O *Object Request Broker* (ORB)

- Via de comunicação entre objetos (*object bus*), na arquitetura do OMG
- Definido pela especificação CORBA (*Common Object Request Broker Architecture*)
  - 1991: CORBA 1.0
  - 1995: CORBA 2.0
  - revisão atual: CORBA 2.2
  - próxima grande revisão: CORBA 3.0 (1999?)

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)

O Desenvolvimento de um Sistema

- Etapas Gerais:
  - Defina as interfaces dos objetos e expresse-as em OMG IDL (*Interface Definition Language*)
  - Gere *stubs* e esqueletos
  - Implemente os objetos e o programa servidor
  - Implemente aplicações clientes

OMG IDL

- Linguagem declarativa usada para especificar interfaces de objetos
  - Sintaxe baseada em C++
  - Suporta herança de interfaces
  - Todas as interfaces de objetos são derivadas de **`CORBA::Object`**
  - CORBA é especificada em IDL
    - de modo a ser independente de linguagem de programação

IDL: Tipos Básicos

- **`short`** (**`signed`** e **`unsigned`**): inteiros de 16 bits
- **`long`** (**`signed`** e **`unsigned`**): inteiros de 32 bits
- **`long long`** (**`signed`** e **`unsigned`**): 64 bits
- **`float`**, **`double`** , **`long double`**: reais em ponto flutuante, conforme o padrão IEEE 754
- **`char`** e **`wchar`**: caracteres de 8 bits e de 16 bits
- **`boolean`**: **`FALSE`** e **`TRUE`**
- **`octet`**: byte de 8 bits (usado para "*raw data*")
- **`any`**: pode conter qualquer tipo IDL (tanto um tipo básico como um tipo definido pelo usuário)

IDL: Tipos Definidos pelo Usuário

- Estruturas:

```
struct Item { string s; short n; };
```

- Enumerações:

```
enum TipoInfo {texto, valor};
```

- Uniões:

```
union Info switch(TipoInfo) {
    case texto: string s;
    case valor: double d;
    default:    long l;
};
```

- Definição de novo nome para um tipo:

```
typedef string Nome;
```

IDL: Tipos Parametrizados

- Seqüências (limitadas ou ilimitadas) :

```
typedef sequence<long,32> LongSeq32;
typedef sequence<long> LongSeq;
```

- Strings e wide strings (limitadas ou ilimitadas):

```
typedef string<40> NomeLimitado;
typedef string NomeIlimitado;
typedef wstring<40> Ident;
typedef wstring Texto;
```

- Números decimais em ponto fixo:

```
typedef Troco fixed<3,2>;
typedef Moeda fixed<12,2>;
```


IDL: Vetores e Matrizes

- Seqüências e strings: comprimento variável
- Vetores e matrizes multidimensionais: tamanho fixo

```
const short Lim = 200;
typedef long LongVec[Lim];

typedef short NLins = 100;
typedef short NCols = 100;
typedef double Matriz[NLins][NCols];

typedef double Mat3D[40][40][40];
```

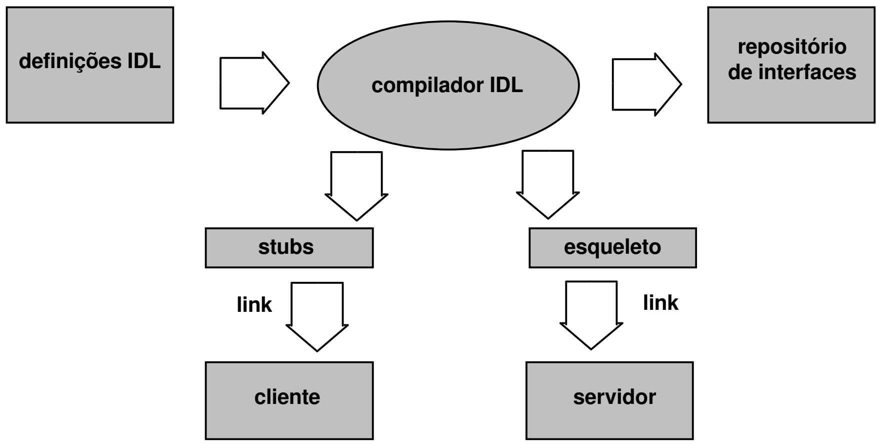
Exemplo de Interface IDL

```
// IDL
module Stock {
    exception UnknownStock {
        string name;
    };
    interface Quoter {
        string name();
        long value(in string stock)
            raises(UnknownStock);
    };
};
```


