CORBA:
IDL, Callback y otros detalles

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid

Contenidos

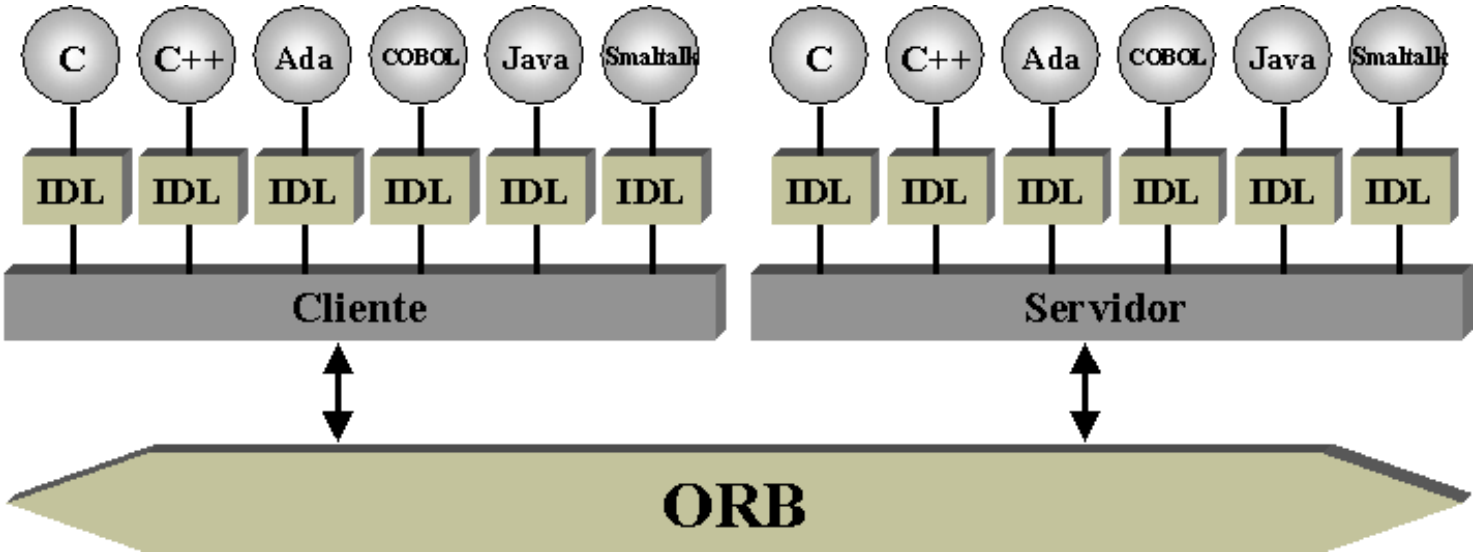
1. El lenguaje de definición de interfaces (IDL)
2. Correspondencia de IDL con Java
3. *Callback* de cliente
4. *Dynamic Invocation Interface*

Contenidos

1. **El lenguaje de definición de interfaces (IDL)**
2. Correspondencia de IDL con Java
3. *Callback* de cliente
4. *Dynamic Invocation Interface*

El lenguaje de definición de interfaces

- ¿Qué es el IDL?
  - Es un lenguaje que permite definir las interfaces de manera independiente del lenguaje de implementación.
  - Existen proyecciones a diversos lenguajes de programación.



http://www.omg.org/

Tipos simples

- Enteros
  - long
  - unsigned long
  - short
  - unsigned short
  - octet
- Decimales
  - Float
  - Double

- Caracteres
  - char
  - string
    - string variable;
    - string fijo<20>;
- Otros
  - void
  - boolean

Enumerados


```
enum estaciones {
  primavera, verano, otonyo, invierno
};

enum colores {
  rojo, verde, azul
};
```

Estructuras


```
struct Persona {
  string nombre;
  long edad;
  Fecha fechaNacimiento;
};
```

Uniones


```
union direccion switch(long) {
  case 1:
    DirIP ip;
  case 2:
    DirURL url;
};
```

- Equivalentes a las uniones de C
- Permite definir un registro con variantes basado en un campo discriminante

Arrays


```
typedef string estaciones[4];
typedef float Calificaciones[100];
typedef long Matriz[10][10];
```

- Definen colecciones de elementos del mismo tipo
- Tienen longitud fija
- Se debe definir un tipo (*typedef*) antes de usarlo

Secuencias


```
typedef sequence<string> contactos;

typedef sequence<float,365> temperaturas;
```

- Son vectores de longitud variable
- Pueden tener longitud máxima
- Puede contener secuencias

Tipos definidos por el usuario


```
typedef string Nombre;

typedef float Calificacion;
```

- Permite renombrar tipos existentes.
- Son de uso obligatorio para arrays y secuencias.

