

Middleware OO

Estudo de Caso: CORBA

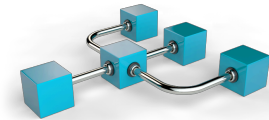


Arquitetura de Sistemas Distribuídos,
Paralelos e Concorrentes
Escola Politécnica – PUCPR
Luiz A. de P. Lima Jr. • luiz.lima@pucpr.br

1

Tópicos

Introdução
Transparências
O Modelo de Programação
IDL
Arquitetura CORBA
Invocações Estáticas e Dinâmicas
O Modelo de Interoperabilidade CORBA



2

2

Object Management Group (OMG)

- **OMG:** <http://www.omg.org>
- **Consórcio:**
 - ✦ Desenvolvedores
 - ✦ Organizações governamentais
 - ✦ Grupos de usuários
- **Objetivos:**
 - ✦ Promover tecnologias OO
 - ✦ Criar especificações
 - ✦ Definir padrões de interoperabilidade



3

3

A Especificação CORBA

- **CORBA:**
 - ✦ Suporte para **programação distribuída aberta**
 - ✦ Padrão aberto
 - ✦ **Interoperabilidade** + **serviços** adicionais
- **RFP + RFC:**
 - ✦ Muitas implementações comerciais

4

4

Implementações da especificação CORBA

- ACE/TAO
- ORBIX (Micro Focus)
- VisiBroker (Micro Focus)
- omniORB (+ omniORBpy)
- JacORB
- Orbacus
- Component Broker – WebSphere (IBM)
- ObjectBroker (BEA)
- JavalDL (Sun)
- CORBAplus (Expersofts)
- COOL-ORB (Chorus Systems)
- Orblite (HP)
- DSOM (IBM)
- DOE (Sun)
- ISIS, Orblite, etc.

5

5

Implementações da especificação CORBA

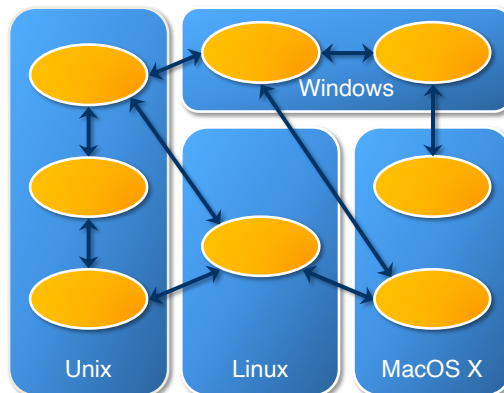
- Produtos são:
 - ✦ Implementações das especificações CORBA mais ou menos conformes às normas e mais ou menos estendidas por serviços adicionais.
- Free CORBA:
 - ✦ ACE/TAO (C++)
 - ✦ JacORB (Java)
 - ✦ OmniCORBA, outros...

6

6

Objetivos de CORBA

- Fornecer **interoperabilidade** entre objetos rodando em sistemas distribuídos (*heterogêneos*) e possibilitar a sua composição em aplicações.



7

7

Como garantir interoperabilidade?

1. Padronizar a “aparência externa” dos objetos (**interfaces**).
2. Propor um mecanismo genérico de ligações entre objetos (**bindings**).
3. Esconder as dificuldades da distribuição dos programadores (**transparências**).
4. Oferecer um conjunto padronizado de **serviços adicionais** aos programadores.

8

8

CORBA fornece...

- Uma base para computação OO **distribuída**:
 - ✦ Suporte para **invocação de métodos remotos**
 - ✦ **Referências transparentes** para objetos remotos
- Uma plataforma OO **aberta**:
 - ✦ **Independência de linguagens** de implementação
 - ✦ Transparência de HW e SO
- Um conjunto de **serviços** distribuídos
 - ✦ **Naming, events, trader...**
 - ✦ Acessados como objetos CORBA

9

9

CORBA

**CORBA =
interoperabilidade
+ serviços**

10

10

Transparências CORBA

- Transparência de **Acesso**
 - ✦ A interação A-B é idêntica seja B local ou remoto (representação de dados).
- Transparência de **Localização**
 - ✦ A implementação da interação A-B é independente da localização exata de B.

11

11

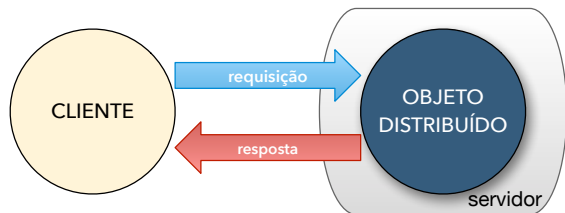
Interações entre Objetos

- Objetos CORBA:
 - ✦ Podem estar em **qualquer lugar** da rede.
 - ✦ São agrupados em **componentes binários** acessados por meio da invocação de métodos.
- Linguagem de implementação, compilador usado e localização do objeto são **transparentes** para o cliente.
- Clientes precisam conhecer somente a **interface** publicada pelos objetos servidores.

12

12

O Modelo de Programação



13

13

Modelo de Programação

- **Objeto distribuído**
 - ✦ Encapsula estado interno
- **Operação/método**
 - ✦ Ponto de acesso a um procedimento executável
- **Interface**
 - ✦ Conjunto de operações/métodos (única por objeto)
- **Interações**
 - ✦ Cliente/servidor (síncronas ou assíncronas)

14

14



Interfaces

Middleware OO – CORBA

15

Interface Definition Language (IDL)

- Para padronizar a **aparência externa** dos objetos.
- Especificação da interface (e não do código) em uma linguagem neutra – **IDL**.
 - ✦ IDL é uma linguagem **puramente declarativa** (i.e., sem detalhes de implementação).
 - ✦ IDL define **interfaces contratuais** entre clientes e servidores.

16

16

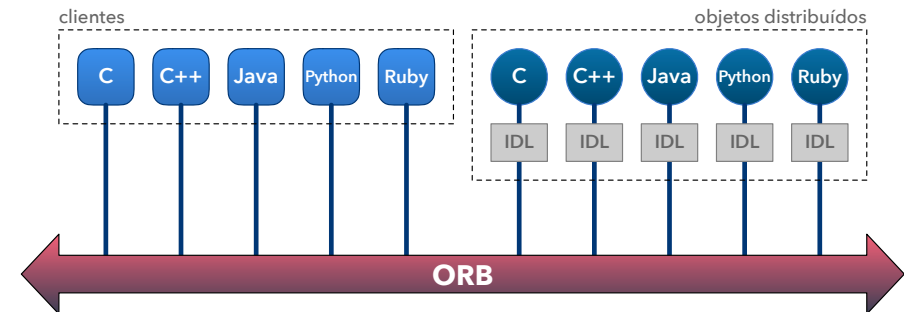
IDL permite a definição da interface **independentemente** de sistema operacional, linguagem de programação e rede para todos os serviços e componentes do ambiente CORBA.

IDL = *Interface Definition Language*

17

17

Independência de IDL da Linguagem de Programação



Arquivos IDL podem ser compilados.

