Sistemas de Información

Tecnologías de objetos distribuidos: OMG-CORBA

Agradecimientos: Jesus Villamor Lugo, Simon Pickin de IT/UCIIIM, Juan José Gil Ríos de Terra.

Índice

- ¿Qué es?. Historia
- ¿Para qué sirve?. Objetivos
- ¿Cómo funciona?
- ¿Cómo se usa?. Ejemplos
- Arquitectura OMA
- Arquitectura Corba
- Corba y sus competidores

CORBA

Concepto ¿Qué es?

- **CORBA** (***C****ommon* ***O****bject* ***R****equest* ***B****roker* ***A****rchitecture*).
- Es una herramienta que facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos (HW y SW)
  - Distintos sistemas operativos (Unix, Windows, MacOS, OS/2)
  - Distintos protocolos de comunicación (TCP/IP, IPX, …)
  - Distintos lenguajes de programación (Java, C, C++, …)
- Define la infraestructura para la arquitectura **OMA** (***O****bject* ***M****anagement* ***A****rchitecture*) especificando los entándares necesarios para la invocación de métodos sobre objetos en entornos heterogéneos

CORBA

¿Para qué sirve?

- Permitir invocación de métodos de un objeto por objetos que residen en diferentes máquinas en entornos heterogéneos
  - Los objetos pueden estar desarrollados en diferentes lenguajes.
  - Los equipos pueden tener diferente:
    - Hardware
    - Sistema operativo
  - Los equipos pueden estar conectados entre sí usando distintos protocolos de comunicación
- Facilitar el desarrollo de aplicaciones distribuidas

CORBA

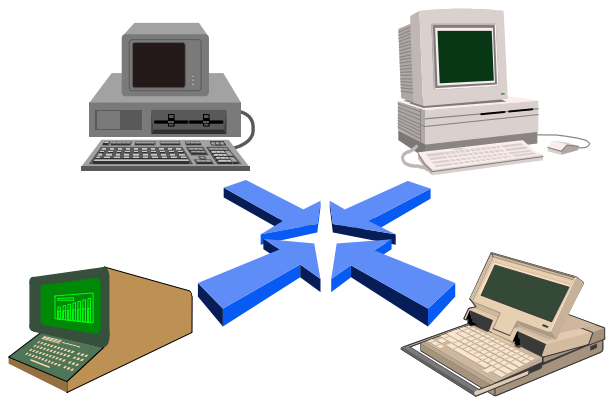
Historia. Especificaciones

- **OMG** (*Object Management Group*)
  - consorcio creado en 1989, primer “producto”: CORBA
  - inicialmente **8 empresas** (Sun, HP, 3Com,...)
    - Hoy: más de 800 socios
      - Proveedores de SW y equipos, operadores de telecomunicaciones, empresas, universidades,...
  - http://www.omg.org/
- **OMA** (*Object Management Archictecture*)
  - Inicial: 1990
- **CORBA** (*Common Object Request Broker Arquitecture*)
  - **CORBA 1**. 1991
  - **CORBA 2**. 1995
  - **CORBA 3**. 2002

CORBA

Objetivos

- Especificar un *middleware* para construir aplicaciones
  - cliente-servidor multi-nivel
  - distribuidos y centralizados
  - flexibles / extensibles



- Poder con la heterogeneidad
  - hardware distinto
  - redes distintas
  - sistemas operativos distintos
  - lenguajes de programación distintos

$\Rightarrow$ interoperabilidad

CORBA

Otras propiedades deseadas

- Transparencia de **distribución**
  - ni cliente ni servidor necesitan saber si la aplicación está distribuido o centralizado
- Transparencia de **localización** (caso distribuido)
  - el cliente no necesita saber donde ejecuta el servicio
  - el servicio no necesita saber donde ejecuta el cliente
- **Integración** de software existente
  - amortizar inversión previa
    - sistemas heredados (*legacy systems*)
- **Activación** de objetos
  - objetos remotos no tienen que estar en memoria permanentemente
  - se hace de manera invisible para el cliente

CORBA

Otras propiedades deseadas

- **Comunicación** flexible
  - múltiples modos de comunicación
    - síncrona: tipo invocación de métodos / RPC
    - asíncrona: sin respuesta
  - múltiples modelos de comunicación
    - invocación estática
    - invocación dinámica

CORBA

Objetivos comerciales

- CORBA 1 (1991)
  - llegar al mercado antes de Microsoft DCOM/COM+
  - competir con DCE, RPC: un éxito
- CORBA 2 (1995)
  - interoperabilidad entre implementaciones de distintos vendedores
  - competir con Java RMI, COM+: un éxito
- CORBA 3 (2002)
  - permitir desarrollo basado en componentes (CBD)
  - competir con EJB, .NET
    - ¿llegó demasiado tarde?
  - competir con los *Web Services*
    - difícil: *Web Services* actualmente en auge
  - hasta ahora no ha tenido éxito: ¿CORBA se muere?

CORBA

Medios para alcanzar los objetivos

- Orientado objetos
  - encapsulación
  - herencia
  - *late binding* & polimorfismo
- Comunicación por invocación de métodos remotos
  - más fácil de programar que IPC, *sockets*, RPC, etc.
  - *stub*:
    - representante local del objeto remoto
    - encargado de la comunicación con el objeto remoto
  - *skeleton*:
    - encargado de la comunicación con el cliente
  - cliente-servidor de grano fino
    - interacciones son de tipo cliente-servidor
    - papeles: misma entidad puede actuar como cliente o servidor

CORBA

Medios para alcanzar los objetivos

- Separación interfaz-implementación
  - la interfaz define un contrato
  - CORBA IDL (*Interface Definition Language*)
  - multi-lenguaje: mapeo IDL → lenguajes de programación
- Referencia de objeto: identificador único de un objeto
  - como un puntero
    - puede ser *nil* : no apunta a ningún objeto
    - puede estar colgando: apunta a un objeto inalcanzable o que no existe
  - puede ser *transient* o *persistent*
  - puede convertirse entre forma interna y forma cadena
  - CORBA 2 IOR (*Interoperable Object Reference*)
    - para evitar conflictos entre implementaciones de diferentes vendedores

CORBA

Medios para alcanzar los objetivos

- Protocolo estándar
  - IIOP: *Internet Inter-ORB Protocol* (implementación del GIOP)
  - Implementado encima de TCP/IP
- Modos de comunicación
  - síncrona:
    - invocación de métodos remotos normal
    - repuesta puede ser de tipo **`void`**
  - asíncrona:
    - invocación de métodos definidos como *oneway*
    - CORBA 3: añade servicio de mensajería (patrón *publisher-subscriber*)
- Uso de envolturas (*wrappers*)
  - permite integrar los sistemas heredados
  - normalmente, una sola instancia de cada clase

CORBA

Medios para alcanzar los objetivos

- ORB (*Object Request Broker*)
  - núcleo de CORBA: un bus de software
  - oculta la heterogeneidad a las dos partes
  - oculta los detalles de la comunicación a las dos partes
  - proporciona transparencia de localización via referencias de objeto
    - interpreta los referencias de objeto
    - canaliza las invocaciones del cliente al objeto remoto correcto
    - canaliza las respuestas del objeto remoto al cliente
  - proporciona transparencia de distribución
    - comportamiento igual en centralizado o distribuido
  - se ocupa de la activación y desactivación de objetos
  - proporciona servicios para construir peticiones dinámicamente

