras exigem um pouco mais de trabalho como grupo, recurso e federação, as que exigem mais elaboração seriam migração, falha e transação..

4.4 O Protótipo como foi Implementado

Para executar o acesso a objetos remotos, uma casca deve ser colocada em volta do sistema RPC acima descrito. Essa casca fará um mapeamento entre o RPC da SUN (procedural) e os objetos (no caso, através da linguagem C++).

O código fonte dessa casca é diferente para cada uma das classes de objeto e dependente da interface do objeto. Para evitar que esse código tenha que ser reescrito para cada nova classe criada faz-se a descrição da interface dos objetos da classe em IDL e desenvolve-se um compilador que, a partir dessa descrição em IDL, gera o código da casca correspondente.

O código gerado por esse compilador incluiria um arquivo de descrição da interface do objeto para o *rpcgen*, dois arquivos cabeçalho C++ (um para o cliente e outro para o servidor) e dois arquivos fontes C++, que implementariam os *stubs* e os esqueletos. Haveria também bibliotecas de suporte para o cliente e para o servidor, independente de objeto. O esquema de bibliotecas para clientes e servidores ficaria então da forma mostrada na Figura 4-3.

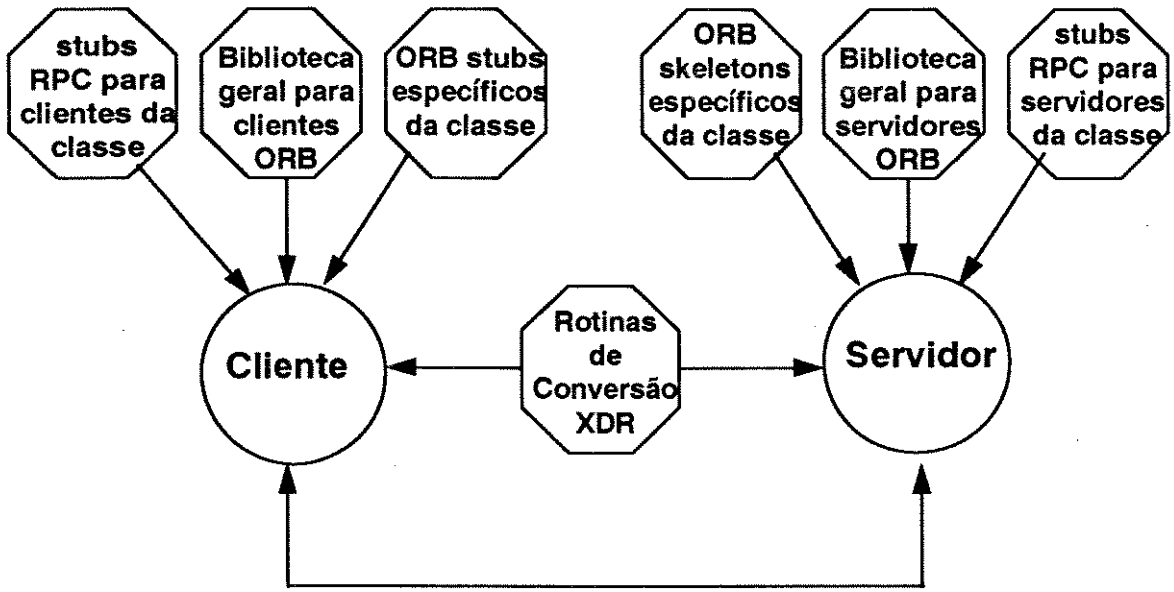


Figura 4-3. Módulos para Geração dos Executáveis do Cliente e do Servidor no Protótipo

4.4.1 Restrições do sistema em relação ao padrão CORBA

O protótipo implementa praticamente o subconjunto mínimo funcional do modelo CORBA. Alguns módulos não foram implementados porque a sua função não era fundamental para a operação do protótipo, outros tiveram sua implementação feita de forma bem simples apenas para dar ao protótipo uma funcionalidade mínima.

A interface por invocação dinâmica não foi implementada, pois embora ela adicione uma grande flexibilidade ao sistema, sua implementação é um tanto quanto complexa. O repositório de interfaces, fundamental para o funcionamento da interface de invocação dinâmica, é outro módulo complexo que não foi implementado. O ORB *core* só tem funcionalidade real no lado cliente, no lado servidor suas funções são exercidas pelos esqueletos RPC gerados automaticamente pelo *rpcgen*. O adaptador de objetos tem apenas a função de gerenciar o repositório de objetos local para a implementação e fazer as traduções entre referências de objetos e eles próprios.

