Object Adapters (OAs)

- Conectam implementações de objetos ao ORB
  - » diferentes OAs podem suportar diferentes estilos de implementação
- Ativam implementações de objetos (servidores)
- Encaminham requisições de serviço para implementações de objetos
- Geram *object references*

O *Basic Object Adapter* (BOA)

- Presente desde a versão inicial da especificação CORBA
- Problemas na especificação do BOA:
  - » aspectos importantes ficaram indefinidos
  - » muita ambiguidade
- É o adaptador de objetos oferecido pelos ORBs hoje existentes
  - » servidores atuais não são portáveis

Responsabilidades do BOA

- Ativação e desativação
  - » de implementações de objeto (servidores)
  - » de objetos individuais num servidor
- Geração e interpretação de *object references*
  - » para tanto, colabora com o servidor
- Invocação de métodos via esqueletos (faz *upcalls* para os esqueletos)

Ativação de Implementações e Ativação de Objetos

- Ao tratar uma requisição de servico o BOA pode ter de fazer dois tipos de ativação
  - » **Ativação de implementação** é necessária quando o servidor não estiver disponível
  - » **Ativação de objeto** é necessária quando o servidor estiver disponível, mas não o particular objeto ao qual a requisição se destina
  - » O BOA só pode chamar o um método do esqueleto se tanto o servidor como o objeto destino estiverem ativos

Políticas de Ativação de Implementação

- Servidor **compartilhado** (*shared server*)
  - » múltiplos objetos num servidor
- Servidor **não compartilhado** (*unshared)*
  - » servidor por objeto
- Servidor **por método**
  - » *à la* CGI
- Servidor **persistente** (o nome é muito ruim)
  - » servidor compartilhado, mas sem ativação automatica

A Interface do BOA

```
// pseudo IDL
module CORBA {
    interface InterfaceDef; // forward decls
    interface ImplementationDef;
    interface Object;
    typedef sequence<octet,1024> ReferenceData;
    ...
    interface BOA {
        ... // operações na próxima figura
    }; ...
}
```

Operações na Interface do BOA

```
// ativação e desativação de implementaçao

void impl_is_ready(in ImplementationDef impl);
void deactivate_impl(in ImplementationDef impl);
void obj_is_ready(in Object obj,
                  in ImplementationDef impl);
void deactivate_object(in Object obj);
```


Operações na Interface do BOA (cont.)

```
// geração e interpretação de object references

Object create(in ReferenceData id,
              in InterfaceDef intf,
              in ImplementationDef impl);
ReferenceData get_id(in Object obj);
void dispose(in Object obj);

// outras operações

...
```

Geração e Interpretação de *Object References*

- Um campo da *object reference* é definido pelo servidor que a criou
- Esse campo, denominado **`id`**, tem o propósito de distinguir os objetos implementados pelo servidor
- Quando o BOA faz uma *upcall* a um servidor para ativar um certo objeto, ele passa ao servidor o **`id`** do objeto

Geração e Interpretação de *Object References* (cont.)

- Um **`id`** pode conter o nome de um arquivo ou uma chave de acesso a uma base de dados com os estados de muitos objetos
- É responsabilidade do servidor prover uma rotina de ativação de objetos que interpreta o **`id`**
  - » O BOA faz *upcalls* a essa rotina

Problemas com o BOA

- No que diz respeito à ativação de objetos, a especificação do BOA deixou de definir muitas coisas:
  - » Como o BOA conhece a rotina de ativação de objetos a ser chamada?
  - » Quais os parâmetros dessa rotina?
  - » Que eventos podem causar ativação de objetos?
- Cada implementador de ORB fez suas escolhas...

"BOAs Proprietários"

- No Orbix, a ativação de objetos é chamada *loading* e é feita por *loaders*
  - » Para instalar um *loader*, um servidor cria uma instância de uma classe derivada da classe **`CORBA::LoaderClass`**
  - » A rotina de ativação de objetos é a função virtual **`load`**, um método da classe base **`CORBA::LoaderClass`**
    - – Esse método pode ser redefinido no servidor

“BOAs Proprietários” (cont.)

- O Visibroker para C++ adota um esquema semelhante
  - » uma instância da classe **`Activator`** é passada como parâmetro para **`impl_is_ready`**
  - » a rotina de ativação de objetos é a função virtual **`activate`**, um método da classe **`Activator`**

“BOAs Proprietários” (cont.)

- O Orbix e o Visibroker permitem que um servidor instancie objetos CORBA diretamente
  - » o construtor de um objeto CORBA registra o objeto com o ORB e cria uma *object reference;* o destrutor faz o oposto
  - » não se usam as operações **`BOA::create`** e **`BOA::dispose`**

Implementando Interfaces IDL em C++

- Como indicar que uma classe C++ implementa uma interface IDL?
  - » Através de uma conexão entre essa *implementation class* e a classe esqueleto gerada pelo compilador IDL
- Duas abordagens tem sido usadas:
  - » heranca
  - » delegação

Implementando Interfaces IDL em C++ (cont.)

- Nos próximos slides vamos considerar:
  - » uma **interface IDL**
  - » as correspondentes **classe interface** (*interface class)* e **classe esqueleto**, ambas geradas automàticamente pelo compilador IDL
  - » a **classe implementação** (*implementation class*) fornecida pelo implementador da interface
- Nosso interesse é a conexão entre a classe esqueleto e a classe implementação

Interface IDL

```
// IDL
interface A {
    short op1();
    void op2(in long l);
};
```


Classe Interface

```
// C++
class A : public virtual CORBA::Object {
public:
    virtual CORBA::Short op1() = 0;
    virtual void op2(CORBA::Long l) = 0;
    ...
};
```

Classe Esqueleto

```
// C++
class _sk_A : public A {
public:
    // ... detalhes específicos de ORB
    //     omitidos daqui ...
    virtual CORBA::Short op1() = 0;
    virtual void op2(CORBA::Long l) = 0;
    ...
};
```


Classe Implementação (Abordagem: Herança)

```
// C++
class A_impl : public _sk_A {
public:
    CORBA::Short op1();
    void op2(CORBA::Long l);
    ...
};
```

Classe Esqueleto Voltada Para Delegação (*Tie Class*)

```
// C++
template <class T>
class _tie_A : public A {
public:
    _tie_A(T& t) : _ref(t) {}
    CORBA::Short op1() { return _ref.op1(); }
    void op2(CORBA::Long l) { _ref.op2(); }
private:
    T& _ref;
};
```


Classe Implementação (Abordagem: Delegação)

```
// C++
class A_impl {
public:
    CORBA::Short op1();
    void op2(CORBA::Long l);
    ...
};
```

- Objetos de implementação são "amarrados" (*tied*) a interfaces IDL em tempo de execução:

  `_tie_A<A_impl> my_obj(my_impl);`

  (aqui **`my_impl`** é uma instância da classe **`A_impl`**)