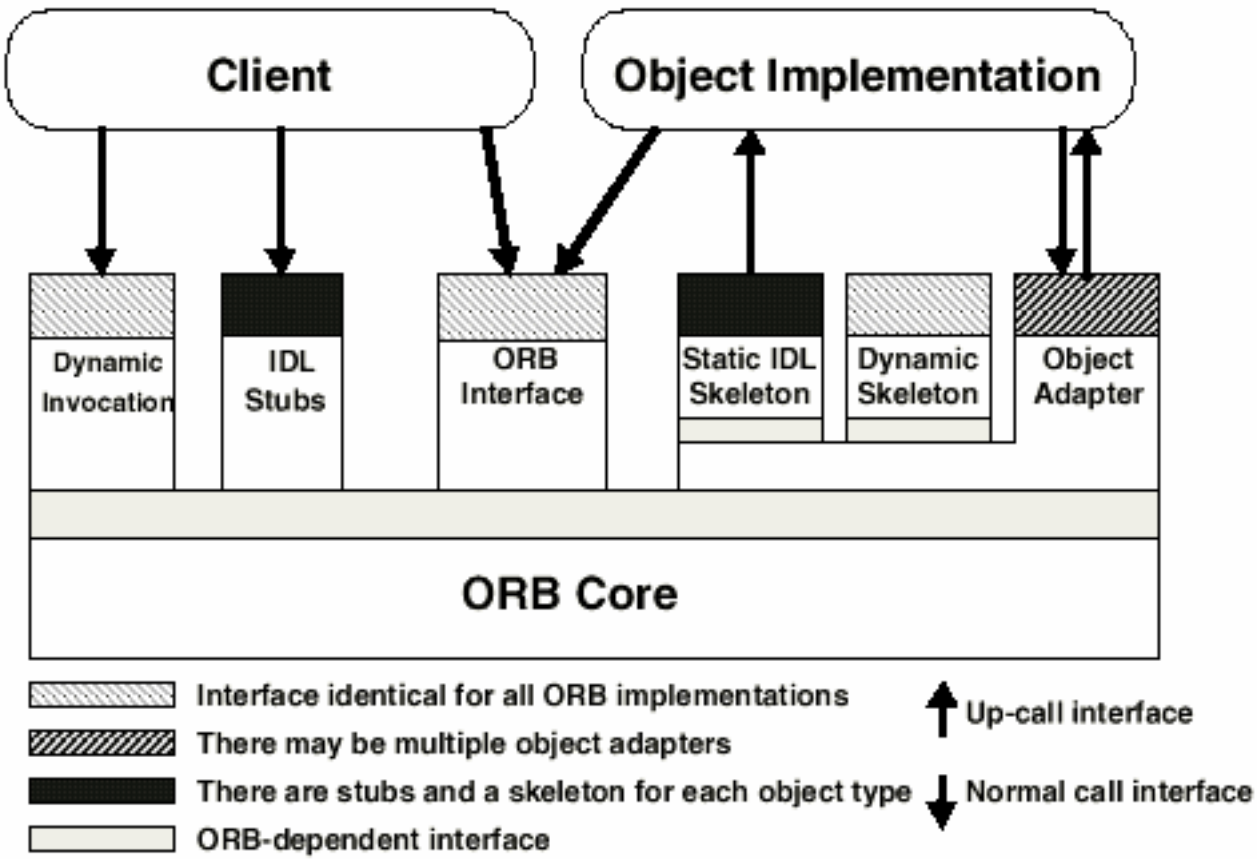
CORBA

¿Cómo funciona?

- **El servidor**
  - Crea objetos remotos
  - Hace accesibles refs a objetos remotos
  - Espera a que los clientes invoquen a estos objetos remotos o a sus métodos
- **El cliente**
  - Obtiene una referencia de uno o más objetos remotos en el servidor
  - Invoca a sus métodos

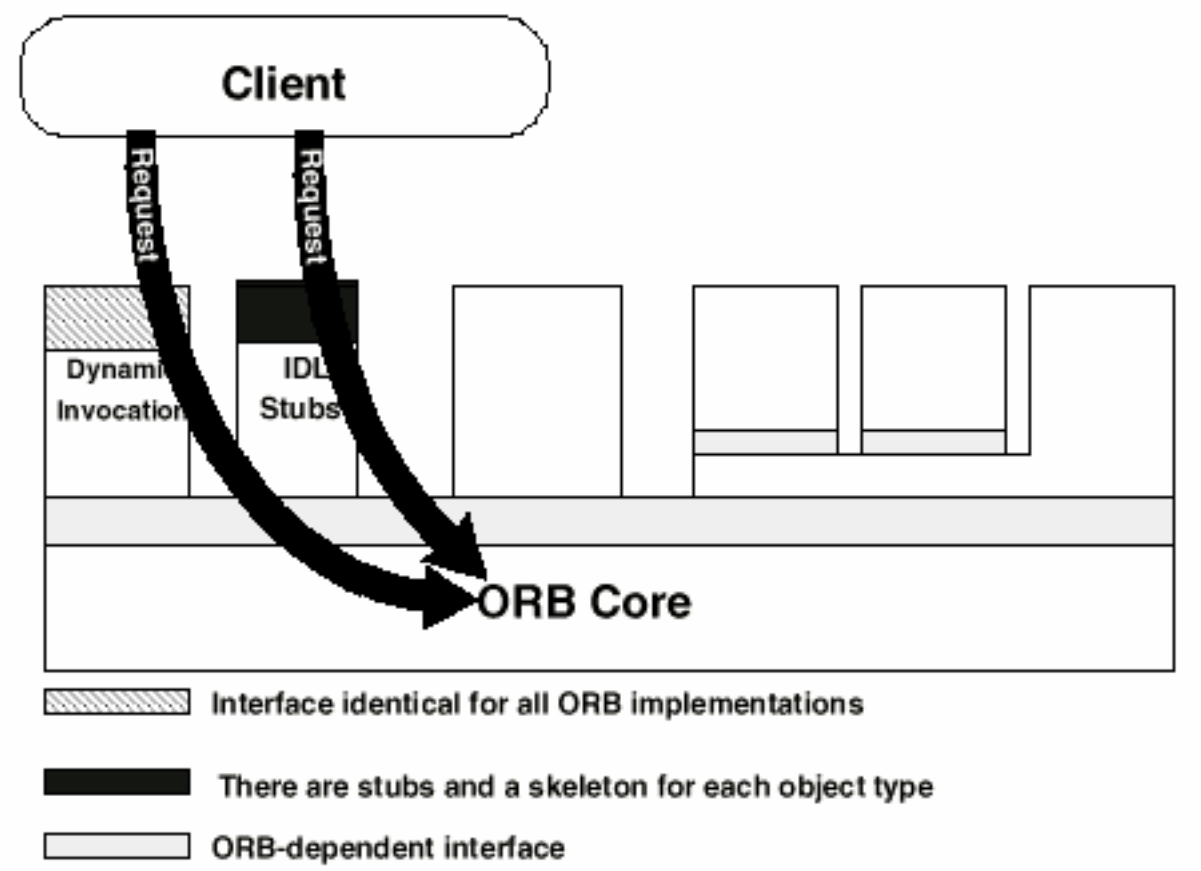
CORBA ¿Cómo funciona?

Elementos de la arquitectura



CORBA ¿Cómo funciona?

¿Cómo se realiza la invocación?



CORBA ¿Cómo funciona?

¿Cómo se realiza la invocación?

- El cliente
  - ¿Qué necesita?
    - Referencia del objeto remoto IOR
    - Tipo del objeto
    - Nombre de la operación que se desea invocar
  - ¿Cómo realiza la invocación?
    - Usando stub generado a partir del IDL
    - Usando invocación dinámica a través de DII
  - Tiene que gestionar las excepciones producidas por su llamada
- El ORB a partir de la petición del cliente
  - Encuentra el código de la implementación apropiada,
  - Transmite los parámetros
  - Transfiere el control a la Implementación de la Interfaz a través de:
    - esqueleto IDL,
    - esqueleto dinámico (DII)
  - Informa de excepciones en el proceso (ej referencias IOR o IDL no válidas)

