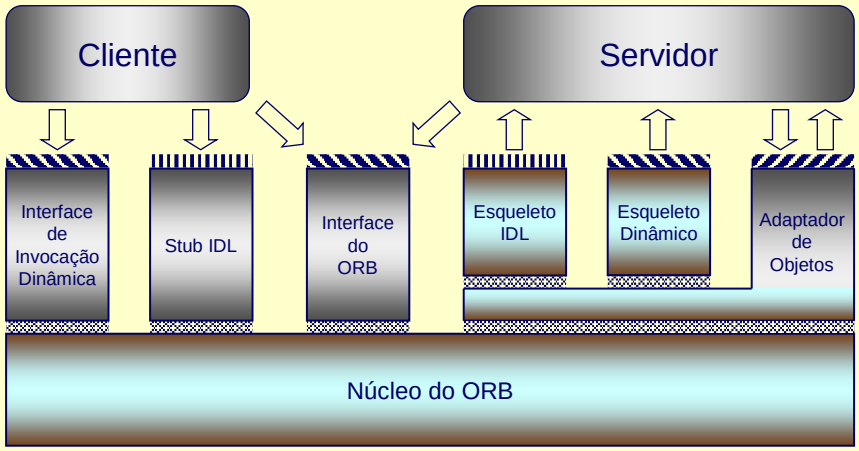
IORs, GIOP e IIOP

Partes da Especificação CORBA

- O núcleo (*core*) do ORB
- A linguagem de definição de interfaces (IDL)
- O repositório de interfaces
- Mapeamentos de IDL para linguagens de programação
- *Stubs* e esqueletos estáticos
- Interfaces de invocação dinâmica (DII) e de esqueleto dinâmico (DSI)
- Adaptadores de objetos (*Object Adapters*)
- O repositório de implementações
- Protocolos (GIOP e IIOP)

Componentes de CORBA



Independe de ORB

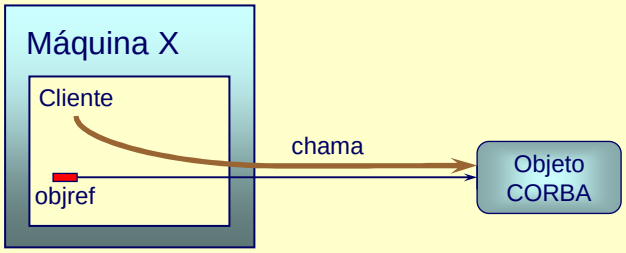
Depende das definições IDL

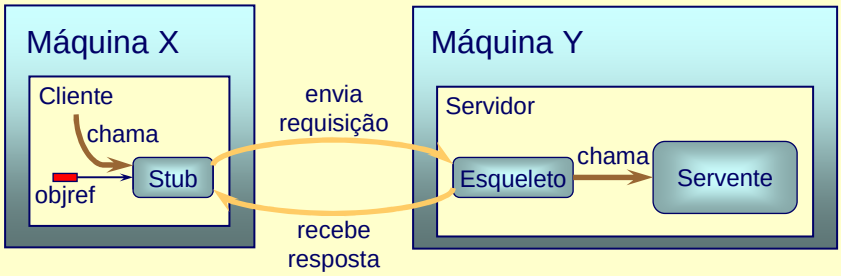
Depende do adaptador

Interface proprietária

Chamada Remota de Métodos

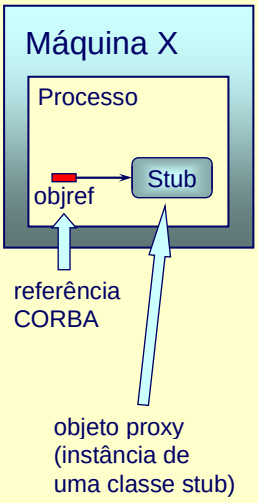
- O que parece acontecer
- O que acontece na verdade