18

18

Compilação IDL

- **Compilação:**
 - ✦ para gerar código de comunicação (*stubs e skeletons*).
- Várias **linguagens-alvo**: C/C++, Java, Ruby, Python, Smalltalk, COBOL, etc.
- Tipos IDL são mapeados em tipos e código auxiliar na linguagem-alvo.

19

19

Componentes da Linguagem IDL

- **Nome da interface**
- **Protótipos dos métodos**
 - ✦ Tipo do valor de retorno
 - ✦ Nome do método
 - ✦ Parâmetros:
 - Tipos
 - Direção de transferência: *in, out, inout*
- **Atributos públicos**
- **Lista de exceções**

20

20

Exemplo de Arquivo IDL

```
exception SaldoInsuficiente {};
interface Conta
{
    readonly attribute float saldo;
    void deposito(in float valor);
    void saque(in float valor) raises
    (SaldoInsuficiente);
    oneway void shutdown(in string senha);
};
```

21

21

Protótipos dos Métodos

- `[oneway] <tipo> <id> (param1, ...) [raises (exc1,...)] [context (name1, ...)]`
 - ✦ “oneway”: operação é “anúncio” (“best-effort”)
 - ✦ `<tipo>`: tipo do valor de retorno
 - ✦ `<id>`: identificador (nome) da operação
 - ✦ Parâmetros: `in`, `out` ou `inout` + `<tipo>`
 - ✦ “raises”: opcional – exceções
 - ✦ “context”: opcional – fornece informações adicionais específicas de uma operação que podem afetar a sua execução (geralmente requisitos não funcionais).

[] =
opcional

22

22

Tipos Pré-definidos

- short
- long
- float
- char
- byte
- boolean
- string
- sequence
- (+ outros tipos derivados)

23

23

Exceções IDL

- Padronizadas
 - ✦ Memória insuficiente
 - ✦ Comunicação interrompida
 - ✦ Erro na conversão de dados
- Exceções dependentes da aplicação
- IDL suporta herança.

24

24



ASDPC

Exercício: Interface em IDL

Middleware OO – CORBA

25



Exercício IDL

- Um servidor de nome **Server** é capaz de responder às operações da tabela abaixo:

Operação	Parâmetros	Resposta do Servidor
eco	"ola"	"ola_ECO"
soma	1.1 ; 2.2	3.3
checkpoint	1616519478	-

- Especifique **em IDL** a interface deste servidor.

26



ASDPC

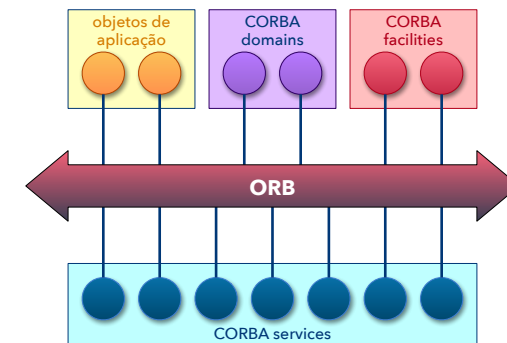
A Arquitetura CORBA

Middleware OO – CORBA

27



A Arquitetura CORBA



28

A Arquitetura CORBA

● ORB: *Object Request Broker*

- ✦ Suporte para invocação de métodos remotos

● CORBAServices

- ✦ **Extensões** do ORB
 - Naming, trading, segurança, etc.

● CORBAfacilities

- ✦ Infra-estruturas **horizontais**:
 - Funções de mais alto nível
 - Interface com o usuário, gerenciamento de informações, impressão, etc.
- ✦ Infra-estruturas **verticais** ("domínios"):
 - Funções específicas do domínio
 - Telemedicina, telecomunicações, etc.

29

29

O ORB

- Permite envio de requisições e respostas de forma transparente entre objetos em uma arquitetura possivelmente distribuída.



30

30

O ORB

- A interface percebida pelo cliente é independente da:

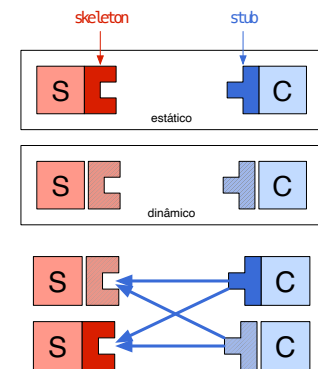
- ✦ **Localização** do objeto
- ✦ Linguagem de **implementação**

31

31

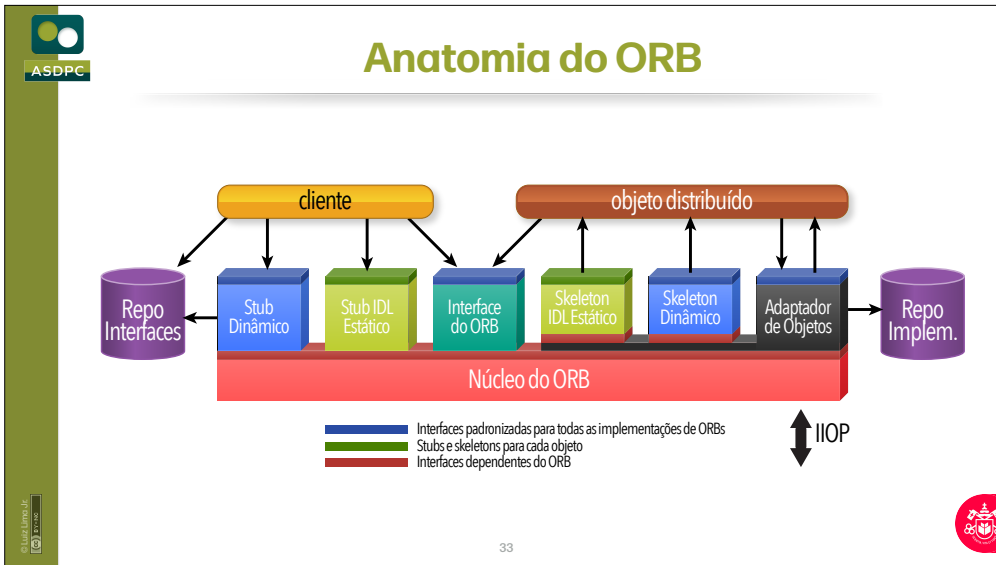
Benefícios do ORB

- Invocações de métodos de forma **estática** (tempo de compilação) + **dinâmica** (tempo de execução)
- Ligações entre **linguagens de alto nível**
- Sistemas **auto-descritivos** (repositórios de interfaces)
- **Transparência** local/remota
- **Transações** e **segurança** embutidas
- Mensagens **polimórficas**
- Co-existência com sistemas legados ("**IDL wrappers**")



32

32



33

No Lado CLIENTE

- **Stubs IDL**
 - ✦ Gerados pela compilação do arquivo IDL
 - ✦ Interfaces **estáticas** de objetos
 - ✦ **Marshalling** + **unmarshalling**
- **Invocação dinâmica**
 - ✦ Descoberta dinâmica de interfaces
 - ✦ Permite construir invocações em **tempo de execução**.
- **Interface ORB**
 - ✦ Funções úteis (e.g. `object_to_string()`)