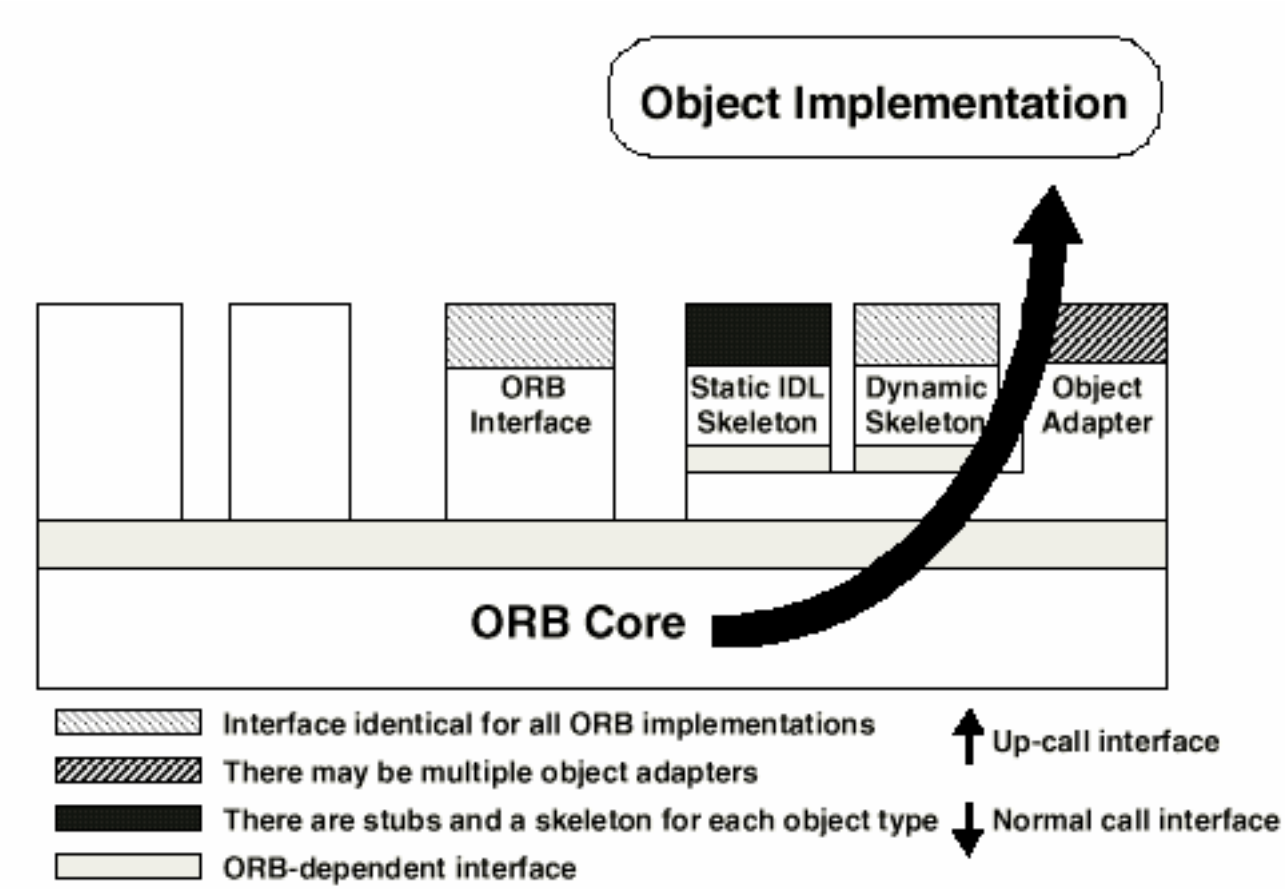
CORBA ¿Cómo funciona?

¿Cómo se realiza la invocación?

- Para el desarrollador es una invocación corriente
- En tiempo de ejecución la invocación pasa por:
  - El stub del cliente
  - El ORB
  - El adaptador de objetos
  - Encontrar el objeto adecuado
  - Viajar por los skeletons del servidor
  - Realizar la invocación sobre el objeto
  - Realizar el mismo camino de vuelta

CORBA ¿Cómo funciona?

¿Cómo se recibe la petición?



CORBA ¿Cómo funciona?

¿Cómo se recibe la petición?

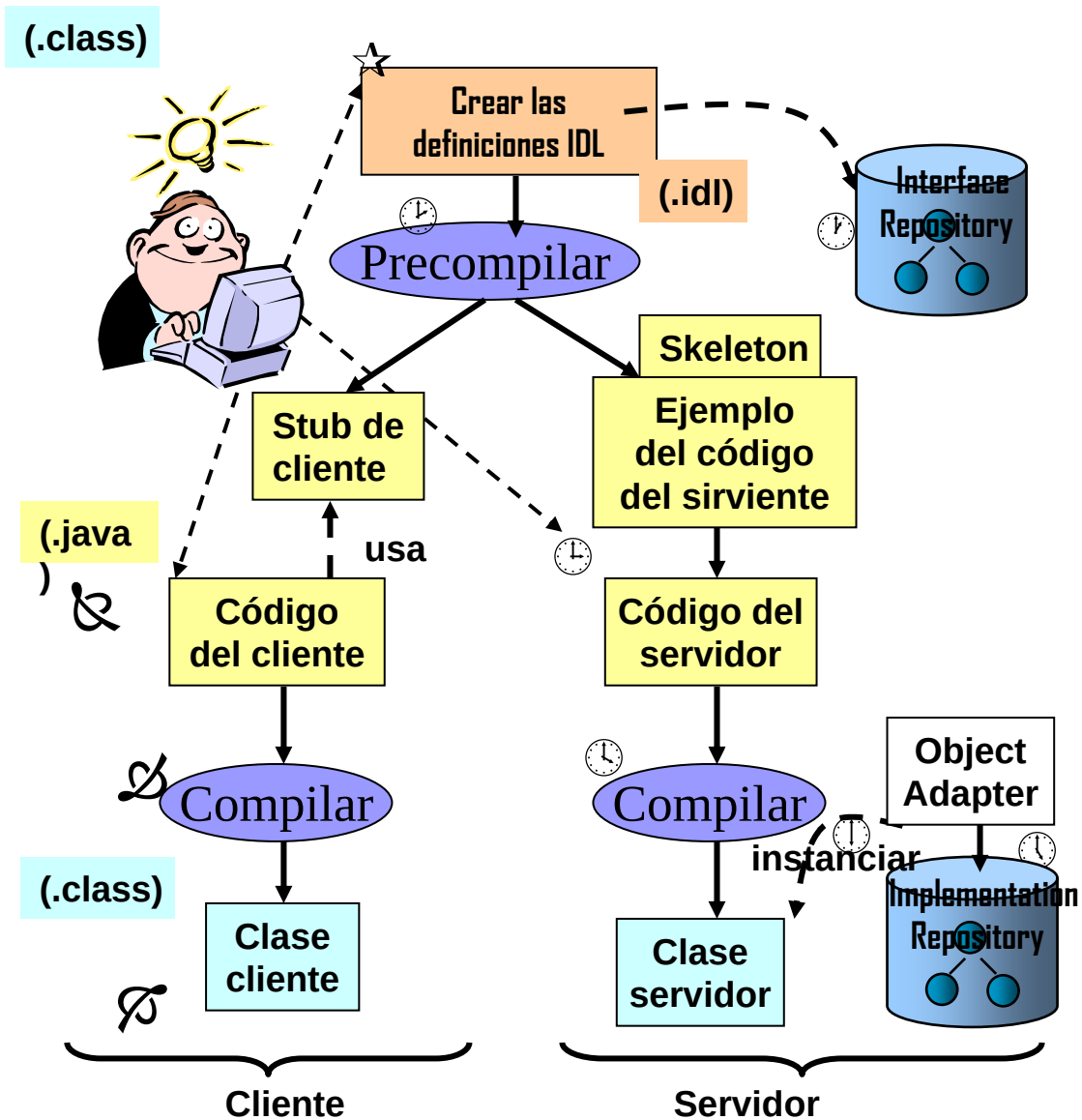
- La implementación del objeto que proporciona el servicio
  - Recibe la invocación de uno de sus métodos como llamadas desde el ORB hacia la Implementación de la Interfaz.
    - La llamada puede venir de un cliente :
      - que ha utilizado los stubs IDL,
      - Que ha utilizado DII
    - Los esqueletos de la interfaz IDL son específicos de cada interfaz y del adaptador de objetos que exista en la implementación de CORBA.
  - Cuando la invocación ha sido completada, el control y los valores de retorno son devueltos al cliente.
  - Puede utilizar los servicios que proporciona el adaptador de objetos e incluso los que proporciona el ORB, mientras procesa la petición que ha recibido del cliente
    - Puede elegir un Adaptador de Objetos entre un conjunto de ellos, una decisión que estará basada en la clase de servicios que pueda requerir la Implementación de la Interfaz
    - Inicialmente OMG propuso como adaptador de objetos el BOA, pero debido a sus limitaciones los fabricantes introducían muchas extensiones propietarias.
    - Para solucionarlo en CORBA 2.2 se introdujo POA, el adaptador de objetos estándar dentro de CORBA 2.2.

CORBA ¿Cómo se usa?

Metodología de desarrollo

- Diseñar el servicio y crear interfaz IDL
- Elegir plataforma y lenguaje de programación
- Conseguir herramienta CORBA para esa plataforma y ese lenguaje.
- La herramienta CORBA debe tener un compilador IDL que, a partir de la interfaz IDL, genere los stubs que permitan enganchar la implementación de la interfaz con la arquitectura CORBA.
- Implementar la interfaz IDL en el lenguaje elegido.
- Crear un servidor CORBA que se encargue de registrar la nueva funcionalidad en CORBA, utilizando la implementación realizada por nosotros y los stubs generados a partir de la interfaz IDL.
- Los nuevos servicios ya están disponibles en CORBA a la espera de la llegada de clientes que los quieran utilizar.

CORBA ¿Cómo se usa?



