Tema 3. Objetos distribuidos

SCS – Sistemas Cliente/Servidor
4º informática

http://ccia.ei.uvigo.es/docencia/SCS

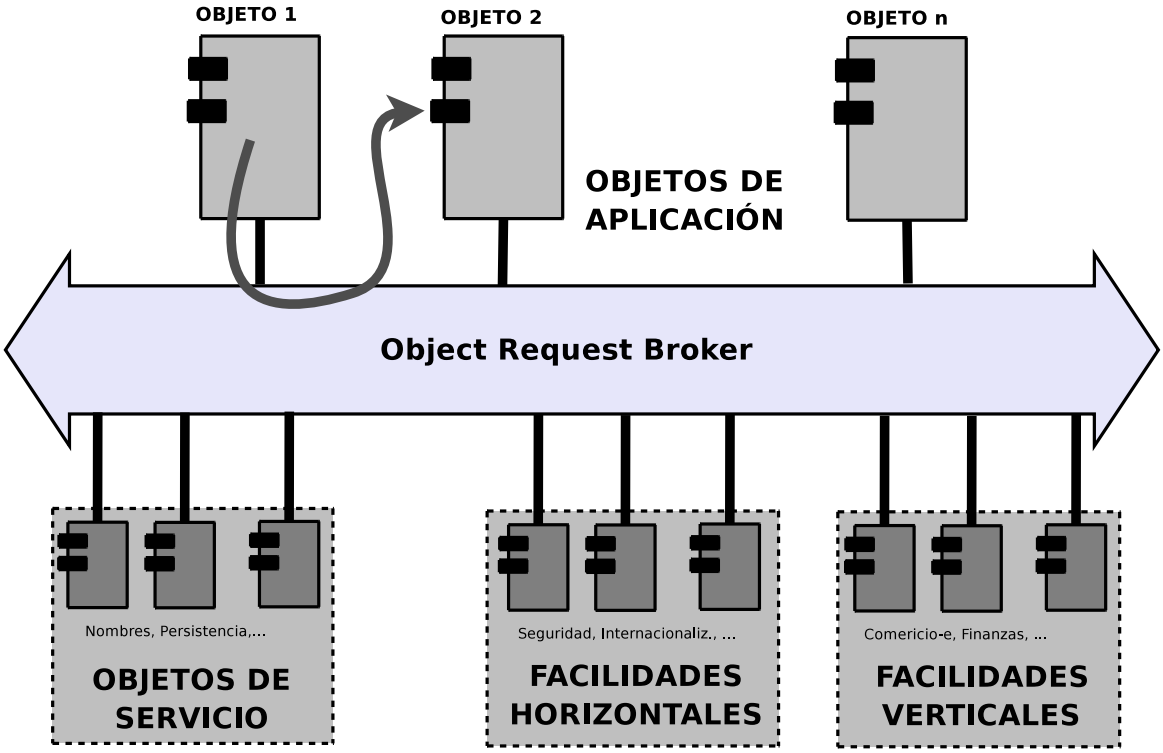
23 de noviembre de 2009

3.3 CORBA

CORBA (*Common Object Request Broker Architecture*): arquitectura común de intermediarios en peticiones a objetos

- *Middleware* que define una arquitectura estándar basada en el modelo de objetos para el desarrllo de aplicaciones distribuidas en entornos heterogéneos
  - distintos
    - hardware + sistemas operativos (Unix, Windows, ...)
    - protocolos de comunicación (TCP/IP, IPX, ...)
    - lenguajes de programación (Java, C, C++, ...)
- Ideas clave
  - Separación entre interfaz e implementación
    - Uso de lenguaje IDL para definir interfaces
    - Independiente del lenguaje usado en la implementación
    - Todos los componentes CORBA se especifican usando IDL
  - Independiencia de localización
    - ORB se encarga de hacer transparente la localización de los objetos CORBA
  - Independencia de fabricante e interoperabilidad
    - CORBA define una especificación, no una implementación
    - ORBs de distintos fabricantes pueden integrarse (GIOP, IIOP)
      - ◇ independencia de vendedor
      - ◇ invocación independiente de si ambas partes están en el mismo ORB o en 2 distintos
- Definido por el consorcio de fabricantes OMG (*Object Management Group*)
  - compuesto por más de 800 compañias
  - dedicado a establecer estándares abiertos para entornos orientados a objetos
  - también define UML (*unified modeling language*) y MDA (*model driven architecture*)
  - `http://www.omg.org`
- OMG ha desarrollado una arquitectura de referencia, OMA (*Object Management Architecture*), de la que forma parte CORBA
  - CORBA especifica los estándares necesarios para la invocación de métodos sobre objetos en entornos heterogéneos
  - Conceptualmente el modelo CORBA ofrece un "*bus software*" al que pueden "conectarse" los objetos del sistema distribuido.