Algumas restrições foram feitas em relação aos recursos da IDL que podem ser utilizados, ou seja, foi implementada a infra-estrutura para um subconjunto da IDL. Se

houvesse um compilador C++ que aceitasse exceções como especificado na última versão da linguagem as exceções poderiam ter sido implementadas com relativa facilidade. Uma restrição adicional é que as operações tenham apenas parâmetros de entrada, sendo a única informação devolvida pela implementação o valor de retorno da operação. O sistema de contextos não foi implementado, pois seu mapeamento para RPC exigiria um mecanismo complexo.

A política de ativação da implementação de objeto utilizada foi a de um processo para cada classe de objetos, segundo o critério apresentado em [ORB91],. Esse modelo foi considerado o mais interessante por permitir que o *ORB core* seja integrado ao adaptador de objetos e aos esqueletos no mesmo processo. Esta opção se adapta ao mecanismo de RPC utilizado, pois cada conjunto composto de um *ORB core*, de um adaptador de objetos, de esqueletos IDL de uma classe e de sua implementação pode ser mapeado em um programa servidor RPC.

4.4.2 Fluxo de Informação no Sistema

O sistema contém um *stub* e um esqueleto C++ para cada método da classe que está sendo implementada e uma biblioteca de suporte a RPC da SUN. Além desses módulos integrados aos processos servidores e clientes, existem o *portmapper* do RPC e o repositório de implementações. O *portmapper* encontra a porta que o servidor está atendendo através da sua localização e do identificador de classe . O repositório de implementações localiza um servidor através da classe de objetos por ele implementada . As camadas existentes e a estrutura geral do sistema podem ser observadas na Figura 4-4.

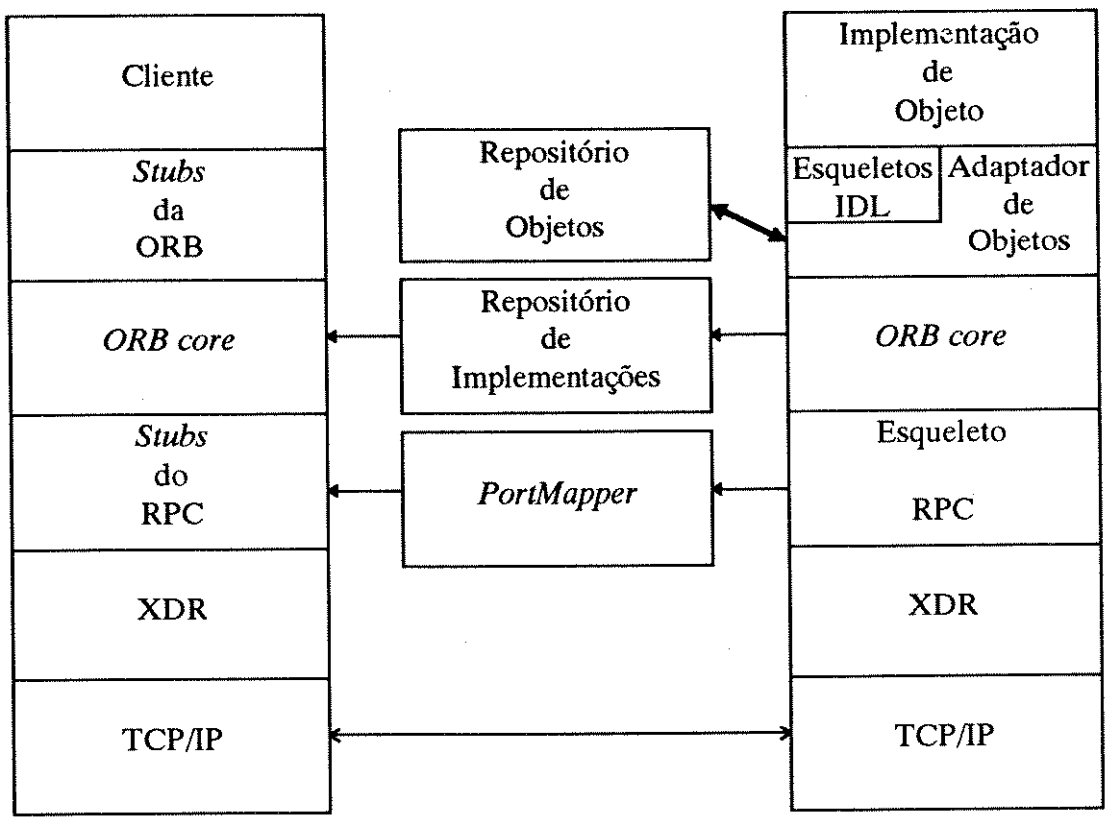


Figura 4-4. A Arquitetura do Sistema

O repositório de implementações é apenas um arquivo com um registro para cada servidor da rede, contendo seu identificador de classe e seu endereço IP. Basta ler esse arquivo registro a registro comparando o identificador de classe para descobrir se há um servidor ativo no sistema que implementa aquela classe de objetos e em que nó ele está localizado.