Proceso de desarrollo

1. Definir los objetos del servidor mediante **IDL.**
2. Cargar la información de IDL en el **Interface Repository**
3. Pre-compilar el **IDL** para generar *stubs* y esqueletos.
4. Escribir el código del servidor, heredando de los esqueletos (y siguiendo el ejemplo)
5. Compilar los ficheros servidores.
6. Registrar los objetos en tiempo de ejecución en el **Implementation Repository**
7. Instanciación de los sirvientes (por el adaptador, normalmente bajo demanda)
8. Escribir el código del cliente, (incluye los *stubs* generados)
9. Compilar el cliente
10. Ejecutar servidor, luego cliente

CORBA Ejemplo 1

Paso 1: Interfaz en IDL

```
// Descripcion de una excepcion
exception DivisionPorCero{
  float op1;
  float op2;
};
```

```
interface Calculator{
  // operacion de Suma
  float add ( in float nb1, in float nb2 );

  // operacion de Division
  float div ( in float nb1, in float nb2 ) raises ( DivisionPorCero );
};
```

(*) Ejemplo tomado de http://www.programacion.com/tutorial/acscorba

CORBA Ejemplo 1

Paso 2: Implementación de calculator

```
public class CalculatorPOAImpl extends CalculatorPOA{
  public float add(float nb1, float nb2){
    System.out.println("Suma = "+nb1+" + "+nb2);
    return nb1 + nb2;
  }
  public float div(float nb1, float nb2) throws
     DivisionByZero {
    System.out.println("Division = "+nb1+" / "+nb2);
    if ( nb2 == 0 ){
            throw new DivisionPorCero(nb1,nb2);
    }
    return nb1 / nb2;
  }
}
```

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

- Inicializar ORB
- Contactar con el objeto remoto
  - Contactar con serv de nombres y obtener ref a él
  - Construir el nombre del objeto en el servidor de nombres
  - Obtener ref a objeto remoto en el servidor de nombres
  - Narrow de objeto genérico Corba al tipo específico del objeto remoto
- Utilizar el objeto remoto como si fuese local

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    org.omg.CORBA.Object obj = null;
    org.omg.CosNaming.NamingContext naming = null;
    try{
    }
    org.omg.CosNaming.NameComponent [] name = new
    org.omg.CosNaming.NameComponent[1];
    name[0] = new org.omg.CosNaming.NameComponent();
    name[0].id = "Calculator";
    name[0].kind = "Example";
    (...)
 } // fin del main
} // fin de la clase
```

1) Inicializar el ORB

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

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try{
            obj = orb.resolve_initial_references("NamingService");
            naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
      }catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
            System.out.println("No se ha podido obtener el NamingService");
            System.exit(0);
      }
      org.omg.CosNaming.NameComponent [] name = new
      org.omg.CosNaming.NameComponent[1];
```

2) Localizar el naming Service y obtener ref a él:

```
obj = orb.resolve_initial_references("NamingService");
naming = org.omg.CosNaming.NamingContextHelper.narrow(obj)
```

```
} // fin de la clase
```

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

3) Construir nombre del objeto calculator:

```
org.omg.CosNaming.NameComponent [ ] name =
                    new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "Calculator";
name[0].kind = "Example";
```

```
pu
    l
            naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        }catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
              System.out.println("No se ha podido obtener el NamingService");
              System.exit(0);
        }
        org.omg.CosNaming.NameComponent [] name = new org.omg.CosNaming.NameComponent[1];
        name[0] = new org.omg.CosNaming.NameComponent();
        name[0].id = "Calculator";
        name[0].kind = "Example";
        (...)
    } // fin del main
} // fin de la clase
```

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
  (...)
      try{
            obj = naming.resolve(name);
      }catch ( org.omg.CosNaming.NamingContextPackage.NotFound ex ){
            System.out.println("Objeto no encontrado en el NamingService");
            System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
            System.out.println("No se ha podido continuar");
            System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
            System.out.println("Nombre invalido");
            System.exit(0);
      }
 (...)
```

4) Localizar ref del objeto calculator en el naming Service:
obj = naming.resolve(name);

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
(...)
    Calculator calc = CalculatorHelper.narrow(obj);
    try{
            System.out.println("5 + 3 = " + calc.add(5,3) );
            System.out.println("5 / 0 = " + calc.div(5,0) );
    }
    catch ( DivisionPorCero ex ){
            System.out.println("Interceptada intento de divisón por cero");
            System.out.println(ex.getMessage());
    }
 } // fin del main
} // fin de la clase
```

5) Narrow de obj genérico Corba al objeto que buscamos:

```
Calculator calc = CalculatorHelper.narrow(obj);
```

CORBA Ejemplo 1

Paso 3: Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
(...)
    Calculator calc = CalculatorHelper.narrow(obj);
    try{
            System.out.println("5 + 3 = " + calc.add(5,3) );
            System.out.println("5 / 0 = " + calc.div(5,0) );
    }
    catch ( DivisionPorCero ex ){
            System.out.println("Interceptada intento de divisón por cero");
            System.out.println("La division era "+ex.op1+" / "+ex.op2);
    }
    catch ( org.omg.CORBA.SystemException ex ){
            System.out.println("Interceptada una excepcion CORBA System");
            System.out.println(ex.getMessage());
    }
```

6) Uso de la ref objeto remoto como si fuera local:

```
System.out.println("5+3 = " + calc.add(5,3));
System.out.println("5/0 = " + calc.div(5,0));
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

- Inicializar ORB (1)
- Contactar con el serv de nombre
  - Contactar con serv de nombr y obtener ref a POA (2)
  - Pasar la referencia CORBA genérica a una específica POA (3)
- Poner objeto remoto disponible en el POA
  - Instanciar objeto remoto (4)
  - Activar el objeto remoto en el orb y obtener id (5)
  - Obtener referencia a objeto remoto a partir del id (6)
  - Registrar el nombre del objeto en el servidor de nombres (7)
  - Enlazar en el servidor de nombres la ref a objeto remoto con su nombre (8)
  - Activar gestor de invocaciones del POA (9)
- Pasar el control al ORB para que acepte peticiones (10)

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
     org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
     org.omg.CORBA.Object objPoa = null;
     org.omg.PortableServer.POA rootPOA = null;
     try{
     objPoa = orb.resolve_initial_references("RootPOA");
     }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
     rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
     CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

1) Inicializar el ORB

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

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

2) Resolver referencia al adaptador de objetos raíz RootPOA

```
objPoa = orb.resolve_initial_references("RootPOA");
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

3) Narrow de ref genérica a ref de adaptador (POA)

```
org.omg.PortableServer.POA rootPOA = null;
rootPOA =
org.omg.PortableServer.POAHelper.narrow(objPoa);
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

4) Crear el objeto Calculator

```
CalculatorPOAImpl calc = new CalculatorPOAImpl();
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
 (...)
    try {
      byte[] servantId = rootPOA.activate_object(calc);
      org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try {
        obj = orb.resolve_initial_references("NamingService");
        System.out.println("Localizado el NamingService");
        naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        System.out.println("Narrow del NamingService");
      } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
 } // fin del main
} // fin de la clase
```

5) Activar el servidor dentro del ORB

```
byte[] servantId = rootPOA.activate_object(calc);
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
 (...)
    try {
      byte[] servantId = rootPOA.activate_object(calc);
      org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try {
        obj = orb.resolve_initial_references("NamingService");
        System.out.println("Localizado el NamingService");
        naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        System.out.println("Narrow del NamingService");
      } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
```

6)Obtener la referencia al servidor

`org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);`

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

7) Acceder al Naming Service

```
obj = orb.resolve_initial_references("NamingService");
naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
```

```
        byte[] servantId = rootPOA.activate_object(calc);
        org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
        org.omg.CORBA.Object obj = null;
        org.omg.CosNaming.NamingContext naming = null;
        try {
          obj = orb.resolve_initial_references("NamingService");
          System.out.println("Locallizado el NamingService");
          naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
          System.out.println("Narrow del NamingService");
        } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
          System.out.println("No se ha podido obtener el NamingService");
          System.exit(0);
        }
 (...)
 } // fin del main
} // fin de la clase
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
       if ( naming == null ) {
         System.out.println("No se ha encontrado el NamingService");
         System.exit(0);
       }
      org.omg.CosNaming.NameComponent [] name = new
      org.omg.CosNaming.NameComponent[1];
      name[0] = new org.omg.CosNaming.NameComponent();
      name[0].id = "Calculator";
      name[0].kind = "Example";
 (...)
```