(a) Componentes de la arquitectura de referencia OMA



Object Request Broker (ORB) : núcleo de OMA, *bus* de comunicación de objetos

- Mediador entre los elementos que forman el sistema distribuido
- *Finalidad*: posibilitar la comunicación entre objetos CORBA
  - Permite que cliente interactúe con los métodos de los objetos remotos
- *Objetivo*: proporcionar transparencia
- ORB oculta:
  - ubicación de los objetos
  - implementación de los objetos
  - estado de ejecución del objeto (activación y persistencia)
  - mecanismos de comunicación entre objetos
- *Tareas*: cuando un cliente invoca una operación
  - ORB localiza la implementación del objeto a invocar y lo activa si es necesario
  - transmite la petición y devuelve la respuesta, gestionando el intercambio de datos en un formato independiente
- Basado en el uso de un lenguaje IDL (*Interface Definition Language*)
  - Sólo se definen tipos e interfaces, no implementaciones
  - Existen *mappings* del IDL a diversos lenguajes de implementación (C++, Java, C, Ada, Pascal, Cobol, Perl, Python)
  - Los interfaces del propio ORB están especificados usando el IDL de CORBA
- ORB es un concepto lógico (abstracción)
  - CORBA especifica qué debe hacer el ORB, pero no cómo implementarlo
  - Puede ser una combinación de uno o varios servidores (*daemons*), liberías, etc
- Permite la conexión con otros ORBs

Application Objects (AO): los objetos que se conectan al ORB y conforman la aplicación distribuida

- Definen los servicios específicos de una aplicación

Object Services (OS): colección de servicios ofrecidos a los objetos que complementan la funcionalidad de ORB.

- servicio de nombres, servicio de tiempo, servicio de *Trading*, servicio de eventos, servicio de persistencia, ...

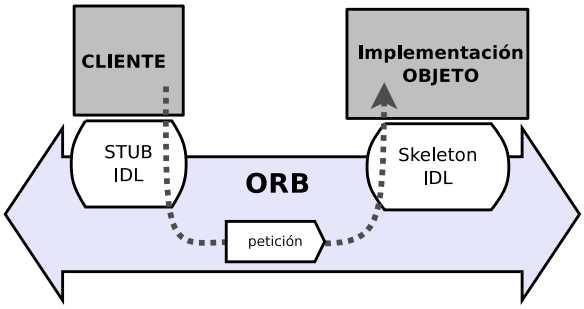
Common Facilities (CF): servicios concretos ofrecidos a las aplicaciones

- *Horizontal Facilities*: facilidades comunes para soportar servicios de seguridad, internacionalización, agentes móviles, impresión, gestión de tareas, ...
- *Vertical Facilities*: facilidades específicas para aplicaciones de dominios concretos: comercio-e, gestión de bases de datos, agentes móviles, negocios, finanzas, ...

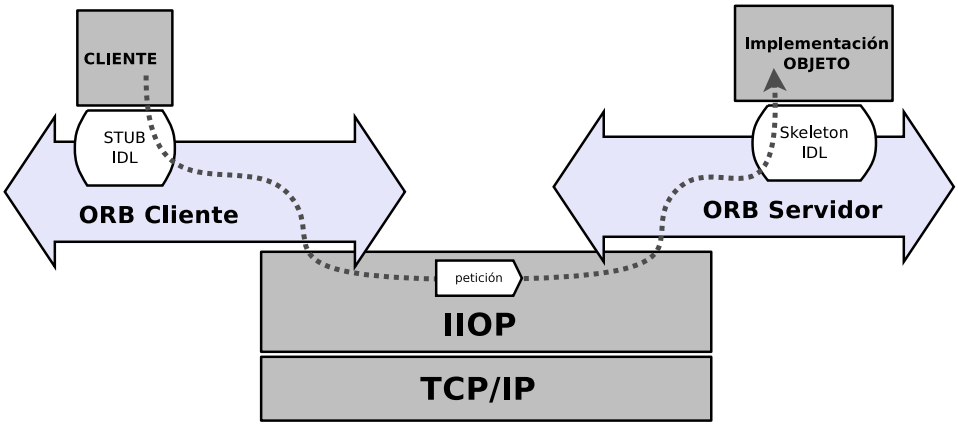
(b) Arquitectura Inter-ORB

- CORBA sólo especifica los elementos necesarios y sus interfaces
  - cada proveedor/empresa ofrecerá sus propias implementaciones
- Evolución de la especificación CORBA (interoperabilidad)
  - CORBA 1.1 (1991): no especificaba la implementación de ORB
    - difícil la comunicación entre ORBs de distintos fabricantes
    - no garantizaba interoperabilidad
  - CORBA 2.0 (1996): establece GIOP+IIOP como protocolo estandarizado para comunicación entre ORBs
    - toda implementación de un ORB conforme a CORBA debe implementarlo
- Protocolos de comunicación entre ORBs
  - GIOP (*General Inter-ORB Protocol*): especificación abstracta de los mecanismos de interacción entre ORBs
    - Conjunto de mensajes a emplear
    - CDR (*Common Data Representation*): Representación externa de Datos Comunes (formato común de intercambio [*marshalling*])
    - IOR (*Interoperable Object Reference*): Referencias a Objetos con soporte para comunicación entre ORBs
  - IIOP (*Internet Inter-ORB protocol*): especifica como implementar GIOP en una red TCP/IP
    - Se usan redes TCP/IP para dar soporte a la comunicación entre ORBs
    - Define el formateo y el intercambio de mensajes sobre sockets TCP