O repositório de objetos é implementado como uma lista de referência cruzada entre as referências de objeto e seus ponteiros. Na ativação de uma implementação de objeto, é executada uma rotina de cadastramento no adaptador de objetos que cria uma nova referência para aquele objeto. No recebimento de uma requisição, a referência de objeto é pesquisada nessa lista e é feita a chamada ao método empregando o ponteiro correspondente. O adaptador de objetos tem chamadas apenas para acesso a esse repositório, não tendo nenhuma outra função nesse protótipo.

O *portmapper* é utilizado apenas para informar em que porta o servidor está atendendo chamadas naquele nó. O *rpcgen* gera o código para cadastramento da implementação junto ao *portmapper* e para permitir ao cliente consultas sobre o número da porta sob a qual a implementação está atendendo. Seu protocolo é interno ao sistema de RPC.

Os *stubs* da ORB fazem chamadas à ORB *core* sob a forma de rotinas de biblioteca para assegurar a existência de uma estrutura cliente RPC com os parâmetros corretos para a invocação. A invocação é então enviada ao sistema de RPC que é responsável pelo estabelecimento do *binding* entre o cliente e a implementação de objeto.

Os esqueletos da ORB consultam o repositório de objetos através do adaptador de objetos para conhecer a que objeto a referência recebida pertence e enviam a chamada ao objeto correspondente.

4.4.3 O Funcionamento no Cliente

No cliente, como pode ser observado na Figura 4-5, cada *stub* ORB utiliza rotinas de uma biblioteca de apoio ORB e então envia a invocação para o sistema RPC. O *stub* RPC empacota os parâmetros e utiliza uma rotina de apoio responsável pela execução da invocação. Essa rotina faz a conversão dos parâmetros empregando a biblioteca XDR e os transmite. Com a recepção dos resultados, esses são desconvertidos utilizando a biblioteca XDR e enviados ao *stub* RPC que os repassa ao *stub* ORB.