7) Construir nombre del objeto calculator: *(=que 2 de cliente)*

```
org.omg.CosNaming.NameComponent [ ] name =
                  new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "Calculator";
name[0].kind = "Example";
```

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
      try{
        naming.bind(name,ref);
      } catch ( org.omg.CosNaming.NamingContextPackage.NotFound ex ){
        System.out.println("Objeto no encontrado");
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.AlreadyBound ex ){
        System.out.println("Ya hay un objeto con ese nombre");
        naming.unbind(name);
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
        System.out.println("Nombre inválido");
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
        System.out.println("No se ha podido continuar");
        System.exit(0);
      }
```

8) Enlazar ref del objeto calculator con su nombre
`naming.bind(name,ref);`

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
     (   )
     rootPOA.the_POAManager().activate();
      System.out.println("El servidor está preparado...");
      orb.run();
    } // fin del try
    catch ( java.lang.Exception ex )
    {
      System.out.println("Se ha capturado una excepción");
      ex.printStackTrace();
    }

  } // fin del main
} // fin de la clase
```

9) Activar gestor de invocaciones del POA
`rootPOA.the_POAManager().activate();`

CORBA Ejemplo 1

Paso 4: Desarrollo del servidor

```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
      rootPOA.the_POAManager().activate();
      System.out.println("El servidor está preparado...");
      orb.run();
    } // fin del try
    catch ( java.lang.Exception ex )
    {
      System.out.println("Se ha capturado una excepción");
      ex.printStackTrace();
    }

  } // fin del main
} // fin de la clase
```

10) Ceder control a ORB para que escuche peticiones
`orb.run();`

CORBA Ejemplo 1

Paso 5: Compilar la aplicación

- Compilar interfaces
- Compilar el servidor
- Generar stubs y skeletons
- Compilar la clase cliente

CORBA Ejemplo 1

Paso 6: Ejecutar la aplicación

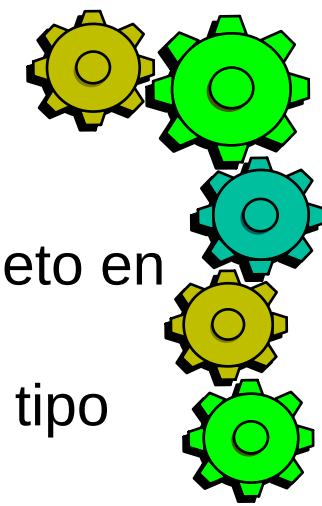
- Arrancar ORB
- Arrancar Servidor de Nombres
- Arrancar el Servidor
- Ejecutar el cliente

Object Management Architecture

Modelo de objetos

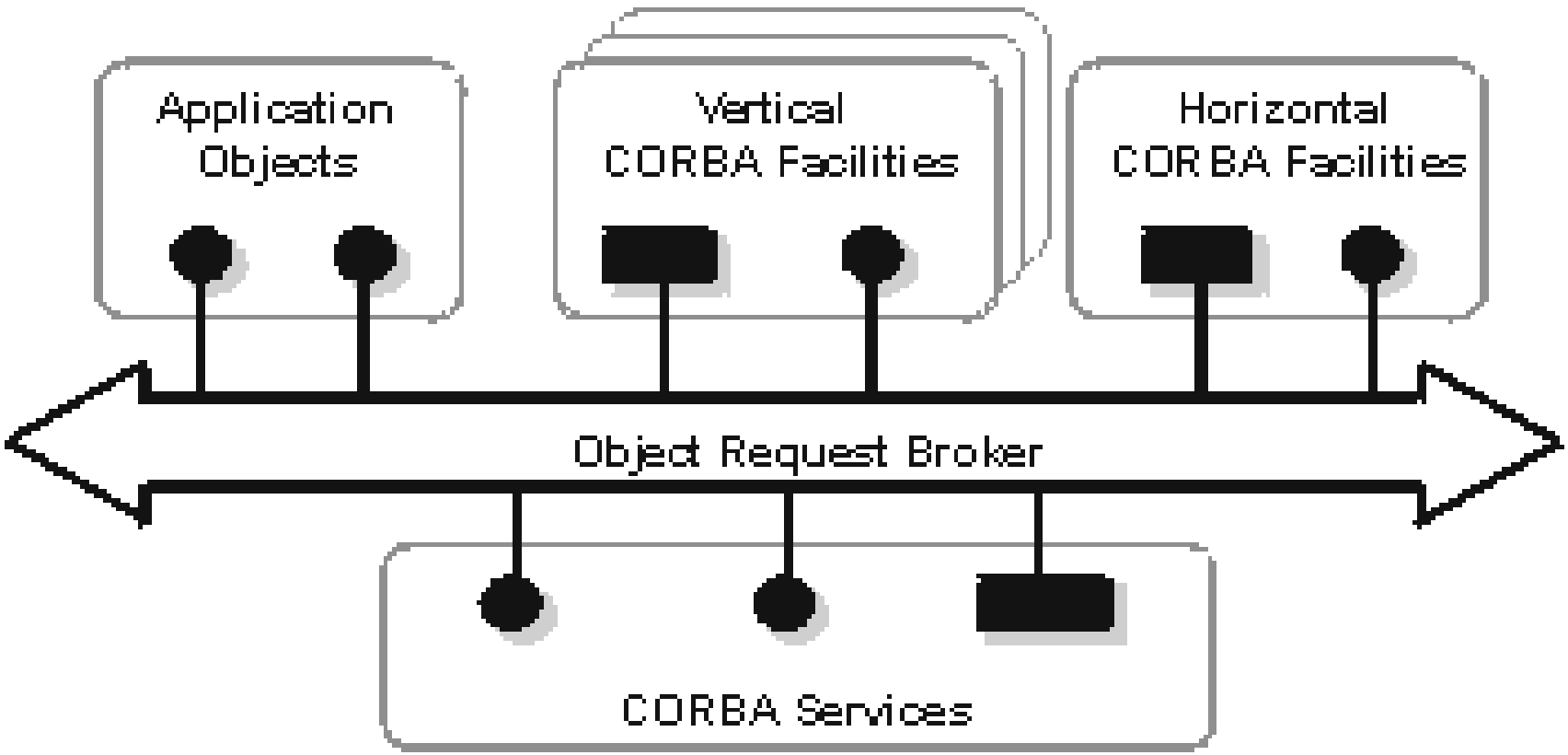
Un *sistema de objetos* de servicios a clientes:

- Cliente. Entidad capaz de hacer *petición* de un servicio (a un *objeto*).
- Objeto. Entidad identificable y encapsulable que da servicios.
- Petición. Evento que tiene asociado operación, parámetros, *target object*, contexto.
  - Formato de una petición se define por:
    - Un lenguaje de "*binding*"
    - Por una llamada de invocación dinámica
- Referencia de un Objeto. Nombre que denota a un objeto en particular.
- Parámetros. Identificados por posición (pueden ser de tipo entrada, salida o entrada-salida)