34

No Lado SERVIDOR

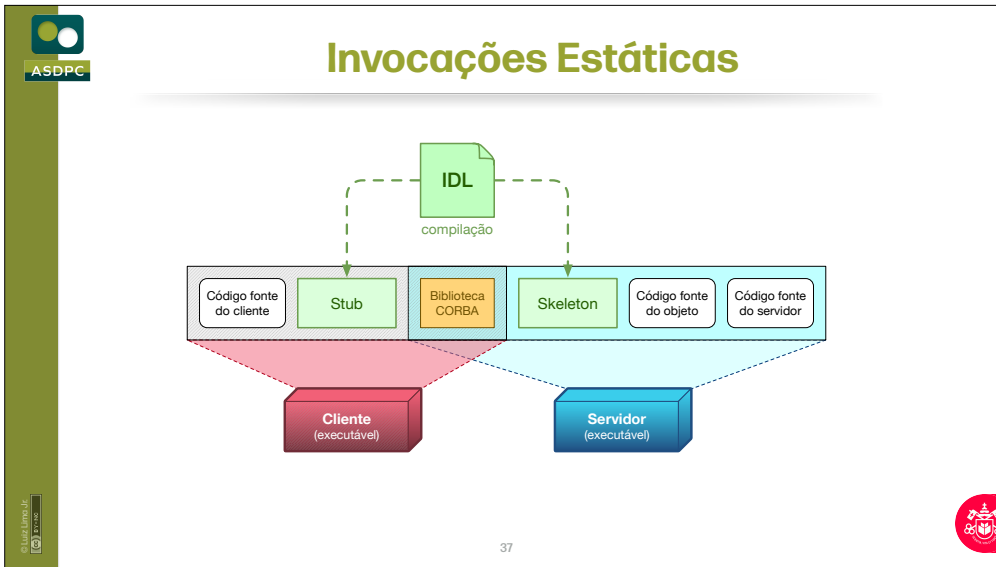
- **Skeletons IDL**
 - ✦ Gerados pela compilação do arquivo IDL
 - ✦ Interfaces **estáticas** para a ativação de serviços
 - ✦ Contrapartida dos *stubs* no lado cliente
- **Skeletons dinâmicos**
 - ✦ Contrapartida das interfaces de invocação dinâmica (DII)
- **Adaptador de objetos (POA)**
 - ✦ Gerencia ciclo de vida dos objetos servidores (**"servants"**)
 - ✦ Cria **referências** para os objetos servidores
 - ✦ Execução, passagem de parâmetros, multiplexação da requisição, etc.

35

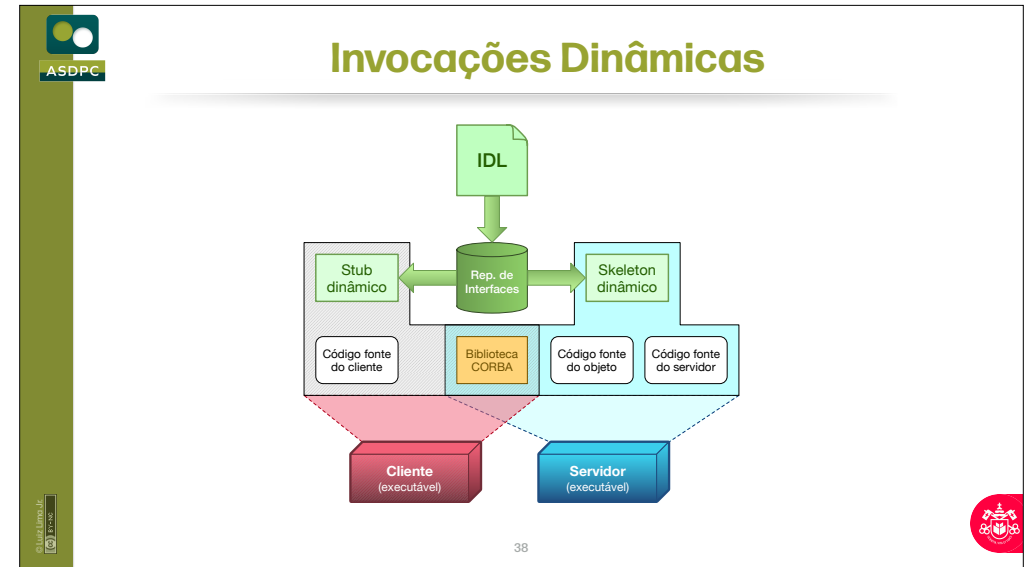
Repositórios

- **Repositório de interfaces (IR)**
 - ✦ Banco de dados de descrições de interfaces
 - ✦ Definições de interfaces: typecodes (metadado)
- **Repositório de implementações (ImR)**
 - ✦ Informações sobre objetos servidores (localização do executável, modos de ativação, ...)
 - ✦ Fornece facilidades para a **execução automática** de servidores.
- **Acessíveis por meio de interfaces IDL**

36



37



38

ASDPC

O Modelo de Interoperabilidade CORBA

Middleware OO – CORBA

39

39

ASDPC

Interoperabilidade CORBA

- Interação requer:
 - ✦ Referências de objetos
- **IOR = Interoperable Object Reference**
 - ✦ Referência de objeto compreendida por todas as implementações de ORBs.

40

40



PRÁTICA CORBA

Objetos Distribuídos

41



Implementação do Cliente

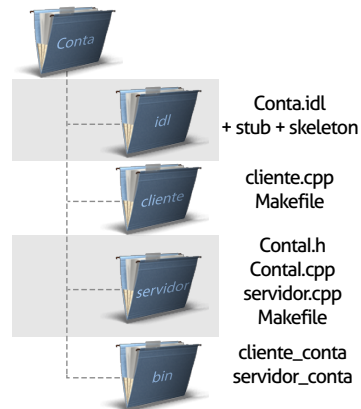
Middleware OO – CORBA

42



EXEMPLO: Cliente de Conta

- Criar **estrutura de diretórios** (recomendado):
- Gerar **stub**:
 - ✦ `tao_idl -Gstl Conta.idl`
- Criar `cliente.cpp`



43

43



Cliente CORBA: Passos Típicos

- Cliente
 - ✦ Declarar variável do **tipo da interface**:
 - `Interface ref;`
 - ✦ Obter **referência** para o objeto servidor:
 - `ref = obtém_referência (...);`
 - ✦ Usar a referência para **chamar os métodos**:
 - `res = ref.método (parâmetros);`

44

44

cliente.cpp

```
...
#include "ContaC.h" // stub
...
using namespace CORBA;
ORB_var orb = ORB::_nil();
int main(int argc, char * argv[])
{
    try {
        orb = ORB_init(argc, argv, "ORB");
```

