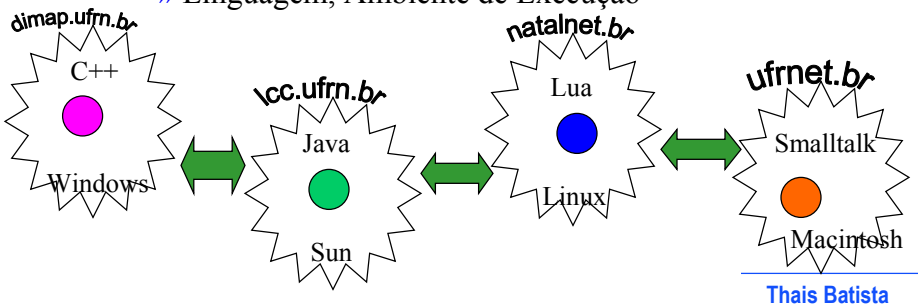
Plataformas de Distribuição de Objetos

- Denominações Equivalentes:
  - Modelos de Componentes
  - Modelos de Integração de Objetos
- Motivação:
  - Reuso de objetos
  - Redução do Tempo e do Custo de Desenvolvimento de Software

Objetivo

- TRANSPARÊNCIAS:
  - Distribuição
  - Heterogeneidade
    - » Linguagem, Ambiente de Execução



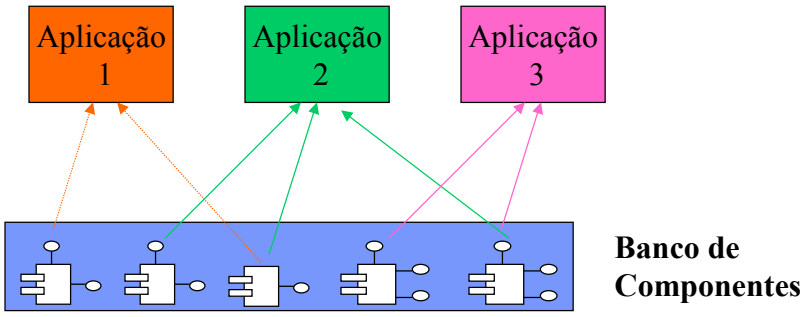
Plataformas de Distribuição de Objetos

- Exemplos:
  - CORBA (Common Object Request Broker Architecture): OMG
  - COM (Component Object Model) e .NET: MicroSoft
  - JavaBeans & Enterprise JavaBeans: Sun

Componentes

- Entidades de Software compiladas separadamente que interagem com outras entidades apenas através das suas *interfaces*.
- Componentes visam *Composição*
- Composição é a junção de componentes pré-fabricados

Desenvolvimento Baseado em Componentes



Interface

- Conjunto de operações (métodos) que podem ser chamadas pelos clientes

```
Objeto pessoa
{
    string nome;
    short identidade;
    long data_nascimento;

    void cadastro(in string nome, in short id, in long data)
    void modifica(in string nome, in short id, in long data)
     void imprime_todas( )
}
```

CORBA

- CORBA (Common Object Request Broker Architecture)
  - padrão para interoperabilidade de objetos distribuídos
    - » modelo cliente-servidor
    - » heterogeneidade de linguagens
    - » distribuição
  - produto de um consórcio chamado OMG (Object Management Group)

CORBA

- Programação Modular + Programação Orientada a Objetos
  - » encapsulamento de dados
  - » objetos em interação que encapsulam a implementação
  - » definição de interfaces dos objetos
    - substituição da implementação sem afetar o uso do objeto
    - desenvolvimento de software *plug-and-play*

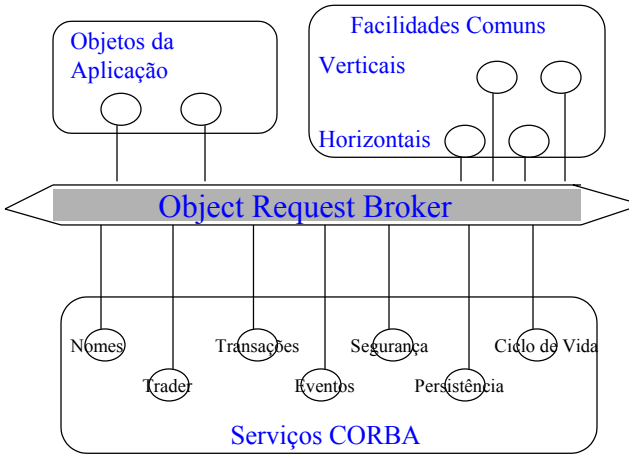
CORBA

- ORB (Object Request Broker): barramento através do qual objetos trocam mensagens
- Interface de Objetos escritas em IDL (Interface Definition Language)
- Implementação dos Objetos em qualquer linguagem que possua o mapeamento (binding) para CORBA