Mapeamentos de IDL

- Definem como IDL é traduzida para as diferentes linguagens de programação
- O OMG padronizou mapeamentos de IDL para C, C++, Smalltalk, COBOL, Ada e Java
  - Há mapeamentos para outras linguagens, mas ainda são proprietários (não padronizados)

Gerando Stubs e Esqueletos



A Interface do ORB

```
// pseudo IDL
module CORBA {
    interface Object; // forward decl
    ...
    interface ORB {
        string object_to_string(in Object obj);
        Object string_to_object(in string str);
        ...
    };
    ...
};
```

Inicialização do ORB

```
// pseudo IDL
module CORBA {
    ...
    typedef string ORBid;
    typedef sequence<string> arg_list;
    ORB ORB_init(inout arg_list argv,
                 in ORBid orb_identifier);
    ...
};
```


A Interface Object

```
// pseudo IDL
module CORBA {
    interface InterfaceDef;  // forward decl
    ...
    interface Object {
        InterfaceDef get_interface();
        boolean is_nil();
        Object duplicate();
        void release();
        ...
    };
    ...
};
```

Operações Sobre *Object References* (cont.)

```
// pseudo IDL
module CORBA {
    ...
    interface Object {
        ...
        boolean is_a(in string logical_typeid);
        boolean non_existent();
        boolean is_equivalent(in Object other);
        unsigned long hash(in unsigned long max);
        ...
    };
    ...
};
```


Mapeamento de IDL para C++

- Cada **`module`** é mapeado num **`namespace`** (ou **`class`**)
- Cada **`interface`** é mapeada numa **`class`**
- Cada operação é mapeada num método com os parametros adequados
- Cada *read/write* **`attribute`** é mapeado num par de métodos *get/set*
- Cada **`readonly attribute`** é mapeado num método *get*

Mapeamento de IDL para C++ (cont.)

- Referências a objetos tipo **`interface`** (*object references*) podem ser mapeadas para **`_ptr`**s ou para **`_var`**s
  - tanto **`_ptr`**s como **`_var`**s podem ser usados para invocar operações sobre objetos remotos
  - **`_ptr`**s tem semântica de ponteiros
  - **`_var`**s são mais “espertos” (fazem gerenciamento de memória)

Exemplo de Cliente C++

```
int main(int argc, char** argv)
{
    using namespace CORBA;
    using namespace Stock;
    string objstr; // classe string ANSI/ISO
    ifstream is("quoter_string_objref");
    is >> objstr;
    ORB_ptr orbref = ORB_init(argc, argv);
    Object_ptr objref =
        orbref->string_to_object(objstr.c_str());
```

Exemplo de Cliente C++ (cont.)

```
    Quoter_ptr quoter = Quoter::_narrow(objref);
    char* name = quoter->name();
    Long value = quoter->value(argv[1]);
    cout << ”Cotação de ” << argv[1]
         << ”, fornecida por ” << name << ”: ”
         << value << endl;
    string_free(name);
    release(quoter);
    release(objref);
    release(orbref);
    return 0;
}
```


Tratamento de Erros

- Exceções IDL são normalmente mapeadas para exceções C++
  - Caso o compilador C++ ainda não suporte exceções, um parâmetro especial de tipo **`CORBA::Environment`** é passado
  - Esse parâmetro deve ser testado antes de se usar quaisquer valores retornados por uma operação

Uso de Exceções no Cliente

```
try {
    char* name = quoter->name();
    Long value = quoter->value(argv[1]);
    cout << ”Cotação de ” << argv[1]
         << ”, fornecida por ” << name << ”: ”
         << value << endl;
} catch(UnknownStock& us) {
    cerr << ”Não disponível cotação de ”
         << us.name << endl;
} catch(SystemException&) {
    cerr << ”Exceção de sistema!” << endl;
}
```


`_ptr`s e `_var`s

- Dois tipos de *object references* em C++
  - dada uma interface **`A`**, o compilador IDL gera definições de dois tipos C++, **`A_ptr`** e **`A_var`**
  - **`A_ptr`** requer gerenciamento explicito
  - **`A_var`** é gerenciado automaticamente

Usando tipos `_var`

```
ORB_var orbref = ORB_init(argc, argv);
Object_var objref =
    orbref->string_to_object(objstr.c_str());
Quoter_var quoter = Quoter::_narrow(objref);
String_var name = quoter->name();
Long value = quoter->value(argv[1]);

// não é mais preciso chamar release para
// quoter, objref ou orbref, nem chamar
// string_free para name
```