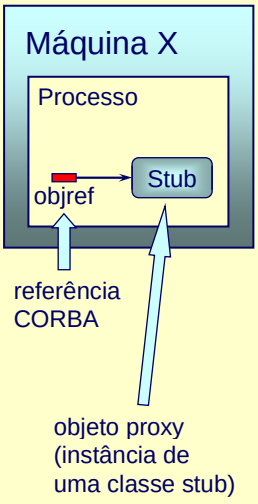
Referências para Objetos CORBA

- Podem estar no "formato interno", durante a execução de um programa que usa objetos CORBA
- Uma referência no "formato interno" é uma "apontador" da linguagem de programação utilizada
  - Ponteiro C ou C++, referência Java, ...
  - Numa linguagem orientada a objetos (C++ ou Java) a referência aponta para uma instância de uma classe stub
  - Essa instância é um objeto proxy que tem o papel de "representante local" do objeto CORBA remoto



Instanciação de Stubs

- Quem cria o objeto proxy?
  - O ORB
- A instanciação acontece quando a referência CORBA "aparece no processo"
- Como uma referência pode "aparecer num processo"?...



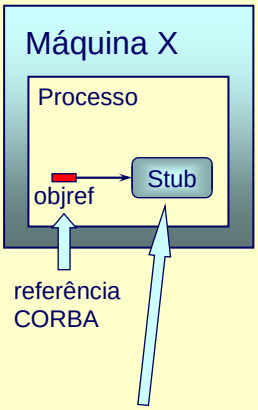
A referência pode aparecer

- No lado servidor, como resultado de uma chamada que cria uma nova referência

```
org.omg.CORBA.Object quoter =
      quoterServant._this_object(orb);
```

- A criação de referências é uma responsabilidade do adaptador de objetos

```
interface POA {
    Object create_reference(
           in CORBA::RepositoryId intf
        ) raises(WrongPolicy);
    Object create_reference_with_id(
           in ObjectId oid,
           in CORBA::RepositoryId intf
        ) raises(WrongPolicy);
    ...
};
```



A referência também pode aparecer

- Via conversão de string em referência

```
obj = orb.string_to_object(str);
```

- Por estreitamento (narrow) de outra referencia já presente no processo

```
quoter = QuoterHelper.narrow(obj);
```

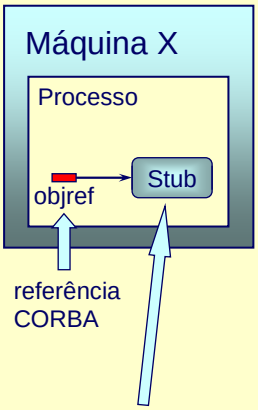
- Como resultado de uma operação invocada sobre um objeto remoto

```
obj = nc.resolve_str("srv/quoter");
```

(aqui **nc** é uma referência para um **NamingContextExt**)

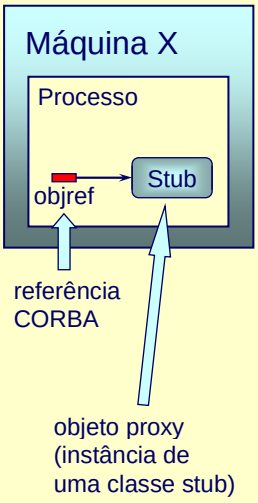
- Como resultado de resolve_initial_references

```
obj = orb.resolve_initial_references(
                              "NameService");
```



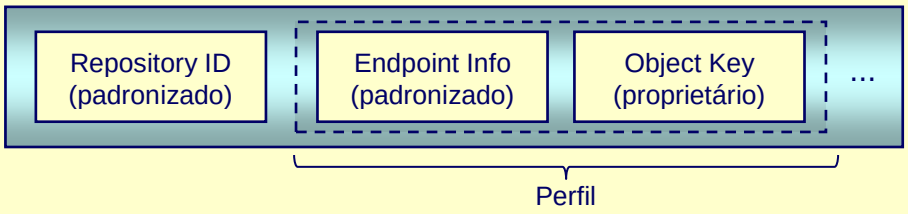
Referências transitam entre processos

- Num formato "para viagem"
  - A especificação CORBA padroniza esse "formato externo"
  - Interoperabilidade entre ORBs de diferentes fornecedores
- Formato externo: Interoperable Object Reference (IOR)
  - Uma IOR entre em cena quando
    - uma referência CORBA é passada como parâmetro ou como resultado de uma operação
    - uma referência CORBA é convertida em string