45

45

cliente.cpp (2)

```
Object_ptr ref;
ref = orb->string_to_object(argv[1]);
Conta_var conta = Conta::_narrow(ref);

conta->deposito(123.45);
cout << "Saldo = " << conta->saldo() << endl;

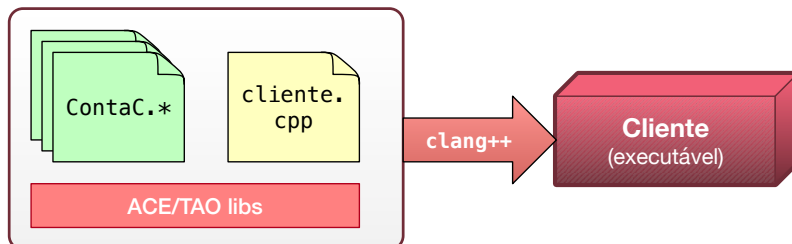
orb->destroy();
} catch (Exception& e) {
    ...
}
```

46

46

Implementação Cliente Conta (C++ ACE/TAO)

- Compilar arquivos do cliente

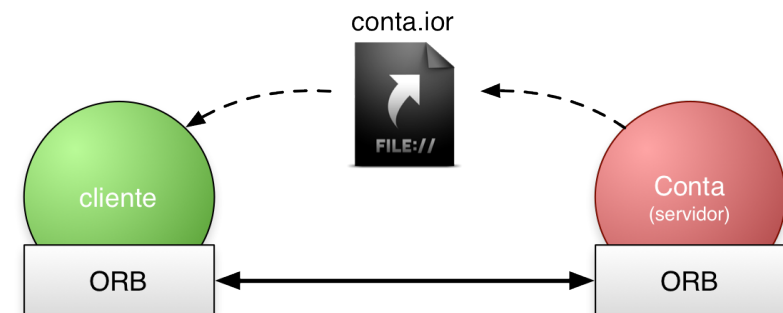


47

47

Implementação Cliente Conta (C++ ACE/TAO)

- Executar `conta` e depois, `cliente`



48

48



IDL: Visão Geral

Middleware OO – CORBA

49



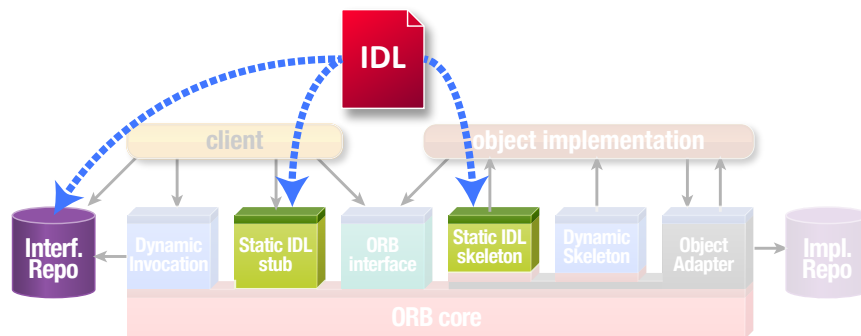
IDL: Interface Definition Language

- Linguagem puramente declarativa:
 - ✦ Não é possível programar em IDL
- Sintaxe similar a C++ ou Java
- Compilada para gerar:
 - ✦ Código de comunicação (stub e skeleton) na linguagem-alvo;
 - ✦ Metadados da interface para o repositório de interfaces.

50



IDL: Interface Definition Language



51

51



Principais Componentes da IDL

- Módulos (*module*)
- Interfaces (*interface*)
- Tipos de dados
- Constantes
- Atributos (*attribute*)
- Operações
- Exceções (*exception*)

52

52



Tipos de Dados IDL

- Tipos básicos
 - ✦ `short long float boolean ...`
- Tipos derivados
 - ✦ Usando a palavra-chave `typedef`
- Tipos estruturados
 - ✦ `enum struct union array`
- Tipos variáveis:
 - ✦ Vetores dinâmicos, `string`, `sequence`
- O tipo `any`

53

53

Tipos e Constantes Básicas

- Inteiros: `[unsigned] short long`
- Reais: `float double`
- 8 bits: `char octet boolean`
- Genéricos: `any`
- Exemplos (constantes):


```
const double Pi = 3.1415926;
const string Msg = "Mensagem";
const unsigned long Mask = (1<<5)|(1<<7);
```

54

54

Tipos Estruturados

```
enum CartaoCredito {Master, Visa, nenhum};
struct RegistroPessoa {
    string nome;
    short idade;
}
union Cliente switch (CartaoCredito) {
    case Master:
        string noCartao ;
    ...
}
```

55

55

Vetores, sequências e strings

```
// vetores
typedef long longVect[30];
typedef long longArray[2][10];

// sequências
typedef sequence <short> shortSeq;
typedef sequence <short,20> shortSeq20;

// strings de tamanho limitada
typedef string<1024> boundedString;
```

56

56

Métodos

```
<tipo_retorno> <nome> (<parâmetros>)
    [raises <exceções>]
    [context];
```

● Parâmetros de métodos podem ser:

- ✦ **in**: enviados ao servidor
- ✦ **out**: recebidos do servidor
- ✦ **inout**: ambas as direções

57

57

Atributos

```
attribute string nome;
readonly attribute short idade;
```

● Atributos:

- São declarados como variáveis.
- Métodos **get** e **set** são gerados (**readonly**: somente **get()**).
- TAO (C++):
 - `Float saldo();` → *get*
 - `void saldo(Float valor);` → *set*

58

58

Exemplo

```
module Utility {
    typedef long id_type;
    interface Unid {
        id_type GetID();
        void PutID(in id_type id);
    };
};
```

59

59

Semânticas de Invocação

● **Síncrona:**


- ✦ Chama método e espera por resultado
- ✦ Comportamento padrão

● **Assíncrona:**

- ✦ Chama método e continua execução.
- ✦ Chamadas de métodos não-confiáveis (sem garantia de sucesso).
- ✦ Só parâmetros **in** e retorno **void** são aceitos .
- ✦ Usa declaração “**oneway**”.

60

60



Mapeamento IDL

- Mapeamento:
 - ✦ Define como as estruturas declaradas em IDL são vistas na linguagem alvo.
 - ✦ Mapeamentos são padrões estabelecidos pela OMG.
- Porque o mapeamento é importante:
 - ✦ Construir parâmetros para a invocação de métodos
 - ✦ Interpretar valores de retorno e exceções
- Mapeamentos atuais:
 - ✦ C++, C, Java, Smalltalk, Ada, Cobol, Ruby, Python...

61

61




Implementação do Servidor

Middleware OO – CORBA

62

62



Visão Geral

- Servidor
 - ✦ Definir a **interface do objeto servidor** em IDL.
 - ✦ **Compilar** o arquivo **IDL** para gerar o código do *skeleton* na linguagem-alvo (Java, C++, C, ...).
 - ✦ Escrever o **código dos métodos** na linguagem-alvo.
 - ✦ Criar código de **inicialização e instanciação** de servants.