Constantes


```
const float   PI = 3.1416;
const string  SALUDO = “HOLA”;
const boolean MULTIHILO = TRUE;
```

- Admitidas para enteros, flotantes, carácter, cadenas, booleanos, bytes y enumerados

Métodos


```
interface Cuenta {
  float saldo ();
  void ingresa(in float cantidad);
  void dispon (in float cantidad);
};
```

- Definen las operaciones admisibles sobre objetos que implementan una interfaz.

Parámetros


```
interface GestorMensajes {
    void envia  (in string mensaje);
    void recibe (out string mensaje);
    void firma  (inout string mesaje);
};
```

- Se debe indicar el sentido del parámetro:
  - Entrada: in
  - Salida: out
  - Entrada/Salida: inout

Métodos de un solo sentido


```
interface Temporizador {
    oneway void desactiva();
    ...
};
```

- Por defecto todos los métodos CORBA son bloqueantes:
  - El cliente no recupera el control hasta que el objeto no finaliza la operación.
- Se puede especificar que un método sea no bloqueante, pero:
  - El tipo de retorno ha de ser void.
  - Los parámetros han de ser solo *in*.
  - No puede lanzar una excepción.

Interfaces y atributos


```
interface alumno {
    readonly attribute string id;
    attribute string titulacion;
    ...
};
```

- Un interfaz puede tener atributos
- Se generan métodos de acceso para el atributo de forma automática

Herencia de interfaces


```
interface Cuenta {
  void ingresa(in float cantidad);
  void dispon(in float cantidad);
};


interface CuentaCredito : Cuenta {
  void fijaLimite(in float limite);
};
```

Módulos


```
module Modulo1 {
    interface I1 { ... };
    interface I2 { ... };
    ...
};
```

Contenidos

1. El lenguaje de definición de interfaces (IDL)
2. **Correspondencia de IDL con Java**
3. *Callback* de cliente
4. *Dynamic Invocation Interface*

Correspondencia desde IDL a Java

- En general, para cada elemento del IDL hay un elemento de Java similar
  - Ejemplo:
    las excepciones en IDL se proyectan a clases que extienden a org.omg.CORBA.UserException

IDL	Java

exception SaldoInsuficiente {
  float cantidad;
} ;
	Class SaldoInsuficiente
	  extends org.omg.CORBA.UserException
	{
	  public float cantidad = (float)0;
	  public SaldoInsuficiente () {… }
	  public SaldoInsuficiente (float _cantidad) { … }
	  public SaldoInsuficiente (String $reason,
	  float _cantidad) { … }
	}

Correspondencia desde IDL a Java

- Para facilitar el envío/recepción de objetos, se crean clases auxiliares que facilitan las operaciones de aplanamiento y conversión
  - Importante el uso de la clase Any (org.omg.CORBA.Any) que sirve como contenedor de cualquier tipo que puede ser descrito en IDL o por cualquier tipo primitivo IDL.
  - La clase Any contiene dos elementos importantes:
    - *Value*: el valor de un dato.
    - *TypeCode*: un objeto que describe el tipo del valor del dato guardado.
  - Gran parte de la clase consiste en pares de métodos para insertar y extraer valores de un objeto Any
  - Principales usos son en el paso de argumentos o resultados en las peticiones, y en los objetos *Context*

Tipos básicos

IDL	Java
void	void
short	short
unsigned short	short
long	int
unsigned long	int
long long	long
unsigned long long	long
float	float
double	double
char	char
wchar	char
string	java.lang.String
wstring	java.lang.String
boolean	boolean
octet	byte

Para las variables se generan 1 clases:

- nombrevariableHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.

Enumerados (1/2)

IDL	Java

```
enum Color  {
   rojo, verde, azul
};
```

Se generan 3 clases:

- Color:
  Es la clase que implementa el enumerado.
- ColorHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.
- ColorHolder:
  Permite el paso de parámetros como *out* o como *inout*.

Enumerados (2/2)


```
public class Color implements org.omg.CORBA.portable.IDLEntity
{
  private        int __value;
  private static int __size = 3;
  private static Color[] __array = new Color [__size];

  public static final int _rojo = 0;
  public static final Color rojo = new Color(_rojo);
  public static final int _verde = 1;
  public static final Color verde = new Color(_verde);
  public static final int _azul = 2;
  public static final Color azul = new Color(_azul);

  public int value () {…} ;
  public static Color from_int (int value) {…} ;
  protected Color (int value) {…} ;
}
```

Estructuras (1/2)

IDL	Java

```
struct Persona {
  string  nombre;
  long    edad;
  long    dni;
};
```

Se generan 3 clases:

- Persona:
  Es la clase que implementa la estructura.
- PersonaHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.
- PersonaHolder:
  Permite el paso de parámetros como *out* o como *inout*.

Estructuras (2/2)


```
public final class Persona implements org.omg.CORBA.portable.IDLEntity
{
  public String nombre = null;
  public int edad = (int)0;
  public int dni = (int)0;

  public Persona () { ... }
  public Persona (String _nombre, int _edad, int _dni) { ... }
}
```

Uniones (1/2)

IDL	Java

```
enum lenguaje {
   Java, Cpp, C
} ;

union Compilador
          switch(lenguaje)
{
   case Java:
      string textoVersion;
   default:
      long numeroVersion;
 };
```

Se generan 3 clases:

- Compilador:
  Es la clase que implementa la unión.
- CompiladorHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.
- CompiladorHolder:
  Permite el paso de parámetros como *out* o como *inout*.