Binding CORBA

- Mapeamento entre uma linguagem e a IDL que especifica como tipos IDL são convertidos em tipos e chamadas de funções da linguagem

Arquitetura Global da OMG



IDL

- Linguagem Declarativa
- Usada para expressar
  - atributos
  - operações que o objeto pode realizar
  - parâmetros de entrada e de saída
  - exceções
  - eventos que um objeto pode emitir

Estrutura da Interface IDL

```
Module <identificador>
{
<declarações_de_tipo>;
<declarações_de_constantes>;
<declarações_de_exceções>;

    interface <identificador> [:<lista_interfaces_base>]
    {
        <declarações_de_tipo>;
        <declarações_de_constantes>;
        <declarações_de_atributos>;
        <declarações_de_exceções>;

        [<tipo_operação>] <identificador> (<parâmetros>)
        [ raises exceção ] [contexto];
    }
}
```


Exemplo IDL

```
Interface pessoa
{
    string nome;
    short identidade;
    long data_nascimento;

    void add_pessoa(in string nome, in short id, in long data)
         raises (JaCadastrado);
}

module pessoas_da_escola
{
    interface estudante:pessoa
    {
       short matricula;
       string curso;
     void add_estud(in string nome, in short id, in long datanasc,
                 in short mat, in string cur)
          raises(JaCadastrado);
    }
```

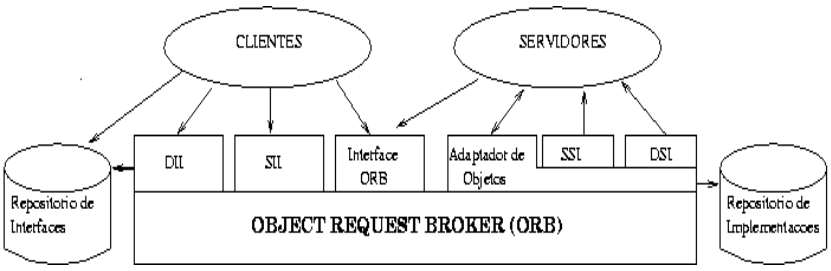
```
Interface professor: pessoa
{
    string disciplina;

    void add_prof(in string nome, in short id,
                 in long datanasc, in string disc)
         raises (JaCadastrado);
    }
}
```

Referência do Objeto

- Identificação do Objeto CORBA
- Atribuída pelo ORB na criação do objeto.
- Para usar o objeto, o cliente deve obter a referência do objeto pois em uma invocação de um método sobre o objeto, o ORB o identifica através da sua referência.
- Pode ser obtida através do serviço de Nomes ou do serviço de Trading

Arquitetura CORBA



ORB - Object Request Broker

- Realiza as tarefas de Comunicação e Coordenação entre objetos
- Intercepta as chamadas e direciona-as para o objeto apropriado

Repositório de Interfaces (RI)

- O RI armazena todas as definições IDL de objetos CORBA disponíveis
- CORBA oferece um conjunto de métodos para leitura e navegação no RI
- O *RepositoryID* identifica globalmente os elementos do RI
  - Exemplo: o RepositoryID da interface *professor* no módulo *pessoas_da_escola* poderia ser:

    IDL:Escola/pessoas_da_escola/professor/:1.0”

Repositório de Implementações

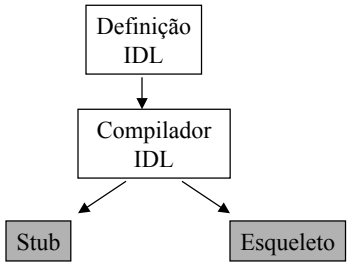
- Armazena informações sobre a implementação dos objetos
  - as classes que um servidor implementa
  - os objetos que estão instanciados
  - as referências dos objetos