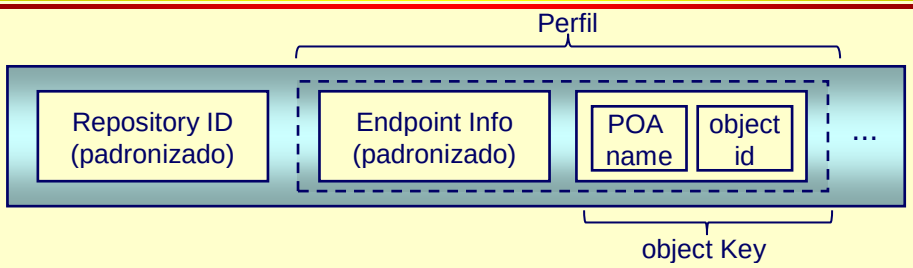
Anatomia de uma IOR

CORBA Interoperable Object Reference (IOR):



- Um repository ID + uma seqüência de perfis
  - Repository ID: identifica a interface do objeto
    - É o ID de uma entrada do repositório de interfaces
  - Endpoint Info: host + port (no caso de um perfil IIOP)
  - Object Key: nome do Object Adapter + object ID

Anatomia de uma IOR



- A object key identifica um dentre os múltiplos objetos CORBA implementados pelo servidor
  - O nome do POA identifica um dos adaptadores de objetos que podem existir no servidor
  - O object id especifica um dos objetos registrados com o POA
    - Pode ser escolhido pela aplicação servidora
      - Exemplo: chave para uma tabela num BD relacional)

Visão Geral do GIOP

- General Inter-ORB Protocol (GIOP): protocolo “abstrato” (família de protocolos)
- O Internet Inter-ORB Protocol (IIOP) é uma realização concreta do GIOP
- Tres elementos principais:
  - Hipóteses sobre a camada de transporte
  - Common Data Representation (CDR)
  - Mensagens e seus formatos

Hipóteses Sobre a Camada de Transporte

- Orientada a conexões
- Conexões são *full-duplex*
- Conexões são simétricas
  - Qualquer lado pode fechar uma conexão
- O transporte é confiável
  - Dados são entregues não mais que uma vez, na ordem que foram enviados
  - Se não forem entregues o remetente é informado do erro
- O transporte provê uma abstração de *byte-stream*
- O transporte indica se houve uma quebra inesperada de conexão

Common Data Representation (CDR)

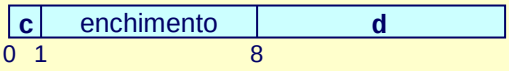
- Formato de transmissão de dados
  - Define o *layout* binário de cada dado IDL
- Características:
  - Suporta tanto a representação *big-endian* como a representação *little-endian*
    - "Receiver makes it right": o transmissor manda na sua representação e o receptor faz o *byte-swapping* se necessário
  - Alinhamento natural de tipos primitivos
    - **`short`** a 2 cada bytes, **`long`** a cada 4 bytes, etc.
  - Os tipos de dados não se auto-identificam
    - Transmissor e receptor precisam estar de acordo quanto aos tipos dos dados transmitidos

Alinhamento de Tipos Primitivos

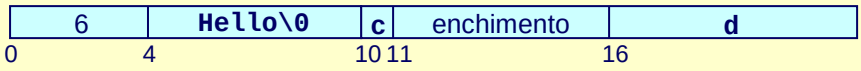
Alinhamento	Tipo IDL primitivo
1	**`char`**, **`octet`**, **`boolean`**
2	**`short`**, **`unsigned short`**
4	**`long`**, **`unsigned long`**, **`float`**, tipos enumerados
8	**`long long`**, **`unsigned long long`**, **`double`**, **`long double`**
1, 2 ou 4	**`wchar`** (o alinhamento depende do *codeset*)