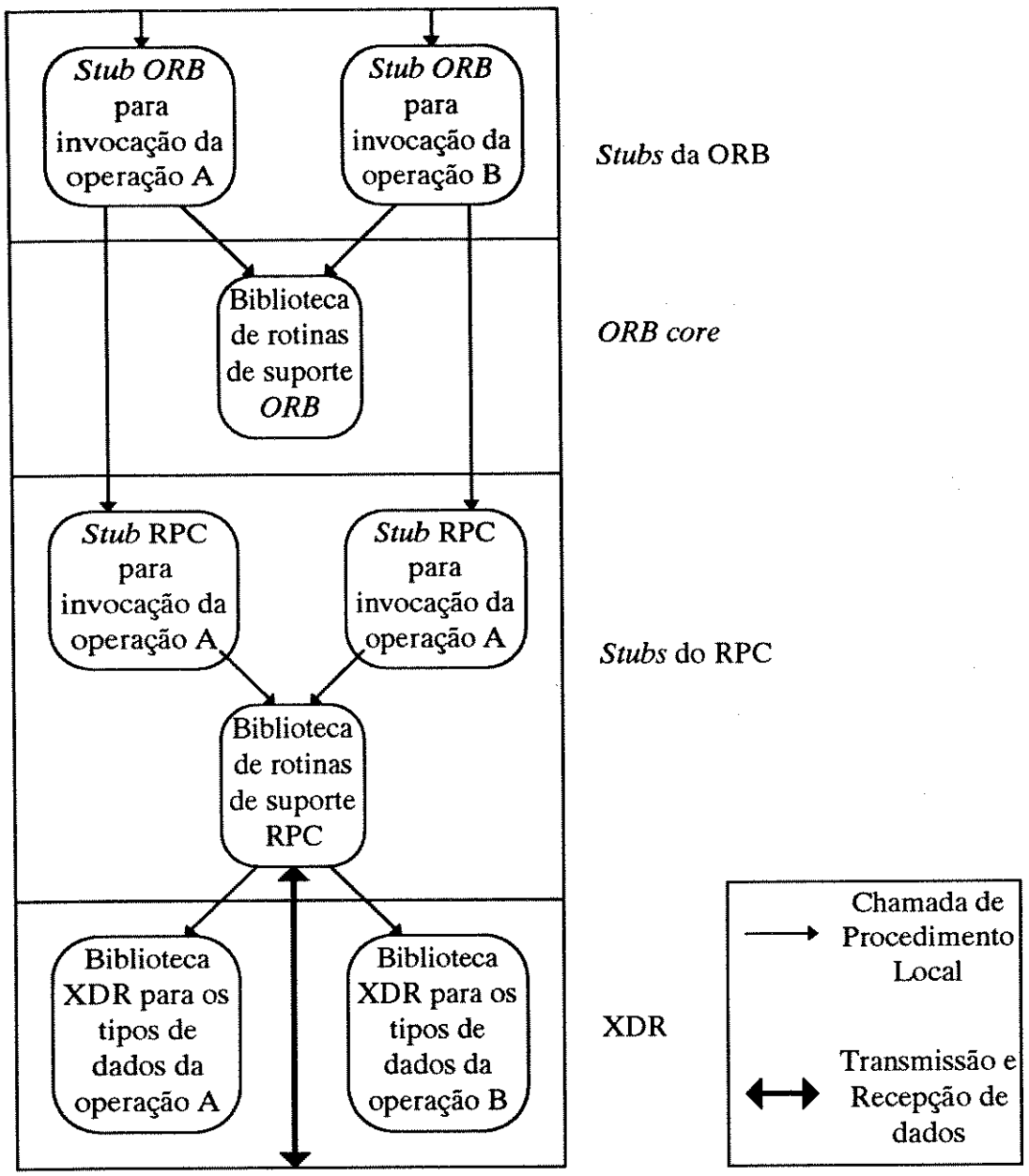


Figura 4-5. Funcionamento no Cliente

4.4.4 O Funcionamento na Implementação do Objeto

Como podemos observar na Figura 4-6, o esqueleto RPC é responsável pela recepção das invocações e por providenciar a sua execução. Ele converte os parâmetros utilizando as rotinas XDR e executa o esqueleto ORB correspondente à operação desejada. Esse esqueleto ORB pode empregar rotinas de apoio presentes na biblioteca do adaptador de objetos.

