1. Introducción

1.1 Construcción de una aplicación CORBA

Toda aplicación CORBA empieza con la definición de las interfaces de los objetos que pueden distribuirse. Para ello se utiliza el lenguaje IDL. Esta especificación establece un contrato entre cliente y servidor: qué servicios (operaciones) ofrece el servidor al cliente.

A partir del archivo con la especificación IDL de las interfaces, se pueden implementar por separado (e incluso en distintos lenguajes de programación y sobre plataformas diferentes, tanto producto CORBA como sistema operativo y hardware) el cliente y el servidor. Tanto cliente como servidor normalmente usan el método de invocación estático, y compilarán el IDL para generar el *stub* y *skeleton*, respectivamente.

Al programador del cliente le basta con realizar un programa que obtenga las referencias de los objetos CORBA que necesite, y que invoque las operaciones oportunas. El programador del servidor tiene que escribir la implementación de la clase del objeto de implementación y el programa del servidor donde se dará vida a los objetos de implementación.

1.2 Descripción de la aplicación

Se trata de implementar un programa cliente/servidor muy sencillo: un Contador distribuido

- Los clientes pueden consultar y modificar el valor del contador. También hay operaciones para incrementar y decrementar en una unidad el valor del contador
- La aplicación muestra el modo de invocación estático (*stubs* y *skeletons* generados por el compilador IDL)
- El programa cliente invoca 1000 veces la operación incrementar sobre el contador y al final muestra el tiempo medio de respuesta

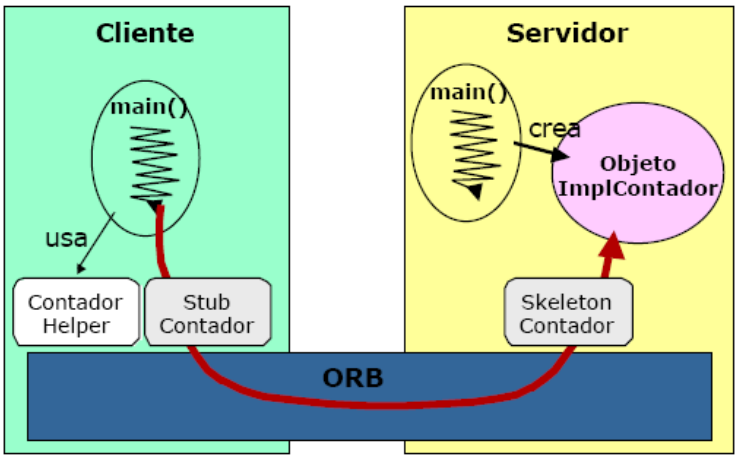
El objetivo de esta aplicación es dar una visión completa de la utilización de CORBA desde el punto de vista del programador.

La aplicación es extremadamente sencilla para no perderse en los detalles de la misma. Se muestra el modo de invocación estático por ser el utilizado habitualmente.

La aplicación permite también explorar la eficiencia de la invocación de operaciones en objetos CORBA locales o remotos.

1.3 Arquitectura de la aplicación

El cliente usa el **`ContadorHelper`** para acceder a funciones del ORB (por ejemplo, conseguir la referencia del objeto remoto) y el **`StubContador`** para invocar las operaciones en el objeto remoto. El servidor usa el **`SkeletonContador`** para recibir las invocaciones y devolver los resultados.



2. Especificación de la interfaz con IDL

El ejemplo muestra una interfaz muy sencilla, llamada *Contador*, que se define dentro del módulo **`PrimerEjemplo`** (un módulo en IDL es similar al **`package`** en Java).