Uniones (2/2)


```
// ejemplo de uso

...
Compilador c = new Compilador().
c.textoVersion("3.2");
...
switch (c.discriminator())
{
   case Java:
            System.out.println(c.textoVersion());
            break;
   default:
            System.out.println(c.numeroVersion());
            break;
}
...
```

Arrays y secuencias

IDL	Java
typedef string estaciones[4];	string []
typedef sequence<string> contactos;	string []
typedef sequence<float,365> temperaturas;	float[]

Los tipos array y secuencias se proyectan a array en Java

Para las variables se generan 2 clases:

- nombretipoHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.
- nombretipoHolder:
  Permite el paso de parámetros como *out* o como *inout*.

Definidos por el usuario

IDL	Java
typedef otroTipo nuevoTipo;	otroTipo

Se utiliza el tipo ya definido.

Para las variables se generan 1 clases:

- nuevotipoHelper:
  Proporciona métodos para su inserción y extracción en un *Any*.

Constantes

IDL	Java

// Constantes 'globales'
const float PI = 3.1416;
	public interface PI {
	public static final
	float value = (float)(3.1416);
	}

// Constantes de una interfaz
interface Circulo {
const float PI = 3.1416;
} ;
	public interface Circulo extends … {
	public static final
	float PI = (float)(3.1416);
	…
	}

Para cada constante 'global' se genera 1 interfaz:

- nombreConstante:
  define una constante *value* con el valor.

Para las constantes en una interfaz, se aprovecha la interfaz ya generada

Métodos (1/2)

IDL	Java

interface Iface
{
void register (
inout string v,
in long l
);
};
	public interface IfaceOperations
	{
	void register (
	org.omg.CORBA.StringHolder v,
	int l
	);
	}

- Los métodos de las interfaces CORBA se transforman en métodos públicos de las interfaces Java.
- Las clases Java generadas relacionadas son:
  - IfaceOperations: contiene los métodos públicos en Java.
  - _IfaceStub: contiene el *stub* del cliente para la invocación del método.
  - IfaceHolder: apoyo para el paso de parámetros (especialmente *inout* y *out*)

Métodos (1/2)

IDL	Java

interface Iface {
oneway void register1(in long l);
void register2 (in long l);
};
	_IfaceStub.java
	…
	org.omg.CORBA.portable.OutputStream $out = _request ("register1", false);
	…
	org.omg.CORBA.portable.OutputStream $out = _request ("register2", true);
	…

- Los métodos de un solo sentido se indican como un indicador en la función de invocación de la clase _IfaceStub.

Interfaces

IDL	Java

interface Cuenta {
…
};
	public interface Cuenta extends
	CuentaOperations,
	org.omg.CORBA.Object,
	org.omg.CORBA.portable.IDLEntity
	{ … }

- Una interfaz CORBA se proyecta a una interfaz Java que extiende, al menos, org.omg.CORBA.Object y org.omg.CORBA.portable.IDLEntity

Interfaces: atributos (1/2)

IDL	Java

interface Cuenta {
  readonly attribute string id;
  attribute float dinero;
};
	Se generan 5 clases:
	
	❑ Cuenta:
	Es la clase que implementa la interfaz.
	❑ CuentaHelper:
	Proporciona métodos para su inserción y extracción en un *Any*.
	❑ CuentaHolder:
	Permite el paso de parámetros como *out* o como *inout*.
	❑ CuentaOperations:
	Proporciona los métodos de acceso y modificación a los atributos.
	❑ _CuentaStub:
	Implementa el stub del cliente para todos los métodos (incluidos los anteriores).

Interfaces: atributos (2/2)


```
// interface CuentaOperations

public interface CuentaOperations
{

   String id ();
   float  dinero ();
   void   dinero (float newDinero);

}
```

Interfaces: herencia

IDL	Java

interface Cuenta {
  void ingresa(in float cantidad);
};
 	
interface CuentaCredito : Cuenta {
  void fijaLimite(in float limite);
};
	public interface CuentaCredito extends
	  CuentaCreditoOperations,
	  Cuenta,
	  org.omg.CORBA.portable.IDLEntity
	{ … }

- La herencia de interfaces CORBA se transforma en herencia de interfaces Java.

Widening y Narrowing

```
interface Cuenta {
   void ingresa(in float cantidad);
};

interface CuentaCredito : Cuenta {
   void fijaLimite(in float limite);
};
```

- Conversiones relacionadas con la herencia:
  - **Widening**: convertir una referencia de una clase derivada en una referencia a una clase base.

CuentaCredito -> Cuenta

  - **Narrowing**: convertir una referencia de una clase base en una referencia a una clase derivada.