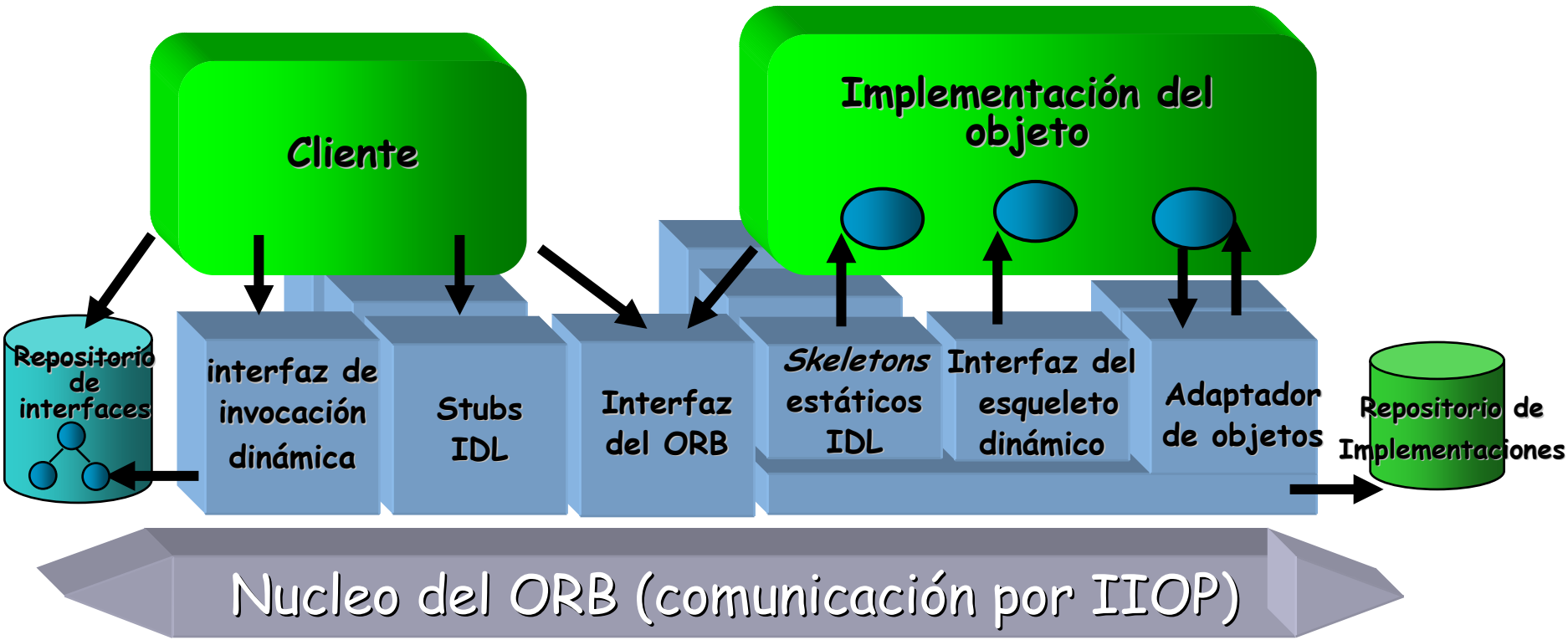


Object Management Architecture

Arquitectura del modelo de referencia



CORBA
Arquitectura CORBA

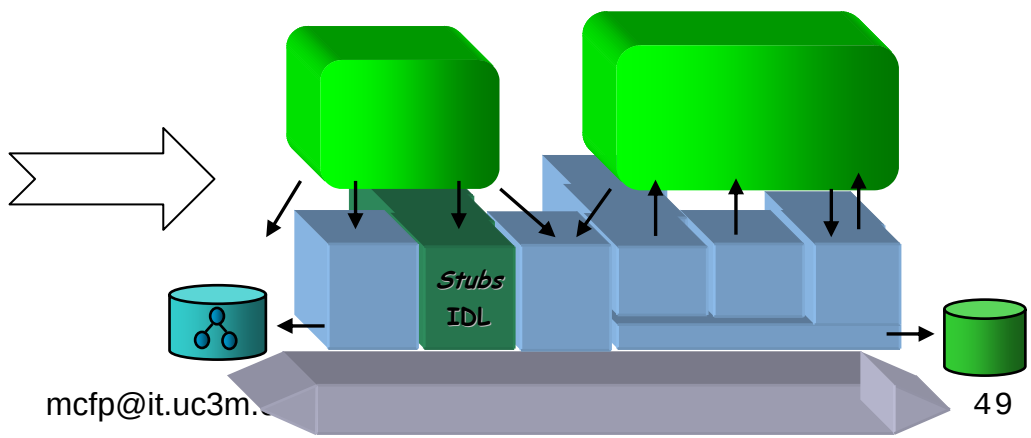


CORBA

Lado cliente (1/6)

- *Stub* IDL
  - interfaz estática a los servicios declarados en las interfaces IDL
  - para el cliente todas las llamadas parecen locales.
  - actúa como *proxy* (representante) del objeto remoto, realizando
    - la invocación de métodos remotos, incluyendo el *marshalling*
    - la recepción de respuestas, incluyendo el *unmarshalling*
  - el cliente puede tener tantos *stubs* como interfaces IDL existan.
  - generado a partir del IDL en el lenguaje de programación del cliente (C, C++, Java, Smalltalk, ADA,…) por un compilador IDL.

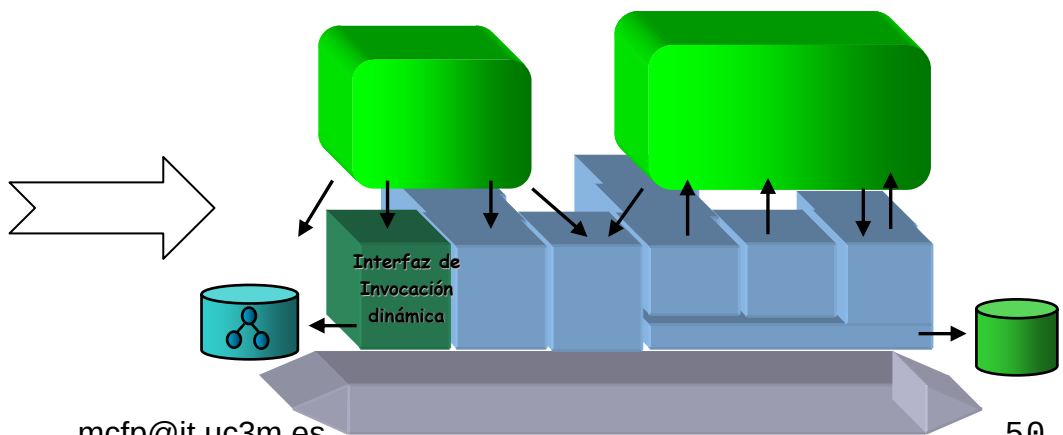
Estructura de CORBA ORB



CORBA
Lado cliente (2/6)

- Interfaz de Invocación Dinámica (DII)
  - utilizado para hacer una invocación dinámica después de haberla construida mediante la interfaz del ORB
  - un solo DII para todas las instancias de todos los tipos
  - el ORB hace el *marshalling*, no la DII

Estructura de CORBA ORB



CORBA

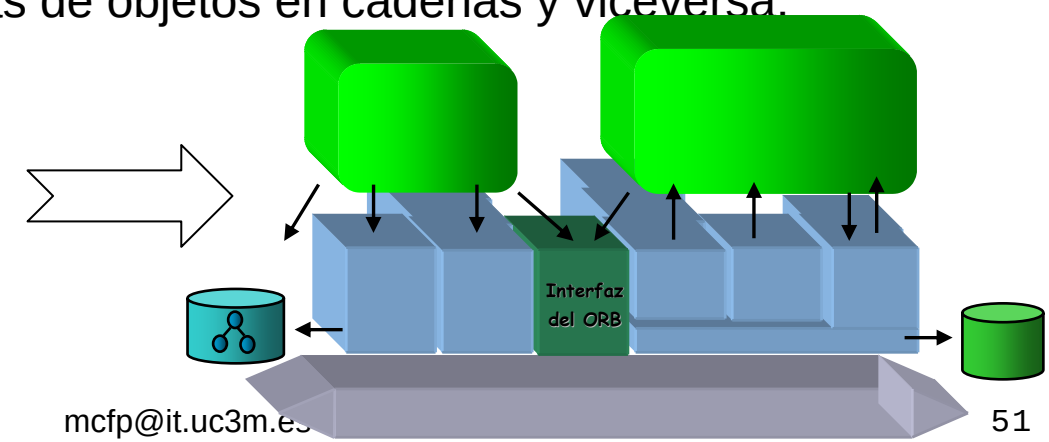
Lado cliente (3/6)

- La interfaz ORB

  Conjunto de librerías o APIs que definen funciones del ORB y que pueden ser accedidas por el código cliente.

  - Acceso a servicios iniciales como *naming* o *trader*
  - Acceso al repositorio de interfaces
    - recuperar la información sobre una interfaz (metadatos)
    - permite descubrir en *run-time* los métodos que ha de ser invocados
  - Acceso a métodos para la construcción de invocaciones dinámicas
    - e.g. para construir listas de argumentos
  - Conversión de referencias de objetos en cadenas y viceversa.