Alinhamento de Tipos Não-Primitivos

```
struct CD {
    char c;
    double d;
};
```



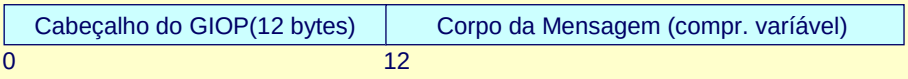

```
interface Foo {
    void op(in string s, in CD cd);
};
```



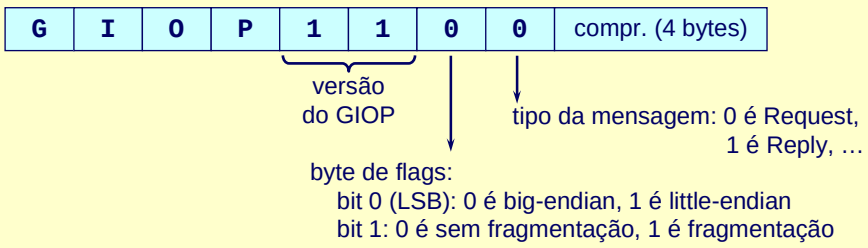
Mensagens do GIOP

- Request (cliente → servidor)
- Reply (servidor → cliente)
  - pode ter status LOCATION_FORWARD
- CancelRequest (cliente → servidor)
- LocateRequest (cliente → servidor)
- LocateReply (servidor → cliente)
- CloseConnection (até GIOP 1.1: servidor → cliente)
- MessageError (cliente ou servidor podem enviar)
- Fragment (cliente ou servidor podem enviar)

Estrutura Básica de uma Mensagem



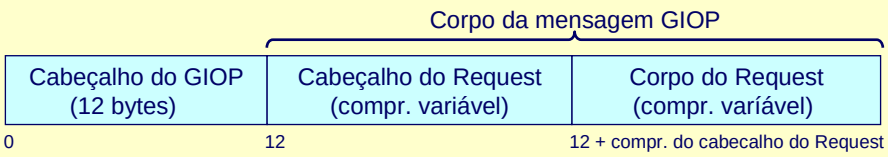
- Cabeçalho de uma mensagem do GIOP 1.1 indicando mensagem Request, com ordenação de bytes big-endian e sem fragmentação:



Cabeçalho das Mensagens do GIOP

```
module GIOP {      // PIDL
    struct Version {
        octet major;
        octet minor;
    };
    enum MsgType_1_1 {
        Request, Reply, CancelRequest, LocateRequest,
        LocateReply, CloseConnection, MessageError, Fragment
    };
    struct MessageHeader_1_1 {
        char          magic[4];     // the string "GIOP"
        Version       GIOP_version;
        octet         flags;
        octet         message_type;
        unsigned long message_size;
    };
    ...
};
```


Formato da Mensagem **Request**




```
module GIOP {      // PIDL
    ...
    struct RequestHeader_1_1 {
        IOP:ServiceContextList service_context;
        unsigned long          request_id;
        boolean                response_expected;
        octet                  reserved[3];
        sequence<octet>        object_key;
        string                 operation;
        Principal              requesting_principal.
    };
    ...
};
```

Campos do **Request**

- **service_context**
  - Dados transmitidos “de carona” com o **Request** e de forma transparente para o cliente
  - Usado para propagar informações necessárias para certos serviços
    - Transaction context (Transaction Service), security context (Security Service)
- **request_id**
  - Número gerado pelo cliente para associar a requisição com sua resposta
- **response_expected**
  - Indica se a chamada é **oneway** ou não
- **reserved**
  - Campo reservado para uso futuro

Campos do **Request** (cont.)