63

63



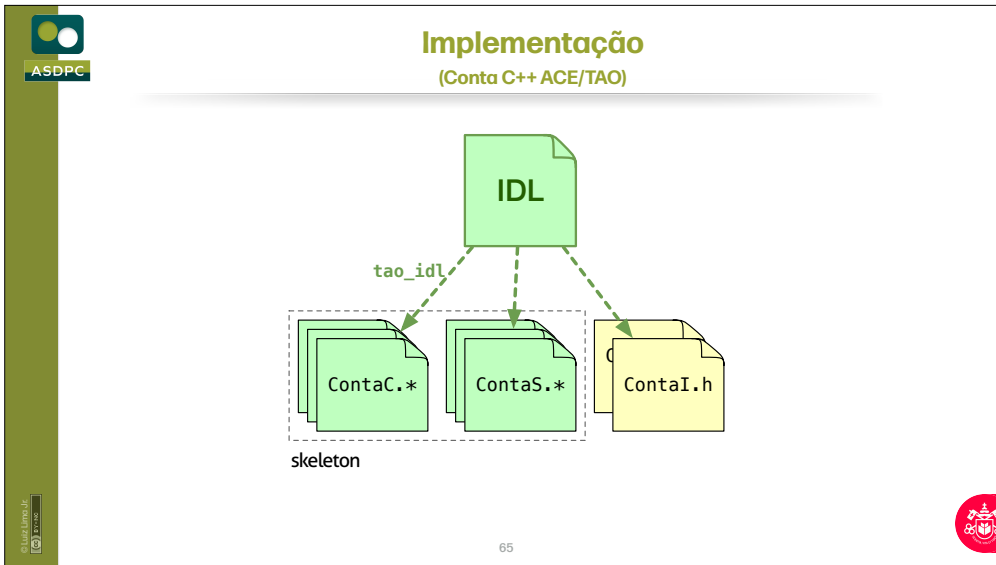
Implementação

(Conta C++ ACE/TAO)

- Criar arquivo IDL
- Compilar arquivo IDL para gerar:
 - ✦ Stub (`ContaC.*`)
 - ✦ Skeleton (`ContaS.*` + `ContaC.*`)
 - ✦ Classes de implementação (`ContaI.*`)
- TAO:
 - ✦ `tao_idl -Gstl -GI Conta.idl`

64

64



65

ASDPC

Implementação

(Conta C++ ACE/TAO)

- Customizar arquivos de implementação:
 - ✦ ContaI.h
 - ✦ ContaI.cpp

66

66

ASDPC

Implementação: Contal.cpp

(Conta C++ ACE/TAO)

```
void Conta_i::deposito(CORBA::Float valor)
{
    saldo_ += valor;
}
string Conta_i::id()
{
    return string("1234-5");
}
```

67

67

ASDPC

Implementação

(Conta C++ ACE/TAO)

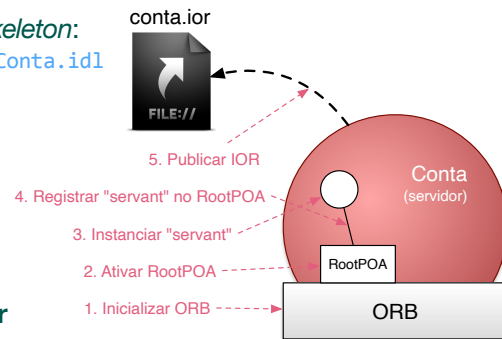
- Criar arquivo do servidor (**servidor.cpp**):
- Tipicamente:
 1. Inicializar **ORB**
 2. Ativar **gerente do POA** (**root POA**) = ativar POA
 3. Instanciar **servants**
 4. **Registrar servants** no POA (tornam-se "objetos CORBA")
 5. **Publicar** referências dos objetos CORBA (arquivo, NS, etc.)
 6. **Aguardar** requisições
 7. **Finalizar** (poa + orb)

68

68

Servidor: resumo da implementação

- Estrutura de **diretórios**
- Geração do *stub* + *skeleton*:
✦ `tao_idl -Gstl -GI Conta.idl`
- Copiar `ContaI.*`
→ `servidor/`
- Implementar **métodos remotos**
- Implementar **servidor**



69

69

servidor.cpp

```
#include "ContaI.h"
...
CORBA::ORB_var orb = CORBA::ORB::_nil();
...
int main(int argc, char* argv[])
{
    try { // exceções CORBA podem ocorrer
        orb = CORBA::ORB_init(argc, argv, "ORB");
```

70

70

servidor.cpp

```
// Ativa Root POA
CORBA::Object_ptr obj;
obj = orb->resolve_initial_references("RootPOA");
PortableServer::POA_var root_poa =
    PortableServer::POA::_narrow(obj);
PortableServer::POAManager_var poa_mgr =
    root_poa->the_POAManager();
poa_mgr->activate();
// Instancia servants
Conta_i conta_i;
// registra servant no POA + obtém ref
Conta_var conta = conta_i._this();
```

71

71

servidor.cpp

```
// Exporta IOR (salva em arquivo aqui)
CORBA::String_var ior = orb->object_to_string(conta.in());
ofstream arqior("conta.ior");
arqior << ior << endl;
arqior.close();
// Bloqueia aguardando requisições
orb->run();
// Cleanup
root_poa->destroy(true,true);
orb->destroy();
```

72

72

servidor.cpp

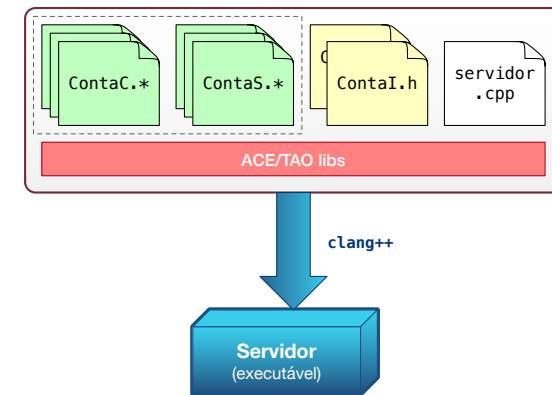
```
// fim do bloco try-catch + fim do programa
} catch (CORBA::Exception& e) {
    cerr << "Exceção CORBA: " << e << endl;
}
return 0;
}
```

73

73

Compilação do Servidor

(Conta C++ ACE/TAO)



74

74



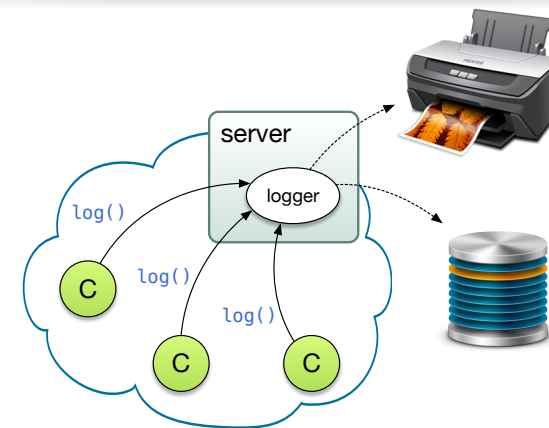
ASDPC

EXERCÍCIO: Logger

Middleware OO – CORBA

75

Logger



76

76

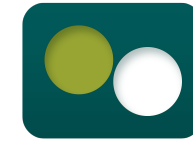
Logger

Operações:

- ✦ `log(prioridade, endereço, pid, hora, mensagem)`
 - *prioridade* = *DEBUG, WARNING, ERROR, CRITICAL*
 - *endereço* = "IP:PORTA" (string)
 - *pid* = inteiro curto sem sinal
 - *hora* = inteiro longo sem sinal
 - *mensagem* = string
- ✦ `verbose(V ou F)`
 - se *V*, servidor imprime dados quando receber (*log()*)
 - Se *F*, não imprime nada

77

77



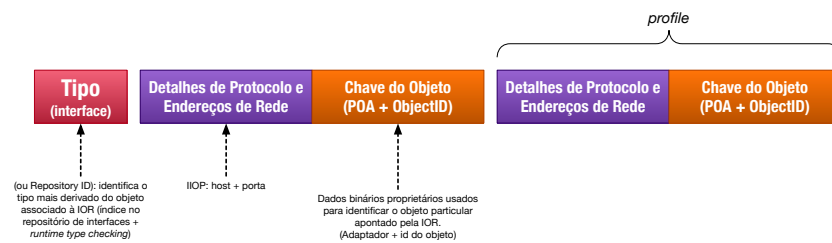
ASDPC

Referências Transientes

Middleware OO – CORBA

78

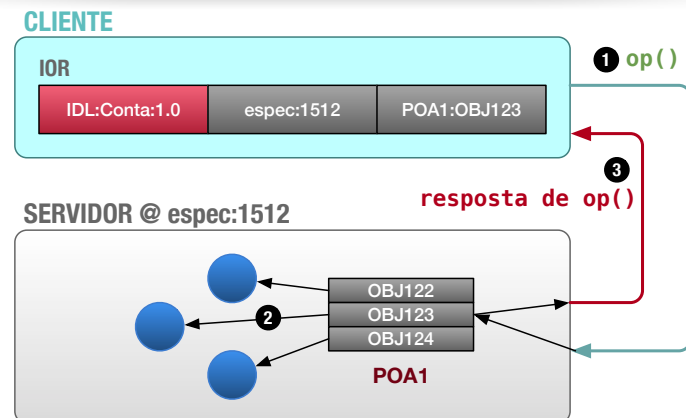
IOR



79

79

Binding de Referências Transientes



80

80

Binding de Referências Transientes

1. ORB cliente abre conexão com *host* + porta especificadas na IOR e envia requisição:
 - ✦ Tamanho da mensagem
 - ✦ Identificador da requisição (único)
 - ✦ Chave do objeto (nome do POA + objeto)
 - ✦ Operações + parâmetros

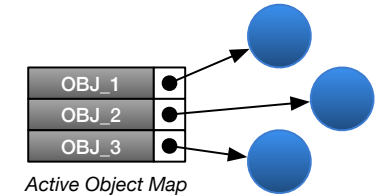
(o cliente aguarda resposta na conexão)

81

81

Binding de Referências Transientes

2. Servidor usa chave do objeto para localizar servant adequado:
 - **Active Object Map:**
 - Para cada adaptador
 - Mapa:
 - nome <=> ponteiro para servant
3. Servidor envia resposta ao cliente:
 - usa mesmo identificador da requisição



82

82

Cenários para Referências Transientes

- **Servidor está rodando** no host e porta especificados pela IOR:
 - ✦ Sucesso
- **Nenhum processo** escutando no endereço especificado
 - ✦ OBJECT_NOT_EXIST
- **Outro servidor** escutando no endereço especificado na IOR:
 - ✦ OBJECT_NOT_EXIST
- **Servidor original reiniciou** no mesmo endereço especificado na IOR:
 - ✦ OBJECT_NOT_EXIST (pois a referência é transiente)

83

83



ASDPC

Experimento: IORmerger

Middleware OO – CORBA

84



Referências Persistentes e o Repositório de Implementações (ImR)

Middleware OO – CORBA

85



Referências Persistentes e o Repositório de Implementações (ImR)

- IOR persistentes:
 - ✦ resistem ao *shutdown* do servidor
 - ✦ devem funcionar mesmo se o servidor for executado em um endereço diferente (ou se a porta for dinamicamente atribuída)
- Não é boa idéia atribuir *host* + porta do servidor em IOR persistentes.
- Ligações por meio do **Repositório de Implementações (ImR)**.

86



O Repositório de Implementações (Imr)

- Migração, escalabilidade, desempenho e tolerância a faltas:
 - ✦ Dependem do repositório de implementações
- Responsabilidades:
 - ✦ Manter **registro de serviços conhecidos**;
 - ✦ Registrar **endereços de servidores** em execução;
 - ✦ **Iniciar** servidores sob demanda.
- Rodam tipicamente em **endereços fixos**.
- Servidores que criam IORs persistentes devem conhecer o endereço do ImR.
- Servidores usando o **mesmo ImR**:
 - ✦ Mesmo “domínio de localização”

87

87



O Repositório de Implementações (Imr)

- Exemplo de **tabela** mantida pelo ImR:

Adaptador	Comando	Endereço
Banco	ssh espec "conta c1"	espec:1513
TempSensor	/usr/local/tsensor	
DBManager		host.com.br:3330

Ferramenta Administrativa (→ tao_imr)

88

88

O Repositório de Implementações (Imr)

- **Nome do Adaptador:**
 - ✦ Identifica adaptador de objetos no servidor
 - ✦ (servidores podem conter vários POAs)
- **Comando de inicialização:**
 - ✦ Como iniciar servidores sob demanda
 - ✦ (se nenhum comando registrado: servidor deve ser executado manualmente)
- **Endereço:**
 - ✦ Host + porta
 - ✦ (**vazio**: servidor não está rodando)

89



89

Ligação de Referências Persistentes

- **Referências persistentes** criadas pelo servidor contém:
 - ✦ Repository ID referente à interface
 - ✦ Host + porta do ImR
 - ✦ Chave do objeto:
 - Nome do adaptador de objetos
 - Nome do objeto: para ligá-lo a um servant
- **O cliente:**
 - ✦ Se comporta da mesma forma como para uma referência transitente:
 - Abre conexão com endereço na IOR
 - Envia requisição

90



90

Ligação de Referências Persistentes

- **Porém:**
 - ✦ Endereço na IOR = endereço do ImR (e não do servidor propriamente dito)
- **ImR:**
 - ✦ extrai o nome do adaptador (POA) da **chave do objeto**, usando-o como índice na tabela.