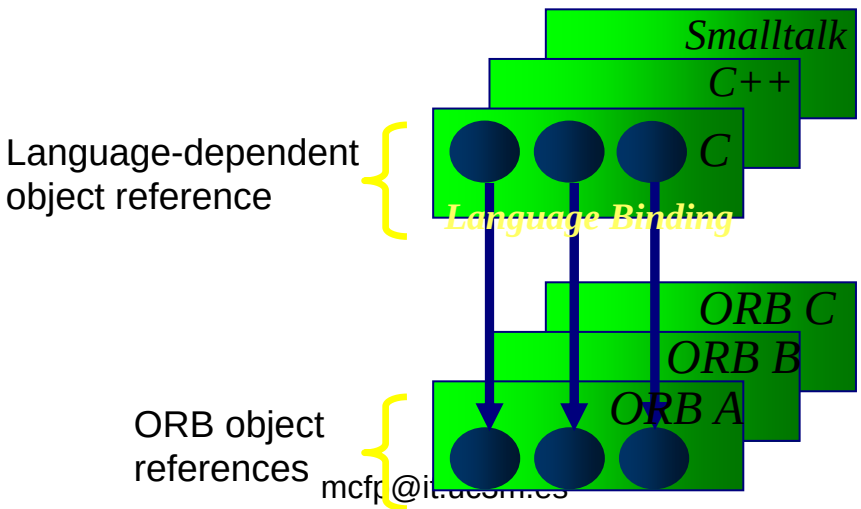
Estructura de CORBA ORB



CORBA

Lado cliente (4/6)

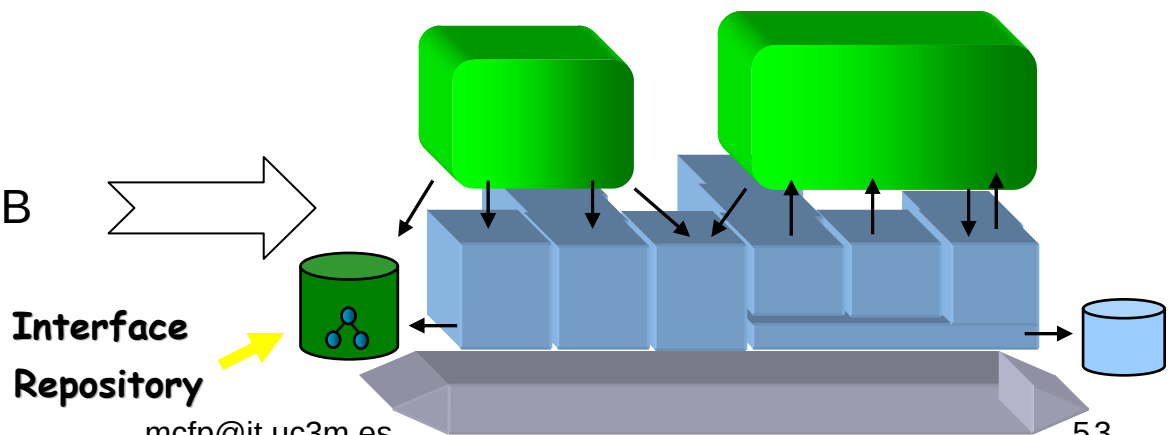
- Referencia de Objetos
  - Información necesaria para un cliente que desea interoperar con un *target object* a través del ORB.
  - Para evitar conflictos entre ORB's de diferentes vendedores CORBA 2.0 define los IOR's (*Interoperable Object References*).
  - *Target Object* es el objeto al cual un cliente hace una petición, la implementación de dicho objeto no es definida por CORBA.



CORBA desde el punto de vista del cliente (5/6)

- Repositorio de Interfaces (IR) (1/2)
  - Base de datos en tiempo de ejecución con versiones máquina de las interfaces IDL (y posiblemente información asociada).
  - Permite obtener y modificar dinámicamente las descripciones (interfaces, métodos y parámetros) de todos los objetos registrados.
  - Es un mecanismo de auto-descripción (metadatos) de los objetos
  - IRs de distintos ORBs pueden formar “federaciones”.
  - El IR en si es un objeto más, accesible a través de un ORB.

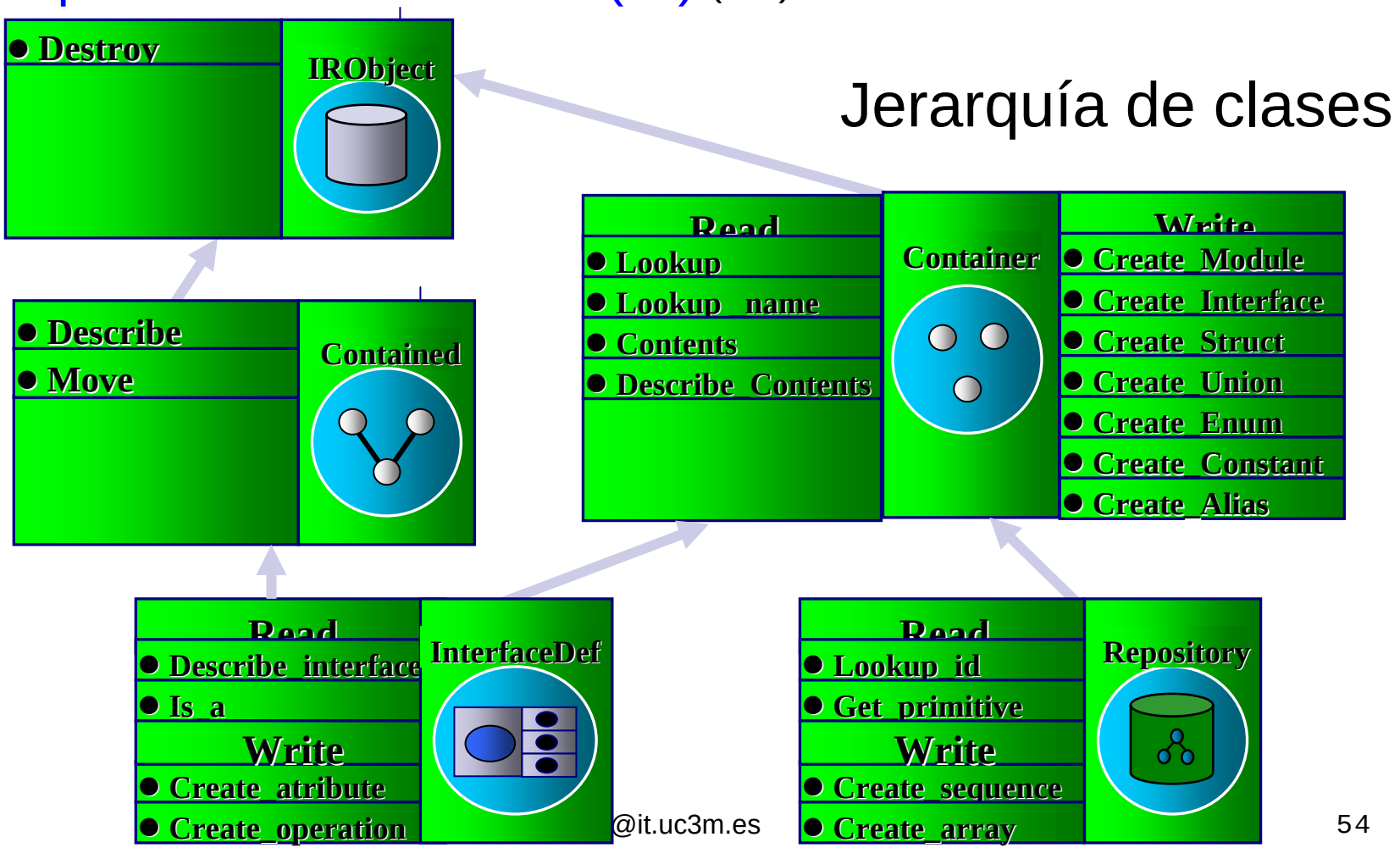
Estructura de CORBA 2.0 ORB



CORBA

Lado cliente (6/6)

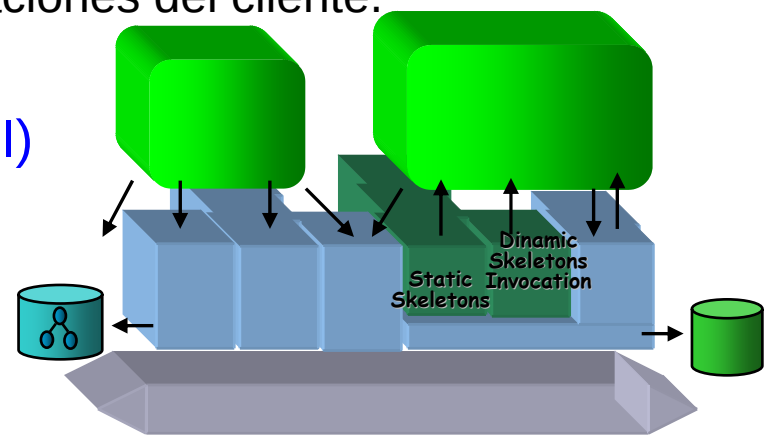
- Repositorio de Interfaces (IR) (2/2)



CORBA

Lado servidor (1/2)

- *Skeleton* estático IDL
  - Representante estático del cliente en el servidor.
  - Para el servidor todas las llamadas parecen locales.
  - Generado a partir del IDL por un compilador IDL.
  - Realiza el *unmarshalling* de las invocaciones del cliente.