Solicitações a Objetos

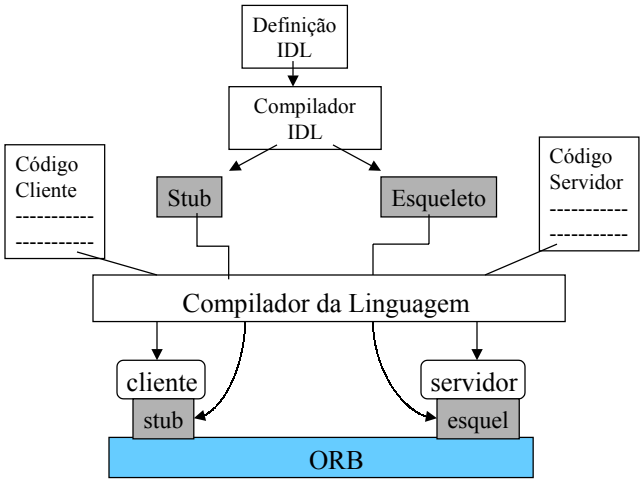
- Podem ser elaboradas de duas maneiras:
  - Estaticamente: usando a Interface de Invocação Estática (SII)
  - Dinamicamente: usando a Interface de Invocação Dinâmica (DII)

SII

- Especificações IDL são compiladas por um compilador IDL que gera stubs do cliente (simplesmente stubs) e stubs do servidor (esqueletos de programas)



Programação CORBA Estática Visão Geral



Problemas da Invocação Estática

- Cliente deve conhecer o stub IDL para cada interface usada
- Ligação estática entre programas clientes e as interfaces IDL dos objetos servidores
- Necessidade de modificação das interfaces implica em rever todos os programas clientes e servidores

DII

- Permite a invocação de métodos sem necessidade de stubs.
  - Descobrir os novos objetos
  - Descobrir suas interfaces (operação get_interface)
  - Obter a definição do método
  - Criar uma lista de argumentos
  - Criar a solicitação
  - Fazer a chamada

Interface de Esqueleto Estático (SSI)

- Para o servidor é transparente se a invocação do método é estática ou dinâmica (o ORB provê esta transparência)
- Funções da SSI:
  - desempacota as solicitações para transmitir para as implementações dos objetos
  - empacota os resultados para envia-los ao cliente

Interface de Esqueleto Dinâmico (DSI)

- Trata chamadas sem stubs
- Funções da DSI:
  - para obter o objeto e o método destino da invocação, consulta os valores dos parâmetros que vêm na mensagem
  - Todas as operações são direcionadas a um esqueleto dinâmico que repassa a chamada

Adaptador de Objetos

- Registra componentes no Repositório de Implementações
- Cria Instâncias de novos objetos em tempo de execução
- Gera e Gerencia Referências de Objetos
- Realiza o mapeamento entre Referências de Objetos e suas implementações
- Ativa e Desativa implementações de objetos

Exemplo: “Hello World”

- Código C++

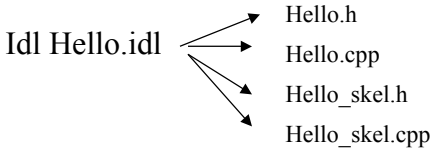
```
#include <iostream.h>
int
main(int, char*[], char*[])
{
    cout <<“Hello World!”<< endl;
    return 0;
}
```

Exemplo: “Hello World” Cliente-Servidor usando CORBA

- Passo 1: Definir a interface IDL

```
Interface Hello
{
   void hello();
};
```

- Passo 2: Traduzir o código IDL para C++ usando o tradutor IDL-C++



Continuação

- Passo 3: Implementando o servidor

```
// O arquivo Hello_impl.h
#include <Hello_skel.h>
class Hello_impl: public Hello_skel
{
public:
      virtual void hello();
};
```

```
// A implementação  Hello_impl.cpp
#include <OB/CORBA.h>
#include <Hello_impl.h>
void
Hello_impl::hello()
{
   cout <<“Hello World!”<<endl;
}
```

Continuação

- Passo 4: O programa principal do servidor (Server.cpp)

```
#include <OB/CORBA.h>
#include <Hello_impl.h>
#include <fstream.h>
int
main(int argc, char *argv[], char *[])
{
   CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
   CORBA_BOA_var boa = orb->BOA_init(argc, argv);

   Hello_var p = new Hello_impl;
   CORBA_String_var s = orb -> object_to_string(p);  // converte uma referencia CORBA
                                                        em um string
    const char* refFile = "Hello.ref";  // arquivo que vai conter a referencia
    ofstream out(refFile);
    out << s << endl;
    boa -> impl_is_ready(CORBA_ImplementationDef::_nil());  // loop do servidor para
                                                               esperar solicitacoes
}
```