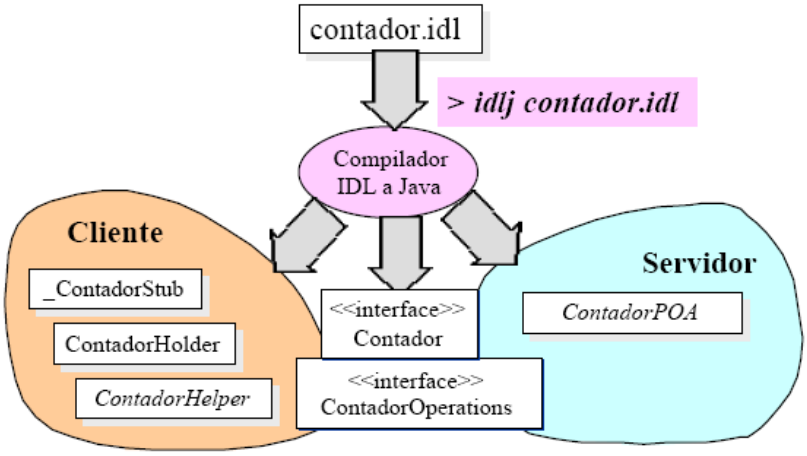
La interfaz define dos operaciones (sin parámetros, y que devuelven un entero **`long`** que es el equivalente al **`int`** en Java), **`inc()`** y **`dec()`**.

El atributo, *valor*, equivale a dos operaciones, una de modificación del valor: **`valor(long)`**, y otra de consulta del valor: **`long valor()`**

Obsérvese que en la interfaz no hay constructor. ¿Cómo puede, entonces, un cliente crear y destruir objetos remotos? Más adelante se verá que la creación de los objetos remotos es responsabilidad del servidor.

```
// contador.idl
module PrimerEjemplo {
     interface Contador {
          attribute long valor;
          long inc();
          long dec();
     };
};
```

3. Compilación de la interfaz



El compilador de IDL a Java genera dos interfaces Java y varias clases:

- **`PrimerEjemplo.Contador`**: es la interfaz de IDL en Java.
- **`PrimerEjemplo._ContadorStub`**: clase Java que implementa el *stub* de la interfaz **`Contador`** en el lado del cliente.
- **`PrimerEjemplo.ContadorHelper`**: clase Java que proporciona métodos útiles para los clientes de objetos **`Contador`**. Por ejemplo los métodos **`bind()`** para obtener una referencia a un objeto **`Contador`**, y **`narrow()`** para convertir referencias a objetos CORBA al tipo **`Contador`**.
- **`PrimerEjemplo.ContadorHolder`**: clase Java que se utiliza si hiciera falta pasar objetos Contador como parámetros **`out`** o **`inout`** en operaciones de otra interfaz.
- **`PrimerEjemplo.ContadorPOA`**: clase abstracta que sirve de base para la clase que tiene el código que implementa las operaciones de la interfaz.
- **`PrimerEjemplo.ContadorOperations`**: es la interfaz de IDL en Java que deben satisfacer los objetos en los que delegan los objetos de la clase **`_tie_`**.

3.1 Código generado por el compilador IDL: `Contador.java` y `ContadorOperations.java`

Obsérvese que en la traducción de IDL a Java:

- Cada operación se corresponde con un método Java
- Cada atributo se corresponde con dos métodos: uno para leer y otro para modificar el valor del atributo

La clase que implementa la interfaz en Java debe implementar los métodos descritos en esta interfaz Java.

```
package PrimerEjemplo;
// IDL:PrimerEjemplo/Contador:1.0
public interface Contador extends ContadorOperations,
                                  org.omg.CORBA.Object,
                                  org.omg.CORBA.portable.IDLEntity
{
}
```

```
package PrimerEjemplo;
public interface ContadorOperations {
     int valor();
     void valor(int val);
     int inc();
     int dec();
}
```

3.2 Código generado por el compilador IDL: `_ContadorStub.java`

Para cada método de la interfaz el código del *stub* se encarga de hacer el *marshalling* de los parámetros (es decir, su serialización o alineamiento), y pasárselos al ORB.