Cuenta -> CuentaCredito

Widening y Narrowing

- Conversiones relacionadas con la herencia:
  - **Widening**: se realiza de forma automática.

```
CuentaCredito cuentaCredito = ... ;
Cuenta cuenta;
cuenta = cuentaCredito;
```

  - **Narrowing**: es necesario utilizar la función *narrow* que se genera en la clase *Helper*.

```
CuentaCredito cuentaCredito = ... ;
Cuenta cuenta;
try {
  cuentaCredito = CuentaCreditoHelper.narrow(cuenta);
}
catch (org.omg.CORBA.SystemException e) { ... }
```

Módulos

IDL	Java

module Finanzas
{
  interface Cuenta
  {
  // …
  };
 	
  // …
 	
};
 	
	package Finanzas;
	
	public interface Cuenta extends
	  CuentaOperations,
	  org.omg.CORBA.Object,
	  org.omg.CORBA.portable.IDLEntity
	{
	  // …
	}

- Un módulo CORBA se proyecta a un paquete Java.

Contenidos

1. El lenguaje de definición de interfaces (IDL)
2. Correspondencia de IDL con Java
3. ***Callback* de cliente**
4. *Dynamic Invocation Interface*

mchat.idl

- Interfaz Listener: escuchador en el cliente
- Interfaz MessageServer: registro de clientes

mchat.idl

```
interface Listener {
        void message   ( in string msg );
};

interface MessageServer {
        void register  ( in Listener lt );
};
```

Preprocesado del IDL a Java

```
acaldero@guernika# idlj -fall mchat.idl
```

- Genera un conjunto de archivos Java:
  - ListenerHelper.java
  - ListenerHolder.java
  - Listener.java
  - ListenerOperations.java
  - ListenerPOA.java
  - _ListenerStub.java
  - MessageServerHelper.java
  - MessageServerHolder.java
  - MessageServer.java
  - MessageServerOperations.java
  - MessageServerPOA.java
  - _MessageServerStub.java

Preprocesado del IDL a Java

```
acaldero@guernika# idlj -fall Hola.idl
```

- NO genera (y el programador ha de escribir):
  - MessageServerImpl.java: Implementación de register.
  - Servidor.java: Método principal del servidor CORBA.
  - ListenerImpl.java: Implementación de message.
  - Cliente.java: Método principal del cliente CORBA.

MessageServerImpl.java (1/3)

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Vector;
import java.util.Iterator;

public class MessageServerImpl extends MessageServerPOA {

    private Vector clients = new Vector();
    private ReadThread rt = null;

    public MessageServerImpl() {
        rt = new ReadThread(this);
    }

    public void register (Listener lt) {
        clients.add(lt);
    }
```

MessageServerImpl.java (2/3)

```
    public void startReadThread () {
         rt.start();
    }

    public void message (String msg)
    {
        Iterator it = clients.iterator();
        while (it.hasNext())
        {
            Listener lt = (Listener) it.next();
            lt.message(msg);
        }
    }

} // MessageServerImpl
```

MessageServerImpl.java (3/3)

```
class ReadThread extends Thread {
    MessageServerImpl msImpl = null;

    public ReadThread (MessageServerImpl msImpl) { this.msImpl = msImpl;}

    public void run() {
       BufferedReader br = new BufferedReader(
                                    new InputStreamReader(System.in));
       try {
         for (;;) {
               System.out.print("message > ");
               String msg = br.readLine();
               msImpl.message(msg);
         }
       } catch (Exception e) {e.printStackTrace(); }
    }
}
```

Servidor.java (1/3)

```
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Servidor {
    public static void main(String[] args) {
       try {
            // + Crear e inicializar el ORB
            Properties props = System.getProperties();
            props.put("org.omg.CORBA.ORBInitialPort", "1050");
            props.put("org.omg.CORBA.ORBInitialHost", "localhost");
            ORB orb = ORB.init(args, props);
            System.out.println("Initialized ORB");
```

Servidor.java (2/3)

```
// + Obtener la referencia al servicio activado en el POA
POA rootPOA = POAHelper.narrow(
        orb.resolve_initial_references("RootPOA"));

MessageServerImpl msImpl = new MessageServerImpl();
rootPOA.activate_object(msImpl);
MessageServer msRef = MessageServerHelper.narrow(
        rootPOA.servant_to_reference(msImpl));

// + Asociar la referencia al servicio de nombres
NamingContext namingContext = NamingContextHelper.narrow(
        orb.resolve_initial_references("NameService"));
System.out.println("Resolved NameService");
NameComponent[] nc = { new NameComponent("MessageServer", "") };
namingContext.rebind(nc, msRef);
```

Servidor.java (3/3)

```
            // + Activar el rootpoa
            rootPOA.the_POAManager().activate();

            // + Arrancar el hilo y esperar por peticiones
            System.out.println("Server ready and running ....");
            msImpl.startReadThread();
            orb.run();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