91



91

Cenários para Referências Persistentes

- **Servidor não está registrado:**
 - ✦ `OBJECT_NOT_EXIST`
- **Servidor está registrado para inicialização manual e não está rodando:**
 - ✦ `TRANSIENT`
- **Servidor está registrado para inicialização mas não está rodando:**
 - ✦ Inicia servidor e espera por mensagem com endereço
- **Servidor rodando:**
 - ✦ `LOCATION_FORWARD`

92



92

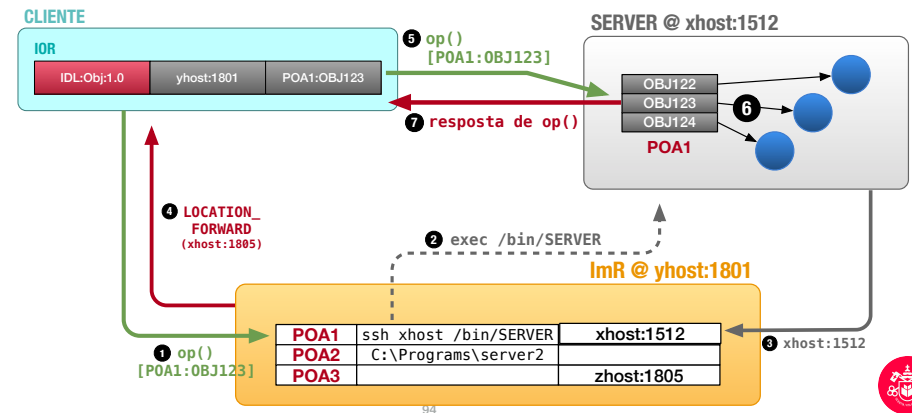
Exceção LOCATION_FORWARD

- Indica ao runtime do lado cliente que a **requisição deve ser enviada a um outro destinatário**.
- A exceção retorna novo endereço.
- Runtime* cliente envia nova requisição para a nova IOR.
- Transparente para o código da aplicação.

93

93

Ligação de Referências Persistentes



94

Ligações de Referências Persistentes

1. Cliente invoca `op()` enviando requisição ao ImR.
2. ImR inicia servidor.
3. Servidor informa ImR seu endereço atual.
4. ImR retorna endereço do servidor ao cliente (`LOCATION_FORWARD`).
5. Cliente reenvia `op()` para a nova localização.
6. Servidor usa chave do objeto para localizar servent.
7. Servidor retorna resultado de `op()`.

95

95

Protocolos de interoperabilidade

- **GIOP: General Inter-ORB Protocol**
 - ✦ Especifica mensagens e formatos básicos comuns entre ORBs.
 - ✦ **CDR: Common Data Representation**
 - ✦ Independente do protocolo de transporte.
- **IIOP: Internet Inter-ORB Protocol**
 - ✦ Especifica como GIOP funciona sobre TCP/IP.
 - ✦ Basicamente: definição do tipo de endereço na IOR (*host + porta*).
 - ✦ Protocolo padrão para conformidade CORBA.

96

96



CORBA services

Middleware OO – CORBA

97



CORBA services

- Coleções de **serviços de infraestrutura**:
 - ✦ Estendem/complementam a funcionalidade do ORB.
- Serviços comuns a **todas as aplicações**.
- Acessíveis como **objetos CORBA** (com interfaces definidas em IDL).
- Interações seguem padrões da OMG.

98



CORBA services

- | | |
|----------------------------|-----------------|
| ● Ciclo de vida | ● Consulta |
| ● Persistência | ● Licenciamento |
| ● Nomes | ● Propriedades |
| ● Eventos | ● Tempo |
| ● Controle de concorrência | ● Segurança |
| ● Transações | ● Trading |
| ● Relacionamentos | ● Coleções |
| ● Externalização | |

99

99



Serviço de Ciclo de Vida

- Operações para:
 - ✦ Criar
 - ✦ Copiar
 - ✦ Mover
 - ✦ Remover
- objetos.

100

100

Persistência

- Interface comum para:
 - ✦ Armazenar componentes
- Armazenamento persistente de objetos:
 - ✦ ODBMS
 - ✦ RDBMS
 - ✦ Arquivos simples

101

101

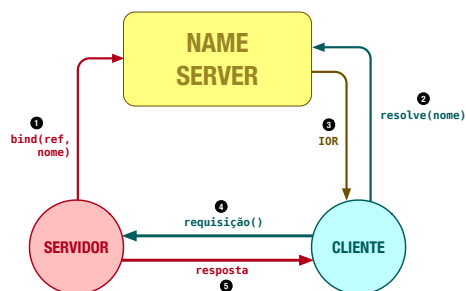
Serviço de Nomes

- Permite obter referências de objetos a partir de seus **nomes**.
- “**Páginas brancas**”
- Suporte para **contextos** e **nomes federados**.

102

102

Serviços de Nomes



103

103

Serviço de Eventos

- Componentes podem:
 - ✦ **Registrar** seu interesse em eventos específicos.
 - ✦ **Criar, gerenciar** e **acessar** “canais de eventos”.
- Canais de eventos:
 - ✦ Coletam e distribuem eventos entre objetos.
 - ✦ Objetos podem não saber da existência uns dos outros.

104

104

Serviço de Controle de Concorrência

- Gerenciamento de *locks* distribuídos e semáforos para transações e *threads*.
 - ✦ (semelhante Zookeeper)

105

105

Serviço de Transações

- Coordenação entre componentes usando transações planas ou aninhadas
- Propriedades **ACID**:
 - ✦ Atomicidade
 - ✦ Consistências
 - ✦ Isolamento
 - ✦ Durabilidade

106

106

Serviço de Relacionamento

- Provê meio de:
 - ✦ Criar associações dinâmicas (*links*) entre objetos.
 - ✦ Percorrer *links* entre componentes.
- Usados para qualquer tipo de associação entre objetos (e.g., grupos).

107

107

Serviço de Externalização

- Provê meio para:
 - ✦ enviar/receber dados entre objetos usando mecanismos de *streams*.
- Serialização de objetos.

108

108

Serviço de Consulta

- Operações para consultar objetos.
- Superconjunto do SQL.

109

109

Serviço de Licenciamento

- Operações para:
 - ✦ Medir o uso do objeto.
 - ✦ Assegurar compensação justa por seu uso.
- Dá suporte a qualquer modelo de controle de uso.
- Pode ser usado em qualquer ponto do ciclo de vida de um objeto.
- Suporte a bilhetagem por sessão, por nó, por criação de instância e por site.

110

110

Serviço de Temporização

- Interface para sincronização de tempo.
- Operações para:
 - ✦ Definir e gerenciar eventos dependentes de tempo.