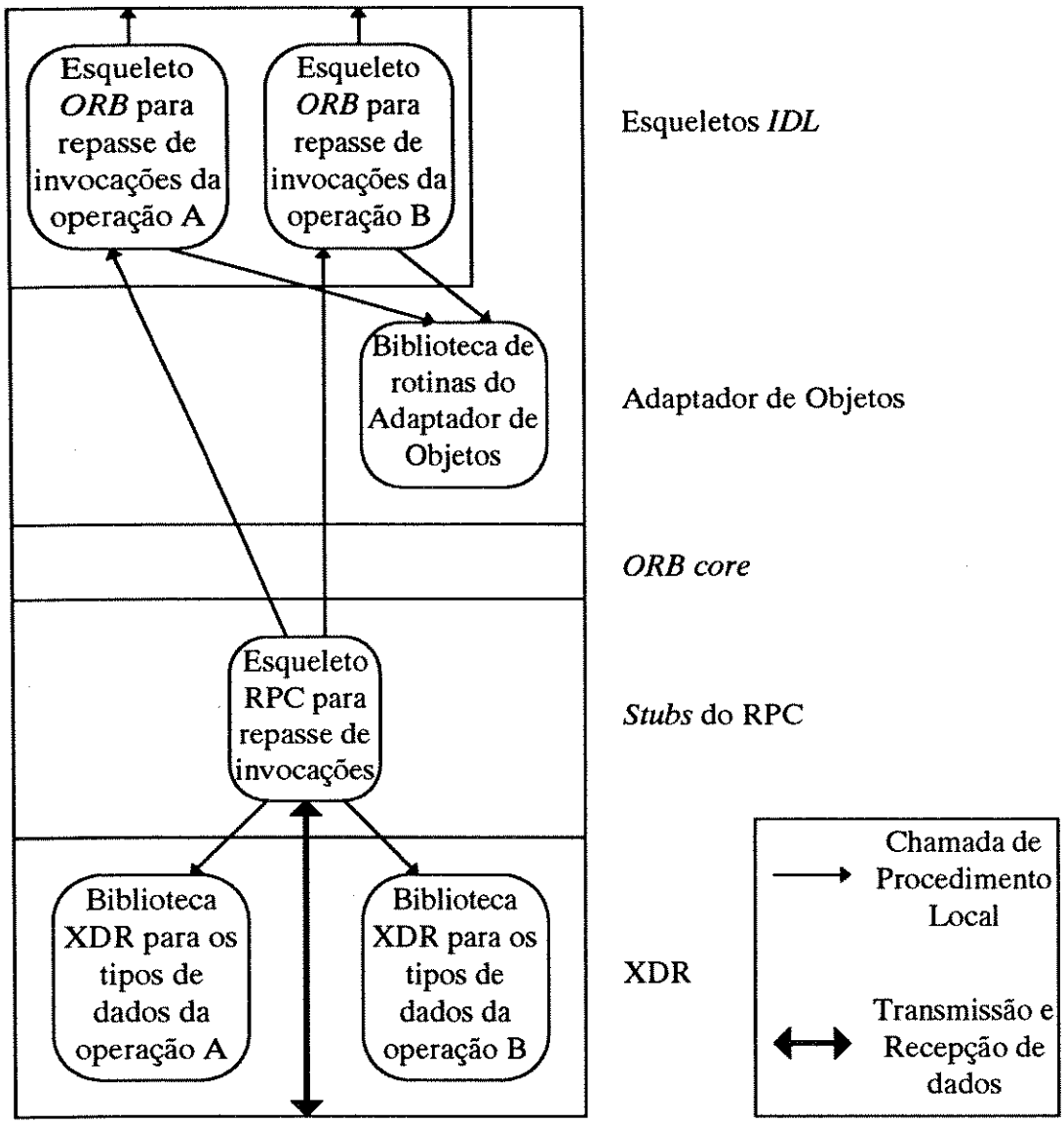


Figura 4-6. Funcionamento na Implementação de Objeto

4.4.5 A Referência de Objeto

A determinação dos dados necessários nas referências de objeto é uma opção importante. É citada na [ORB91] uma referência de objeto que identifica univocamente um objeto no sistema e é mencionado um valor de referência de objeto que não se refere a nenhum objeto (semelhante ao valor *NULL* para ponteiros do C). Como conclusão tem-se que uma referência de objeto é um ponteiro para um objeto num ambiente distribuído.

No sistema distribuído analisado não há possibilidade de migração de objetos. Nesse caso, uma decisão que facilita a implementação e incrementa a performance do sistema é a inclusão do endereço do nó onde se encontra a implementação do objeto na referência de objeto. Fazendo-se essa opção a localização do objeto, que é fixa, fica conhecida uma vez que se tem a referência do objeto. É importante salientar que essa localização é para uso apenas interno ao sistema para garantia da transparência de localização descrita na Tabela 3-5.

Num nó podem funcionar diversas implementações de objetos, cada uma implementando uma classe diferente. Para distinção entre essas implementações é conveniente que se tenha um identificador da classe do objeto na referência. Esse identificador servirá para se saber qual a implementação de objeto que deve ser ativada.

Um terceiro componente para a referência de objeto seria um identificador de objeto local à implementação de objeto. Os valores a serem atribuídos a esse identificador podem ser sequenciais com a criação de objetos.

Como consequência a referência de objeto possui o endereço do nó onde se encontra a implementação de objeto, o identificador da classe implementada por essa implementação e um identificador para o objeto local a implementação. O valor de nenhum objeto (*NULL*) pode ser convencionado como sendo os valores de todos os componentes iguais a zero.

4.4.6 O Mecanismo de Objeto Cliente e Objeto Servidor

Para a implementação das transparências de acesso e localização, pode ser empregado um mecanismo extremamente interessante descrito na literatura [WU93]. Nesse mecanismo, para cada objeto na implementação de objeto (objeto servidor) utilizado pelo cliente, existe um objeto correspondente no cliente (objeto cliente) que faz

as vezes de procurador do objeto servidor. O objeto cliente contém a referência de objeto para o objeto servidor e tem a sua interface.

O mecanismo de funcionamento pode ser observado na Figura 4-7. Com a referência do objeto servidor, o objeto cliente envia as chamadas que ele recebe ao seu objeto servidor. Como as invocações feitas ao objeto cliente são invocações a um objeto local, a transparência de acesso já foi assegurada, consequentemente o cliente não sente nenhuma mudança ao acessar um objeto local ou remoto. A transparência de localização atingida através desse mecanismo é uma decorrência da localização do objeto estar encapsulada no objeto cliente e este se encarrega de contatar o objeto servidor na sua localização. O cliente não precisa considerar que cada objeto tem uma localização.