ListenerImpl.java

```
public class ListenerImpl extends ListenerPOA
{

    public void message(String msg)
    {
          System.out.println("Message from server : " + msg);
    }
}
```

Cliente.java (1/3)

```
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Cliente {
    public static void main(String[] args) {
        try {
            //initialize orb
            Properties props = System.getProperties();
            props.put("org.omg.CORBA.ORBInitialPort", "1050");
            props.put("org.omg.CORBA.ORBInitialHost", "localhost");
            ORB orb = ORB.init(args, props);
            System.out.println("Initialized ORB");
```

Cliente.java (2/3)

```
//Instantiate Servant and create reference
POA rootPOA = POAHelper.narrow(
              orb.resolve_initial_references("RootPOA"));
ListenerImpl listener  = new ListenerImpl();
rootPOA.activate_object(listener);
Listener ref = ListenerHelper.narrow(
               rootPOA.servant_to_reference(listener));

//Resolve MessageServer
MessageServer msgServer = MessageServerHelper.narrow(
    orb.string_to_object("corbaname::localhost:1050#MessageServer"));

//Register listener reference (callback object) with MessageServer
msgServer.register(ref);
System.out.println("Listener registered with MessageServer");
```

Cliente.java (3/3)

```
            //Activate rootpoa
            rootPOA.the_POAManager().activate();

            //Wait for messages
            System.out.println("Wait for incoming messages");
            orb.run();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Compilación del ejemplo

guernika.lab.inf.uc3m.es

```
acaldero@guernika # javac1.6 –g  *.java
```

Ejecución del ejemplo

guernika.lab.inf.uc3m.es