Continuação

- Passo 5: Implementando o Cliente (Client.cpp)

```
#include <OB/CORBA.h>
#include <Hello.h>
#include <fstream.h>
int
main(int argc, char *argv[], char *[])
{
   CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);

    const char* refFile = "Hello.ref";  // ler a referencia escrita pelo servidor
    ifstream in(refFile);
    in >> s;
    CORBA_Object_var obj = orb -> string_to_object(s);
    Hello_var hello = Hello::_narrow(obj); // a partir da referencia CORBA gera uma referencia para
                                              objeto
    hello -> hello();  // chama a operacao hello no objeto hello
}
```

Continuação

- Passo 6: Compilação e Linking
  - Dependente de plataforma. Leia o arquivo README do ORBacus
- Passo 7: Execução
  - Executar primeiro o Servidor que vai criar o arquivo Hello.ref necessário para o cliente comunicar-se com o servidor
  - Executar o Cliente

Exemplo de Programação CORBA sem stubs

- LuaOrb
  - *binding* entre Lua e CORBA que usa a DII (do lado do cliente) e a DSI (do lado do servidor), permitindo que se instale e use objetos CORBA sem necessidade de geração de *stubs*.
  - Linguagem de programação: Lua

Lua

- dinamicamente tipada (não há necessidade de declaração prévia de tipos)

```
a = 10
...
a = "Oi"
```

- Forma de estruturar dados: Tabelas
  - criação de uma tabela vazia: { }
  - criação com atribuição de campos: t = {primeiro = "maria", ultimo = "joão" }
  - a representação **t.nome** é açúcar sintático para **t["nome"]**

Lua - continuação

- Programação Orientada a Objetos
  - através de funções e tabelas
  - Definição de funções:

```
function objeto:método(parâmetros)
...
end
```

  - Chamada de método:

```
objeto:método(parâmetros)
```

Lua - continuação

- Tag Methods (métodos de Tag)
  - permite ao programador mudar o comportamento de Lua em condições especiais (presença de erros durante a execução ou acesso a um campo inexistente de uma tabela). Por exemplo:

    receiver:foo(parâmetros)

    receiver não é um objeto Lua e

    foo não é um método de *receiver* => ERRO

Lua - continuação

- Com tag methods um comportamento diferente pode ser estabelecido para esta situação

  receiver:foo(parâmetros)

  receiver não é um objeto Lua e

  foo não é um método de *receiver* => Repasse para LuaOrb tratar

LuaOrb

- Explora a idéia de *proxy* (representante)
  - » Cliente: cria um proxy que é um objeto Lua que representa o objeto CORBA

    Createproxy()
  - » Servidor: é criado um proxy (objeto C++) para um objeto Lua que age como servidor CORBA

    CreateDSIServer()

Exemplo: "Hello World" Cliente-Servidor usando LuaOrb

- Passo 1: Definir a interface IDL

```
Interface HelloLua
{
   void hello();
};
```

- Passo 2: Alimenta o repositório de interfaces

```
Irfeed("/usr/local/luaorb/demo/hello.idl")
```

Continuação

- Passo 3: Implementando o servidor em Lua

```
LuaOrb_cfg.irep = createproxy{interface = "CORBA::Repository", ior_file= "ir.ref"}

hello_impl={} -- criou o objeto hello_impl
function hello_impl:hello() -- cria o método hello do objeto hello_impl
      print("Hello World")
end

hello_server = CreateDSIServer("HelloLua", hello_impl) -- estabelece que a
                                                        hello_impl corresponde a IDL HelloLua
writeto("hello.ref")
write(hello_server:_get_ior())
writeto()
```


Continuação

- Passo 4: Implementando o cliente em Lua

```
LuaOrb_cfg.irep = createproxy{interface = "CORBA::Repository", ior_file= "ir.ref"}

h = createproxy{interface = "HelloLua", ior_file="hello.ref"}
h:hello()
```

Continuação

- Passo 5: Execução
  - Executar primeiro o Servidor que vai criar o arquivo Hello.ref necessário para o cliente comunicar-se com o servidor
  - Executar o Cliente

Serviços CORBA

- Serviço de Nomes
  - mapeamento entre um nome simbólico e uma referência de objeto
  - define interfaces para inserção e remoção de nomes, consulta e navegação pelo banco de nomes (repositório)
  - permite que nomes sejam estruturados hierarquicamente em *Contextos* - estruturas similares a diretórios

Continuação Serviço de Nomes

```
Interface NamingContext {
   void bind(in Name n, in Object obj)  // liga um nome a um objeto
         raises(NotFound, AlreadyBound, ...)
  ....
  Object resolve(in Name n) // procurar um objeto com nome n
  ...
  void unbind(in Name n)
        raises(NotFound, ...)
  ...
};
```