Comunic. en ORB simple



Comunic. entre ORBs



3.3.1 Arquitectura CORBA

(a) Terminología básica de CORBA

objeto CORBA: Entidad virtual sobre la que se pueden hacer invocaciones remotas (no existe si no hay una implementación)

- Tiene asociados { una identidad propia (referencia); un interfaz IDL; una implementación (sirviente); una localización }
- Modelo de objetos del IDL de CORBA no soporta el concepto de clase ($\neq$ C++ y Java)
  - Sólo hay objetos CORBA independientes que implementan determinados interfaces IDL
- Pueden ser implementados en cualquier lenguaje (incluso lenguajes no orientados a objetos)

referencia a un objeto CORBA: Estructura opaca (no accesible) e inmutable que contiene la dirección de un objeto CORBA ($\approx$ puntero distribuido)

- Usada por cliente para comunicarse con implementaciones del objeto CORBA
- Identifica de forma única al objeto CORBA en todo el sistema
- Uso de IOR (*Interoperable Object Reference*): formato portable de referencias a objetos
- IOR contiene:
  - Identificador de tipo (info. sobre interfaz IDL implementada por el objeto)
  - Información del extremo (dirección física: nombre anfitrón + nº puerto)
  - Clave del objeto: identificador único dentro del ORB
    - Nombre del POA (adaptador objetos) + nombre del objeto en POA
- Puede haber múltiples referencias a un objeto CORBA
- Se puede convertir en una cadena de caracteres y viceversa
  - util para transferir y almacenar IOR

sirviente (servant): Entidad del lenguaje de programación que implementa uno o más objetos CORBA

- Define (implementa) las operaciones que soporta un determinado interfaz CORBA IDL
- En lenguajes OO suele ser una instancia de una clase *sirviente* que implementa las operaciones del interfaz IDL

cliente CORBA: Entidad (componente de la aplicación) que realiza peticiones a objetos CORBA

- No tratan con los objetos CORBA directamente, sino con referencias CORBA
- Cuando un componente de aplicación quiere utilizar un servicio proporcionado por otro componente, primero debe obtener una referencia para el objeto CORBA que proporciona ese servicio
- Clientes pueden obtener referencias a objetos:
  - en la creación y registro de objetos (cliente puede crear un nuevo objeto)
  - a través de consultas al servicio de nombres de CORBA
  - convirtiendo la referencia a objeto en una cadena y recuperándola

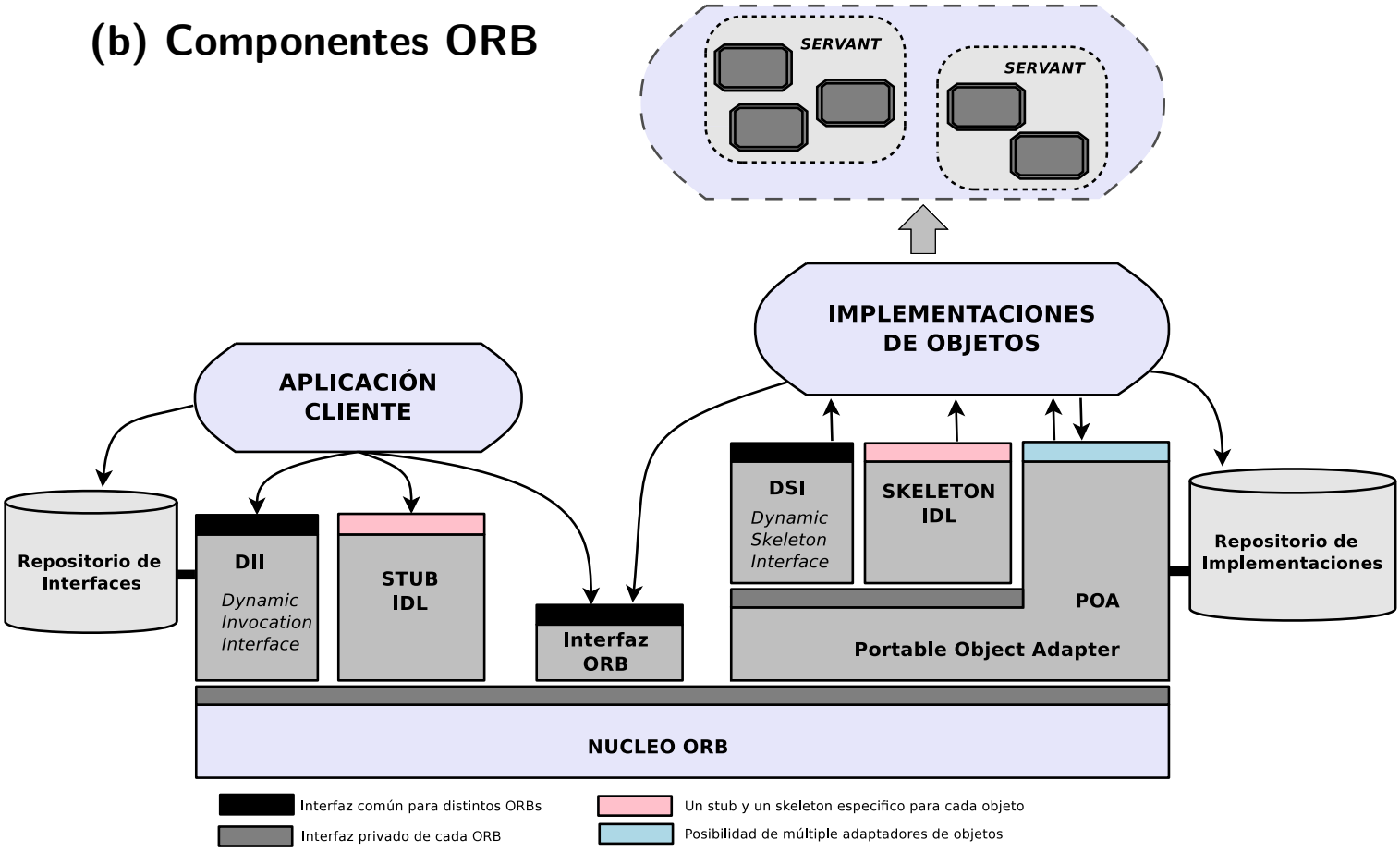
servidor CORBA: Proceso que contiene uno o más objetos CORBA y los pone a disposición de los clientes (registro y activación)