```
acaldero@guernika # orbd -ORBInitialPort 1050

acaldero@guernika # java1.6  Servidor
```

```
acaldero@guernika # java1.6  Cliente &

acaldero@guernika # java1.6  Cliente &
```

Contenidos

1. El lenguaje de definición de interfaces (IDL)
2. Correspondencia de IDL con Java
3. *Callback* de cliente
4. ***Dynamic Invocation Interface***

Dynamic Invocation Interface (DII)

- Clientes necesitan un *stub* precompilado.
- *Dynamic Invocation Interface*:
  - Servidores ofrecen dinámicamente interfaces
  - Clientes acceden a los interfaces deseados
- Detección de los objetos remotos:
  - Servicio de nombres de CORBA
  - Servicio de trading OMG
  - Herramientas de búsqueda

Dynamic Invocation Interface (DII)

- Implicaciones de diseño:
  - La misma implementación del servidor
  - Aumento de la complejidad del cliente

- Esquema general:
  - Encontrar el objeto y su referencia
  - Acceder al interface del objeto
  - Acceso al método

Interfaces a usar en la DII

- CORBA::Object
  Define las operaciones de cada objeto CORBA.
  get_interface()
  create_request() crea objeto Request

- CORBA::Request
  Define las operaciones sobre cada objeto remoto.
  add_arg(), invoke(), send_oneway(), delete(), etc.

- CORBA::VNList
  Da soporte a la construcción de la lista de parámetros.
  add_value(), get_count(), remove()

- CORBA::ORB
  Define métodos ORB de propósito general.
  create_list();

Esquema general de DII

1. Obtener el nombre del interface: Objeto InterfaceDef
   CORBA::Object.get_interface();
2. Obtener la descripción del método.
   lookup_name(), describe(), describe_interface()
3. Crear la lista con los argumentos: Objetos NVList.
   CORBA::ORB.create_list(), CORBA::ORB.add_item()