111

111

Serviço de Negociação (*Trading*)

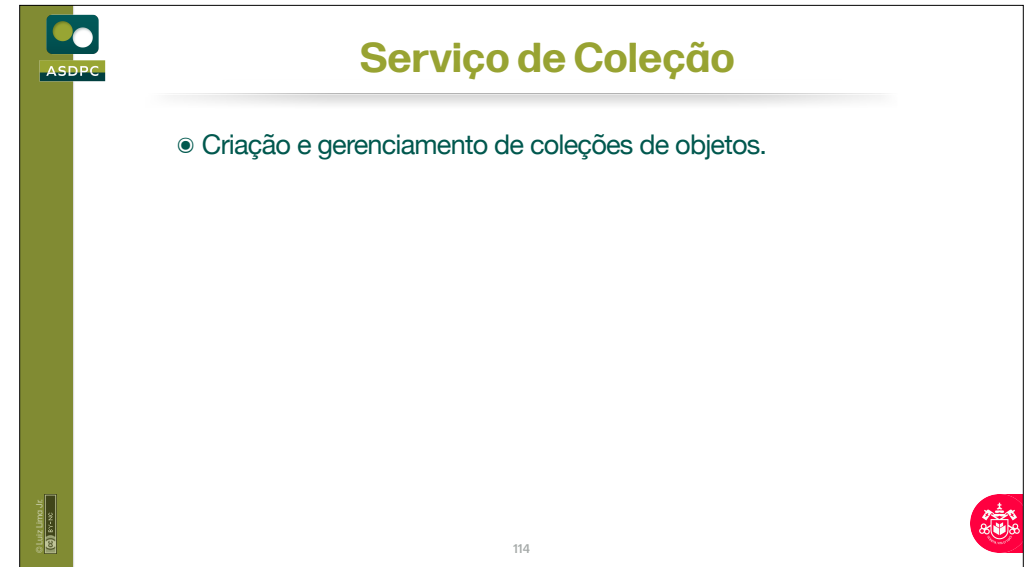
- “Páginas amarelas”
- Objetos servidores:
 - ✦ publicam seus serviços
- Clientes:
 - ✦ encontram servidores apropriados por meio das propriedades do serviço.

112

112



113



114

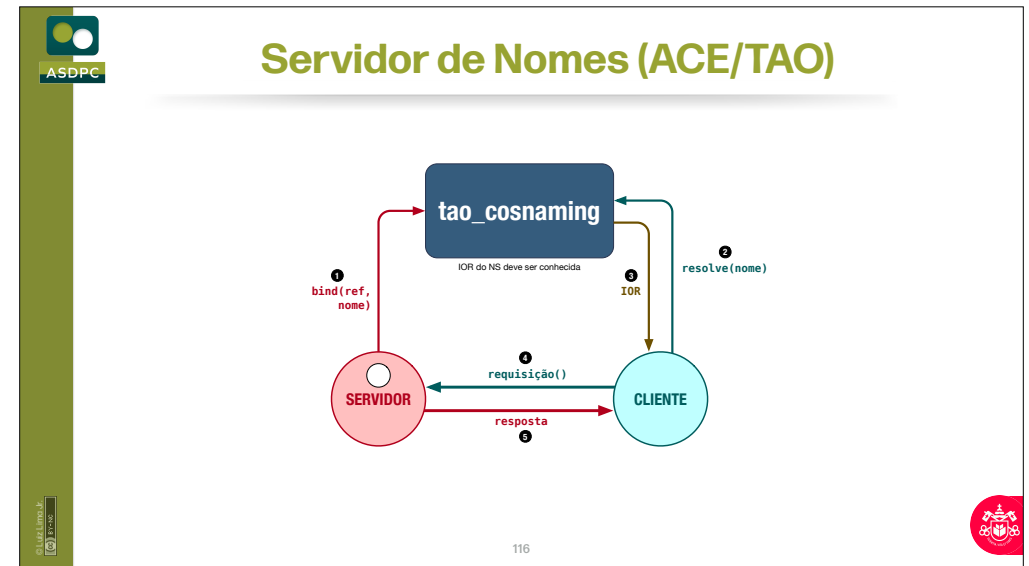


Prática: Servidor de Nomes

Middleware OO – CORBA

115

115



116

Publicando a Referência do Objeto Distribuído

1. Obter referência para NameService:

```
obj = orb->resolve_initial_references("NameService");
CosNaming::NamingContext_var ns =
    CosNaming::NamingContext::_narrow(obj);
```

2. Compor nome a ser publicado:

```
CosNaming::Name name(1);
name.length(1);
name[0].id = CORBA::string_dup("conta1");
```

3. Publicar referência:

```
ns->bind(name, conta.in());
```

117

117

Observações

● Name = sequence<string>:

- ✦ Como caminho de um diretório:
 - *contexto1/contexto2/conta1*

● bind:

- ✦ Produz **exceção** se nome já existe no NS.
- ✦ Utilizar **rebind** para sobrescrever valores sem que haja exceções.

118

118

Cliente

1. Obter referência para **NameService**:

- ✦ como no servidor

2. Compor nome do serviço procurado:

- ✦ como no servidor

3. “Resolver” o nome no NS:

```
obj = ns->resolve(name);
Conta_var conta = Conta::_narrow(obj);
```

119

119

Execução (Servidor de Nomes)

● Executando o NS:

- ✦ `tao_cosnaming [-o ns.ior] [-m 1]`
 - *-o: publica IOR do NS no arquivo*
 - *-m: permite multicasting para descobrir IOR do NS*

● Cliente/servidor (se “ns.ior” disponível):

- ✦ `-ORBInitRef NameService=file://ns.ior`

● Nem sempre é possível compartilhar arquivo IOR:

- ✦ `-ORBListenEndpoints iiop://localhost:1234`

● Clientes/servidores:

- ✦ `-ORBInitRef NameService=corbaloc:iiop:localhost:1234/NameService`

120

120



CORBA Facilities

Middleware OO – CORBA

121



CORBA facilities

- "Facilitar" o desenvolvimento de aplicações.
- Infraestruturas **horizontais**:
 - ✦ estendem a funcionalidade de CORBA sem modificar seu núcleo.
- Infraestruturas verticais (ou "**domínios**"):
 - ✦ usados em contextos específicos de aplicações;
 - ✦ desenvolvidos por Grupos de Interesses Especiais da OMG.

122



122



Infraestruturas Horizontais

- Interface com o usuário:
 - ✦ semelhante a "OpenDoc";
 - ✦ gerenciamento de alocação de espaço de tela para objetos.
- Gerenciamento de informações:
 - ✦ armazenamento de documentos;
 - ✦ troca de dados para documentos compostos.
- Gerenciamento de tarefas:
 - ✦ *Workflow*
 - ✦ *Scripting*
 - ✦ *Agentes...*

123

123



Infraestruturas Horizontais

- Gerenciamento de sistemas (extensão do serviço de notificações):
 - ✦ *Logging*
 - ✦ *Segurança*
 - ✦ *Balanceamento de carga*

124

124



Infraestruturas Verticais

- CORBAmed
- Telecomunicações
- Agentes móveis
- Contabilidade
- *Firewalls*
- ...

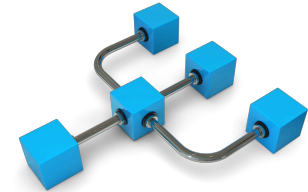
125



125

Mais informações

OMG-CORBA Home Page: www.corba.org
 OMG: www.omg.org
 ORB Zone: www.orbzone.org
 Free ORBs: www.corba.org/corbadownloads.htm
 Vinosky, "Advanced CORBA C++ Programming"
 ACE/TAO docs: www.cs.wustl.edu/~schmidt/TAO.html
 Remedy IT: TAO Programmers Guide (TPG):
www.theaceorb.nl/en/ace-tao-ciao-dance-prod/tpg



126



126