- Contiene implementaciones de los respectivos interfaces IDL

petición CORBA: Encapsula una llamada desde un cliente a un método de un objeto CORBA

- Contiene { referencia al objeto destino (IOR); identificación del método a invocar; parámetros de la llamada }

(b) Componentes ORB



CORBA soporta invocación estática e invocación dinámica

- Objeto CORBA invocado no distingue entre invocaciones estáticas y dinámicas

Invocación estática

- Asume que cliente conoce la interfaz del objeto remoto en tiempo de compilación y que esa interfaz no cambia
  - No permite { utilizar nuevos servicios que puedan aparecer / cambios en el interfaz
  - Suficiente en la mayoría de aplicaciones
- Interfaces estáticas
  - Se definen en *tiempo de compilación* y no cambian
  - Generados automáticamente a partir del interfaz IDL (compilador IDL)
  - Envían y reciben mensajes (*marshalling* y *unmarshalling*)

  Stub IDL: se enlaza con el cliente
  - representación "local" de los métodos remotos
  - recibe las llamadas del cliente
  - existe un *stub* para cada objeto remoto

  Skeleton IDL: se enlaza con el servidor
  - invoca a los métodos de la implementación real (*servant*)
  - existe un *skeleton* para cada objeto remoto

Invocación dinámica

- Mecanismo usado para invocar interfaces no conocidos en tiempo de compilación
- Interfaces dinámicas
  - Generadas a partir de la información del *Repositorio de Interfaces*

  DII (*dynamic invocation interface*) Usado por clientes que no tienen *stub* para un objeto CORBA determinado
  - Permite especificar y construir invocaciones en tiempo de ejecución
  - Usa info. resultante de consultar al *repositorio de interfaces*
  - Posibilidad de invocar objetos CORBA sin conocerlos previamente
  - Operaciones: create_request, invoke, send, get_response, ...

  DSI (*dynamic skeleton interface*) Permite a los objetos CORBA recibir invocaciones sin que tengan un *skeleton* propio
  - Inspecciona las invocaciones que recibe para determinar el método y objeto (interfaz) a invocar
  - Realiza la invocación sobre el *servant* correspondiente
  - ORB le indica el nombre de método y sus parámetros

- Repositorio de interfaces (*Interface repository*)
  - Catálogo donde los *servants* registran las definiciones de los interfaces que implementan
    - Almacena info. sobre { nombres de métodos; nombre y tipo de sus argumentos; excepciones generadas }
  - Permite a los clientes buscar interfaces en tiempo de ejecución y localizar servants que no existían cuando fueron compilados
  - Proporciona funciones (API) para listar y buscar nuevos servicios que serán invocados de forma dinámica (a través de las interfaces DII y DSI)

Repositorio de Implementaciones (*Implementation repository (IMR)*)

- Mantiene lista de objetos CORBA activables y de los *servants* asociados que los implementan
  - Permite cargar bajo demanda los *servants* registrados
  - Soporta el manejo de referencias a objetos persistentes
    - Permite al *servant* cambiar de ubicación sin afectar al cliente
- Los objetos transitorios (no activables) no necesitan usar el repositorio de implementaciones
- Uso de referencias indirectas
  - Cuando un *servant* se registra en el *IMR*, las referencias a objetos apuntan al *IMR* en lugar de al *servant*.
  - Cuando un cliente usa esa referencia:
    - el *IMR* recibe la petición y activa el *servant* (si es necesario)
    - devuelve una nueva referencia al cliente que identifica al *servant* concreto (host y puerto)
    - esa referencia formará parte del IOR usado por el cliente en su invocación

Interfaz del núcleo ORB

- Permite el acceso directo a los servicios básicos del ORB, excepto las invocaciones desde clientes y la activación de objetos.
  - Inicialización del ORB
  - Acceso a los servicios iniciales: servicio de nombres, POA raíz, etc
  - Acceso al repositorio de interfaces y construcción de invocaciones a través del interfaz DII
  - Operaciones sobre referencias de objetos: conversión a/desde IOR a formato texto, duplicación de referencias, ...

