Desarrollo de software basado en componentes

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es

DDBCS

Máster Universitario en Ingeniería Informática

Índice

Índice

Índice

Fundamento del desarrollo basado en componentes

Definition

Sistema *independientemente extensible*

Sistemas Abiertos

- Concurrentes
- Reactivos
- Independientemente extensibles
- *Ingreso* y *salida* dinámica de componentes-software en el sistema

Problemática en el desarrollo y evolución de los sistemas abiertos

Componente

Definición

- Una unidad de composición de aplicaciones software con interfaces especificadas mediante contrato y un conjunto de requisitos que sólo posee dependencias de contexto explícitas.
- Puede ser desarrollado, distribuido independientemente de otros, incorporado al sistema de forma independiente y es propenso a la composición con otros componentes desarrollados por terceras partes, en tiempo y espacio.

Componentes vs. Objetos

Características de un componente

- Unidad de despliegue independiente
- Unidad de composición de terceras partes
- No pueden tener un estado *externamente visible*
- No existe *herencia de implementación* de componentes

Características de un objeto

- Unidad de instanciación que posee 1 sola identidad
- Con estado que puede ser externamente observable
- Encapsula estado y comportamiento

Componentes vs. módulos

- Los módulos son unidades de compilación separada
- Pueden implementar TDA y comprobación de tipos
- No se instancian ni definen *referencias de objetos*
- Módulos que no contienen ninguna clase podrían ser considerados componentes
- Los módulos utilizan variables estáticas globales para mostrar su estado ($\neq$ componentes)
- *Demasiada* dependencia estática externa

- El uso de la modularidad es un pre-requisito para la tecnología de componentes software
- Los componentes necesitan extensión independiente y control explícito sobre dependencias que no les pueden proporcionar los módulos

POC

Concepto

La POC es una extensión natural de la POO de aplicación a entornos abiertos, en los que ésta última presenta limitaciones.

POC II

Limitaciones de la POO

- No diferencia entre aspectos *computacionales* y *composicionales* de las aplicaciones
- Dificultades para reutilización de objetos con estados externamente visibles
- Interfaces de demasiado bajo nivel

Componentes

Definiciones

- Unidad de composición con interfaces especificadas por contrato y sólo dependencias explícitas de su contexto (Szyperski)
- Conjunto de *elementos atómicos*={modulo +recursos} simultáneamente desplegados
  (recursos = “colección congelada de elementos tipados"que parametrizan a los componentes)
- Elemento que cumple con los 7 criterios de Meyer
- Una unidad binaria que exporta e importa funcionalidad utilizando un mecanismo de interfaz estándar (Stal)

Criterios de Meyer

Los 7 criterios:

1. May be used by other software elements
2. May be used by clients without the intervention of the component's developer
3. Includes a specification of all dependencies
4. Includes a specification of the functionality it offers
5. Is usable on the sole basis of its specifications
6. Is composable with other components
7. Can be integrated into a system quickly and smoothly

Componentes II

Estructura común a todas las definiciones

- *Parte técnica*: {interfaces contractuales, composicionalidad, independencia del entorno}
- *Parte mercadotécnica*: papel que juegan las “terceras partes”, facilidad de despliegue
- Establecimiento de *invariantes* (algunas definiciones)

Paradigma diferenciado de programación

Conceptos básicos de la POC

- Composición tardía
- Entornos
- Eventos
- Reutilización (caja blanca, cristal, gris y negra)
- Contratos
- Polimorfismo: reemplazabilidad, parametrización, acotación
- Seguridad
- Reflexión

Conceptos

Modelo de componentes

1. Forma de las interfaces de los componentes
2. Mecanismos de interconexión entre interfaces
3. Asumidos por las plataformas:
   -entorno de desarrollo y ejecución
   -aislar la mayor parte de las dificultades y aspectos específicos

Relación entre modelo de componentes y marcos de trabajo

Modelos de componentes II

Figura: Modelo y Plataforma de Componentes

Ejemplos de plataformas de componentes

Plataforma de componentes

Un entorno de desarrollo y de ejecución que permitirá aislar la mayor parte de dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en componentes que siguen un modelo en particular.

Modelo	COM	JavaBeans	CORBA
Plataforma	ActiveX/OLE	EJB	Orbix

Marcos de Trabajo

Diseño reutilizable de todo o parte de un sistema, representado por un conjunto de clases abstractas y la forma en que estas interactúan.

Características

- Esqueleto de una aplicación que debe ser adaptado
- Conjunto de punto de entrada o ganchos, que conforman la parte variable de la aplicación
- Problema de la documentación y evolución

Patrones

Un patrón de diseño ofrece una solución abstracta a un problema de diseño que aparece muy frecuentemente. Se expresa mediante un conjunto de relaciones e interacciones entre componentes.

Características

- Diseñar la arquitectura de los marcos de trabajo
- Mejor documentación arquitectónica
- Catálogos de patrones dependientes e independientes

Arquitecturas software

Representación de alto nivel de la estructura de un sistema o aplicación, que describe las partes que la integran, las interacciones entre ellas, los patrones que supervisan su composición y las restricciones a la hora de aplicar estos patrones.

Características

- Colección de componentes y sus interacciones
- Comprender y manejar la estructura de aplicaciones complejas
- Reutilizar dicha estructura o partes de ella
- Planificar la evolución de la aplicación

Desarrollo de software orientado a componentes

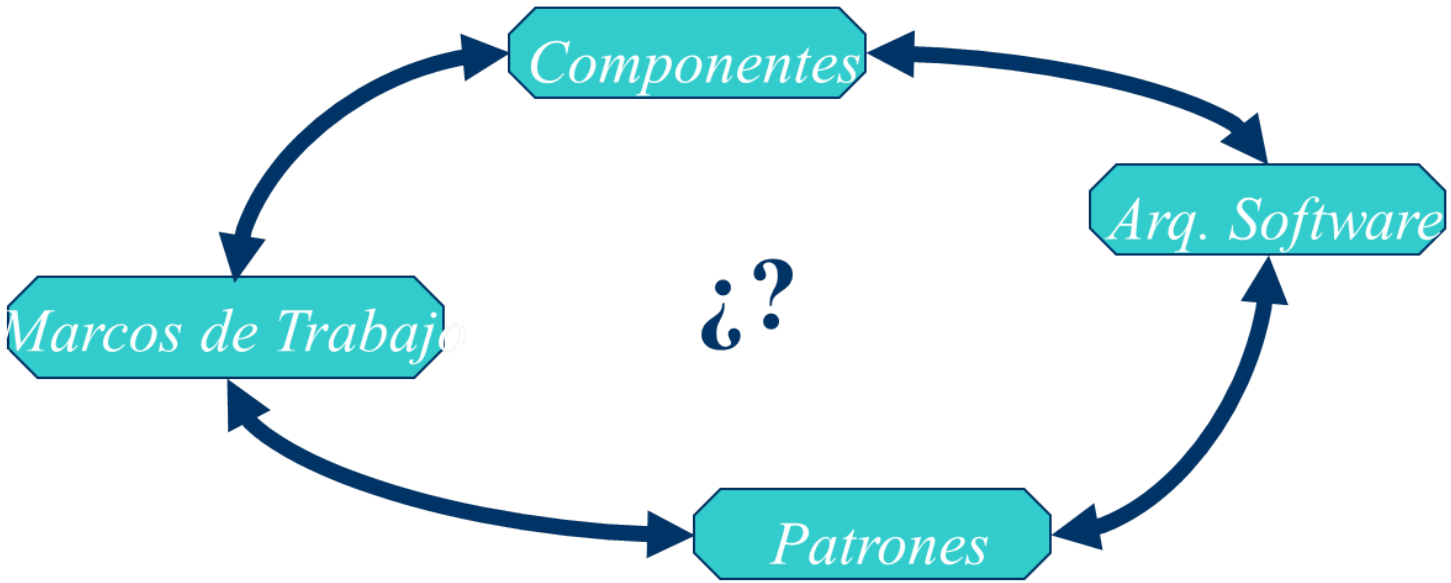

Figura: Ontología Simple- Software Basado en Componentes

Arquitecturas software y componentes

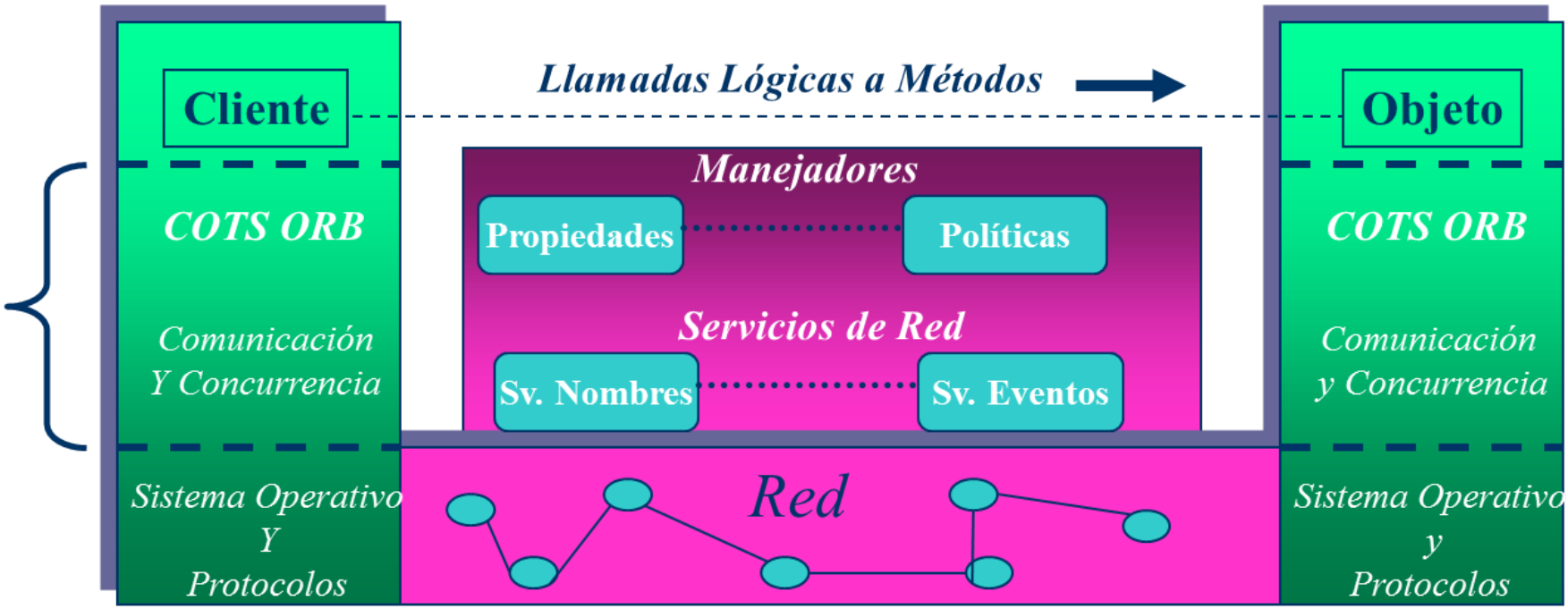

Figura: Componentes y su Integración en Arquitecturas

Modelos de componentes y UML

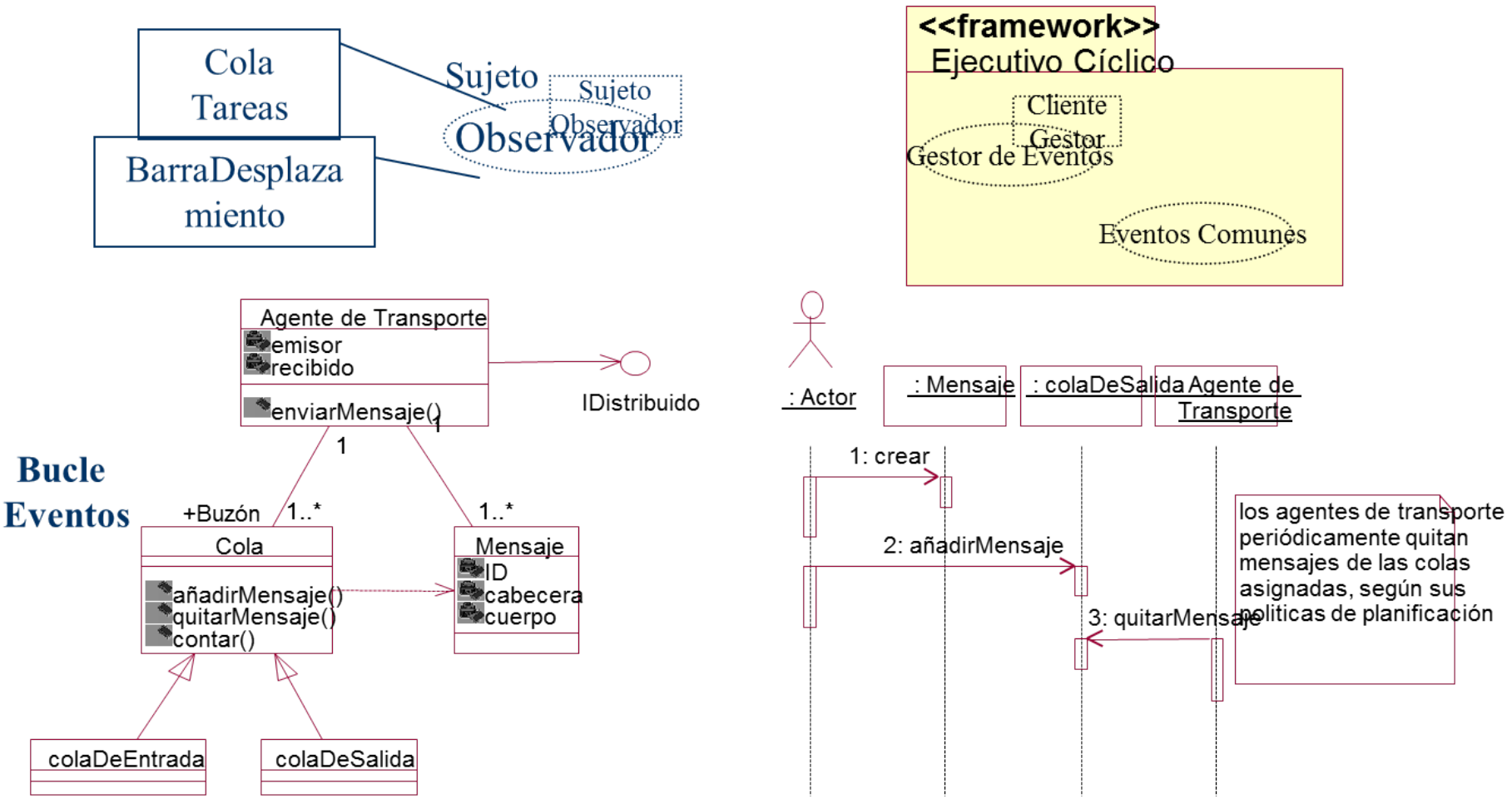

Figura: Correspondencia entre Middleware y UML

Interfaces

- Se ha de interpretar como un contrato del componente: lo que ofrece y lo que recibe a cambio
- Interfaz= {atributos, métodos públicos y eventos}
- Especificación de *signaturas* y condiciones de los eventos
- No sobrespecificación ni de los posibles consumidores ni de la forma de utilización
- Se suelen seguir 2 modelos: {RPCs, Publicar y Suscribir}

Soluciones basadas RPCs vs. Marcos de Trabajo

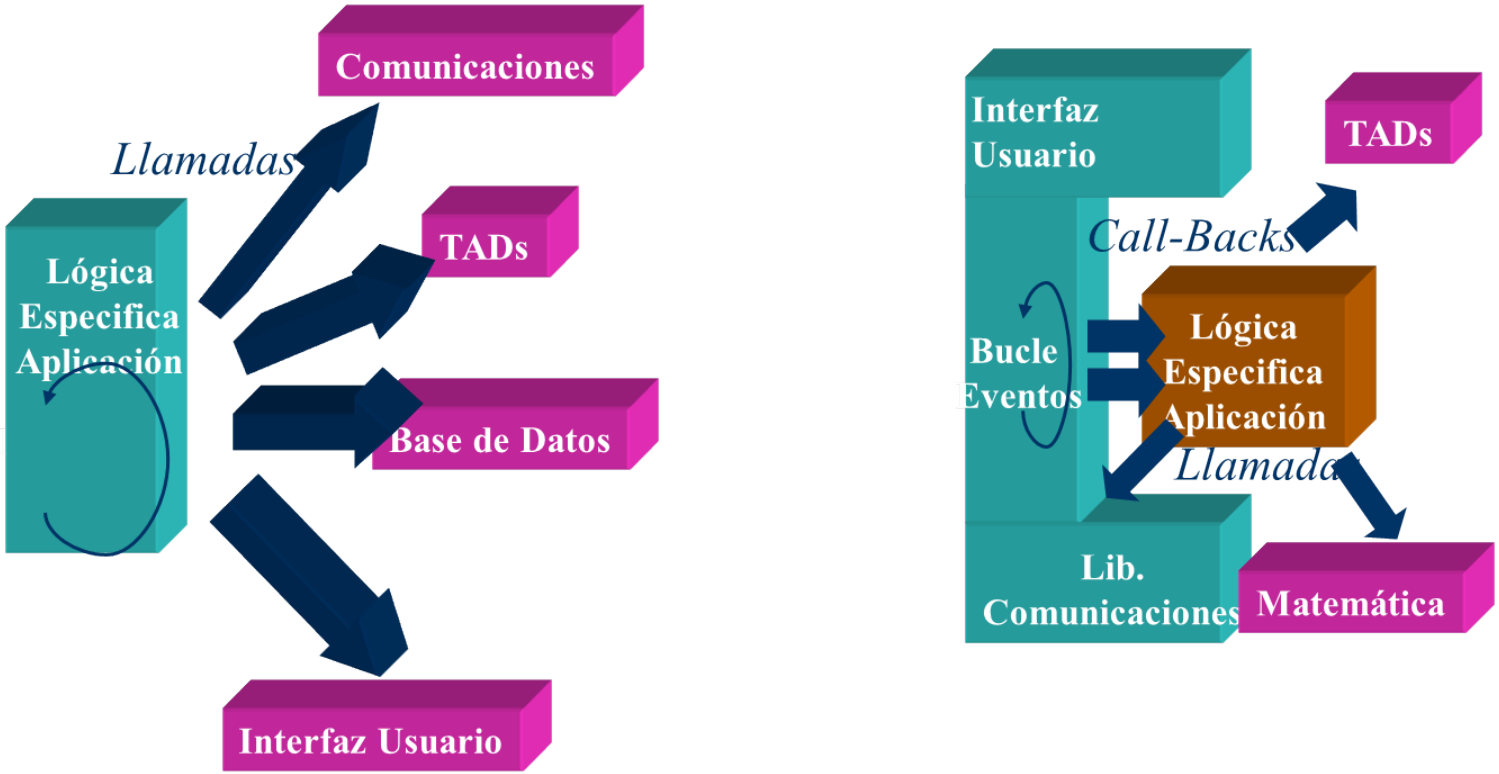

Figura: Diferencias entre marcos de trabajo y desarrollo basado en librerías

Modelo Publish-Subscribe para Eventos

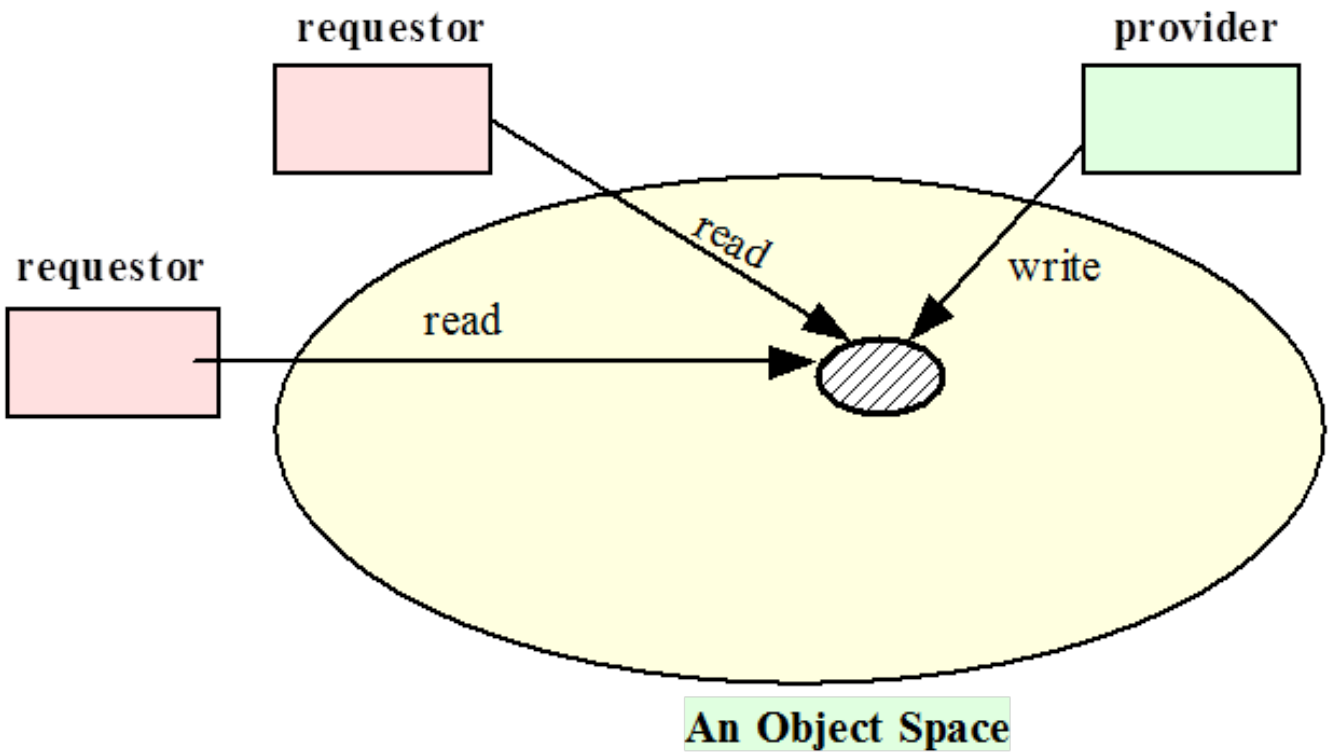

Figura: Modelo Abstracto de Computación Distribuida

se pueden suscribir a los mensajes de ese evento
middleware distribuye el mensaje a los suscriptores
abstracción potente para multicasting o comunicación de grupo

Contenedores

- Los componentes existen y cooperan dentro (p.e.: los controles ActiveX) de *contenedores*:
  - entorno compartido de interacción
  - aplicación muy conveniente para *envolver* objetos visuales
  - adoptan un modelo reactivo de computación
  - no se modelan bien como objetos
- Las relaciones contenedor–componentes sólo mediante eventos
- Los contenedores pueden cambiar el aspecto gráfico de sus componentes
- Pueden pasarles referencias de objetos a los componentes incluidos

Metainformación

Se trata de toda la información que un componente ha de hacer pública sobre sí mismo y sus propiedades.

- Permiten descubrir la funcionalidad que ofrecen los componentes y contenedores para su manipulación por contenedores y otros componentes.
- La *inspección* de la metainformación puede ser estática, dinámica y en tiempo de ejecución
- La representación de metainformación se presta a utilizar *reflexión*
- *Introspección* si el modelo de componentes soporta *reflexión*

IDEs

Definition

Aplicación visual para construir aplicaciones a partir de componentes

Elementos de un IDE

- Lienzo o contenedor
- Editores para configurar y especializar componentes
- Visores y navegadores
- Directorios de componentes
- Herramientas para desarrollar componentes (compiladores, depuradores, pruebas unitarias, etc.)
- Acceso a control y gestión de proyectos
- Soporte para realizar CSCW

Ejemplos de IDEs actuales

Nombre	**Fabricante**
Visual Studio	Microsoft
Visual Age	IBM
Visual Cafe	Symantec
Eclipse	ESF
NetBeans	Oracle

Complementados con lenguajes de configuración: JavaScript, VBScript, etc.

Servicios y facilidades

Definition

La PD y la POC se basan en un conjunto de servicios que proporcionan a los componentes el acceso a los recursos compartidos de una forma segura y eficiente.

Tipos de servicios más comunes

- Comunicaciones remotas
- Directorio (asignación de nombres, localización y acceso)
- Seguridad
- Transacciones
- Gestión (monitorización, gestión y administración de todo)

Estándares

Evolución

1. Superación de los problemas de transportabilidad de llamadas en SOs y primitivos sistemas en red
   problema de las llamadas a procedimientos fuera de los límites de un proceso
   Sólo los *sockets* de Unix USB resultaban transportables
2. Primera solución al problema de la no-transportabilidad de las llamadas a procedimientos: RPCs
   Introduce nuevos conceptos: resguardos, proxies, marshalling, etc.

Arquitectura de una aplicación con RPCs

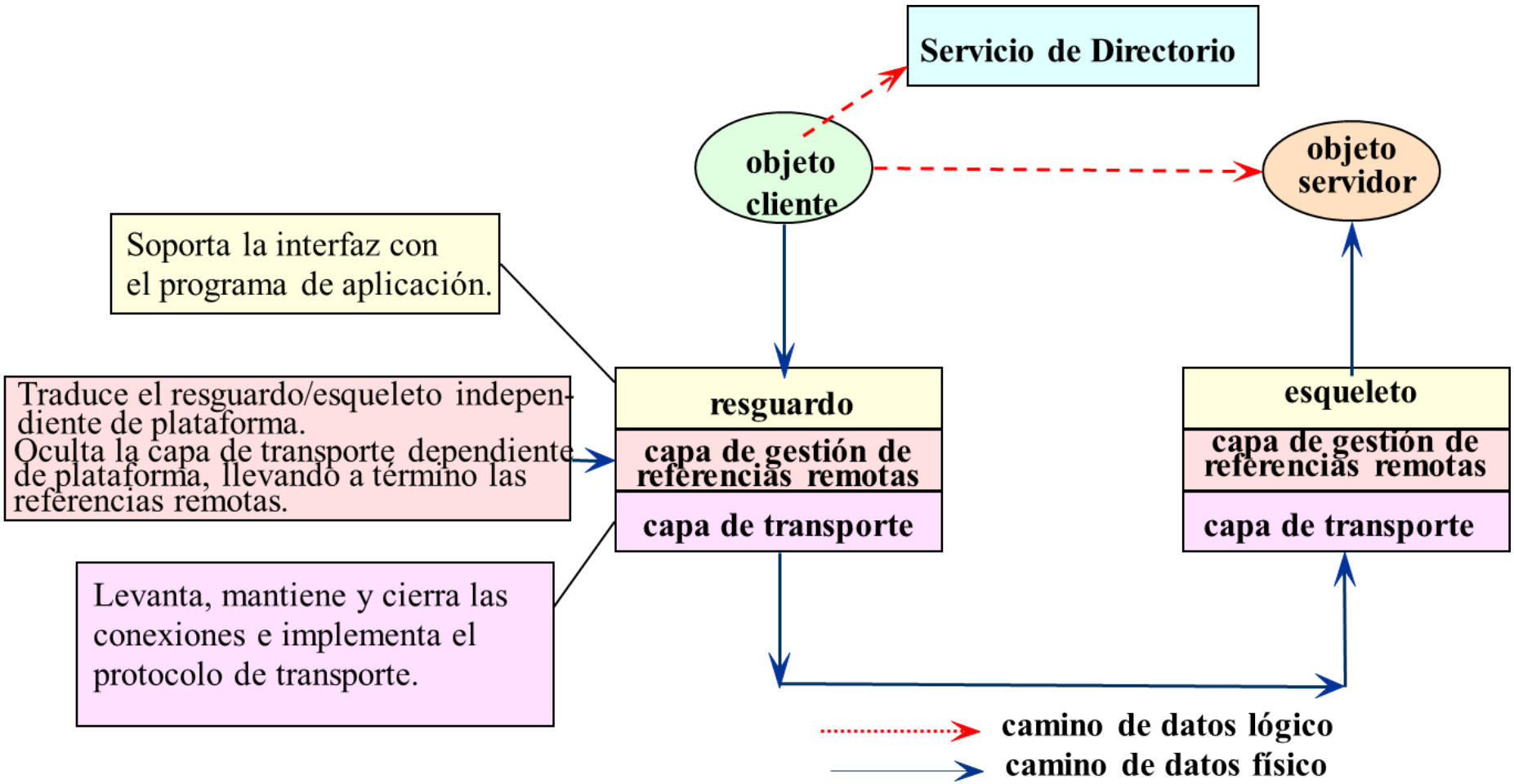

Figura: Arquitectura RPC

Estándares II

Evolución

1. IDL y UUID propuestos por DCE (OSF), etc.
2. Problema de llamadas a objetos remotos
3. librerías dinámicas (DLLs) y la *"solución"* de Microsoft

Distributed Computing Environment (DCE) de OSF es el servicio más importante que implementa el protocolo RPC para plataformas de computación heterogéneas.

DCOM: estructura y función

3 capas fundamentales:

- Función de un servidor COM
- Objeto COM y sus interfaces
- Interacción entre cliente y objeto

DCOM: conceptos iniciales

Interface

"Una colección nombrada de operaciones abstractas (o métodos) que representan una funcionalidad"

Object Class

"Una implementación concreta y nombrada de una o más interfaces"

Object (Object Class Instance)

"Una instanciación de algún *Object Class*"

DCOM: conceptos iniciales II

Object Server

"Un proceso responsable de crear y albergar instancias de objetos"

Client

"Un proceso que invoca un método de un objeto"

DCOM: interfaces

- Las interfaces ha de seguir el mismo diseño estándar de memoria (C++ virtual functions)
- Dicha especificación permite la integración a nivel binario de componentes escritos en diferentes lenguajes (C++,Java,Visual Basic)
- Las interfaces de los objetos se describen con el “Interface Description Language” (IDL):
  -encapsulación de datos
  -polimorfismo
- Implementación a bajo nivel:
- las tablas de *despacho* de DLLs utilizan variables y pueden proporcionar *ligadura dinámica*

DCOM: interfaces

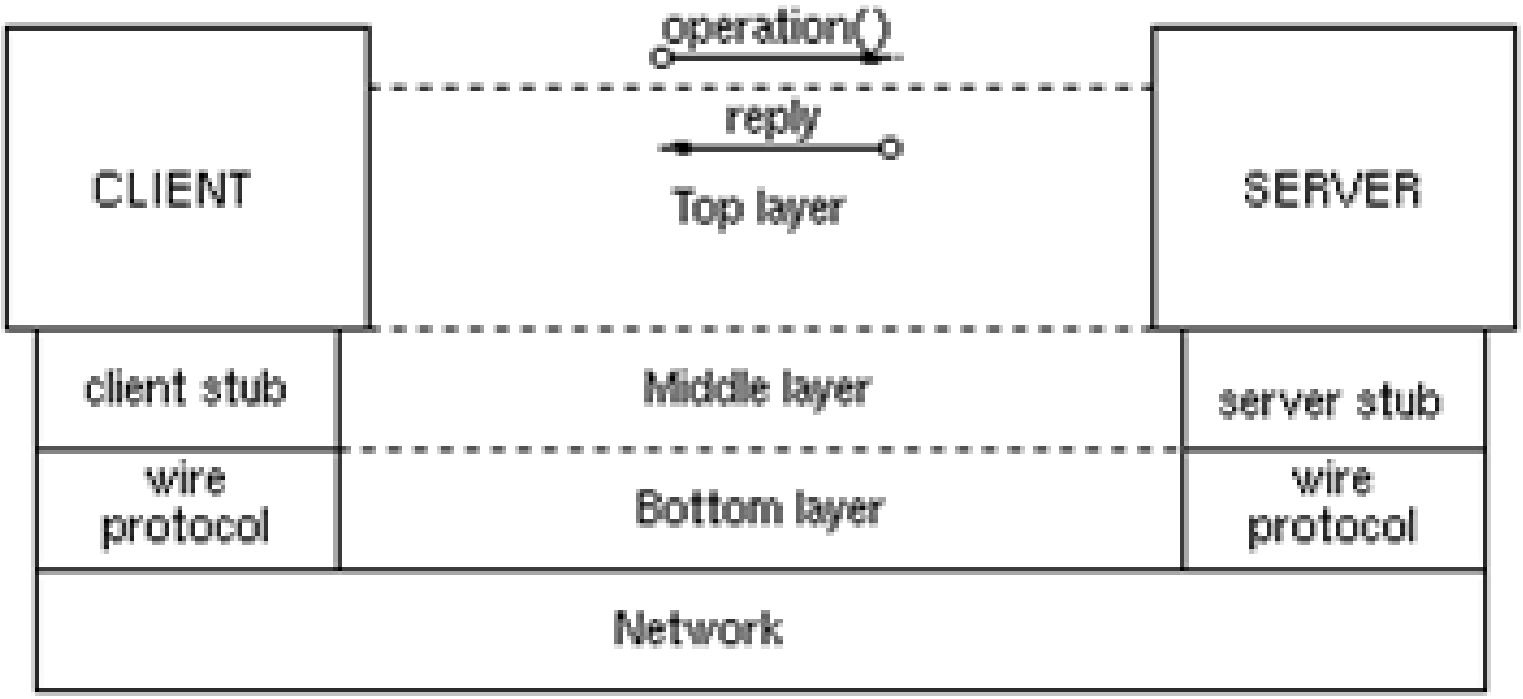

Figura: Interacciones entre Cliente y Servidor DCOM

Arquitectura DCOM: Capa superior

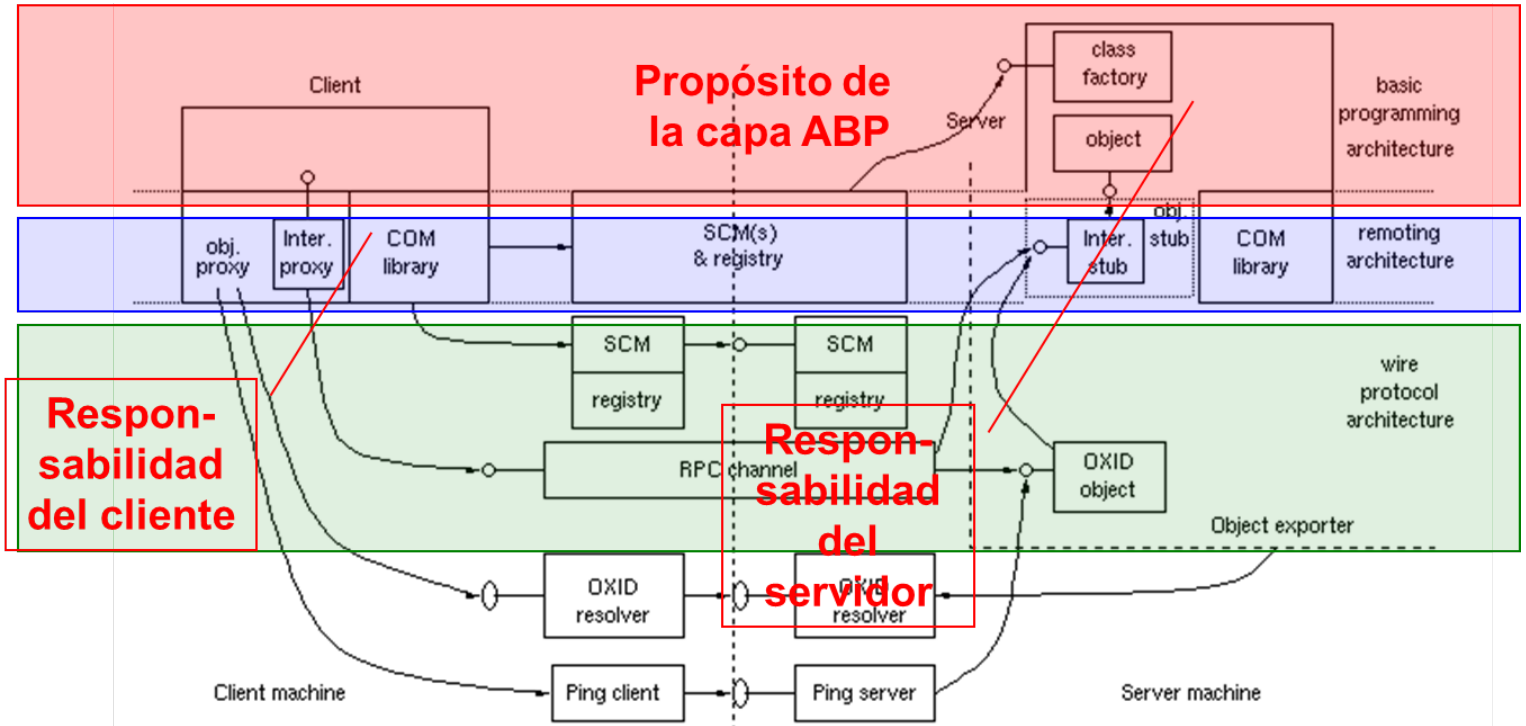

Figura: Comportamiento de la capa ABP de DCOM

El cliente y el servidor interaccionan como si residieran en el mismo espacio de direcciones y en la misma máquina.

Arquitectura DCOM: Capa intermedia

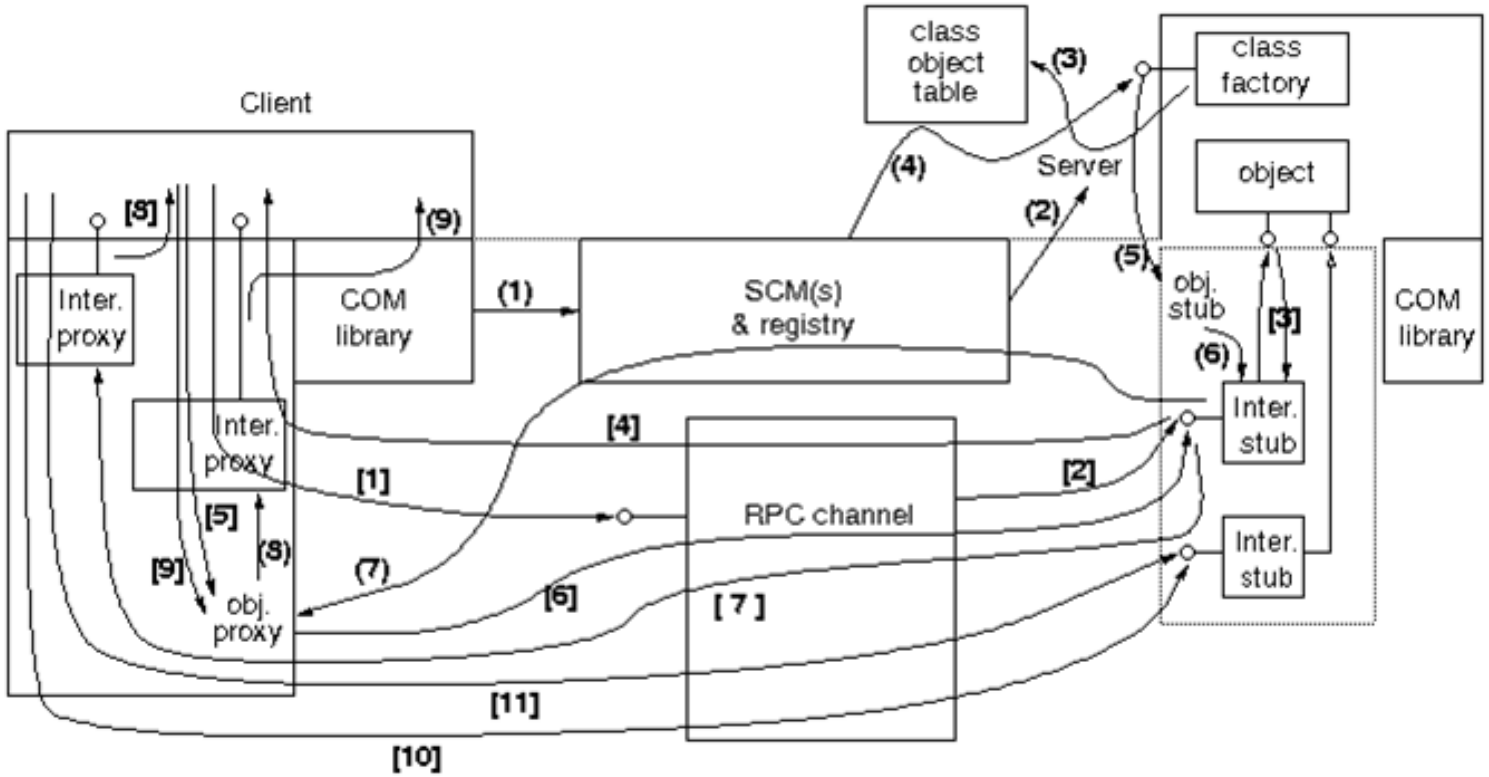

Figura: Funcionamiento capa intermedia de DCOM

Se utilizan “clases factoria” ubicadas en un servidor COM
No siempre se crean “instancias frescas” cuando se llama al método de creación de la factoría

Arquitectura DCOM: Capa intermedia II

Pasos de una llamada de creación a la clase factoría:

1. La biblioteca COM delega la tarea de creación
2. Se comprueba si se ha registrado una clase factoría con el nombre de la aplicación
3. Recupera el puntero a la clase factoría y llama al método de creación de instancias
4. Se crea un objeto *resguardo* para la instancia recién creada
5. El *resguardo* serializa y envía el puntero a la interfaz que se asocia con el objeto servidor
6. Se crea un *proxy* de interfaz y se le asocia un canal que lo conecta con el resguardo en el servidor mediante RPC
7. Devuelve al cliente un puntero al nombre de la interfaz

Arquitectura DCOM: Capa inferior

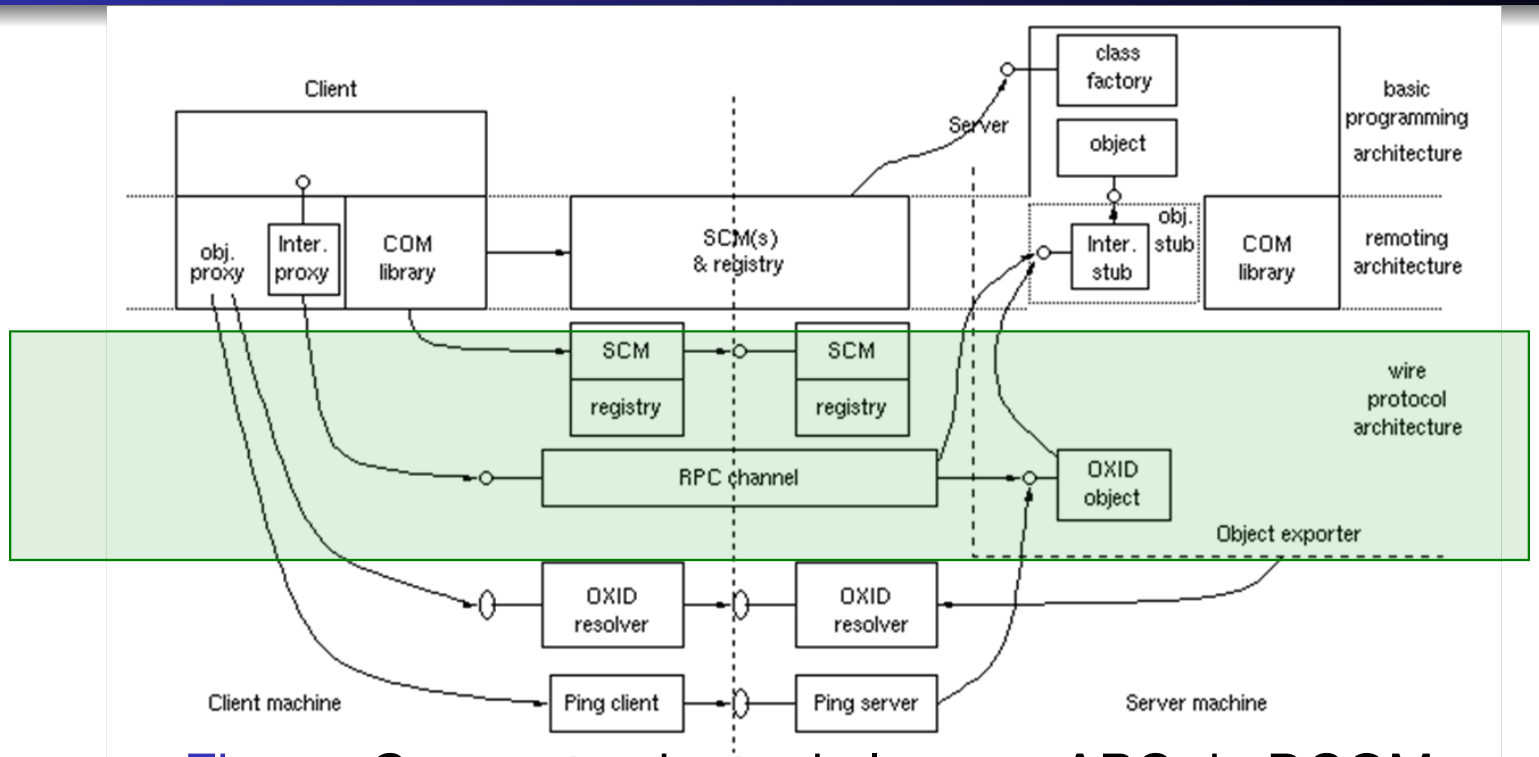

Figura: Comportamiento de la capa APC de DCOM

El protocolo de cables DCOM está basado principalmente en la especificación para RPCs de OSF DCE, con unas cuantas extensiones, que incluyen la representación de referencias remotas.

Estándares III

Evolución

1. Llamadas a métodos remotos con ligadura tardía mediante el ORB de CORBA
2. Implementación de una *Máquina Virtual* sobre el SO y la red: Java y .NET
3. Interoperabilidad más allá de los límites de la MV

CORBA

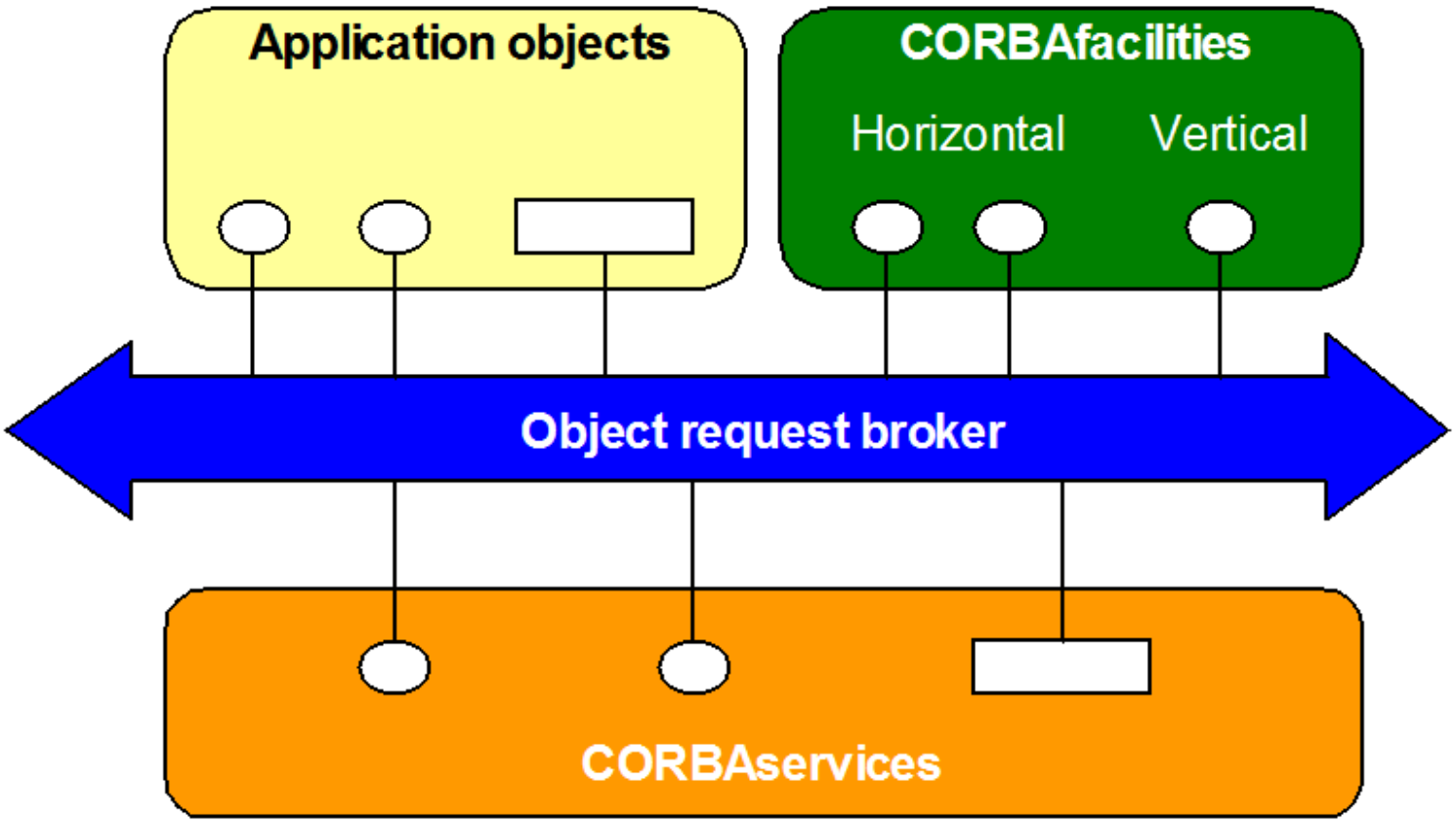

Figura: Arquitectura Software para Sistemas Distribuidos Heterogéneos

CORBA: el “bus de objetos”

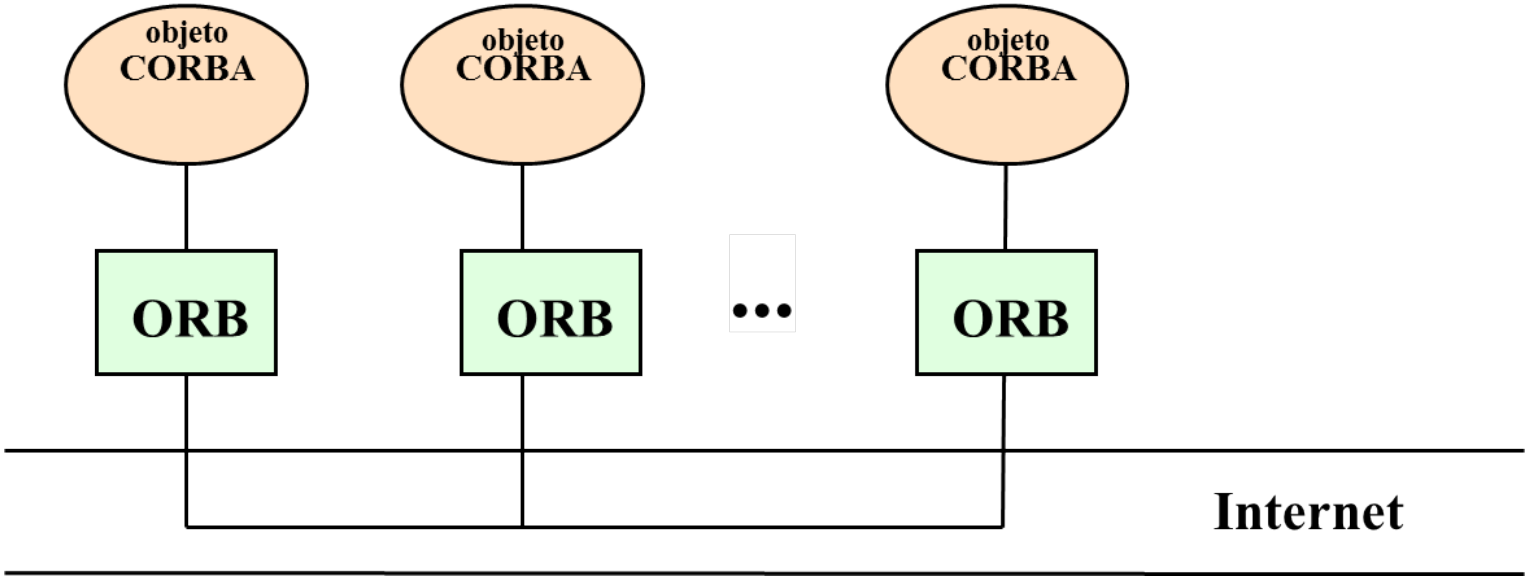

Figura: ORBs receptivos al IIOP conectados por Internet

Aplicación CORBA cruzando plataformas

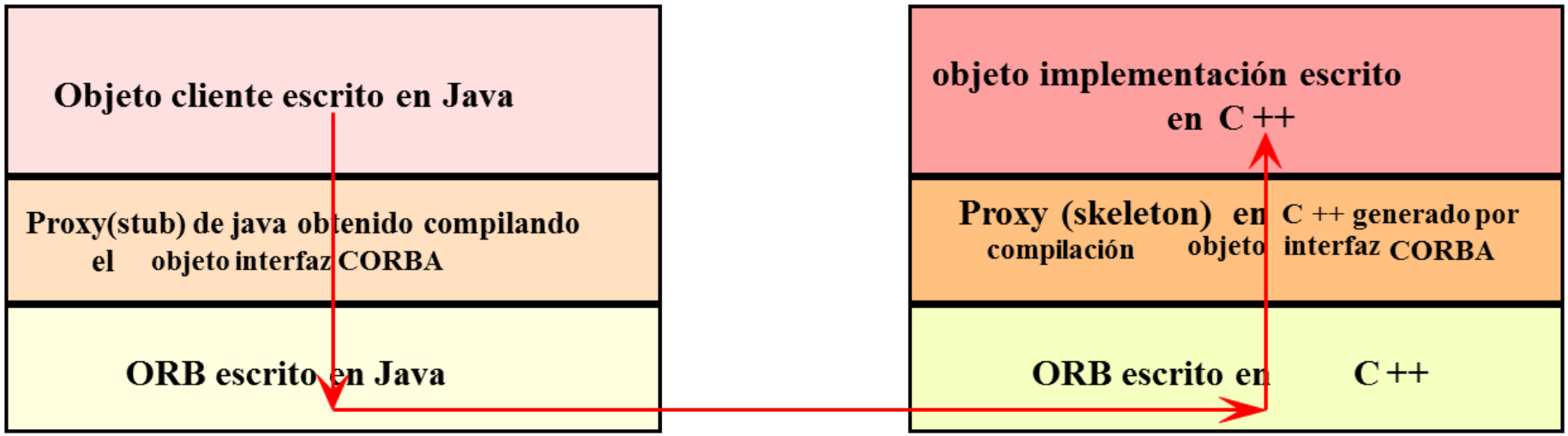

Figura: IDL y la interoperabilidad de lenguajes

Se sitúa en un metanivel con respecto a datos, atributos, métodos, interfaces, etc. de los lenguajes de programación
Su sintaxis es parecida a la Java o C++
IDL se aplica (map) a los lenguajes de programación, mediante un compilador de IDL.

Diferencias entre DCOM y CORBA

- Un cliente DCOM interacciona con un objeto COM adquiriendo un puntero a una de las interfaces del objeto
- El cliente de un objeto CORBA adquiere su referencia de objeto y la usa como un manejador para hacer llamadas a métodos, como si el objeto estuviera ubicado en el espacio de direcciones del cliente
- CORBA también soporta herencia múltiple a nivel de IDL, pero esto no lo tiene DCOM.

Implementación de una MV

Máquina Virtual

- Proporciona soporte para llamadas a objetos remotos uniforme, interoperable y transportable, para cada plataforma específica
- Enfoque seguido por Java Virtual Machine y el entorno `.NET`
- JVM proporciona además soporte especial para ser interoperable más allá de los límites de la MV local

Remote Method Invocation (RMI)

RPC "orientado a objetos" de Java

- En este modelo, un proceso invoca los métodos de un objeto, que puede residir en una máquina remota.
- Como ocurre con el protocolo RPC, los argumentos pueden pasarse con la invocación a un método.
- Un servidor de objeto exporta un *objeto remoto* y lo registra en un servicio de directorio
- Un objeto remoto es aquel que implementa una interfaz Java `remote`
- Se ofrecen métodos remotos, que pueden ser invocados desde clases Java en otras máquinas
- La API RMI permite utilizar varios servicios de directorio para registrar objetos distribuidos

Interoperabilidad

Para conseguirla entre componentes:

- Especificación estándar de interfaces
- Referencias a objetos más allá de los límites de MV
- Descubrimiento, localización y provisión de servicios
- Manejo de la evolución de los componentes

Interoperabilidad II

Cuestiones pendientes

- ¿Una o varias interfaces por objeto?
- Polimorfismo basado en implementación de interfaces
- ¿Herencia múltiple de interfaces?
- Nombrado y localización de servicios:
  - Identificadores únicos (UUID, IID, CATID y CLSID)
  - Localizadores de recursos encriptados y retransmitidos (IOR de CORBA)
  - ¿Servicios de directorio (`Registry` de JVM)o metainformación asociada a cada servicio?
  - Comprobación dinámica de interfaces por *reflexión*
  - Creación dinámica de servicios

Ejercicio

Acceder a la página `http://lsi.ugr.es/mcapel/docencia/tecnologiaobjetos2013/privado2/trabajos.shtml` y crearse aplicaciones CORBA y RMI de Java sencillas utilizando el lenguaje IDL para definir las interfaces y el compilador `idlj` de Java para generar los resguardos y los esqueletos