4. Crear la solicitud: Nombre del método + NVList
   CORBA::Request
5. Invocar la solicitud.
   CORBA::Request.invoke(),
   CORBA::Request.send_deferred()
   CORBA::Request.send_oneway()

Estrategias para implementar DII

- *Do-it-Yourself*

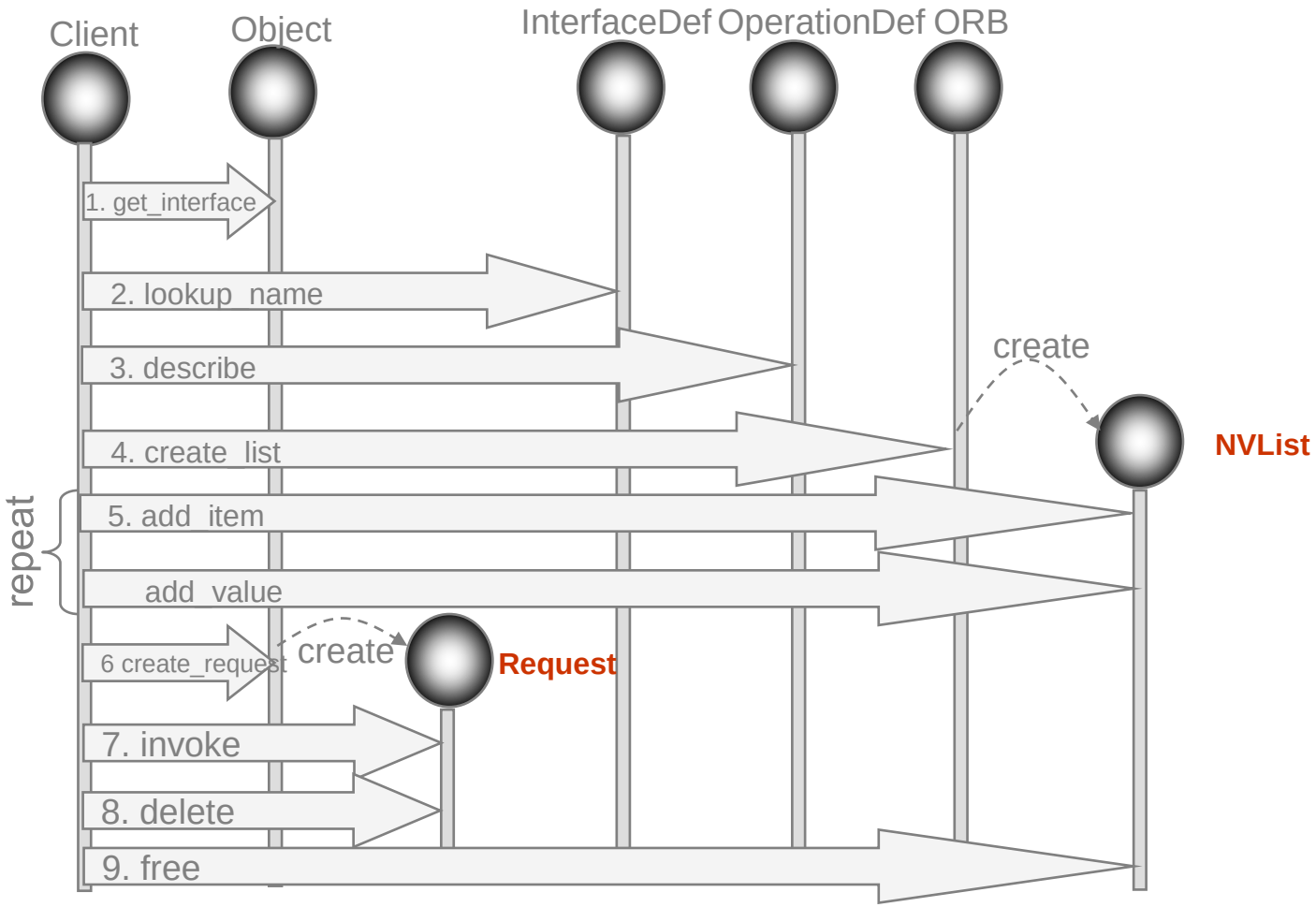
- *ORB-Can-Help*

- *Yet-Another-Way*

Do-it-Yourself (1/2)

- Crear una solicitud
- Ensamblar la solicitud sin ayuda del ORB
- Principales pasos:
  1. Preguntar al objeto por la definición de su interfaz:
     devuelve objeto *InterfaceDef*
  2. Buscar el método deseado:
     devuelve objeto *OperationDef*
  3. Obtener la descripción del método
  4. Crear una lista vacía de parámetros tipo NVList
  5. Rellenar la NVList
  6. Crear el objeto solicitado:
     devuelve objeto *Request*
  7. Invocar el método
  8. Borrar el objeto
  9. Borrar el NVList

Do-it-Yourself (2/2)

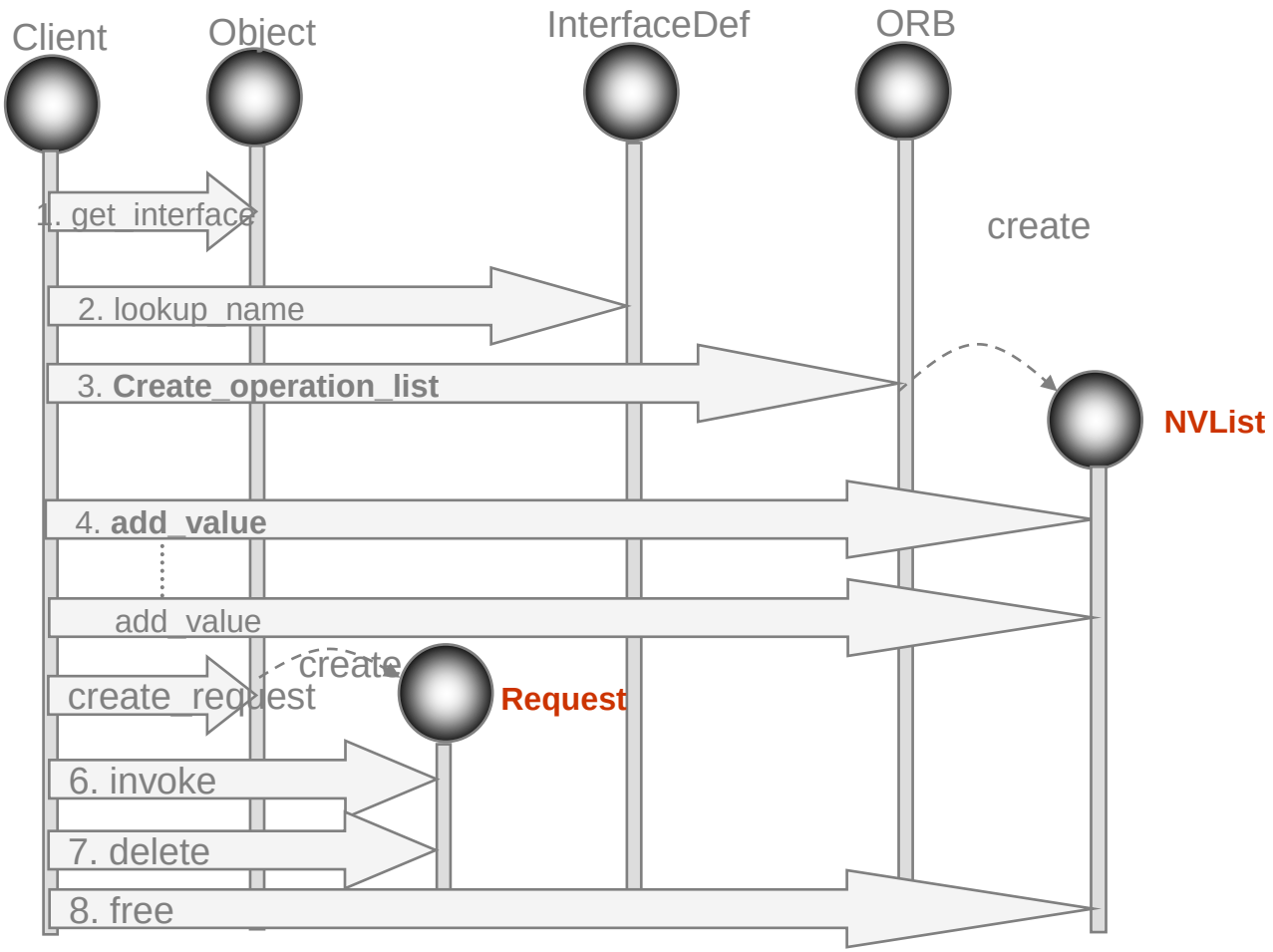


ORB-Can-Help (1/2)

- El ORB crea la NVList

- Principales pasos:
  1. Preguntar al objeto por la definición de su interfaz
  2. Buscar el método deseado
  3. Llamar al ORB para crear la NVList
  4. Llamar al ORB para rellenar la NVList
  5. Crear el objeto solicitado
  6. Invocar el método
  7. Borrar el objeto
  8. Borrar el NVList

ORB-Can-Help (2/2)

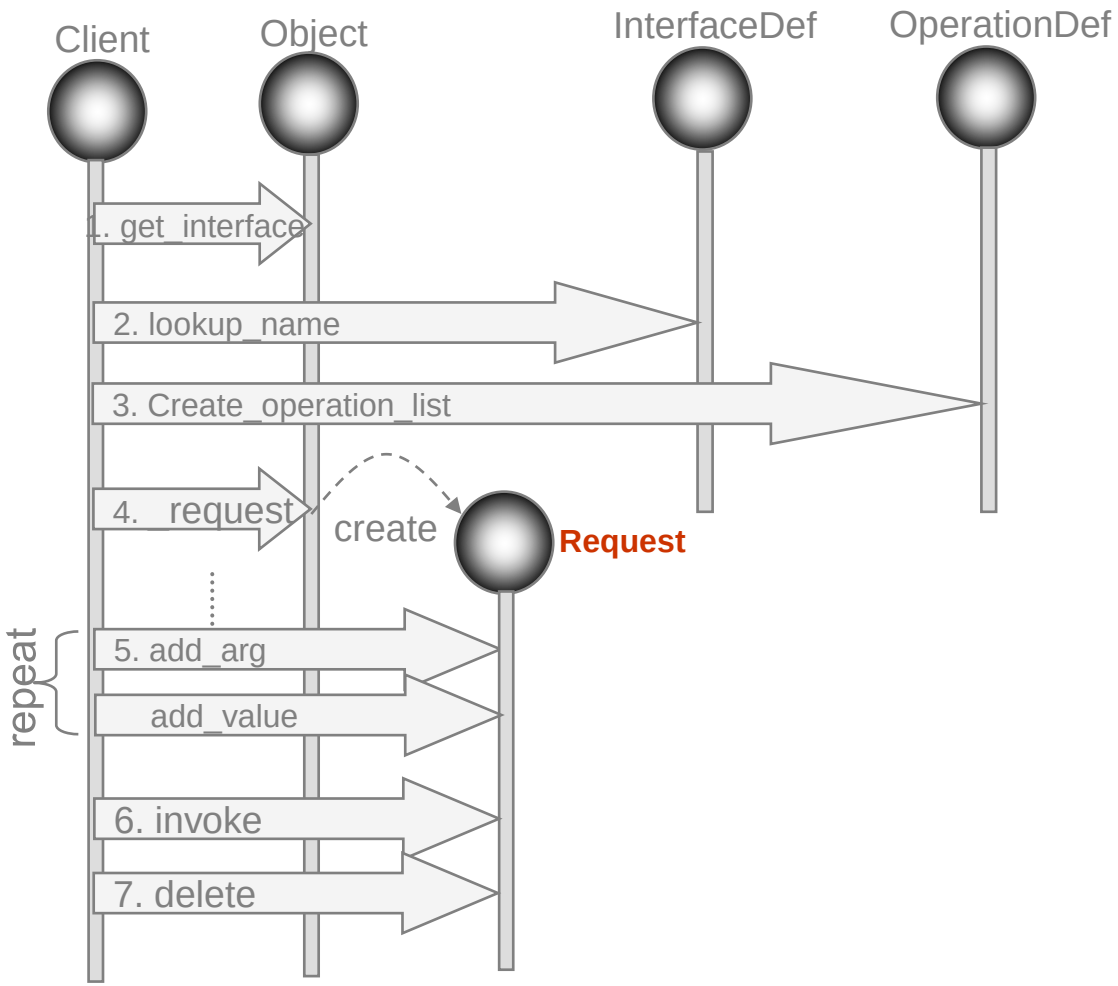


Yet-Another-Way (1/2)

- No utilizar NVList

- Principales pasos:
  1. Preguntar al objeto por la definición de su interfaz
  2. Buscar el método deseado
  3. Obtener la descripción del método
  4. Crear una solicitud vacía de objeto
  5. Rellenar la solicitud
  6. Invocar el método
  7. Borrar el objeto

Yet-Another-Way (2/2)



Ejemplo de *Yet-Another-Way* (1/4)