POA (*Portable Object Adapter*)

- Da soporte a la conexión de la implementación de un objeto CORBA (*servant*) al núcleo del bus ORB
  - Intermediario entre el ORB y los objetos en la parte servidor
  - Adapta los objetos para que los pueda usar el ORB
- Al crear una implementación (*servant*) se debe registrar en el POA
  - Al registrarse pasa a ser un Objeto CORBA y se le asigna un identificador de objeto **(IOR)**

Nombre interfaz IDL ID en repositorio interfaces	Protocolo + dirección IIOP:Máquina:nº puerto	Clave del objeto nombre POA:nombre de Objeto en POA

  - Necesario para que el obejto CORBA pueda recibir invocaciones
  - Equivalente al proceso de *exportación* en Java RMI
- El POA gestionará las peticiones que lleguen hacia ese objeto
- Se encarga de:
  - Manejar un conjunto de objetos servidores (*servants*) y sus respectivos *skeleton*
    - instancia, activa y crea referencias a objetos CORBA (IOR)
    - mantiene una "tabla" donde asocia cada *nombre de objeto* con el *servant* que lo implementa
  - Recibir y redirigir las peticiones del cliente al *servant* adecuado
    - selecciona el *servant* e invoca mediante su *skeleton* al método apropiado
  - Gestionar los identificadores de objetos (mapea IOR a identificadores de implementaciones)
  - Implementar las políticas de objetos persistentes y transitorios
  - Registrar los objetos en el repositorio de implementaciones
- Reemplaza al BOA (basic object adapter) de CORBA 1.1 (era dependiente del fabricante)
- Pueden exitir múltiples POAs en una aplicación CORBA, gestionando distintos *servants*
  - Organizados en una jerárquía de POAs
  - Siempre existe como mínimo un POA raíz (asociado al nombre *"RootPOA"*)
    - La referencia (IOR) al POA raíz se obtiene consultando directamente al ORB
    - Se usa la operación `resolve_initial_references()` del interfaz ORB, pasándole el nombre `''RootPOA''`

(c) Flujo típico en invocaciones estáticas

(1) Funcionamiento general

- El servidor
  - crea objetos remotos
  - hace accesibles referencias a objetos remotos
  - espera que clientes invoquen métodos de esos objetos remotos
- El cliente
  - obtiene referencia a un objeto remoto en servidor
  - invoca a sus métodos

(2) Esquema del proceso de invocación

- El cliente necesita
  - referencia del objeto remoto (IOR)
  - tipo del objeto
  - nombre de la operación a invocar
- Cliente usa
  - stub generado a partir del IDL
  - invocación dinámica a través del interfaz DII
- Cliente gestiona 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
  - Usa el POA para transfierir el control al *servant* que implementa el interfaz, a través de
    - *skeleton* IDL
    - esqueleto dinámico (DII)
  - Informa de excepciones en el proceso

(3) Flujo típico de invocaciones estáticas en tiempo de ejecución

Nota: Previamente el proceso servidor que crea/instancia el *servant* lo habrá registrado en el POA encargado de gestionarlo, que le asignará su IOR

1. Cliente obtiene una referencia (IOR) al objeto remoto
   - consultando el servidor de nombres de CORBA
   - obtenido como valor de retorno en una invocación remota previa
2. Cliente invoca uno de los métodos remotos sobre el *stub* IDL
3. *stub* IDL comunica la invoación al ORB (se crea una petición CORBA)
4. ORB localiza el objeto y le pasa la llamada al adaptador de objetos (POA) encargado de gestionarlo
   - si fuera necesario se activará el *servant* correspondiente (puesta en ejecución)
   - POA localiza la implementación del objeto indicada

5. POA deriva invocación en el *skeleton* IDL
6. *skeleton* realiza la invocación real sobre el *servant*

Completada la ejecución del método:

- El *skeleton IDL* recupera los valores de retorno del *servant*
- Esos valores volverán al cliente pasando a través del *skeleton* IDL (crea el mensaje de respuesta), el ORB y el *stub* IDL, que los recupera del mensaje de respuesta y se los pasa al cliente.

3.3.2 IDL de CORBA

(a) Definición de interfaces CORBA

CORBA utiliza su propio IDL (*Interface Definition Language*) como mecanismo para definir los interfaces de los objetos remotos.