Obsérvese que el atributo valor se plasma en dos operaciones, y que el resultado long de las operaciones **`inc()`** y **`dec()`** se traduce en Java a **`int`**. Además, cada operación de la interfaz se define como **`public`**. Aparte, esta clase define su propio constructor, y algunos otros métodos de apoyo (como ejercicio, compilar el ejemplo y ver el código de las clases resultantes).

```
package PrimerEjemplo;
public class _ContadorStub
     extends org.omg.CORBA.portable.ObjectImpl
     implements Contador
{
     // ...
     public int valor() { /* ... */ }
     public void valor(int value){ /* ... */}
     public int inc() { /* ... */}
     public int dec() { /* ... */}
     // ...
}
```

3.3 Código generado por el compilador IDL: `ContadorHelper.java`

La clase **`Helper`** tiene métodos estáticos que pueden ser útiles para el programador del cliente y el servidor.
El método **`narrow()`** es estándar y sirve para convertir un objeto de tipo **`CORBA::Object`** a uno del tipo de la clase específica, en este caso **`Contador`**.

```
package PrimerEjemplo;
final public class ContadorHelper {
     public static void insert(org.omg.CORBA.Any any, Contador val) {
          // ...
     }
     public static Contador extract(org.omg.CORBA.Any any) {
          // ...
     }
     // ...
     public static Contador narrow (org.omg.CORBA.Object val) {
          //...
     }
}
```

3.4 Código generado por el compilador IDL: ContadorHolder.java

Las clases **`Holder`** sirven para soportar el paso de parámetros **`out`** e **`inout`** de objetos CORBA en otras operaciones definidas en una interfaz.

```
package PrimerEjemplo;
final public class ContadorHolder
                    implements org .omg.CORBA.portable.Streamable {
     public Contador value;
     public ContadorHolder() { }
     public ContadorHolder(Contador initial) { value = initial; }
     //...
}
```

3.5 Código generado por el compilador IDL: `ContadorPOA.java`

Los métodos de la interfaz se dejan como abstractos para que los implemente el programador en la clase que define a partir de ésta.

```
package PrimerEjemplo;
public abstract class ContadorPOA
                    extends org.omg.PortableServer.Servant
                    implements org .omg.CORBA.portable.InvokeHandler,
                    ContadorOperations
{
     // ...
     public org.omg.CORBA.portable.OutputStream _invoke (String opName,
                    org.omg.CORBA.portable.InputStream in,
                    org.omg.CORBA.portable.ResponseHandler handler)
     {
          // ...
     }
}
```

4. Implementación del cliente

El Cliente puede ser una clase Java con método **`main()`** o un **`applet`**. En general un Cliente realiza las siguientes funciones:

1) Inicializar el ORB
2) Obtener la referencia a un objeto CORBA que implemente la interfaz deseada (en este ejemplo, Contador) utilizando para ello, la interfaz de las funciones que proporciona el ORB. Antes de usar un objeto CORBA es necesario tener su referencia (con los objetos Java es igual, hay que conseguir la referencia del objeto Java antes de poder usarlo). En CORBA se obtiene la referencia al objeto CORBA, y lo que ocurre entonces es que se obtiene una referencia al objeto *proxy* que implementa el *stub* de la interfaz del objeto remoto). Hay varias maneras de hacer esto, lo normal usando el servicio de nombres o como resultado de una operación. El más sencillo (aunque no es lo habitual) es usar el método proporcionado por el ORB **`string_to_object()`**, que será utilizado en este ejemplo. **¡ATENCIÓN! En este ejemplo no se utiliza el servicio de nombres para obtener una referencia al objeto remoto**.
3) Utilizar el objeto CORBA. Una vez que se ha obtenido su referencia, el objeto CORBA se puede utilizar como cualquier otro objeto Java (en realidad se usa el *proxy* que es un objeto Java).

```
package PrimerEjemplo;
public class ClienteContador {
   public static void main(String args[]) {
      try {
      // 1. Inicializa el ORB
      // Inicializar el ORB siempre se hace igual, llamando al método
      // init(), método estático de la clase ORB (paquete org.omg.CORBA) y
      // que devuelve una referencia al objeto que soporta la funcionalidad
      // del ORB.
```

```
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);


      // 2. Consigue la referencia al objeto Contador.  En este ejemplo,
      // para conseguir una referencia al objeto que implementa la interfaz
      // Contador se usa la solución del método string_to_object() que
      // proporciona el ORB. Para ello se requiere que previamente se haya
      // dejado en un fichero (o en un URL) la referencia al objeto contador
      // al que se quiere acceder (esto lo habría hecho el servidor).

         org.omg.CORBA.Object obj = null;
         try {
            String refFile = "Contador.ref";
            java.io.BufferedReader in =
               new java.io.BufferedReader(new java.io.FileReader(refFile));
            String ref = in.readLine();
            obj = orb.string_to_object(ref);
         }
         catch(java.io.IOException ex) {
            ex.printStackTrace();
            System.exit(1);
         }
         Contador c = ContadorHelper.narrow(obj);

      // 3. Utiliza el contador. El objeto Contador se puede usar como
      // cualquier objeto Java. Este programa ejemplo, después de
      // incrementar 1000 veces el contador, lista el tiempo medio dedicado
      // en la invocación a la operación inc(). Al ejecutarse se puede
      // comparar el resultado cuando cliente y servidor están en la misma o
      // en distintas máquinas, o se puede probar a ejecutar varios clientes
      // a la vez.
         System.out.println("El Contador 1 tiene el valor: "+c.valor());
         // Llama a la operacion inc() 1000 veces
         System.out.println("Incrementando el contador...");
         long t_inicial = System.currentTimeMillis();
         for (int i = 0 ; i < 1000 ; i++ ) c.inc();
         // Calcula el tiempo final e imprime las estadisticas
         long t_final = System.currentTimeMillis();
         System.out.println("Tiempo medio de invocacion = ”
            + ((t_final - t_inicial)/1000f) + " msegs");
         System.out.println("Valor final del contador = " + c.valor());
   }
   catch(org.omg.CORBA.SystemException e) {
      System.err.println(“CORBA System Exception: " + e);
   }
   }
}
```

5. Implementación del servidor

Un objeto CORBA puede estar implementado por un objeto Java. La clase del objeto de implementación debe tener todos los métodos de clases en la interfaz e incorporar el skeleton. Esto último puede hacerse bien por herencia de la clase ImplBase, o utilizando objetos TIE que se encargan de realizar las tareas del skeleton y delegan la realización de la operación al objeto de implementación (que sería de una clase Java normal que implementa las operaciones de la interfaz).

Por otra parte, los objetos CORBA deben vivir en un proceso. El método **`main()`** del proceso inicializa el ORB y crea algunos objetos CORBA (al menos los necesarios para que el cliente pueda empezar a trabajar con el servidor).

Así pues, el servidor consta de dos partes:

1) La implementación de una o varias interfaces IDL
    - Clases que implementan las operaciones de las interfaces y que se corresponden a objetos CORBA
    - Hay dos maneras de implementar una interfaz:
        - Por herencia de la clase **`interfacePOA`**
        - Por delegación, usando una clase **`_tie_`**
2) El programa principal (**`main`**)
    - Inicializa el ORB y POA
    - Crea objetos que implementan interfaces

5.1 Implementación de la interfaz IDL

En este ejemplo se utiliza la herencia. La implementación se realiza definiendo una clase de implementación que tiene que hacer dos cosas:

1) Heredar de la clase <interfazIDL>POA. Esta clase la genera el compilador de IDL y proporciona el skeleton
2) Implementar cada método de la interfaz. Se pueden usar métodos auxiliares en la clase de implementación de la interfaz