- **object_key**
  - A object key da IOR usada para mandar a requisição
  - Identifica o objeto alvo no servidor destino
- **operation**
  - String com o nome da operação sendo chamada
  - Para acesso a atributos, é **_get_*attr_name*** ou **_set_*attr_name***
  - Para operações na pseudo-interface **Object, pode ser**:
    - **_interface** (operação **get_interface**)
    - **_is_a** (operação **is_a**)
    - **_non_existent** (operação **non_existent**)
  - As outras operações de **Object** (**duplicate**, **release**, **is_nil**, **is_equivalent** e **hash**) são tratadas localmente pelo ORB do cliente
- **requesting_principal**
  - Identificação do cliente (*deprecated*)

Corpo do **Request**

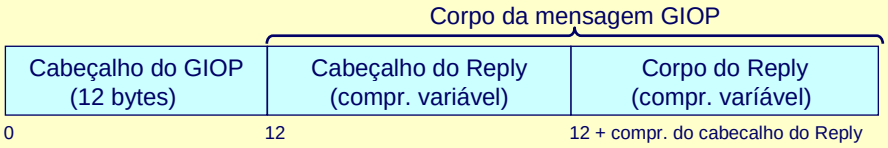
- Contém os parâmetros **in** e **inout** da operação
- Os parâmetros são enviados numa estrutura
  - Exemplo:

```
interface Foo {
    void op(
        in string   param1,
        out double  param2,
        inout octet param3
    );
};
```

- Esta estrutura vai no corpo de um **Request** para a operação **op**:

```
struct params {
    string param1;
    octet  param3;
};
```


Formato da Mensagem **Reply**




```
module GIOP {      // PIDL
    ...
    enum ReplyStatusType {
        NO_EXCEPTION, USER_EXCEPTION,
        SYSTEM_EXCEPTION, LOCATION_FORWARD
    };
    struct ReplyHeader {
        IOP:ServiceContextList service_context;
        unsigned long          request_id;
        ReplyStatusType        reply_status;
    };
    ...
};
```

Corpo do **`Reply`**

- Seu conteúdo depende do **`reply_status`**
  - **`NO_EXCEPTION`**
    - Uma estrutura com o valor de retorno da operação e os parâmetros **`out`** e **`inout`**
  - **`USER_EXCEPTION`** ou **`SYSTEM_EXCEPTION`**
    - O tipo (repository id) da exceção, seguido pelos campos da exceção
  - **`LOCATION_FORWARD`**
    - Uma *object reference* que o ORB do cliente deve usar para tentar novamente a operação

As Outras Mensagens do GIOP

- CancelRequest (cliente → servidor)
- LocateRequest (cliente → servidor)
- LocateReply (servidor → cliente)
- CloseConnection (até GIOP 1.1: servidor → cliente)
- MessageError (cliente ou servidor podem enviar)
- Fragment (cliente ou servidor podem enviar)

Gerenciamento de Conexões

- O modelo de interações entre clientes e servidores CORBA é *connectionless*
  - Uma aplicação cliente pode chamar uma operação quando quiser
  - Nem a aplicação cliente nem a servidora explícitamente abrem ou fecham conexões de transporte
  - O ambiente *run-time* do ORB se encarrega de fazer isso
- CORBA deixa a estratégia de gerenciamento de conexões a critério do ORB
- No lado do cliente, o *run-time* do ORB pode:
  - Abrir e fechar uma conexão para cada requisição
    - Ruim!
  - Multiplexar todas as requisições para objetos de um servidor através de uma única conexão de transporte
    - Todo ORB razoável faz isto

Gerenciamento de Conexões (cont.)

- No lado do servidor, o *run-time* do ORB pode:
  - Parar de aceitar conexões quando o servidor atingir um certo limite no número de conexões abertas
  - Usar a mensagem **`CloseConnection`** para fechar conexões ociosas, de modo a poder aceitar novas conexões
- O ORB do cliente deve estar preparado para receber mensagens **`CloseConnection`**
  - Se precisar chamar o servidor de novo, abre outra conexão
  - Neste caso nenhuma exceção sobe para a aplicação cliente
  - Uma queda de conexão que não for precedida por mensagem **`CloseConnection`** gera a exceção **`COMM_FAILURE`** para a aplicação cliente