- Una especificación IDL es un contrato entre cliente y servidor
  - especifica la sintaxis de { las invocaciones remotas que enviará el cliente / los métodos ofrecidos e implementados por el *servant*
- Lenguaje IDL es independiente del lenguaje de programación y del sistema operativo
  - el *compilador IDL* se encarga de la traducción
- Sintaxis cercana a la de C++
  - permite comandos de preprocesador (`#define, #include, #ifdef, #undef`)
- Permite definir { módulos (ámbitos de declaración, son anidables) / interfaces / tipos compuestos
- Tipos básicos estándar: *long,string,boolean,float,double,char,octet,any,void*
- Tipos compuestos { enumeraciones, estructuras / uniones (con selector) / arrays, secuencias (arrays long. variable)
  - Ejemplo:

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

- Definición de *interfaces*: describen el interfaz del objeto CORBA
  - Definición de *constantes*, *atributos*, *operaciones* y *excepciones*
- Definición de *excepciones*: nombre + estructura (campos)
- Definición de *atributos*: valores asociados al objeto CORBA accesibles mediante operaciones
  - Se implementarán como métodos *getter* y *setter*
  - Admiten modificador `readonly` (sólo se generará el *setter*)
- Definición de *operaciones* (métodos)
  - Para cada operación especifica *tipo de retorno*, *nombre*, *parámetros* de entrada y *excepciones* que puede lanzar
    - Especificación de parámetros { entrada (*in*) / salida (*out*) / entrada/salida (*inout*)

- Modificador *oneway*: cambia semántica de la invocación
  - por defecto CORBA usa semántica AL-MENOS-UNA-VEZ
  - modificador *oneway* establece semántica TAL-VEZ
- Ejemplos:

```
long sumar(in long a, in long b);
oneway void incrementarSaldo(in cliente cli,
                             in double cant) raises (ClienteNoExiste);
```

- IDL CORBA permite herencia de interfaces (no de implementaciones)
  - Sintaxis: `interface Circulo:Figura {...};`
  - Todos los interfaces heredan implicitamente de `CORBA::Object`
  - Es posible herencia múltiple
  - No es posible sobreescritura de operaciones (*overriding*)

Más información: `http://www.omg.org/gettingstarted/omg_idl.htm`

(b) Mapeo de CORBA IDL a C++ y Java

Mapeo entre los constructores de IDL y los de Java y C++

IDL	Java	C++
`module`	`package`	`namespace`
`interface`	`interface`	clase abstracta
`operation`	método Java	función miembro C++
`attribute`	par de métodos	par de métodos
`exception`	`exception`	exception

Mapeo entre tipos IDL y tipos Java y C++

IDL	Java	C++
`boolean`	`boolean`	`CORBA::Boolean`
`char/wchar`	`char`	`CORBA::Char`
`octet`	`byte`	`CORBA::Octet`
`short/unsigned`	`short`	`CORBA::Short/CORBA::UShort`
`long/unsigned long`	`int`	`CORBA::Long/CORBA::ULong`
`float`	`float`	`CORBA::Float`
`double`	`double`	`CORBA::Double`
`string/wstring`	`String`	`char * + CORBA::String_var`
`any`	`org.omg.CORBA.Any`	`CORBA::Any`
`struct`	clase (con atribs. públicos)	`struct`
`union`	clase (con atribs. públicos)	`union`
`sequence`	`array`	`struct` (campos `max`, `length`, `data`)
`array`	`array`	`array`
`constant`	variable `final` en interfaz/clase	`#define`

3.3.3 Servidor de nombres CORBA

La especificación CORBA define un servicio de nombres

- Mantiene asociación entre nombres y referencias (IOR) a objetos CORBA
- Estructura jerárquica basada en **contextos de nombramiento** (*Naming Context (NC)*)
  - Dentro de cada NC se definen un conjunto de pares `(nombre, IOR)`
  - Se puede asociar un nombre a un NC de nivel inferior (crea un nodo un hijo)
  - En cada ORB existe un *"Initial Naming Context"*
    - punto de acceso a la jerarquía de nombres definida en ese ORB
  - Los servicios de nombres de distintos ORBs se pueden fusionar (federar)
    - Un mismo IOR puede aparecer en 2 puntos de la jerarquía, accesibles desde 2 *"Initial Naming Context"* distintos
  - No hay nombres absolutos $\longrightarrow$ todos son relativos a un *Initial Naming Context*
  - Los nombres tienen varios componentes (representan su *path* en la jerarquía)
    - búsqueda recursiva a través de un conjunto de objetos *Naming Context*
    - el último componente es una hoja que almacena el IOR del objeto CORBA
- La interfaz del servicio de nombres está definida en IDL CORBA

```
module CosNaming {
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence<NameComponent> Name;
    enum BindingType {nobject, ncontext};
    struct Binding {
        Name          binding_name;
        BindingType binding_type;
    };
    typedef sequence <Binding> BindingList;
    interface BindingIterator {
        boolean next one(out Binding b);
        boolean next n(in unsigned long how_many,
                               out BindingList bl);
        void destroy();
    };
```

```
    interface NamingContext {
        void bind(in Name n, in Object obj) raises(...);
        void rebind(in Name n, in Object obj) raises(...);
        void bind context(in Name n, in NamingContext nc)
                        raises(...);
        void rebind context(in Name n,
                         in NamingContext nc) raises(...);
        Object resolve(in Name n) raises(...);
        void unbind(in Name n) raises(...);
        NamingContext new context();
        NamingContext bind new context(in Name n)
                        raises(...);
        void destroy() raises(...);
        void list(in unsigned long     how_many,
                  out BindingList      bl,
                  out BindingIterator bi);
    };
   ...
};
```

- Clientes y servidores solicitan el *"Initial Naming Context"* al ORB
  - Se usa la operación `resolve_initial_references()` del interfaz ORB, pasándole el nombre `''NameService''`
  - ORB les devuelve un objeto CORBA que implementa el interfaz `NamingContext`
  - A partir de ese *"Initial Naming Context"* se recorre la jerarquía
- Los **nombres** CORBA son secuencias de instancias de `NameComponent`
  - Dos elementos: { `id`: el nombre en si; `kind`: categoría [manejada internamente] }
  - Operaciones `bind()`, `rebind()` y `resolve()` usan como argumentos objetos de tipo `Name` (secuencias de `NameComponent`)
    - `bind()`, `rebind()`: asocian el IOR de un objeto CORBA con un contexto de nombrado
    - `resolve()`: localiza el IOR de un objeto CORBA en la jerarquía de nombres

3.3.4 Uso de CORBA en Java

El entorno Java SE (*Java Standard Edition*) incluye una implementación compatible con CORBA 2.3 denominada **Java IDL**

- Orientado principalmente al uso de aplicaciones CORBA desde Java

(a) Paquetes

- `org.omg.CORBA`: proporciona el mapeo del API de CORBA a Java
  - clase `org.omg.CORBA.ORB`: proporciona el API para el ORB de CORBA
- `org.omg.CosNaming`: proporciona el servicio de nombres
- `org.omg.PortableServer`: clases e interfaces usados en la parte "servidor" de las aplicaciones CORBA
- `org.omg.PortableInterceptor`: proporciona mecanismos de registro y control de los servicios del ORB
- `org.omg.DynamicAny`: clases e interfaces para manejar el tipo CORBA `any`

(b) Herramientas

- `idlj`: compilador de interfaces IDL para Java
- `orbd`: demonio con los servicios básicos de CORBA
  - servicio de arranque (*Bootstrap Service*)
  - servicio de nombres transitorio (*Transient Naming Service*)
  - servicio de nombres persistente (*Persistent Naming Service*)
  - gestor de servidores (*Server Manager*)
- `servertool`: interfaz de linea de comandos para registrar, inciar y parar *servants*
- `tnameserv`: servicio simple de nombres (transitorio y volátil)

(c) Proceso de desarrollo

1. Definir los objetos del servidor mediante IDL.

```
// Calculadora.idl
module ejemplo {
  interface Calculadora {
      long sumar(in long a, in long b);
      long restar(in long a, in long b);
      long multiplicar(in long a, in long b);
      long dividir(in long a, in long b);
      readonly attribute long contadorOperaciones;
  };
};
```

2. Pre-compilar el IDL para generar *stubs*, *skeleton* y código adicional

```
$ idlj -fall  Calculadora.idl
```

La opción `-fall` indica que se generen los ficheros necesarios tanto para clientes como para servidores

Para cada módulo, el compilador `idlj` genera un paquete Java, en este caso de nombre `ejemplo` con la siguiente estructura:

- Común para parte cliente y servidor
  - `CalculadoraOperations.java`: interfaz con métodos definidos en el IDL
  - `Calculadora.java`: interfaz con la versión Java del IDL (hereda del anterior)
  - `CalculadoraHelper.java`: clase auxiliar con la implementación del método **narrow** que transforma objetos CORBA genéricos (`omg.CORBA.Object`) en objetos que implementan `Calculadora`
  - `CalculadoraHolder.java`: clase auxiliar con operaciones para trabajar/adaptar los parámetros declarados como `out` y `inout`
    - "Simula" el paso de parámetros por valor mediante copia-restauración
- Sólo para parte cliente
  - `_CalculadoraStub.java`: *stub* del cliente (implementa el interfaz `Calculadora`)
- Sólo para parte servidor
  - `CalculadoraPOA.java`: clase abstracta de la que debe heredar el *servant* que implemente el interfaz IDL
    - Juega el papel de *skeleton*
    - Además incluye las funcionalidades necesarias para la implementación del *servant* y su integración (registro) en el ORB y el POA

Nota: si hubiera más interfaces definidos en el fichero IDL se repetiría el mismo esquema de interfaces y clases Java para cada uno e ellos

- Se usa un esquema similar para tipos de datos `struct` y `union`

3. Escribir el código del *servant* (implementación del objeto CORBA)

   En *Java IDL* cada *servant* debe heredar de la clase `xxxxxxPOA.java` que corresponda, e implementar todos los métodos definidos en el interfaz IDL

```
import ejemplo.CalculadoraPOA;

public class CalculadoraImpl extends CalculadoraPOA {
  private int contador;
  public CalculadoraImpl(){
    contador = 0;
  }
  public int sumar(int a, int b){
    contador++;
    return (a+b);
  }
  ...
  public int contadorOperaciones(){
    return (contador);
  }
}
```

4. Escribir un proceso "servidor" que instancie los *servant* y los registre en el POA

   a) Inicializar ORB (método: `org.omg.CORBA.ORB.init(...)`)

   b) Obtener referencias al POA raíz
   - Contactar a través del interfaz básico del ORB con el servidor de nombres y obtener referencia al *POA raíz*
     método: `orb.resolve_initial_references("RootPOA")`
   - Convertir la referencia CORBA genérica a una de tipo POA
     método: `raizPOA = POAHelper.narrow(...)`
   - Activar el gestor de invocaciones del POA raíz
     método: `raizPOA.the_POAManager().activate()`

   c) Hacer al objeto remoto disponible en el POA
   - Instanciar el *servant* del objeto remoto
   - Activar el objeto remoto en el POA raíz y obtener su referencia
     método: `servant_to_reference(...)`

d) Registrar el nombre del objeto en el servidor de nombres

- Obtener el *"Initial Naming Context"* del ORB y hacer el *narrow* correspondiente
  métodos: { `orb.resolve_initial_references("NameService")` ; `NamingContextExtHelper.narrow(...)` }
- Construir el nombre y enlazarlo en el servidor de nombres con la referencia al objeto remoto
  método: `nc.rebind(nc.to_name("<nombre>"), <objeto_corba>)`

e) Pasar control a ORB para que acepte peticiones (método: `orb.run()`)

```
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;

import ejemplo.*;  // Importar el paquete generado por idlj

public class EjemploServidor {
  public static void main(String[] args) {
    try {
      // 1. Inicializar ORB
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

      // 2.1 Obtener POA raiz
      POA raizPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

      // 2.2 Activar el POA manager
      raizPOA.the_POAManager().activate();

      // 3.1 Crear instancia de la implementación (servant)
      CalculadoraImpl calculadoraServant = new CalculadoraImpl();

      // 3.2 Registrar en el POA y obtener referencia al objeto (IOR)
      org.omg.CORBA.Object calculadoraCorba =
                           raizPOA.servant_to_reference(calculadoraServant);

      // 4.1 Obtener el initial naming context
      org.omg.CORBA.Object ncCorba = orb.resolve_initial_references("NameService");
      NamingContextExt nc = NamingContextExtHelper.narrow(ncCorba);

      // 4.2 Asociar un nombre (en el primer nivel)
      nc.rebind(nc.to_name("Calculadora"), calculadoraCorba);

      // 5 Quedar a la espera de peticiones
      System.out.println("Proceso servidor en espera ... ");
      orb.run();
      ...
}
```

5. Compilar los ficheros de la parte servidor

```
$ javac CalculadoraImpl.java
$ javac EjemploServidor.java
```

6. Escribir el código del cliente

   a) Inicializar ORB

   b) Contactar con el objeto remoto

   - Contactar con servidor de nombres y obtener el *"Initial Naming Context"*
   - Construir el nombre del objeto en el servidor de nombres
   - Obtener referencia al objeto remoto en el servidor de nombres
     método: `nc.resolve(nc.to_name("<nombre>"))`
   - `narrow` de objeto genérico CORBA al tipo específico del objeto remoto
     método: `CalculadoraHelper.narrow(...)`

   c) Utilizar el objeto remoto como si fuese local

```
import org.omg.CosNaming.*;

import ejemplo.*;  // Importar el paquete generado por idlj

public class EjemploCliente {
  public static void main(String[] args) {
   try  {
     // 1. Inicializar ORB
     org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

     // 2.1 Obtener instancia del servidor de nombres (initial naming context)
     org.omg.CORBA.Object ncCorba= orb.resolve_initial_references("NameService");
     NamingContextExt nc = NamingContextExtHelper.narrow(ncCorba);

     // 2.2 Construir el nombre del objeto y obtener ref. desde serv. de nombres
     org.omg.CORBA.Object calculadoraCorba=nc.resolve(nc.to_name("Calculadora"));

     // 2.4 Convertir el objeto CORBA al tipo Calculadora (narrow)
     Calculadora calculadora = CalculadoraHelper.narrow(calculadoraCorba);

     // 3 Invocar métodos remotos
     int resultado = calculadora.sumar(100, 20);
     ...
}
```

7. Compilar el cliente

```
$ javac EjemploCliente.java
```

8. Arrancar ORB, lanzar el "servidor" y luego el cliente (en terminales/máquinas distintos)

```
$ orbd -ORBInitialPort 1111
$ java EjemploServidor -ORBInitialHost localhost -ORBInitialPort 1111
$ java EjemploCliente -ORBInitialHost localhost -ORBInitialPort 1111
```

Los parámetros `-ORBInitialHost localhost` y `-ORBInitialPort 1111` especifican la localización del ORB (y del servidor de nombres)

- Tanto el cliente como el servidor los pasan tal cúal al ORB al inicializarlo (método `orb.init(args, null)`)
- Por defecto se usa el puerto 900, para iniciar el ORB sin contar con permisos de administrador debe especificarse un puerto menor que 1024