```
class CountClientDii
{
  public static void main(String args[])
  {
    boolean loop_all = false;
    long startTime, stopTime;
    CORBA.Request request;

    try
    {
      // Initialize the ORB.
      System.out.println("Initializing the ORB");
      CORBA.ORB orb = CORBA.ORB.init();

      // Bind to the Count Object
      System.out.println("Binding to Count Object");
      Counter.Count counter = Counter.Count_var.bind("My Count");
```

Ejemplo de *Yet-Another-Way* (2/4)

```
// Set Sum to initial value of 0
System.out.println("Setting Sum to 0");
counter.sum((int)0);

if ((args.length != 0) && (args[0].compareTo("loop_all") == 0))
     loop_all = true;

System.out.println("Starting invoke only test");
request = buildRequest(counter);

// Calculate Start time
startTime = Systeem.currentTimeMillis();

// Increment 1000 times
for (int i = 0 ; i < 1000 ; i++ )
  { request.invoke();}}
```

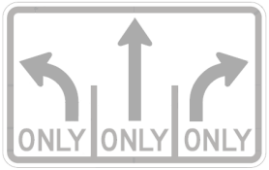
Ejemplo de *Yet-Another-Way* (3/4)

```
      // Calculate stop time; print out statistics
      stopTime = System.currentTimeMillis();

      System.out.println("Avg Ping = "
                         + ((stopTime - startTime)/1000f)
                         + " msecs");
      System.out.println("Sum = " + counter.sum());
    }
    catch(CORBA.SystemException e) {
      System.err.println("System Exception");
      System.err.println(e);
    }
  } // main
```

Ejemplo de *Yet-Another-Way* (4/4)

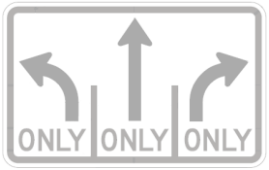
```
   public static CORBA.Request buildRequest(Counter.Count counter)
                                throws CORBA.SystemException
   {
        // (1) get interface for Count object
       CORBA.InterfaceDef CountInterface = counter._get_interface();

        // (3) describe interface for Count object
        CORBA._InterfaceDef.FullInterfaceDescription intDesc =
                                CountInterface.describe_interface();

        if (intDesc.operations[0].name.compareTo("increment") == 0)
        { // (4) create request object for dynamic increment
          CORBA.Request request = counter._request("increment");
          // initialize result value
          request.result().value().from_long(0);
          return request;
        } else System.out.println("Unknown method");
        return null;
   }
} // class
```

Opciones para acceso a método remoto



- *Stubs* estáticos precompilados
- *Dynamic Invocation Interface*
- Descarga dinámica de *stubs*

Recomendaciones



- Clientes invocan con frecuencia y los objetos de los servidores no cambian
  stubs estáticos
- Clientes invocan con poca frecuencia
  DII
- Clientes descubre servidores en tiempo de ejecución
  DII
- Clientes se ejecutan desde un navegador y descubren nuevos objetos
  descarga dinámica de *applets* o *stubs*

CORBA:
IDL, Callback y otros detalles

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid