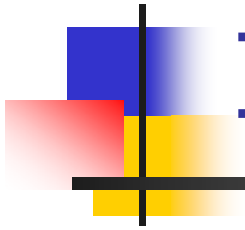
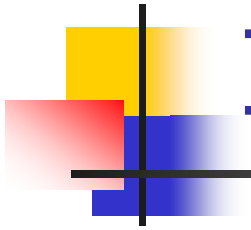


Introducción a CORBA

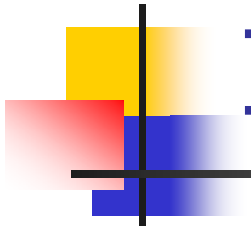
Ingeniería del Software II
Curso 2008/2009

Sergio Ilarri Artigas
silarri@unizar.es



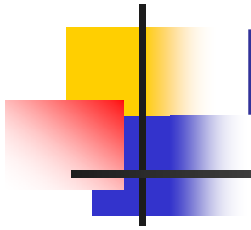
Índice (I)

- *OMG*, *OMA*:
  - *Core Object Model*
  - *Reference Model*
- Conceptos básicos: *stub*, *skeleton*, *ORB*
- El *ORB*
- Invocaciones entre objetos
- El lenguaje de definición de interfaces (*IDL*)
- Interoperabilidad: *GIOP*, *IIOP*



Índice (II)

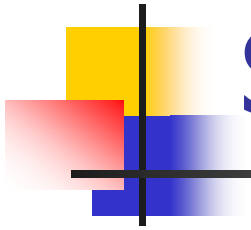
- Adaptadores de objetos: *POA*
- Modelo *CORBA*:
  - Servicios *CORBA*: servicio de nombres
- Comparación con *RMI*
- Resumen de la terminología



Motivación

- Queremos computación distribuida:
  - Entre plataformas hardware distintas
  - Entre plataformas software distintas
  - Manteniendo código legado
  - Con distintos lenguajes de programación
  - Sin depender de un vendedor concreto

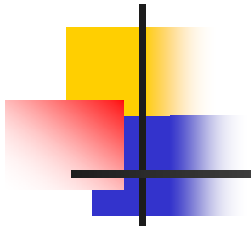
- Sistemas de objetos distribuidos (*objetos cooperativos*)



Solución: *CORBA*

- ***C**ommon **O**bject **R**equest **B**roker **A**rchitecture*

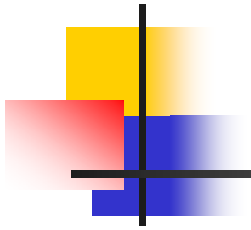
- Introducida por el *Object Management Group* (*OMG*) en 1989



OMG

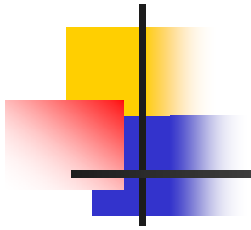
- El mayor consorcio de la industria del software
  - Más de 800 empresas (año 2000) y organizaciones, incluyendo la mayoría de vendedores y desarrolladores de tecnologías de objetos distribuidas y desarrolladores de aplicaciones
- Promueve la ingeniería de software orientada a objetos
- Emiten especificaciones (no hay implementación de referencia) **por consenso** y de forma gratuita
- Propone una **arquitectura común** para el desarrollo de aplicaciones distribuidas orientadas a objetos basada en especificaciones de interfaces
- También estandariza el lenguaje de modelado UML

http://www.omg.org/



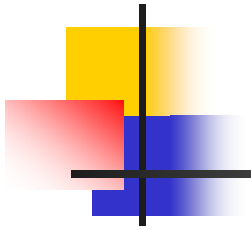
OMA

- *Object Management Architecture*
- *Framework* donde se encajan el resto de tecnologías adoptadas por OMG
- Dos partes fundamentales:
  - *Core Object Model*
  - *Reference Model*



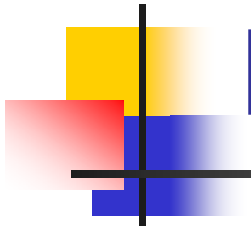
OMA: Core Object Model (I)

- Terminología común productos OMG
- Objetivos:
  - *Portabilidad*
    - Componentes que no dependen de una implementación concreta
  - *Interoperabilidad*
    - Sin importar la plataforma, localización, o lenguaje de implementación



OMA: Core Object Model (II)

- Es un modelo de objetos clásico:
  - Objetos: referencia al objeto (identidad)
  - Operaciones: signaturas, excepciones
  - Interfaces
  - Tipos de datos (que no son objetos):
    - No especificados por el *Core Object Model*
    - Sí por CORBA
  - Interfaces como modo de agrupamiento de operaciones (y como "tipos")



Reflexión

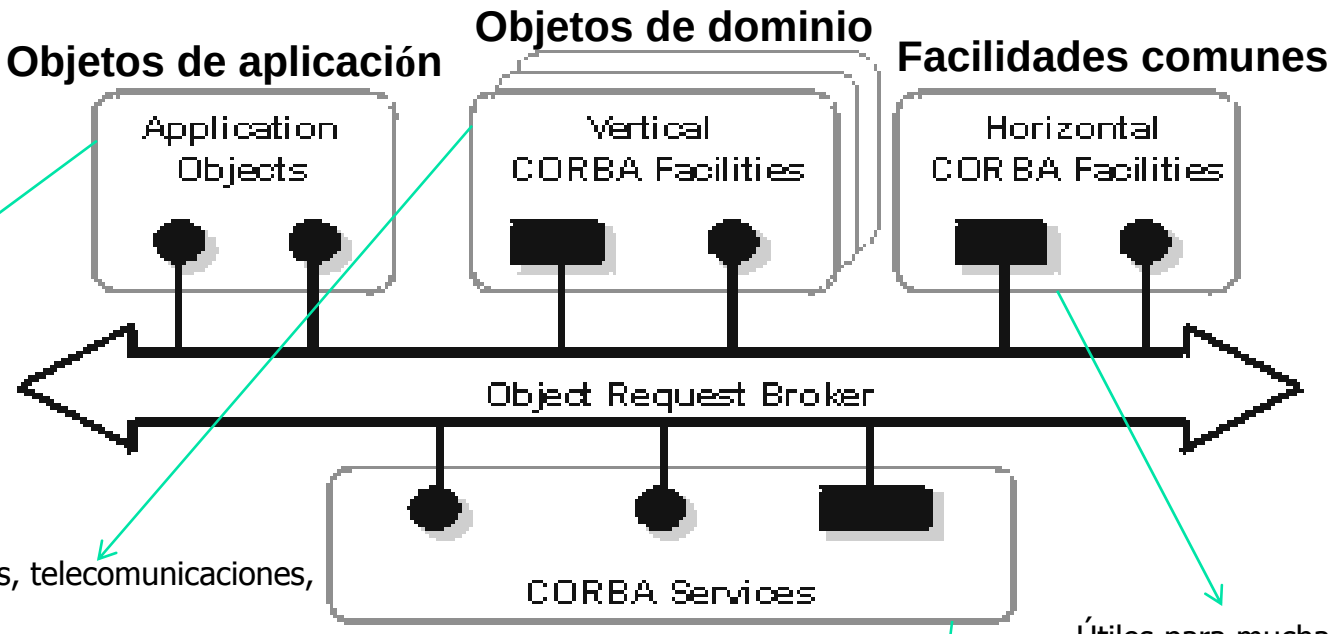
¿Por qué es un *modelo de objetos abstracto*?

No lo implementa ninguna tecnología particular (≈ clases abstractas)

Core Object Model → CORBA Object Model

OMA: Reference Model

4 Categorías de objetos



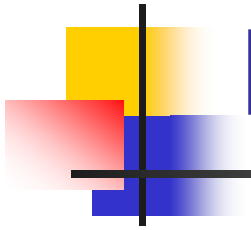
Objetos CORBA programados

Áreas de aplicación:
Fabricación, finanzas, telecomunicaciones, salud, etc.

Útiles para muchas aplicaciones:
intercambio de datos, agentes, internacionalización, etc.

Para aplicaciones de cualquier dominio:
Servicio de nombres, seguridad, etc.

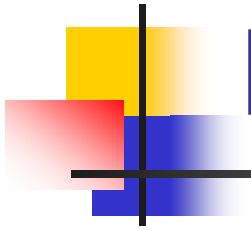
http://www.omg.org/oma/



Reflexión

¿Están los objetos de aplicación estandarizados?

No, los define el programador

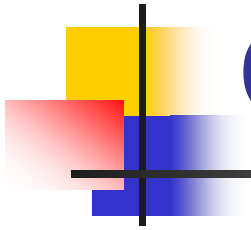


Reflexión

¿Por qué es un *modelo de referencia*?

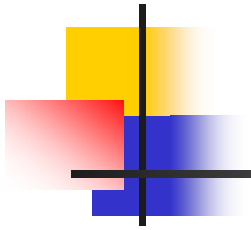
Especificación

OMG define los interfaces, la implementación es cosa de los vendedores



Conceptos Básicos

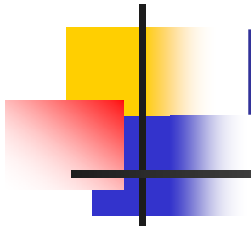
- Objeto CORBA:
  - Una entidad "virtual" que puede ser localizada por un ORB y recibir invocaciones remotas.
- Referencia a un objeto CORBA:
  - Estructura de datos opaca que identifica un objeto CORBA.
- Servant:
  - Entidad de un lenguaje de programación que implementa uno o más objetos CORBA



CORBA

- Como *RMI*, pero independiente del lenguaje y más completo:
  - Definición de interfaces
  - Generación de *stubs* y *skeletons*

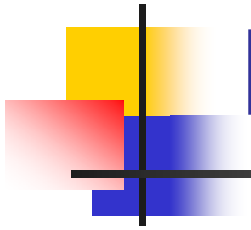
Definido sobre el *Core Object Model* de la *OMA*



Reflexión

¿Cómo podemos lograr interoperar transparentemente con implementaciones distintas?

Interfaces



Reflexión

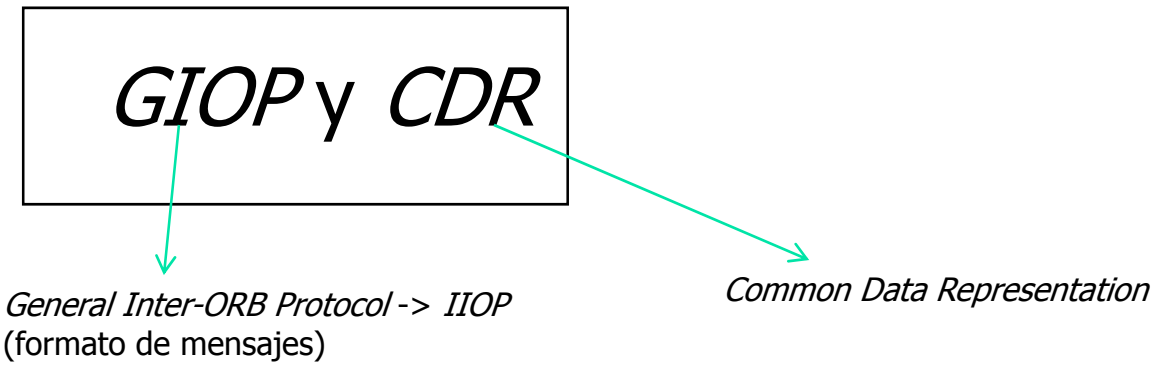
¿Y si las implementaciones están en distintos lenguajes de programación?

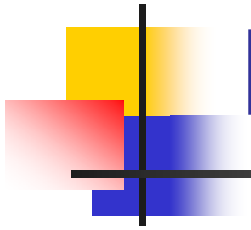
Interfaces en lenguaje neutro (*IDL*)

Interface Definition Language

Reflexión

¿Cómo podemos lograr
la transparencia respecto a la
plataforma hardware y la red?



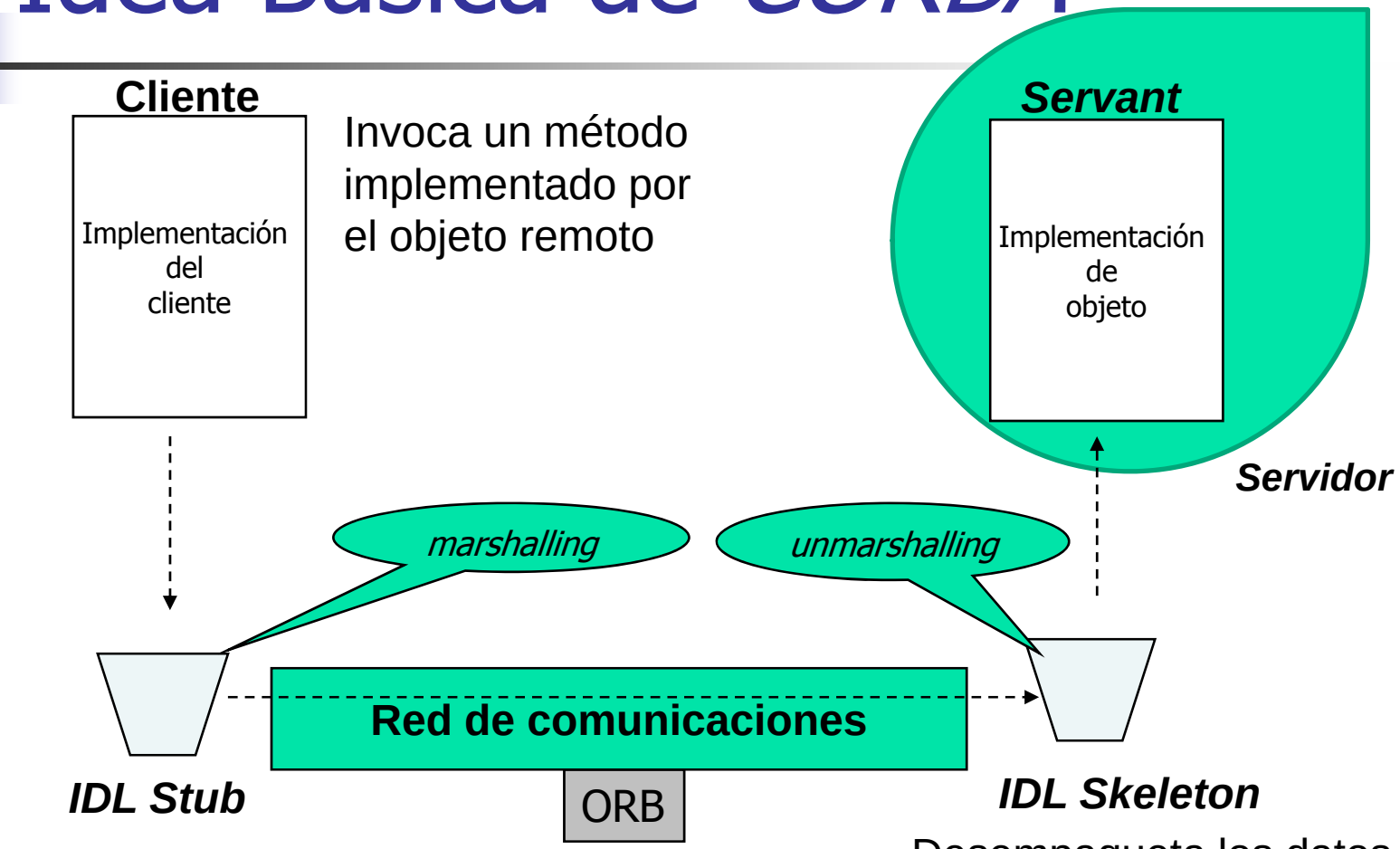


Reflexión

¿Por qué en RMI no necesitamos algo del estilo del CDR?

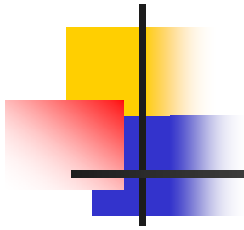
RMI es Java-Java

Idea Básica de *CORBA*



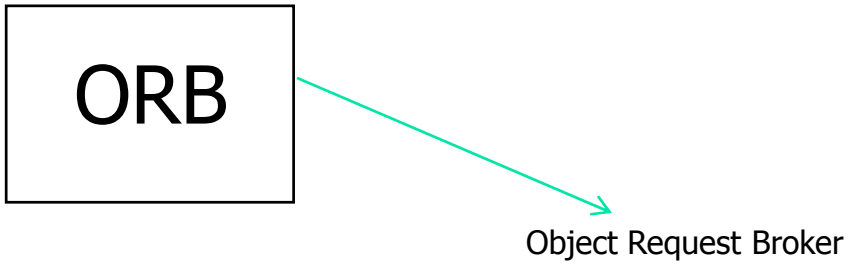
Empaqueta la invocación, envía los datos al *skeleton*

Desempaqueta los datos, invoca método en objeto, devuelve resultados/errores



Reflexión

¿Qué hemos introducido con respecto a *RMI* en el dibujo anterior?

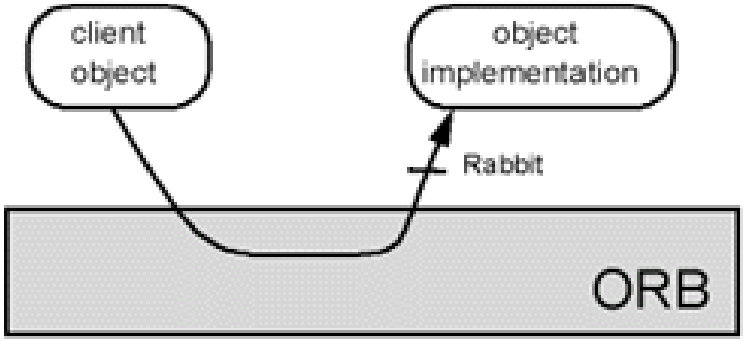


ORB (I)

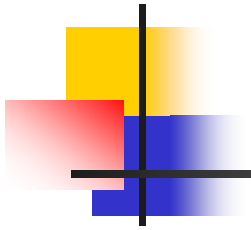
- *Object Request Broker* (*Object Bus*)
- Servicio distribuido que implementa las peticiones al objeto remoto:
  - Localiza al objeto remoto en la red
  - Comunica la petición a dicho objeto
  - Espera los resultados y los pasa al cliente

Petición de servicios a objetos distribuidos

Cada máquina tiene su ORB

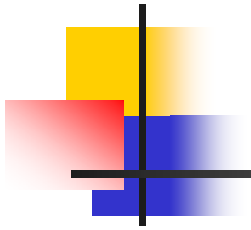



```
interface Rabbit {
        :
    AnotherObject jump(
        in long how_high);
        :
};
```



ORB (II)

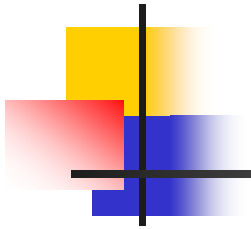
- Proporciona *transparencia de localización*
  - Nada cambia si varía la localización de cliente u objeto remoto
- Proporciona *independencia del lenguaje*
  - Cliente y objeto *CORBA* remoto pueden estar implementados en lenguajes de programación diferentes



ORB (III)

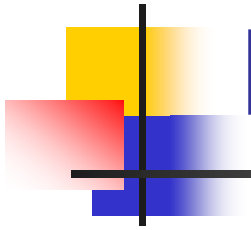
- Funciones principales:
  - Enrutamiento de peticiones y respuestas
  - Transformaciones entre *referencias a objetos remotos* y cadenas de caracteres
  - Invocación dinámica de objetos remotos
  - Activación/desactivación de objetos

También llamadas *CORBA object handles*



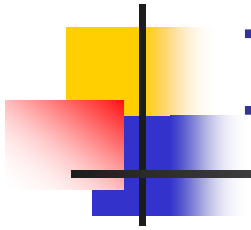
ORB (IV)

- Elementos importantes:
  - *Interface Repository*
    - (no todas las implementaciones, ver siguiente transp.)
  - *Implementation Repository*
    - (no todas las implementaciones, ver siguiente transp.)
  - *Client Stubs*
  - *Server Skeletons*
  - *Portable Object Adapter* (*POA*)
  - *Dynamic Invocation /Skeleton Interface* (*DII /DSI*)



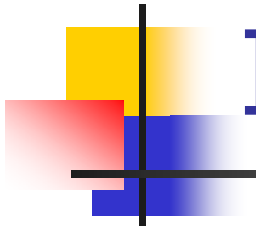
Repositorios

- Repositorio de interfaces:
  - Almacena las definiciones IDL
  - Permite que los objetos CORBA se auto-describan
- Repositorio de implementaciones:
  - Permite al ORB localizar y activar (bajo demanda) implementaciones de objetos



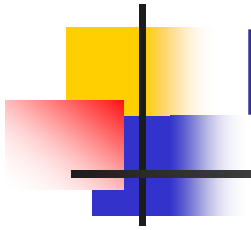
Invocaciones entre Objetos (I)

- Nos centramos en la *invocación estática*
  - La usada con mayor frecuencia (también es posible la invocación dinámica)
- Similar al método convencional de invocación en Java
- Se comprueban tipos en tiempo de compilación
- Las definiciones del interfaz deben estar disponibles cuando se compila el cliente (*stub*)



Invocaciones entre Objetos (II)

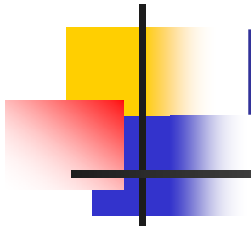
- Un cliente necesita una referencia al objeto remoto (*IOR*: *Interoperable Object Reference*):
  - Es como el número de teléfono del objeto remoto
  - Identifica unívocamente al *servant* independientemente de su localización



Reflexión

¿Algún patrón de diseño relacionado con los *stubs*?

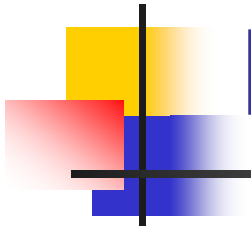
El patrón ***Proxy***



Reflexión

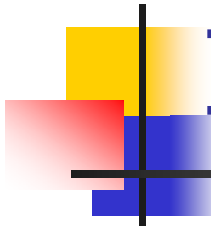
¿Algún patrón de diseño relacionado con los *skeletons*?

El patrón ***Adapter***



Reflexión

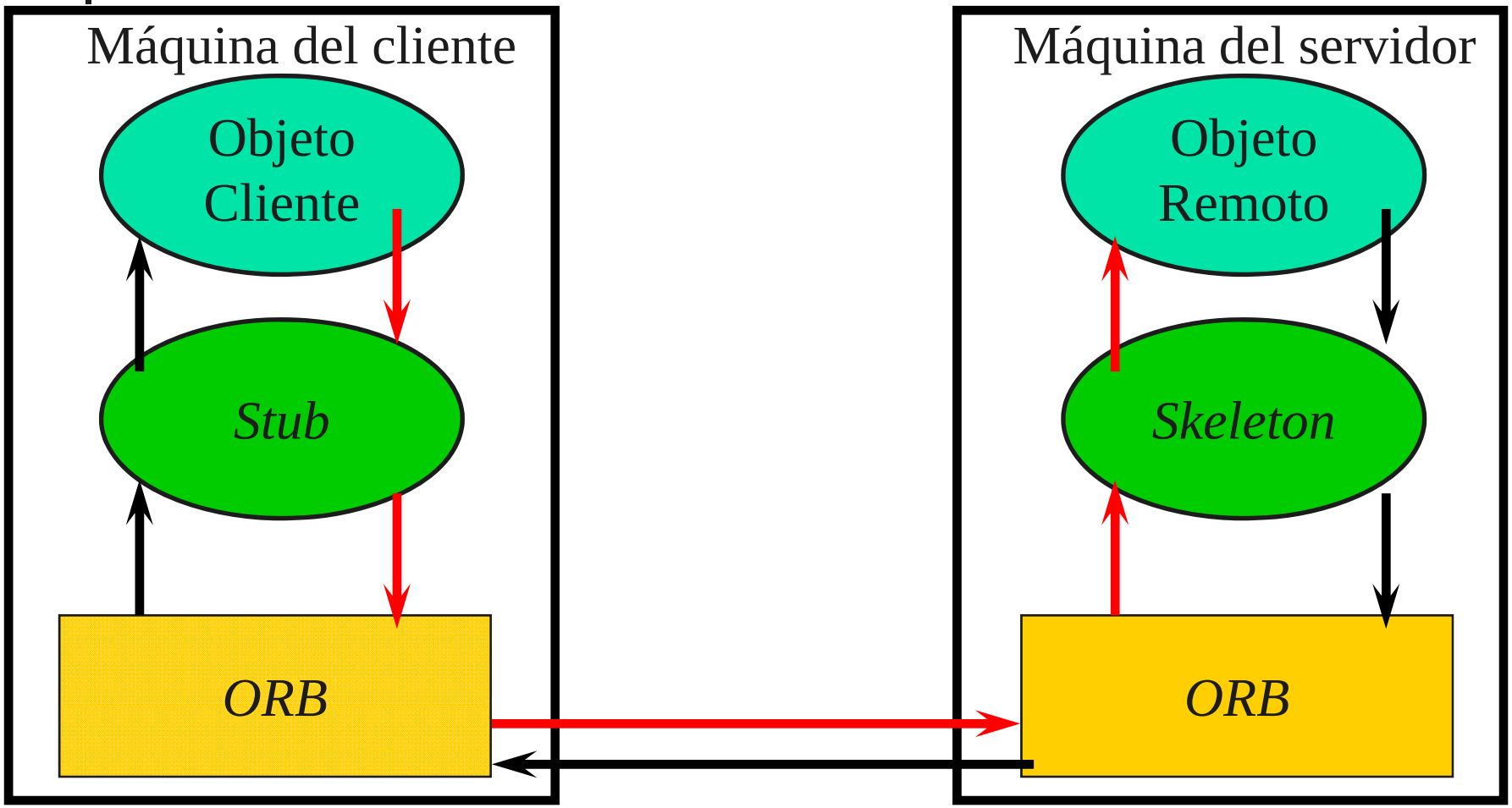
¿Y cómo conseguimos la referencia al objeto remoto?

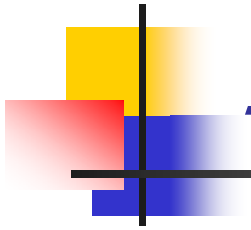


Invocaciones entre Objetos (III)

- Obtención de la referencia:
  - *ORB.resolve_initial_references(objectNam)*
  - Preguntando a otros objetos *CORBA* (como el servicio de nombres)
  - Usando una *stringified object reference:*
    - *object_to_string* y *string_to_object* en *org.omg.CORBA.Object*
    - Puede almacenarse, por ejemplo, en disco

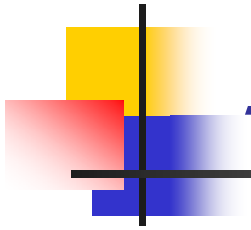
Invocaciones entre Objetos (IV)





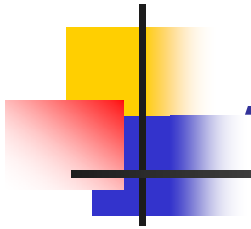
IDL (I)

- *Interface Definition Language*
- Lenguaje declarativo (no sentencias)
- Orientado a objetos, sintaxis tipo C++
- Para especificar el interfaz de un objeto (similar a los ficheros .x de *RPC*):
  - "Contrato" entre el código que implementa el objeto y los posibles clientes
  - Indica las operaciones soportadas, no cómo se implementan



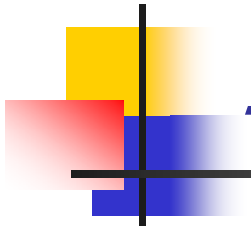
IDL (II)

- El objeto *CORBA* y el cliente puede implementarse en distintos lenguajes (Java, C++, etc.)
- El cliente sólo depende del interfaz
- Un interfaz IDL es independiente del lenguaje de programación:
  - Permite que dos objetos implementados en lenguajes distintos puedan "entenderse"



IDL (III)

- IDL define ***Mappings*** (***bindings***) con diversos lenguajes de programación:
  - C
  - C++
  - Java
  - Ada
  - COBOL
  - Smalltalk
  - Objective C
  - Lisp
  - Python



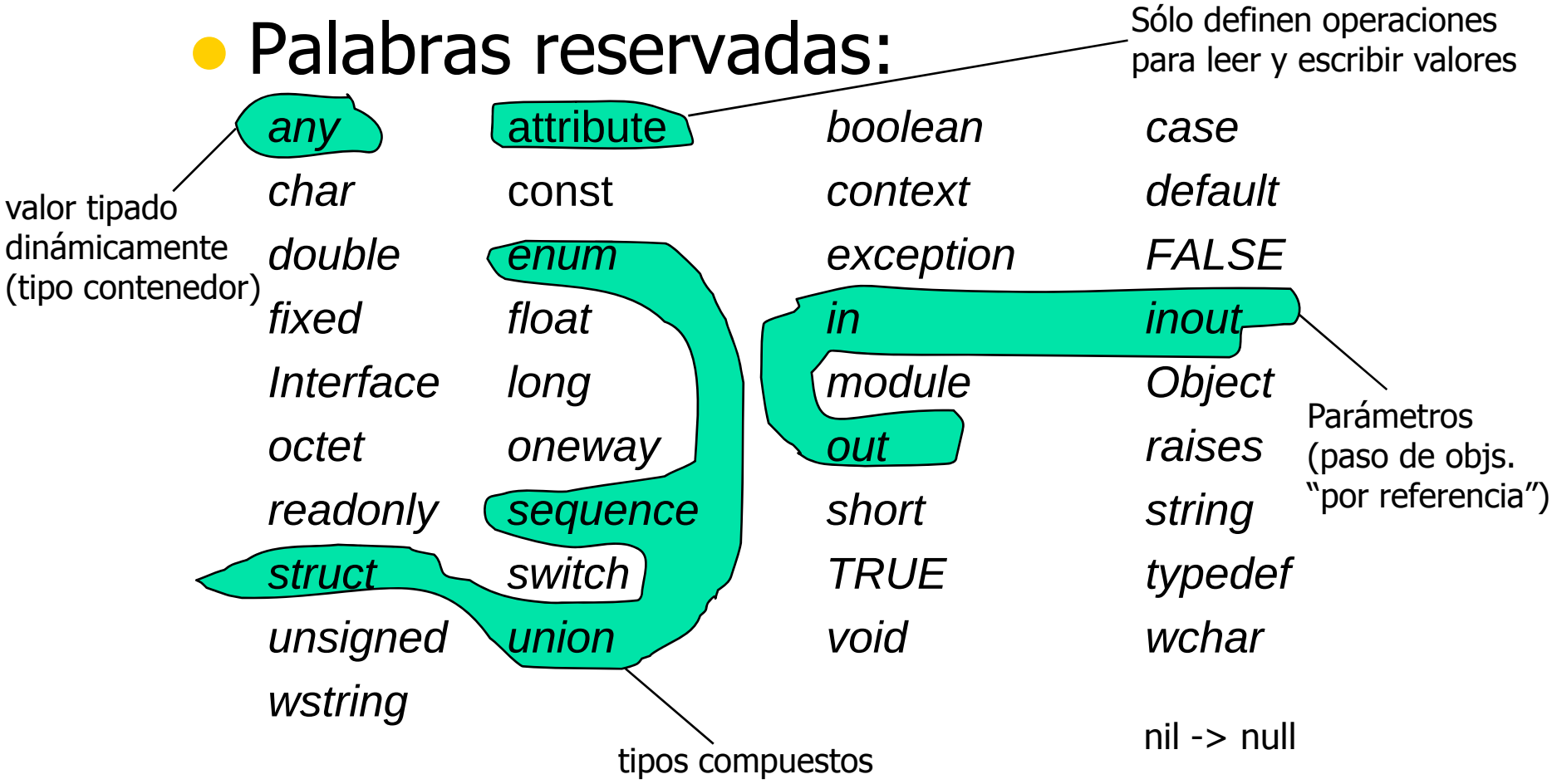
IDL (IV)

- OMG IDL permite definir :
  - Módulos para los interfaces
  - Operaciones y atributos
  - Excepciones que pueden lanzar las operaciones
  - Tipos de datos:
    - Para los parámetros y valores de retorno de operaciones
    - Para los atributos

Los objetos remotos se pasan por referencia

IDL (V)

- Palabras reservadas:



IDL (VI)

- Signos de puntuación:

  herencia

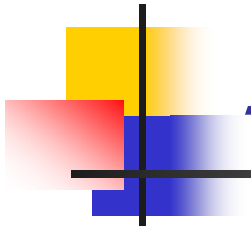
  ; { } : , = + - ( ) < > [ ] ` "

  \ | ^ & * / % ~

- Preprocesado:

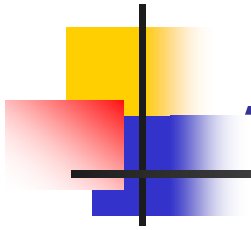
  # ## ! || &&

No insistiremos en la sintaxis (se parece mucho a Java)



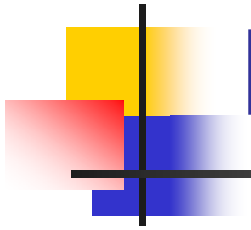
IDL (VII)

- Compiladores de IDL
  - Específicos para un cierto lenguaje
    - Compilar para el cliente (stubs) y para el servidor (skeletons)
    - Cliente y servidor pueden estar en distintos lenguajes
  - Generan, a partir de los interfaces IDL, los *skeletons* y *stubs* para ese lenguaje



IDL (VIII): Excepciones

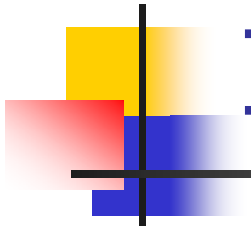
1. Excepciones estándar de sistema:
   - Definidas por la especificación de *OMG*
   - Cualquier operación del *IDL* puede lanzarlas
   - Causas:
     - Excepciones del lado del servidor (fallo de activación, falta de recursos, etc.)
     - Excepciones del lado del cliente (tipo de operando inválido, algo que ocurre antes de enviar la petición o después de obtener el resultado)
     - Excepciones del sistema de comunicaciones (máquina caída, *ORB* inalcanzable, etc.)
2. Excepciones de usuario:
   - Definidas en el *interfaz IDL* con la palabra clave *raise*



Reflexión

¿Es útil "conectar" *ORBs* entre sí?

Sí, por ejemplo, de distintos vendedores



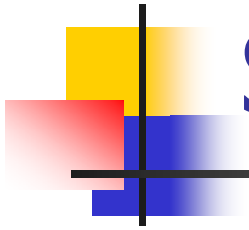
Interoperabilidad

- *IIOR (Interoperable Object Reference)*
- *GIOP (General Inter-ORB Protocol)*
  - Formatos de mensajes
  - Sobre un protocolo de transporte orientado a conexión
- *IIOP (Internet Inter-ORB Protocol)*
  - GIOP + TCP/IP

"Under the hood: IORs, GIOP and IIOP", Dave Bartlett, August 200, http://www.ibm.com/developerworks/library/ws-underhood/

Adaptadores de Objetos

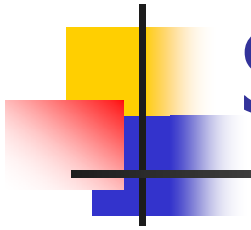
- *Object Adapter (OA)*
  - En el servidor, recibe una petición y la redirige al servant correspondiente
- Hasta CORBA 2.1:
  - *Basic Object Adapter (BOA)*
    - Ambigua y poco potente, obsoleto
- Desde CORBA 2.1
  - *Portable Object Adapter (POA)*
  - Potente, complejo
- Políticas de activación de los objetos



Servicios *CORBA* (I)

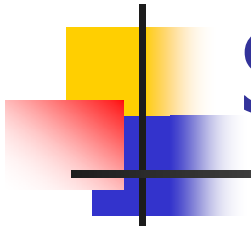
- *CORBA Services* (*COS*), *Object Services*
- Parte del *Reference Model* de *OMA*
- Servicios distribuidos para soportar la integración e interoperación de objetos

http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm



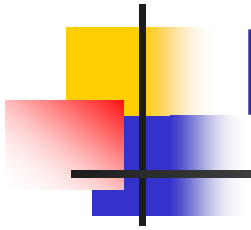
Servicios *CORBA* (II)

Servicio de	Descripción
Ciclo de vida	Define cómo crear, copiar, eliminar y mover objetos (independientemente de su localización)
Eventos, Notificaciones	Objetos que registran dinámicamente su interés en eventos Desacopla la comunicación entre objetos distribuidos
Nombres	Define cómo los objetos *CORBA* pueden tener nombres simbólicos amigables Permite al *ORB* obtener la referencia asociada a un nombre Páginas blancas
Relaciones	Proporciona relaciones tipadas n-arias entre objetos *CORBA*
Externalización	Permite convertir objetos *CORBA* a/desde flujos de datos externos



Servicios *CORBA* (III)

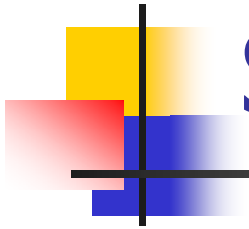
Servicio de	Descripción
Transacciones	Permite objetos de distintos *ORBs* participar en una misma Transacción distribuida
Concurrencia	Gestión de cerrojos para objetos *CORBA*, para garantizar accesos *serializables*
Propiedades	Permite asociar pares nombre-valor a los objetos *CORBA* (fuera del sistema estático del IDL)
Trading	Páginas amarillas para objetos (búsqueda por propiedades)
Consultas	Proporciona operaciones de consulta sobre conjuntos de objetos
Seguridad	Establece un modelo de seguridad para objetos *CORBA*
Persistencia	Permite almacenar de forma persistente los objetos *CORBA* en ficheros o bases de datos
…	…



Reflexión

¿Cómo localiza un cliente al objeto que quiere llamar?

-Leer un fichero con una IOR+obtener referencia
-Pasar por línea de comandos una IOR+obtener referencia
-O, mejor, usar un servicio de nombres



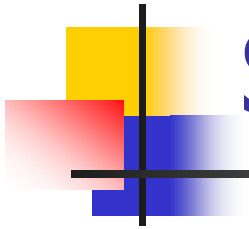
Servicio de Nombres (I)

Conceptos importantes:

- ***Name binding***: asociación nombre-objeto
- ***Name context***: objeto que contiene un conjunto de *name bindings* únicos en él

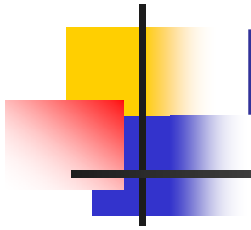
- Un objeto puede tener nombres distintos en diferentes contextos

Todos los nombres son relativos a su contexto



Servicio de Nombres (II)

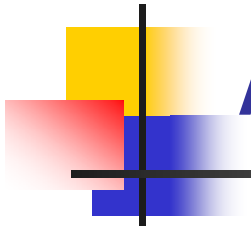
- Operaciones:
  - ***Resolve a name***: obtener el objeto asociado con el nombre en un contexto dado
  - ***Bind a name***: crear un *name binding* en un contexto dado
- Los nombres pueden ser compuestos
- Los contextos se pueden distribuir



Reflexión

¿Qué necesitamos para poder acceder al servicio de nombres?

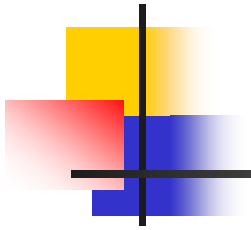
Saber dónde está



Algunos Productos CORBA

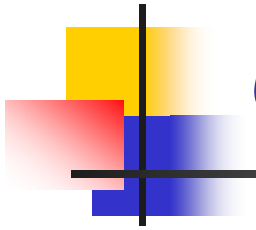
ORB	Description
Java 2 ORB	Parte de Java2 SDK
VisiBroker	Inprise Corporation
OrbixWeb	Iona Technologies
WebSphere	Servidor de aplicaciones con ORB, IBM
Netscape Communicator	Versión de VisiBroker embebida
ORBs de código abierto	JacORB, OpenORB, etc.

Todos los citados en la tabla soportan Java



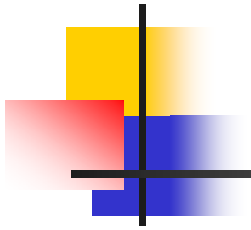
CORBA vs. *RMI* (I)

- Lenguaje:
  - *RMI* => Java para cliente y servidor
  - *CORBA* => distintos lenguajes
- Objetivo (de ambos):
  - Construir sistemas distribuidos
- Usabilidad:
  - *RMI* es más fácil de aprender y usar
  - Código más pequeño y simple



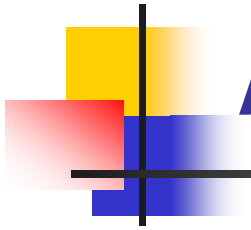
CORBA vs. *RMI* (II)

- *CORBA* no tiene recolección de objetos distribuida, *RMI* sí
- *CORBA* proporciona muchos servicios adicionales (gestión de transacciones, descubrimiento de servicios, etc.)
- *CORBA* usa el protocolo *IIOP* en lugar de *JRMP (Java Remote Method Protocol)*
  - Aunque también está RMI-IIOP (sin IDL)



CORBA vs. *RMI* (III)

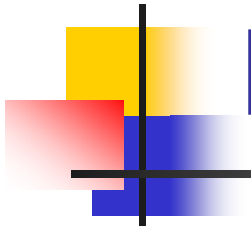
- Tres nuevos elementos en *CORBA*:
  - *Object Adapter*
  - *Implementation Repository*
  - *Interface Repository*



Agradecimientos

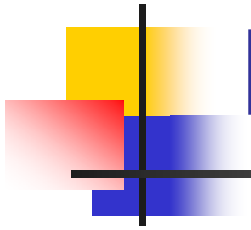
Algunas de las transparencias e ideas de esta presentación están inspiradas o han sido adaptadas de trabajos de:

Celsina Bignoli (http://www.smccd.edu/accounts/bignolic/)
Aurélien Esnard (http://www.labri.fr/perso/esnard/)
Fernando Bellas (http://www.tic.udc.es/~fbellas/)
Alex Chaffee (http://www.purpletech.com/)



Referencias (I)

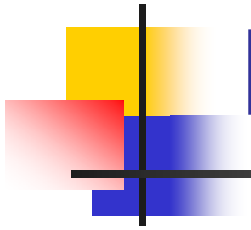
- OMG website: http://www.omg.org/
- OMG's CORBA website: http://www.corba.org/
- CORBA Basics: http://www.omg.org/gettingstarted/corbafaq.htm
- Introduction to CORBA (Sun, sólo hasta Java 1.3):
  http://java.sun.com/developer/onlineTraining/corba/corba.html
- Java IDL (Sun):
  http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html
- Especificaciones CORBA:
  http://www.omg.org/technology/documents/corba_spec_catalog.htm
- CORBA downloads:
  http://www.omg.org/technology/corba/corbadownloads.htm
- The free CORBA page:
  http://adams.patriot.net/~tvalesky/freecorba.html



Referencias (II)

- Java Enterprise in a Nutshell. David Flanagan, Jim Farley, William Crawford, Kris Magnusson, ISBN 1565924835E:

  http://www.unix.org.ua/orelly/java-ent/jenut/index.htm



Fin

Gracias por vuestra atención