La clase, que tiene el nombre que decida el programador (en este ejemplo **`ImplContador`**), hereda de **`ContadorPOA`** dentro del paquete **`PrimerEjemplo`** (el nombre que se le dio al módulo). En este ejemplo, todo contador nuevo empieza con el valor inicial dado en el constructor. Las operaciones son sencillas, y como puede verse no tienen nada de especial por el hecho de tratarse de la implementación de un objeto CORBA. Si fuese necesario, la clase de implementación podría añadir nuevos métodos auxiliares.

```
package PrimerEjemplo;
class ImplContador extends ContadorPOA {
     private int valor_;
     // Constructor
     ImplContador () {
          valor_ = 0;
          System.out.println("Creado Objeto Contador con valor= "+valor_);
     }
```

```
      // attribute valor
      public int valor() { return valor_; }
      public void valor(int val) { valor_ = val; }
      // operaciones:
      public int inc() { return ++valor_; }
      public int dec() { return --valor_; }
}
```

5.2 Implementación del servidor: programa principal

El programa principal (**`main`**) realiza las siguientes funciones:

1) Inicializar el ORB y el POA.

2) Crear los objetos CORBA. Al menos los que sean necesarios inicialmente (se pueden crear otros dinámicamente). Dentro de un servidor puede haber varios objetos que implementan interfaces CORBA. Estos objetos pueden ser creados por otros objetos, pero al menos uno debería ser creado en el flujo principal del programa servidor para que algún cliente pudiera invocar operaciones sobre él. **¡ATENCIÓN! Este ejemplo no utiliza el servicio de nombres para registrar al sirviente**.

3) Pasar el control al ORB (Bucle de eventos). Cuando ya se tienen objetos CORBA inicializados, entonces el servidor está preparado para recibir peticiones de operaciones en los objetos CORBA que contiene. Para eso, el servidor se mete en un bucle de eventos, esto es, se queda esperando recibir alguna petición, y la procesa, vuelve a esperar la siguiente petición y así indefinidamente.
   Para quedarse esperando indefinidamente peticiones se usa el método **`run()`**. Todo el código se ha puesto dentro de **`try-catch`** porque se puede producir la excepción **`SystemException`** (estándar) si alguna de las operaciones sobre el objeto **`orb`** fallara.

```
package PrimerEjemplo;

public class ServidorContador {

   static public void main(String[] args) {
      try {
         // 1. Inicializa el ORB
         org.omg.CORBA.ORB orb= org.omg.CORBA.ORB.init(args, null);
         // Crea un POA y obtiene la referencia al manager del rootPOA
         org.omg.PortableServer.POA rootPOA =
                   org.omg.PortableServer.POAHelper.narrow(
                            orb.resolve_initial_references("RootPOA"));
         org.omg.PortableServer.POAManager manager =
                   rootPOA.the_POAManager();

         // 2. Crea el objeto Contador
         ImplContador unObjetoContador = new ImplContador();
         Contador c = unObjetoContador._this(orb);
         // ... y exporta su referencia en un fichero:
         try {
```

```
            String ref = orb.object_to_string(c);
            String refFile = "Contador.ref";
            java.io.PrintWriter out = new java.io.PrintWriter(
               new java.io.FileOutputStream(refFile));
            out.println(ref);
            out.close();
         }
         catch(java.io.IOException ex) {
            ex.printStackTrace();
            System.exit(1);
         }

         // 3. Se queda esperando peticiones de servicio
         System.out.println("Servidor Contador preparado para
                     recibir peticiones");
         manager.activate();
         orb.run();
      }
      catch(Exception e) {
         e.printStackTrace();
         System.exit(1);
      }
      System.out.println("Fin de actividad del servidor Contador");
      System.exit(0);
   }
}
```

Para probar la aplicación es necesario compilar todo:

```
/PrimerEjemplo$ javac *.java
```

Ejecutar el servidor:

```
$java PrimerEjemplo.ServidorContador &
```

Ejecutar el cliente:

```
$java PrimerEjemplo.ClienteContador
```