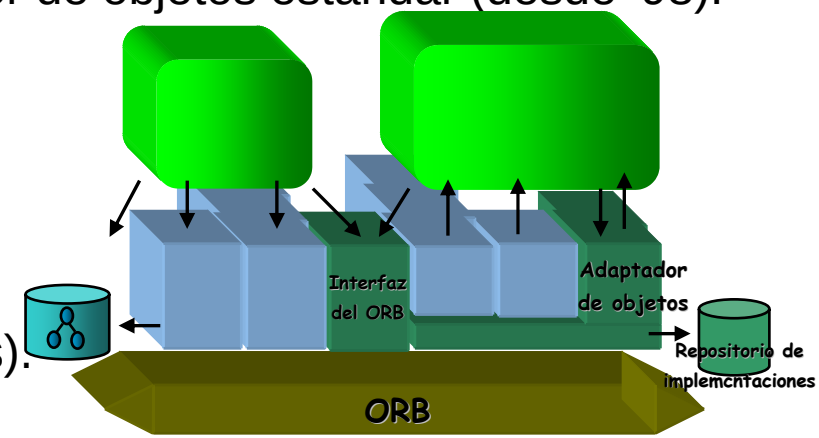


- Interfaz de esqueletos dinámicos (DSI)
  - Da mecanismos de asociación en *run-time* a servidores que necesitan manejar peticiones de componentes que no tienen *stubs*.
  - Recibe las peticiones y averigua a que objetos van dirigidas.
  - Son muy utilizados en la construcción de "*Bridges*" entre **ORB's**.
  - Se pueden usar para generar dinámicamente objetos.
  - Es el equivalente en el servidor al **DII** del cliente.

CORBA

Lado servidor (2/2)

- El Adaptador de objetos
  - Intermediario entre el núcleo del ORB y los objetos
  - Instancia y activa objetos servidores, y crea referencias de objeto
  - Mapea referencias de objetos a IDs de implementaciones de objetos
  - Invoca el método apropiado cuando llega una petición remota (*upcall*).
  - Registra objetos en el repositorio de implementaciones.
  - **POA** (*Portable Object Adapter*): adaptador de objetos estándar (desde '98).
- El Repositorio de Implementaciones
  - Proporciona un registro en *run-time* de las clases, los objetos instanciados y los ID's que un servidor soporta.
  - También puede almacenar otro tipo de información (trazas, datos administrativos).



- La interfaz ORB
  - API que da soporte local, idéntico al proporcionado en el lado del cliente.

CORBA
Adaptadores de objetos estándarizados

- ***B**asic **O**bject **A**dapter* (BOA, 1990)
  - sub-especificado ⇒ el API ofrecido a los sirvientes depende del vendedor del ORB

- ***P**ortable **O**bject **A**dapter* (POA, 1998)
  - propósito principal: **portabilidad**
    - código del servidor independiente de la implementación del ORB
  - estándariza un control más fino del entorno de ejecución del servidor
    - acercar el adaptador de objetos a la noción de contenedor (pero gestión de transacciones, ciclo de vida etc.: *CORBAServices*)

CORBA
Funcionalidades del POA

- Abre los mecanismos internos del servidor a los programadores
  - da soporte flexible a la persistencia y activación de objetos
- Relaciona referencias de objeto, IDs de objeto y sirvientes
- Espacio de nombres para los sirvientes
- Introduce la noción de Gestor de Sirvientes (*Servant Manager*)
  - se ocupa de la creación, activación y desactivación de sirvientes
  - para un control más fino, el usuario puede definir su propio gestor
    - se puede definir un gestor por cada implementación de objeto CORBA
  - interfaz **`ServantActivator`**
    - para gestionar objetos persistentes
  - interfaz **`ServantLocator`**
    - para gestionar objetos transitorios (no persistentes, fugaces)

CORBA
Activación con el POA

- Recordatorio: la implementación de un objeto CORBA
  - sirviente
    - una instancia de un objeto CORBA ejecutándose
  - ID de objeto
    - referencia utilizado por el POA y la implementación de objetos para identificar a un objeto CORBA (y restaurar su estado cuando se activa)
  - referencia de objeto
    - referencia utilizado por un cliente CORBA que encapsula una dirección del nivel transporte, un ID del adaptador de objetos y un ID de objeto
- POA: relación flexible entre IDs de objeto y sirvientes
  - permite que
    - peticiones a distintos objetos sean tratados por el mismo sirviente
    - peticiones sucesivas al mismo objeto sean tratados por distintos sirvientes
  - eficiencia y escalabilidad
    - pooling etc.

CORBA
Políticas del POA

- ***Lifespan***: indica si las referencias de objeto creados por el POA son transitorios o persistentes (nota: **no** se deberían registrar objetos transitorios en el servicio de nombres).
  Por defecto: TRANSIENT
- ***Servant Retention***: indica si el POA mantiene los sirvientes en memoria y almacena la correspondencia entre IDs de objetos y sirvientes en el "mapa de objetos activos" o no
  Por defecto: RETAIN
- ***ID Assignment***: indica si es el POA o la implementación del objeto quien asigna los IDs de objeto
  Por defecto: SYSTEM_ID
- ***ID Uniqueness***: indica si los sirvientes activados por el POA tienen ID de objeto único o no
  Por defecto UNIQUE_ID
- ***Thread***: indica si toda petición es tratada por el mismo hilo (SINGLE_THREAD_MODEL) o no necesariamente
  Por defecto: ORB_CTRL_MODEL
- ***Activation***: indica si un sirviente se activa implícitamente a la hora de crear una referencia de objeto o si debe ser activado explícitamente
  Por defecto: IMPLICIT_ACTIVATION
- ***Request processing***: indica si en caso de no encontrar un objeto en el mapa de objetos activos, se devuelve una excepción (RETAIN obligatorio), o se envía la petición a un gestor de sirvientes (que activará un sirviente) o a un sirviente por defecto; en el caso NO_RETAIN: no hace falta mirar el mapa
  Por defecto: USE_ACTIVE_OBJECT_MAP_ONLY

CORBA y sus competidores

Competencia a aplicaciones CORBA va en 2 direcciones

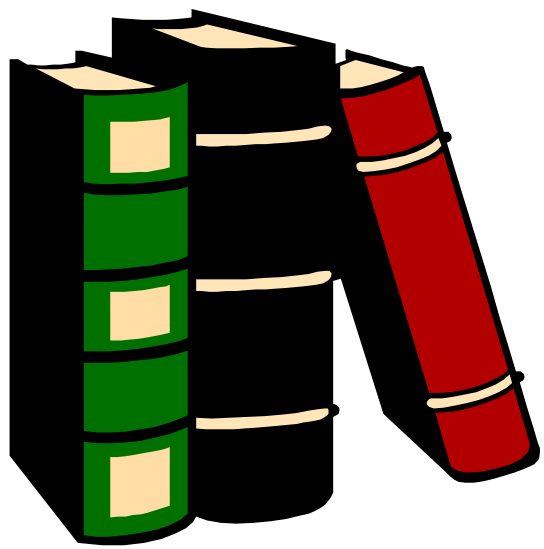
1. Middleware Internet
   - Java Sockets
     - Tecnología sustrato para la programación en red de Java
   - HTTP/CGI, modelo predominante
     - HTTP proporciona una semántica simple tipo RPC sobre sockets
     - CGI proporciona comandos para integrar comandos desde HTTP a una aplicación servidor
     - Servlets, mecanismo parecido a CGI en Java
   - WebServices (competencia más reciente)
2. ORBs no Java/CORBA
   - RMI, ORB nativo de Sun; incluido en el SDK
   - DCOM, ORB industrial, incluido en los S.O. de Microsoft

Algunas referencias

– ***Dan Harkey, Robert Orfali***, ***Client/Server Programming with Java and CORBA, 2nd Edition*** *2nd. Edition (1998) John Wiley & Sons, Inc. ISBN: 0-471-24578-X Mirarse el capítulo 1*

– ***Yang, Z. y Duddy, K. (1996).*** *CORBA: A Platform for Distrbuted Object Computing (A State-of-the-Art Report on OMG/CORBA). ACM Operating System Review. Vol.30, No. 2, Abril 1996*

–***Vinoski, Steve (1997).*** *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine. Febrero 1997.*



Referecias Web

- Tutorial con ejemplos:

  http://www.programacion.com/tutorial/acscorba/3/
- Especificación completa. Página de OMG (Object Management Group)

  http://www.omg.org/