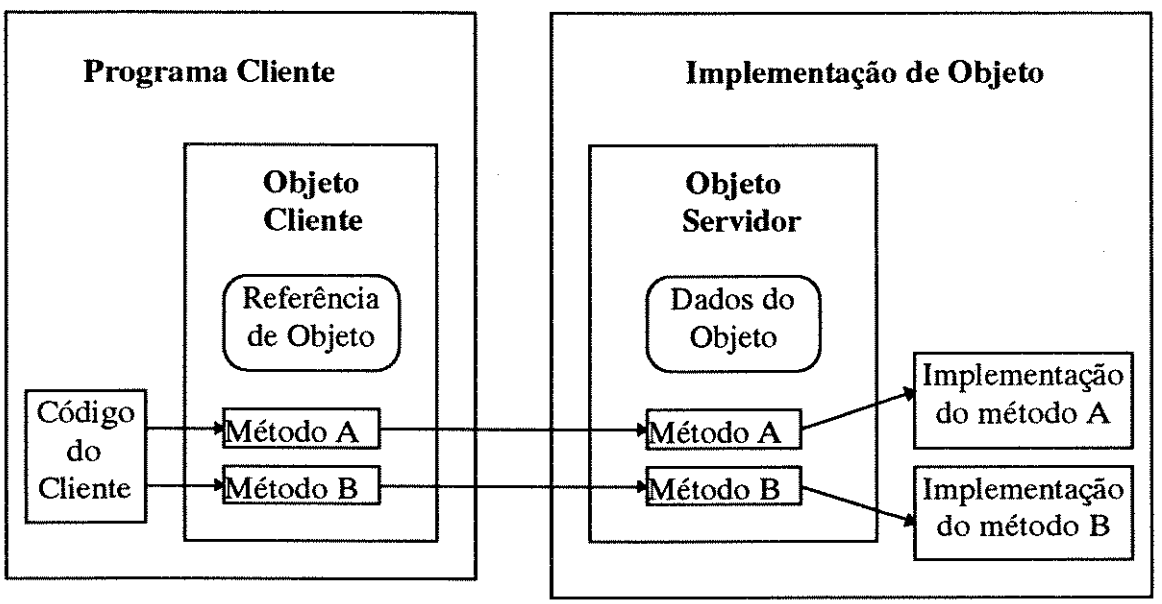


Figura 4-7. O Mecanismo do Objeto Cliente/Objeto Servidor

Esse tipo de abordagem permite que clientes diferentes tenham objetos clientes referenciando o mesmo objeto servidor. Essa situação está representada na Figura 4-8. Podem existir problemas de concorrência no caso de dois clientes diferentes invocarem métodos no mesmo objeto servidor ao mesmo tempo. Como o processamento na implementação de objeto é sequencial (*single-threaded*), no presente sistema isso é

impossível. No caso de uma implementação de objeto com processamento concorrente, deve ser utilizado algum mecanismo para evitar inconsistência.

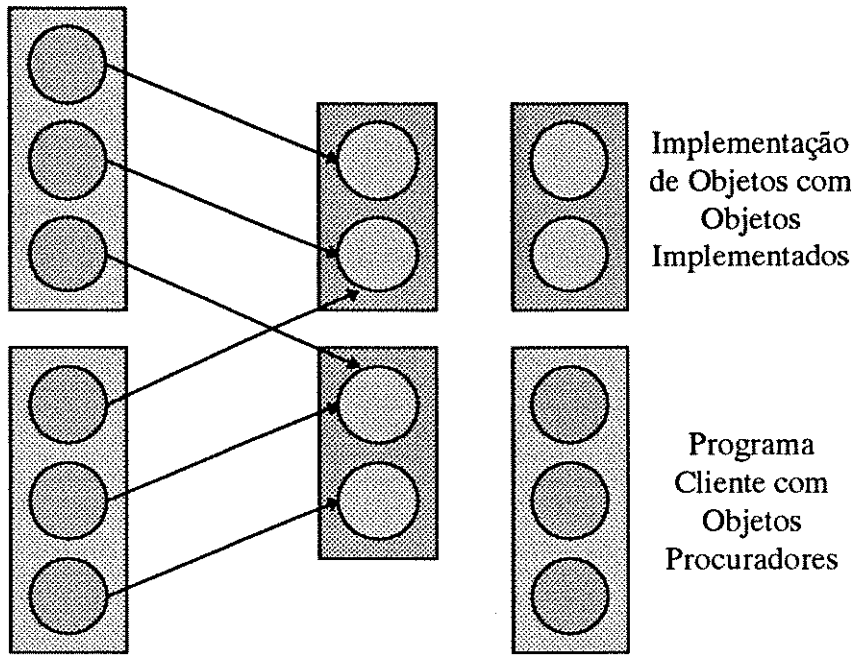


Figura 4-8. Múltiplos Objetos Clientes Referenciando o Mesmo Objeto Servidor

4.4.7 Plataforma e Ferramentas de Desenvolvimento Utilizadas

A plataforma utilizada para o desenvolvimento do protótipo foi a rede de estações de trabalho existente na Faculdade de Engenharia Elétrica da UNICAMP. A implementação de objeto funcionava na servidora da faculdade e os clientes foram executados de vários nós da rede, ligados direta ou indiretamente à servidora. Só foram executados testes em estações com arquitetura *SPARC* pois o compilador *C++* utilizado não tinha versões estáveis nas outras arquiteturas, embora o sistema funcione em qualquer arquitetura devido à isolação de formato de dados provida pelo nível XDR.

Para o desenvolvimento do sistema foi utilizado o compilador *C++* da *Free Software Foundation*, o *g++*, versão 2.5.7. Para a geração das *stubs* e dos esqueletos *RPC* foi utilizado o *rpcgen* incluído no *SunOS* versão 4.1.1.

4.5 Especificação do Mapeamento IDL para *Stubs* e Esqueletos

O sistema permite a elaboração de um compilador cuja entrada seja a descrição da interface em IDL e cujas saídas sejam *stubs* e esqueletos para a implementação do sistema. Um mapeamento a partir do qual torna-se possível construir o compilador é descrito a seguir.

4.5.1 Mapeamento para *Stubs*

Cada operação IDL seria mapeada na seguinte *stub* ou método *C++:*

```
valret     interface::operação(parâmetros)
{
  valret *result;    //Variável que armazenará o resultado
  client(clnt,node,sock);      //Prepara cliente RPC
  result=interface_método(reference,parâmetros,clnt);
     //Faz a invocação
  return *result;    //Volta o resultado
};
```

Onde:

- valret representa o tipo do valor de retorno da operação,
- interface é o nome da interface que se está mapeando,
- operação é o nome da operação que está sendo mapeada,
- parâmetros são os parâmetros de entrada da operação.

4.5.2 Mapeamento para Esqueletos

Para cada operação IDL teríamos o seguinte esqueleto de invocação de método em C++:

```
extern "C"{     //Para o esqueleto poder ser chamado pelo RPC
```

```
valret *interface_operação_1(Object *ref,parâmetros)
{
  static valret result;  //Variável para guardar o resultado
  result=interface_orb_object(*ref)->operação;     //Invoca
  return &result;   //Volta resultado
};
```

Onde são seguidas as mesmas convenções de nomes da subseção anterior.

4.5.3 Mapeamento para Tipos de Dados

O mapeamento para tipos de dados é feito segundo a Tabela 4-1 que apresenta o mapeamento dos tipos de dados CORBA para os tipos de dados XDR. Após esse mapeamento o *rpcgen* gera as rotinas de conversão XDR para os tipos de dados do sistema.

4.6 Análise do Sistema Implementado Segundo o Modelo ODP

O sistema se adapta ao ponto de vista de engenharia, pois ele implementa mecanismos de transparência de distribuição e de *binding* entre objetos. Pode ser feito um mapeamento entre os componentes do sistema e elementos do ponto de vista de engenharia.

Um mapeamento natural para o *stub* ODP é o *stub* ORB em conjunto com o *stub* RPC e a biblioteca de conversão de dados XDR. O conjunto desses três elementos faz as funções de um *stub* ODP.

O mapeamento para o *binder* fica por conta das rotinas RPC. Não foi necessária preocupação com a função de *binding* visto que o sistema de RPC executa essa tarefa.

O objeto de protocolo é o protocolo TCP/IP usado na rede *ethernet* onde o sistema foi implementado. Esse protocolo garante a comunicação fim a fim entre os objetos.

CAPÍTULO 5

CONCLUSÕES E TRABALHOS FUTUROS

As especificações analisadas nesse trabalho representam passos importantes na padronização de sistemas distribuídos. Os documentos da OMG já têm implementações comerciais criadas por grandes empresas fornecedoras de *software*. O modelo de referência ODP servirá como guia para a implementação de sistemas distribuídos, como o modelo de referência OSI tem sido empregado.

ODP é um padrão complexo que cobre com elegância toda a especificação de um sistema distribuído, desde os níveis mais baixos do sistema operacional até o nível de empreendimento, onde as necessidades da organização são muito importantes. Esse padrão cobre praticamente todos os componentes necessários a um sistema distribuído, qualquer que seja sua aplicação. Para atingir este objetivo obrigatoriamente o modelo tem que ser complexo. Uma consequência dessa complexidade é a inexistência até o ano de 1994 de uma empresa fornecedora de *software* que tenha se sujeitado ao risco de anunciar um produto que apresente conformidade parcial ou total com a especificação ODP.

O mercado tem grande expectativa em relação aos documentos emitidos pelo OMG. A especificação CORBA [ORB91] que foi criada pelo OMG é o denominador comum dos sistemas distribuídos orientados a objeto. Sua infra-estrutura permite usar uma funcionalidade extremamente simples para a comunicação entre objetos. Como consequência dessa simplicidade vários produtos que seguem a especificação OMG estão sendo comercializados. A implementação desenvolvida nesse trabalho mostra que um subconjunto mínimo da arquitetura pode ser implementado facilmente utilizando protocolos RPC disponíveis no mercado. A grande desvantagem da arquitetura é não

prover suporte para aplicações multimídia, na forma de interfaces *stream* com características isócronas. Para implementação desse tipo de aplicações será necessário o emprego de algum artifício, provavelmente não muito elegante.

5.1 Sugestões para Trabalhos Futuros

O trabalho aqui apresentado pode dar origem a diversos trabalhos e implementações tendo como objetivo final um sistema distribuído orientado a objeto completo de acordo com uma das especificações analisadas.

Um trabalho interessante seria a extensão do protótipo para toda a IDL CORBA. Essa extensão incluiria recursos como exceções e parâmetros de entrada e saída. O protótipo poderia ficar independente do *rpcgen* para dar maior flexibilidade à implementação, fazendo chamadas diretas à biblioteca do sistema RPC. Essa abordagem traria ganhos em desempenho e tamanho do código objeto.

Outro trabalho sugerido é a extensão do protótipo para a cobertura de toda a especificação CORBA do OMG, incluindo o repositório de interfaces, interface de invocação dinâmica, repositório de implementações, núcleo ORB e um conjunto de adaptadores de objeto.

O trabalho mais complexo mas ao mesmo tempo onde ter-se-ia maiores resultados em termos de funcionalidade é a implementação de todo o modelo de referência ODP. Essa implementação consistiria num trabalho bastante elaborado, para execução a longo prazo e seu resultado seria um sistema com todas as funcionalidades e transparências propostas por ODP.

Deve ser lembrado que tanto as especificações OMG quanto ISO sofrem constantemente alterações. A OMG está elaborando para divulgação o documento de especificação da CORBA versão 2.0. A tendência da ISO é que os seus documentos se

estabilizem, uma parte deles já está no DIS (*Draft of International Standard* ou Minuta de Padrão Internacional) e brevemente serão padrões internacionais.

BIBLIOGRAFIA

[COH81] Cohen, D.: "On Holy Wars and a Plea for Peace," IEEE Computer, vol. 14, pp. 48-54, Out. 1981.

[MEN94] Mendes, M. J. e Madeira, E. R. M., "Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware" - 12o Simpósio Brasileiro de Redes de Computadores - Curitiba PR - Maio 1994, pp. 93-109

[ODP94] International Standards Organization, "ISO/IEC JTC1/S C21, Basic Reference Model ODP - Part 1:Overview and Guide to Use; Part2: Descriptive Model; Part 3: Prescriptive Model" - fevereiro 1994

[OMA92] Object Management Group, "Object Management Architecture Guide", Revisão 2.0, OMG TC Document 92.11.1 - Setembro 1992

[ORB91] Object Management Group, "The Common Object Request Broker: Architecture and Specification", revisão 1.1, OMG TC Document 91.12.1 - Dezembro 1991

[OSA92] Object Management Group, "Object Services Architecture", Revisão 6.0, OMG TC Document 92.8.4 - Outubro 1992

[RFP92] Object Management Group, "Object Services Request for Proposal 1", OMG TC Document 92.8.4 - Outubro 1992

[RFP93] Object Management Group, "Object Services Request for Proposal 2", OMG TC Document 93.6.1 - Novembro 1993

[STR91] Stroustrup, B. - "The C++ Programming Language", 2nd Ed. Addison-Wesley, Reading (MA), 1991.

[TAN87] Tanenbaum, A. S. - "Operating Systems: Design and Implementation", Englewoods Cliffs, NJ: Prentice Hall, 1987.

[TAN88] Tanenbaum, A. S. - "Computer Networks", 3rd Ed. Englewoods Cliffs, NJ: Prentice Hall, 1988.

[TAN92] Tanenbaum, A. S. - "Modern Operating Systems", Englewoods Cliffs, NJ: Prentice Hall, 1992.

[TSC93]Tschammer, V.; Mendes, M. J.; Souza, W. L.; Madeira, E. R. M. e Loyolla, W. P. D. C. - "Processamento Distribuído Aberto e o Modelo RM-ODP/ISO" - 11o Simpósio Brasileiro de Redes de Computadores - Campinas, SP - Maio 1993, pp. 175-195

[WU93] WU, Z. - Making C++ A Distributed Programming Language - Fourth Workshop on Future Trends of Distributed Computing Systems - Lisboa, Portugal - Setembro 1993, pp.228-233