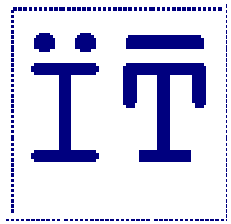
Sistemas de Información



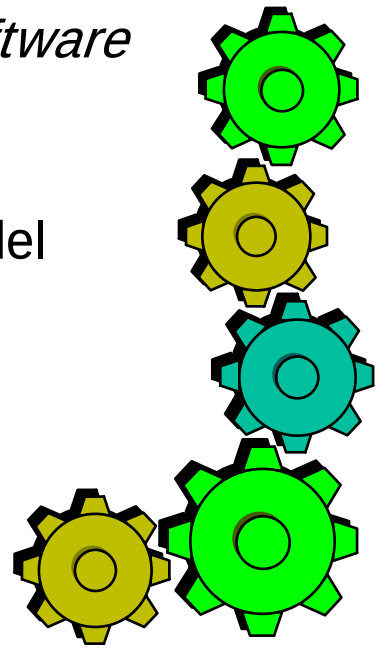
Tecnologías de Objetos Distribuidos

Servicios CORBA

Agradecimientos: esta presentación se basa parcialmente en una presentación de Juan Pavón Mestras de la UCM y en una de Jesus Villamor Lugo de IT/UCIIIM, la última basada a su vez en una presentación de Juan José Gil Ríos de Terra

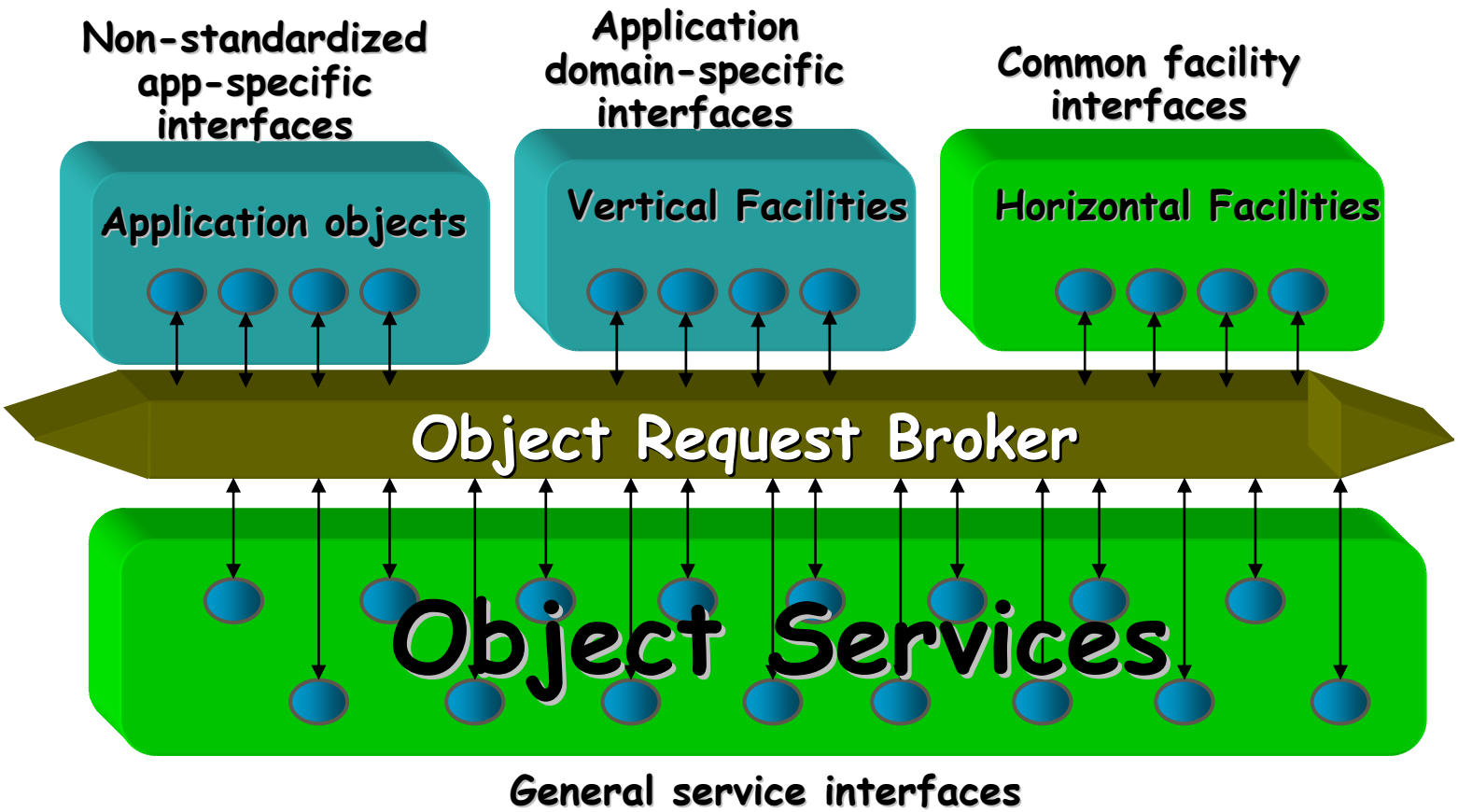
Objetivos de OMA

- Interfaces estándar que permiten extensibilidad del *software*
- Transición tecnológica gradual
- Productividad mejorada debido a la inter-operabilidad del *software* en redes **heterogéneas**
- Interfaces de usuario intuitivas
- Uniformidad y consistencia sobre diferentes e independientes aplicaciones (gracias a los estándares)



OMA ofrece a usuarios y desarrolladores, la filosofía de inter-operabilidad de sistemas software distribuidos, a través de la mayor cantidad de hardware, sistemas operativos y lenguajes de programación

Un breve repaso de OMA



COSS I

Cuatro servicios iniciales:

- **Nombrado** (*Naming*): correspondencia entre nombres convenientes de objetos y referencias a objetos reales.
- **Ciclo de Vida** (*Object Life Cycle*): creación, borrado, copiado y traslado de objetos.
- **Eventos** (*Event*): registro para la notificación requerida y esperada de la ocurrencia de eventos. Extensión: **Notificación** (*Notification*).
- **Estado persistente** (*Persistent State*) , antes **Objeto Persistente** (*Persistent Object*): existencia a largo plazo de objetos, gestión del almacenamiento de objetos.

publicadas por OMG/Wiley como COSS Volumen I en 1994

COSS II

Cuatro servicios más:

- **Relación** (*Relationship*): gestión de representación y consistencia de relaciones entre objetos.
- **Externalización** (*Externalization*): capacidad para almacenar la representación de objetos en medios removibles y permitir más adelante la re-internalización.
- **Transacciones** (*Transaction*): combina el paradigma de transacciones y el de objetos para tratar los problemas del procesamiento de transacciones comercial.
- **Control de Concurrencia** (*Concurrency Control*): gestión de la ejecución concurrente en un entorno distribuido.

publicadas como **CORBAServices** en 1995.

COSS III y COSS IV

- **Consulta** (*Query*): permite a usuarios y objetos invocar consultas en colecciones de otros objetos (1995).
- **Propiedades** (*Property*): define operaciones para crear y manipular conjuntos de propiedades (parejas nombre-valor) asociadas a objetos (1995)
- **Licencia** (*Licencing*): mecanismos para que los productores controlen el uso de su propiedad intelectual (1995).
- **Seguridad** (*Security*): identificación, autenticación, etc. (1996).
- **Tiempo** (*Time*): hora y temporizadores (1996) Extensión: **Visión Realzada del Tiempo** (*Enhanced View of Time*).
- **Negociación** (*Trading*): localización de objetos (paginas amarillas) suministrando información del servicio requerido (1996).
- ...

Servicios CORBA
Servicios bien conocidos

- El ORB está configurado para dar las referencias de los servicios básicos, los servicios CORBA “bien conocidos”:
  - Servicio de Nombres (“NameService”)
  - Servicio de Trader (“TradingService”)
  - Repositorio de Interfaces (“InterfaceRepository”)
- Para obtener la referencia a uno de estos servicios (por ejemplo el Servicio de Nombres):

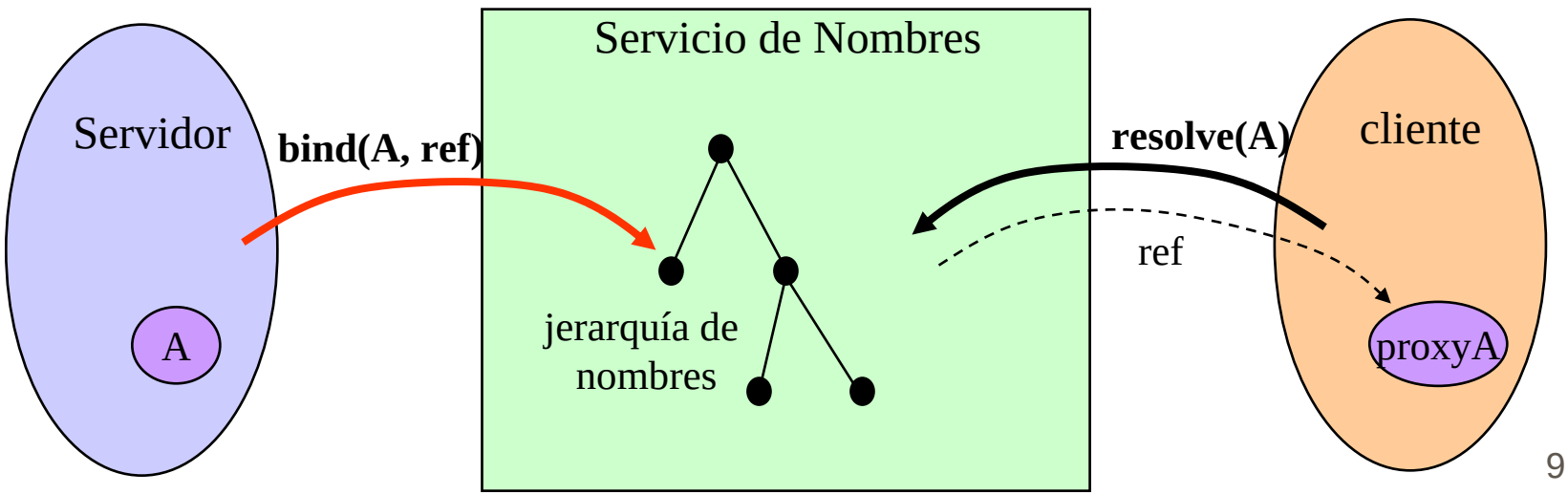
```
org.omg.CORBA.Object objeto =
            orb.resolve_initial_references(“NameService”);
NamingContext ns = NamingContextHelper.narrow(objeto);
```

Servicio de Nombrado

- Concepto básico
  - Cada objeto tiene un único ID (o referencia)
  - Opcionalmente se pueden asociar uno o más nombres a una referencia
  - Siempre se puede definir un nombre relativo a su contexto
- Servicio de Páginas Blancas
  - Encapsula el nombre en directorios tales como DCE CDS, ISO X.500 o SUN NIS+
  - La idea es no reinventar la rueda
  - La jerarquía de nombres no necesita un root universal

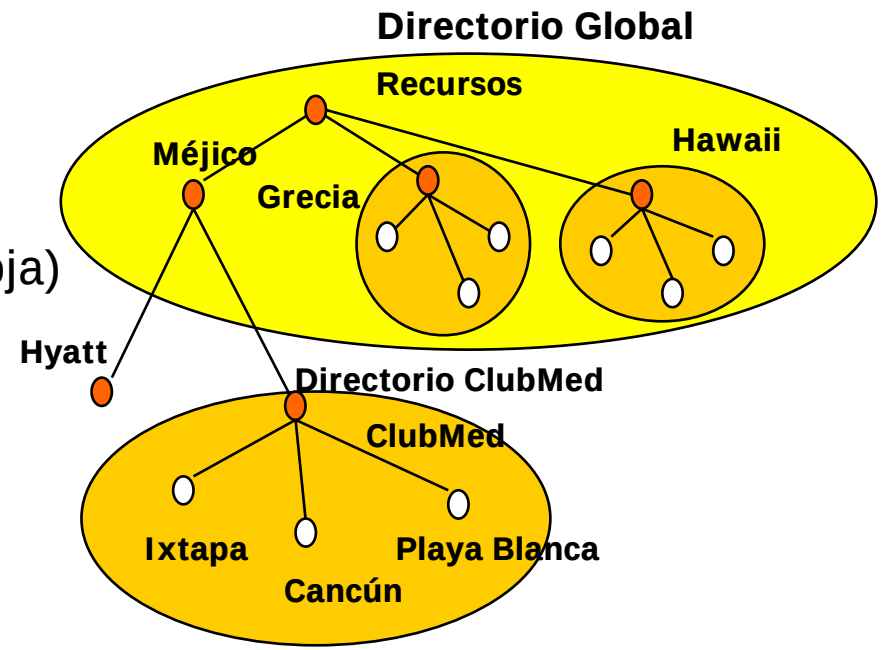
COSNaming
Uso del servicio

- El Servicio de Nombres guarda pares
  <**nombre, referencia a objeto**>
  - Los nombres están organizados en una jerarquía
- El Servicio de Nombres es usado por cliente y servidor:
  - El servidor asocia (**`bind`**) en el Servicio de Nombres una referencia a objeto con un nombre
  - El cliente puede pedirle al Servicio de Nombres que a partir de un nombre le dé (**`resolve`**) una referencia a un objeto CORBA



COSNaming
Jerarquía de nombres

- Los nombres están organizados jerárquicamente
  - Ejemplos de jerarquías: sistema de ficheros, direcciones y dominios en Internet, etc.
- Un nodo en la jerarquía puede ser:
  - Un contexto de nombres
    - define un espacio de nombres
  - Un nombre (necesariamente nodo hoja)
    - puede tener asociado una referencia a un objeto
- Un objeto puede tener asociados varios nombres
- La organización de la jerarquía de nombres es flexible
    - para adaptar otros servicios de directorio fácilmente (por ejemplo, DCS CDS, ISO X.500, Sun NIS+, Internet LDAP)



COSNaming
Nombres

- Cada **Name** está formado por varios NameComponent
- Cada **NameComponent** es un par: <identificador, clase>

```
typedef string Istring;
typedef sequence<NameComponent> Name;
struct NameComponent {
    Istring      id;
    Istring      kind;
};
```

- **id** es el nombre que identifica el objeto **NameComponent**
- **kind** cualifica **id** (no es obligatorio darle un valor)
- Ejemplos:

  (“C:”,”volumen”)(“usuario”,”dir”)(“juan”,”dir”)(“ejemplo.idl”,”archivo idl”)

  (“Compañía”,””)(“Sucursal”,””)(“Madrid”,”Ciudad”)(“Director”,”cargo”)

COSNaming
Interfaz **NamingContext**

```
interface NamingContext {
   NamingContext new_context();
   void destroy() raises (NotEmpty);
   Object resolve (in Name n) raises (NotFound, CannotProceed, InvalidName);
   void list (in unsigned long how_many, out BindingList b,
              out BindingIterator bi);
   void unbind(in Name n)
     raises (NotFound, CannotProceed, InvalidName);
   void bind(in Name n, in Object obj)
     raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
   void rebind(in Name n, in Object obj)
     raises (NotFound, CannotProceed, InvalidName);
   void bind_context(in Name n, in NamingContext nc)
     raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
   void rebind_context(in Name n, in NamingContext nc)
     raises (NotFound, CannotProceed, InvalidName);
   void bind_new_context(in Name n)
     raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
};
```

COSNaming
Interfaz **BindingIterator**

- La operación **list()** devuelve una lista de bindings:

```
enum BindingType { nobject, ncontext };
struct Binding {
   Name binding_name;
   BindingType binding_type;
};
typedef sequence<Binding> BindingList;
void list (in unsigned long how_many, out BindingList bl, out
   BindingIterator bi);
```

- La interfaz **BindingIterator** permite recorrer la lista de bindings:

```
interface BindingIterator {
   boolean next_one(out Binding b);
   boolean next_n(in unsigned long how_many, out BindingList bl);
   void destroy();
};
```

Creación de un nombre y asociación a un objeto

- Normalmente es el servidor quien crea un nombre y le asocia a una referencia a objeto
  1) Obtiene el **`NamingContext`** raíz llamando a **`resolve_initial_references("NameService")`**
  2) Crea los **`NamingContext`** necesarios (si no existen) usando **`bind_new_context()`**
  3) Crea las asociaciones a las referencias a objetos usando **`bind()`**

COSNaming
Excepciones

- En **`NamingContext`** define las siguientes excepciones:
  - **`NotFound`** Algún componente del nombre especificado no está en la jerarquía de nombres
  - **`InvalidName`** El nombre especificado no es válido
  - **`AlreadyBound`** El objeto ya está asociado con el nombre dado
  - **`NotEmpty`** El **`NamingContext`** tiene al menos un binding

- Por lo tanto el código anterior debería estar en un bloque **`try...catch`** para tratarlas

Obtener el objeto asociado a un nombre

- El cliente busca un nombre y obtiene el objeto asociado
  1) Obtiene el **`NamingContext`** raíz llamando a **`resolve_initial_references("NameService")`**
  2) Crea un **`Name`**, que puede ser compuesto por uno o más **`NameComponent`**
  3) Obtiene la referencia a un objeto invocando **`resolve()`** sobre el **`NamingContext`** raíz
  4) Hace **`narrow()`** de la referencia a objeto conseguida

Usando una jerarquía de nombres
Lado servidor: registrar el nombre

```
// Consigue la referencia al servicio de Nombres
org.omg.CORBA.Object nsObj =
                    orb.resolve_initial_references("NameService");
NamingContext ns = NamingContextHelper.narrow(nsObj);

// Primer contexto: “Divisiones”
NameComponent[] divisiones = {new NameComponent(“Divisiones","");};
NamingContext divisionesNC = ns.bind_new_context(divisiones);

// Segundo contexto: “Ventas”
NameComponent[] ventas = { new NameComponent(“Ventas",“Madrid"); };
NamingContext ventasNC = divisionesNC.bind_new_context(ventas);

// Nombre: “Director”
NameComponent[] director = { new NameComponent(“Director",""); };
ImplPersona implp = new ImplPersona();
Persona p = implp._this(orb);
ventasNC.bind (director, p);
```

Usando una jerarquía de nombres
Lado cliente: obtener el objeto

```
// Consigue la referencia al servicio de Nombres
org.omg.CORBA.Object nsObj =
                    orb.resolve_initial_references("NameService");
NamingContext ns = NamingContextHelper.narrow(nsObj);

// Prepara el nombre compuesto: “Divisiones” “Ventas” “Director”
NameComponent[] nombreDirector = new NameComponent[] {
                    new NameComponent(“Divisiones","”),
                    new NameComponent(“Ventas ",“Madrid"),
                    new NameComponent(“Director ","")
                };

// Resuelve el nombre en el contexto raíz
org.omg.CORBA.Object objetoDirector = ns.resolve(nombreDirector);

// Hace narrow a Persona
Persona p = PersonaHelper.narrow(objetoDirector);
```

Nombres interoperables
Esquema URI **Corbaloc**

- Permite especificar referencias de objeto como URIs:

  corbaloc:<*protocol*>:<*version*><*host*>:<*port*>/<*key_string*>

  <*version*> = <*major*> “.” <*minor*> “@”

  - Si no se especifica el protocolo, se supone IIOP (v1.1)
  - Si no se especifica el *host*, se supone *localhost*
  - Si no se especifica el puerto, se supone 2809

- Actualmente, sólo tres protocolos reconocidos:
  - protocolo IIOP: estándar de CORBA (GIOP sobre TCP/IP)

    corbaloc:iiop:<*host*>:<*port*>/<*key_string*>
  - protocolo RIR: **resolve_initial_references(key_string)**

    ej. **corbaloc:rir:/NameService**
  - protocolo IOR: *Interoperable Object Reference*

Nombres interoperables
Stringified names

- *Stringified names*: syntax para nombres de **`COSNaming::Name`** en forma de cadena (tipo *path*):
  - carácter “/”: para separar los **`NamingComponent`**
  - carácter “.”: para separar la cadenas del **`id`** y del **`kind`**
  - si el **`NamingComponent`** no tiene **`kind`** no se escribe el “.”
- Mecanismo de escape
  - Se puede utilizar los caracteres “/” y “.” dentro de la cadena del **`id`** o del **`kind`** escapandoles con un “\”

Nombres interoperables
Stringified names

```
interface NamingContextExt : NamingContext {
     typedef string StringName;
     typedef string Address;
     typedef string URLString;

     // converts a sequence of NameComponent into a stringified name
     StringName to_string(in Name n) raises(InvlaidName);

     // converts a stringified name into a sequence of NameComponent
     Name to_name(in StringName sn) raises(InvalidName);

     // resolves a stringified name into an object reference
     Object resolve_str(in StringName sn)
               raises(NotFound, CannotProceed, InvalidName);

     exception InvalidAddress{};

     // combines corbaloc address and stringified name into URL
     URLString to_URL(in Address addr, in StringName sn)
               raises(InvalidAddress, InvalidName)
};
```

Nombres interoperables
Esquema URI **Corbaname**

- Extensión del esquema *corbaloc* para el servicio de nombres
- "#" seguido por un *stringified name* añadido al final del URL
  - Si no se especifica la clave, se supone la cadena "NameService"
- Dos formas:

```
// resuelve <strname> en el contexto de nombrado definido
// por <host>:<port> y clave <key_string>
corbaname::<host>:<port>/<key_string>#<strname>

// resuelve <strname> en el contexto de nombrado devuelto
// por resolve_initial_references("NameService")
corbaname:rir:#<strname>
```

- Conversión de URLs en formato *corbaloc* o *corbaname* a referencias de objeto: **`CORBA::ORB::string_to_object()`**

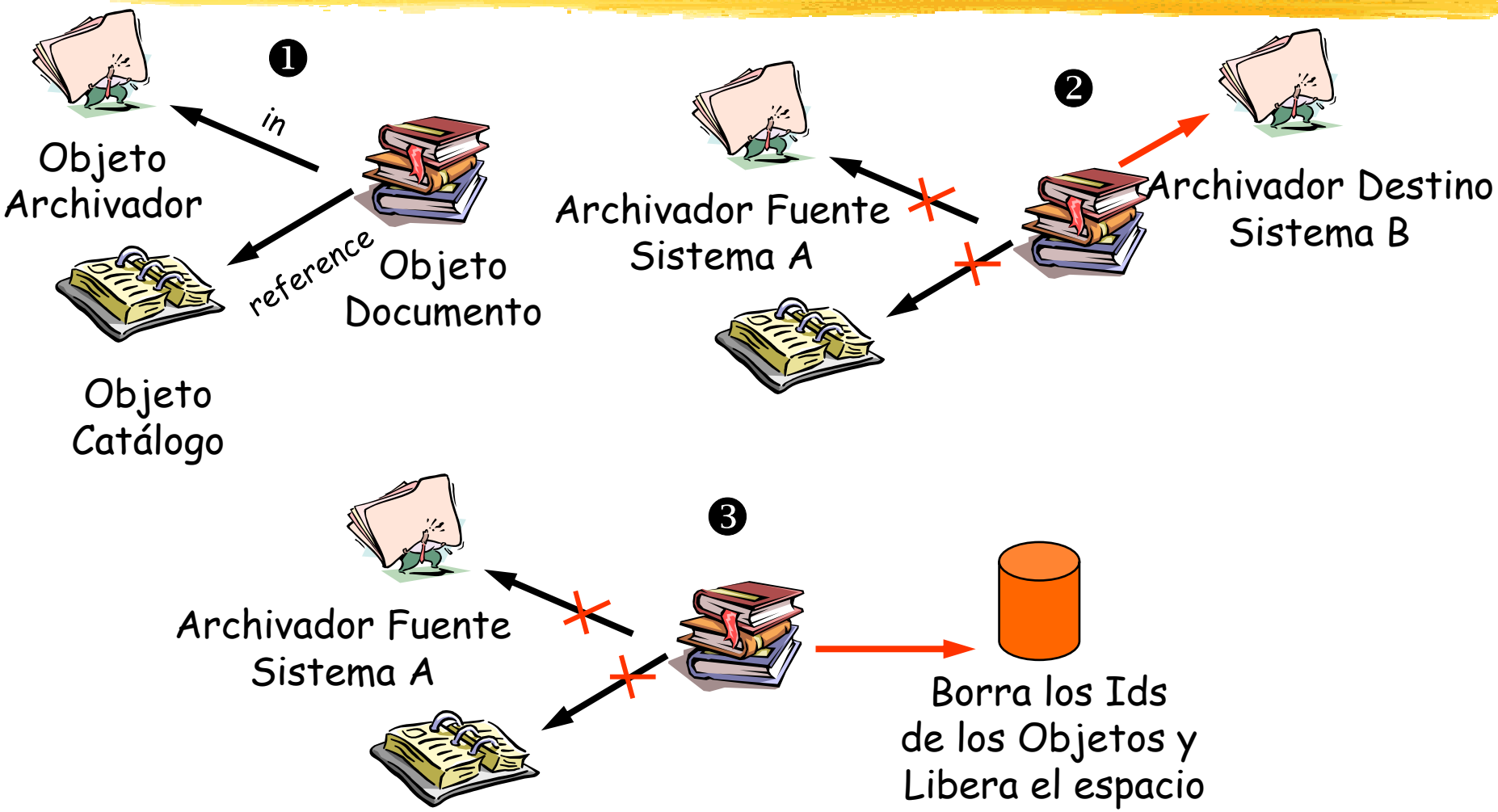
Nombres interoperables
Configuración de referencias iniciales

- -ORBInitRef
  - Se utiliza par dar la referencia de objeto de un servicio inicial
    -ORBInitRef *<ObjectID>=<ObjectURL>*
    - *ObjectID* = NameService, TradingService, NotificationService etc.
    - *ObjectURL* no puede ser de formato **`corbaloc:rir:`**
  - Ejemplo:
    `-ORBInitRef NameService=corbaname::myhost.example.com`
- -ORBDefaultInitRef
  - Proporciona una cadena prefijo para resolver nombres con **`resolve_initial_references`** que no se han podido resolver de otra manera

Servicio de Ciclo de Vida

- Concepto básico
  - Proporciona operaciones para crear, copiar, mover y borrar objetos
  - Trabaja en conjunción con el Servicio de Relación
    - Asociaciones de contenido (in)
    - Asociaciones de referencia (reference)
    - Constricciones entre objetos
  - Efecto del borrado de un objeto
    - Borra la referencia (su ID)
    - Libera espacio en memoria
    - Borra todas sus asociaciones con otros objetos

Un ejemplo

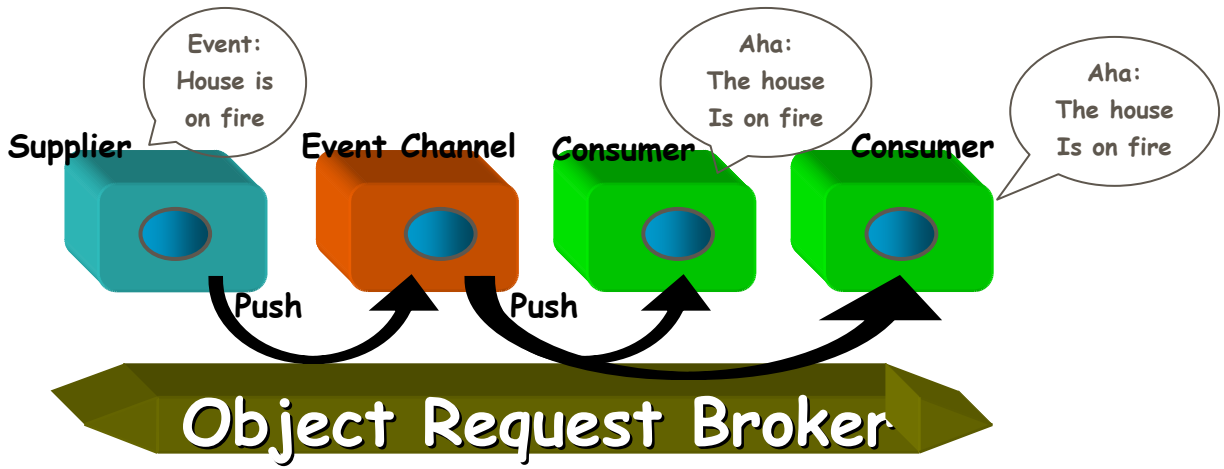

1
in
Objeto
Archivador
reference
Objeto
Documento
Objeto
Catálogo
2
Archivador Fuente
Sistema A
Archivador Destino
Sistema B
3
Archivador Fuente
Sistema A
Borra los Ids
de los Objetos y
Libera el espacio

Servicio de Eventos

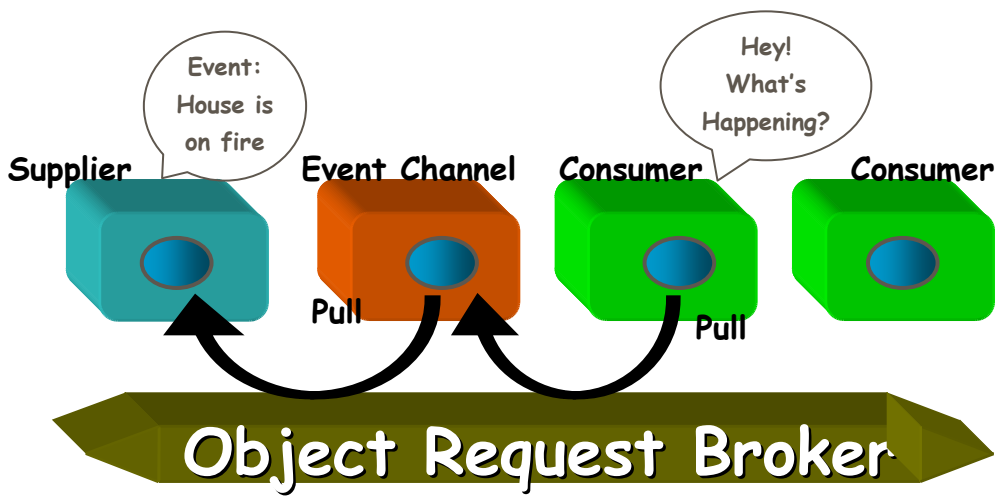
- Concepto básico
  - Un **evento** es una acción sobre un objeto que va a ser de interés para uno o más objetos
  - Una **notificación** es un mensaje que un objeto envía a sus participantes informando que un evento ha ocurrido
- Roles y canales
  - **Proveedores**: Producen eventos
  - **Consumidores**: Los procesan mediante manejadores
  - **Canal de Eventos**: Transmite eventos 1:1 ó 1:N
- Dos modelos
  - ***Push***: El proveedor toma la iniciativa y envía el evento
  - ***Pull***: El consumidor toma la iniciativa y pide el evento

Modelos de Eventos

- Estilo *push*:



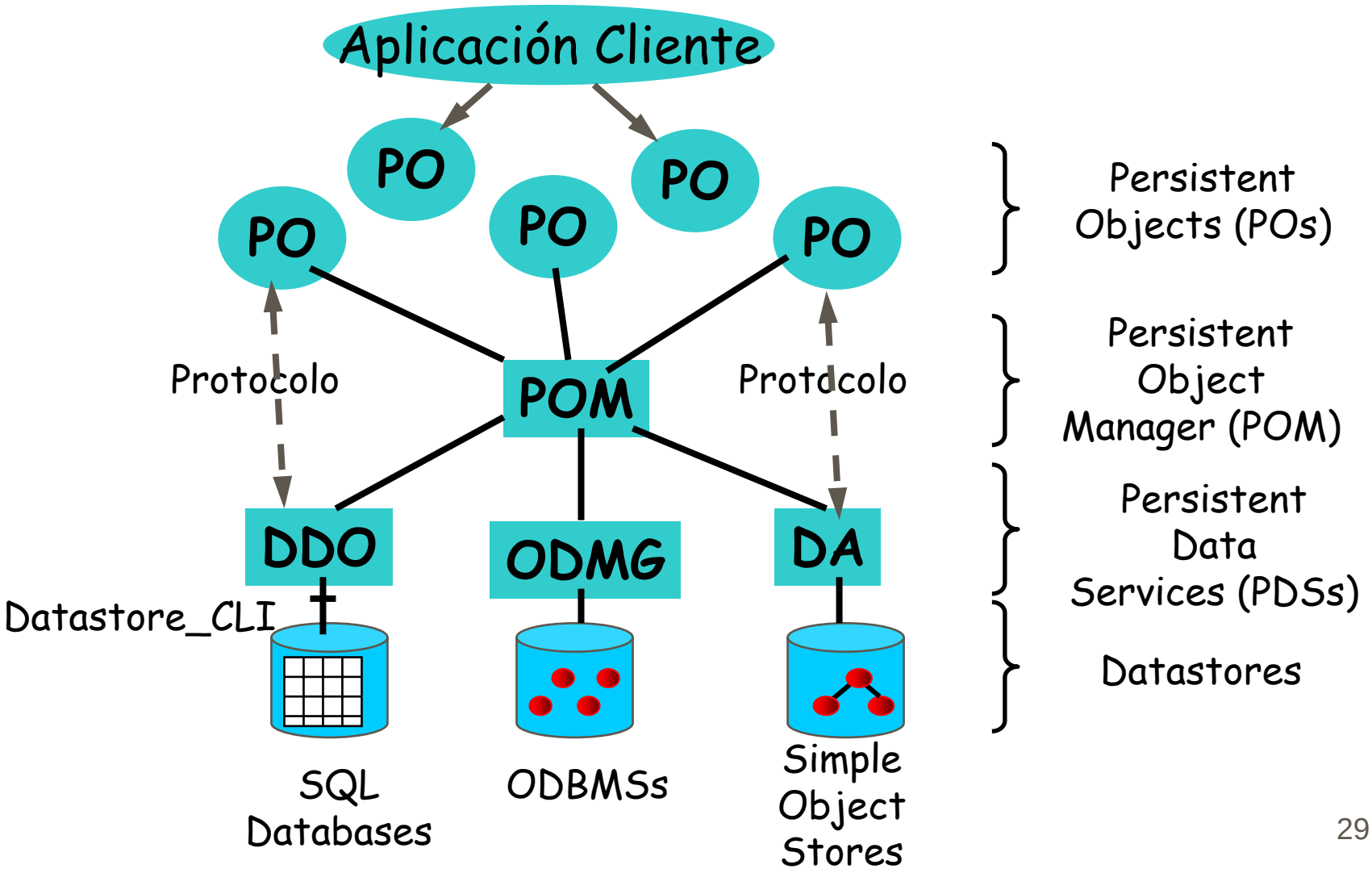
- Estilo *pull*:



Servicio de Persistencia

- Concepto básico
  - La **persistencia** almacena el estado de un objeto en un almacén no volátil de datos
- CORBA PSS *Persistent State Service* (nombre antiguo: POS *Persistent Object Service*)
  - Define interfaces a los datos de los objetos persistentes (*PO*) usando interfaces IDL bien definidas
- ***PO***: Persistent Object
  - Posee el grano más fino de persistencia
    - Colabora con sus Bases de Datos
  - Decide qué protocolo usar para el almacenaje
    - Delega la gestión a los servicios persistentes

Elementos del PSS



Servicio de Relación

- Concepto básico
  - Permite crear dinámicamente relaciones entre objetos inmutables
    - El servicio oculta a los objetos de que forman parte de una relación
- Motivación
  - Relaciones multi-direccionales
  - Manipulación por terceros (ni cliente, ni servidor)
  - Navegación por grafos de objetos
  - Herencia entre relaciones

Tipos de Relaciones y Otros Conceptos

- Tipos de Relaciones
  - Relación de propiedad (*ownership*)
  - Relación de contención (*containment*)
  - Relación de referencia (*reference*)
  - Relación de autoría (*authoring*)
  - Relación de empleo (*employment*)
- Rol, Grado, Cardinalidad
  - Rol: Papel que el objeto juega en la asociación
  - Grado: Número de roles en una asociación
  - Cardinalidad: Número máximo de asociaciones en que un rol está envuelto

Servicio de Externalización

- Concepto
  - Permite externalizar un objeto a un *stream* e internalizarlo desde un *stream*
    - El objeto se puede salvar y restaurar desde un fichero
    - El programador puede mover por el *stream* con un *cursor*
- Poder del *stream*
  - Permite la importación/exportación de objetos
  - Permite una persistencia básica
  - Los objetos se pueden copiar y mover
  - Los objetos se pueden pasar por parámetro

Servicio de Transacciones

- Concepto
  - La transacción es la unidad atómica que garantiza la ejecución consistente de una operación o conjunto de operaciones en un sistema distribuido
    - Implica un cliente junto con uno o más servidores
- Características (OTS)
  - Soporta transacciones planas y anidadas
  - Permite la convivencia con transacciones procedurales
  - Soporta transacciones sobre distinto ORBs
  - Hace transaccional el IDL sin más que heredar la implementación del IDL de una clase abstracta de OTS

Servicio de Control de Concurrencia

- Concepto
  - Provee interfaces que adquieren y liberan *locks* y permiten a los clientes coordinar su acceso a recursos compartidos
    - Permite asociar *locks* a transacciones concurrentes
    - El recurso que debe "candarse" es el servidor, no el cliente
    - Deben prevenirse los deadlock
- Operaciones básicas del *lock*
  - *read, write, intention read, intention write* y *upgrade*
  - Ej. de deadlock: varios clientes tienen un *read lock* y uno de ellos adquiere un *write lock*

COSS III y COSS IV

- **Consulta (*Query*)**. Permite a usuarios y objetos invocar consultas en colecciones de otros objetos (Mar/95).
- **Propiedad (*Property*)**. Define operaciones para crear y manipular conjuntos de propiedades asociadas a objetos (Nov/95)
- **Licencia (*Licensing*)**. Mecanismos para que los productores controlen el uso de su propiedad intelectual (Nov/95).
- **Seguridad (*Security*)**. Identificación, autenticación, etc. (Mar/96).
- **Tiempo (*Time*)**. Hora y temporizadores (Mar/96). Extensión: **Visión Realzada del Tiempo** (*Enhanced View of Time*).
- **Negociación (*Trading*)**. Localización de objetos (paginas amarillas) suministrando información del servicio requerido (Oct/96).

Servicio de Consulta

- Concepto
  - Permite consultar y manipular cualquier objeto CORBA
    - Sea volátil o persistente, local o remoto, individual o colectivo
- Dos lenguajes de consulta con el QS
  - OQL (ODMG-93's Object Query Language)
  - SQL (con extensiones orientadas a objetos)
- Modo de operar
  - Parecido al método de Invocación Dinámica
    - El cliente Crea el "QueryManager", quien crea la "Query"
    - Sobre él se prepara la consulta, se ejecuta y se obtiene el resultado iterando sobre una colección

Servicio de Propiedades

- Concepto
  - Son atributos dinámicos
    - Se pueden definir en tiempo de ejecución sin necesidad de usar IDL
    - Se pueden asociar a cualquier objeto que ya exista
    - No tienen por qué ser atributos definidos en el objeto a caracterizar
  - Interfaces
    - *PropertySet, PropertySetDef*
  - Operaciones
    - *define property, get value, set value, set mode, delete mode*

Servicio de Consulta

- Concepto
  - Permite consultar y manipular cualquier objeto CORBA
    - Sea volátil o persistente, local o remoto, individual o colectivo
- Dos lenguajes de consulta con el QS
  - OQL (ODMG-93's Object Query Language)
  - SQL (con extensiones orientadas a objetos)
- Modo de operar
  - Parecido al método de Invocación Dinámica
    - El cliente Crea el "QueryManager", quien crea la "Query"
    - Sobre él se prepara la consulta, se ejecuta y se obtiene el resultado iterando sobre una colección

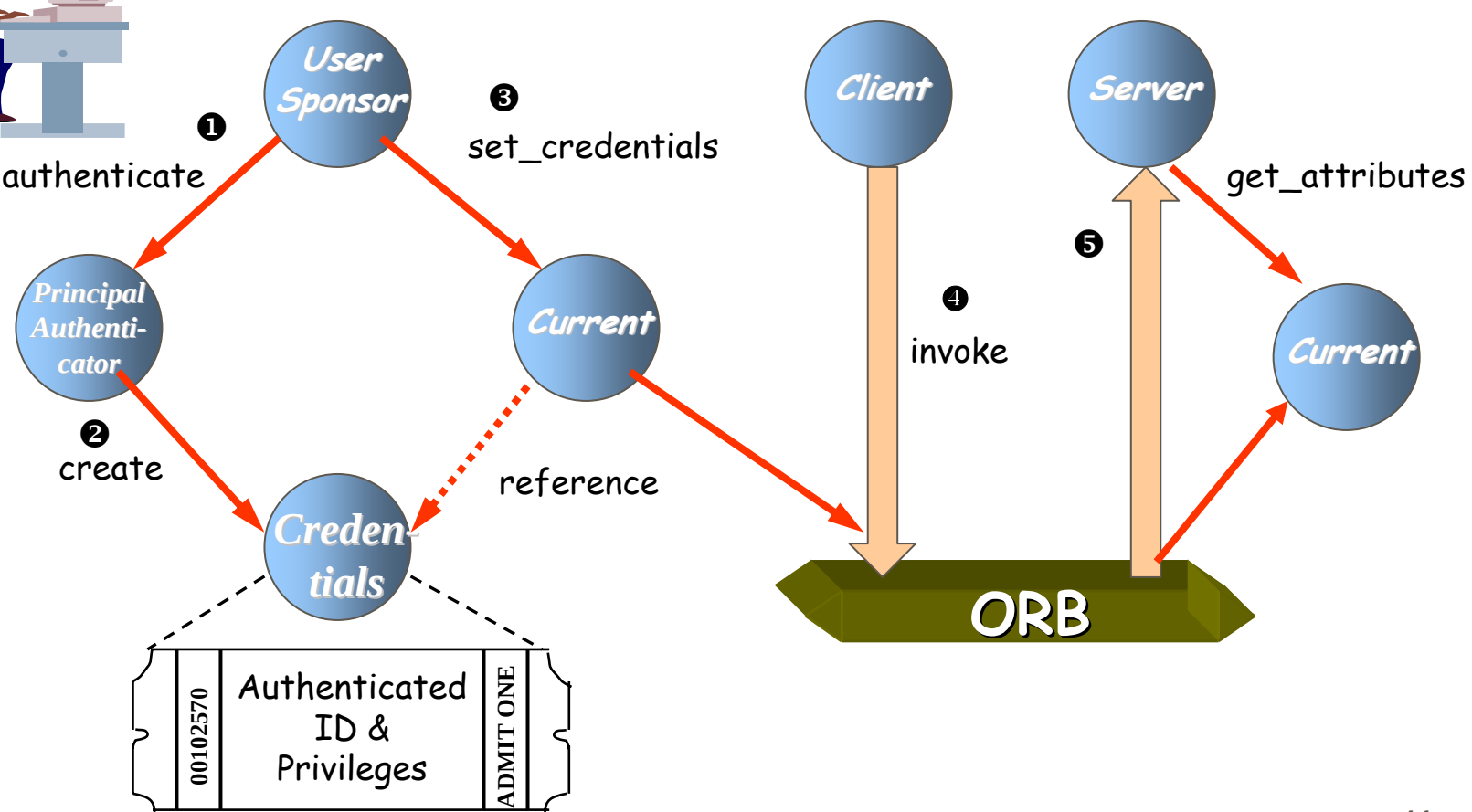
Servicio de Licencia

- Concepto
  - Permite registrar componentes CORBA
    - Se puede fijar un período de uso del componente
    - Las licencias se peden asignar a diferentes usuarios, colecciones de usuarios u organizaciones
    - Los gestores de licencia tienen restricciones de seguridad
      - Para evitar la introducción de Caballos de Troya
- Interfaces
  - **`LicenseServiceManager`**
    - **`obtain_producer_specific_license_service`**
  - **`ProducerSpecificLicenseService`**
    - **`start_use`**
    - **`check_use`**
    - **`end_use`**

Servicio de Seguridad

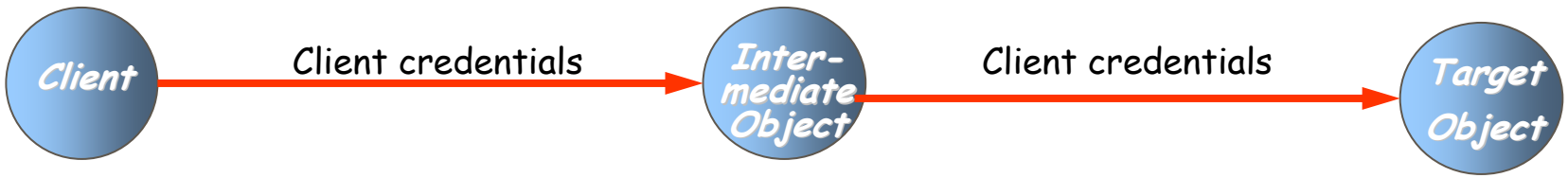
- Concepto
  - ¿Eres tú quien dices ser?
  - Non-Repudation
    - Un ORB puede probar irrefutablemente si una acción tuvo lugar
  - Non-Tampering and Encryption
    - Secure Socket Layer (SSL)
  - Security Domain
    - Conjunto de objetos a los cuales aplicar una política de seguridad
  - Authenticated Id
    - Hace al cliente responsable de sus acciones
    - Permite al servidor determinar qué recursos se van a acceder
    - Identifica únicamente el que envía el mensaje
    - Permite a los proveedores de servicios determinar a quien facturar
    - Sirve privilegios que pueden ser delegados

Servicio de Seguridad
Cómo funciona

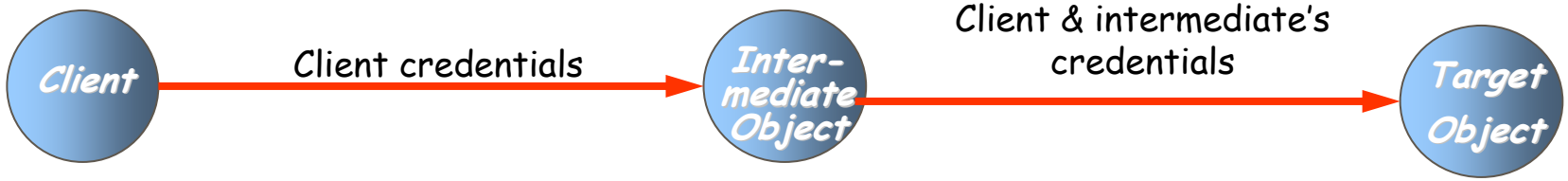


Servicio de Seguridad Delegación de Privilegios

a) No Delegation



b) Simple Delegation



c) Composite Delegation

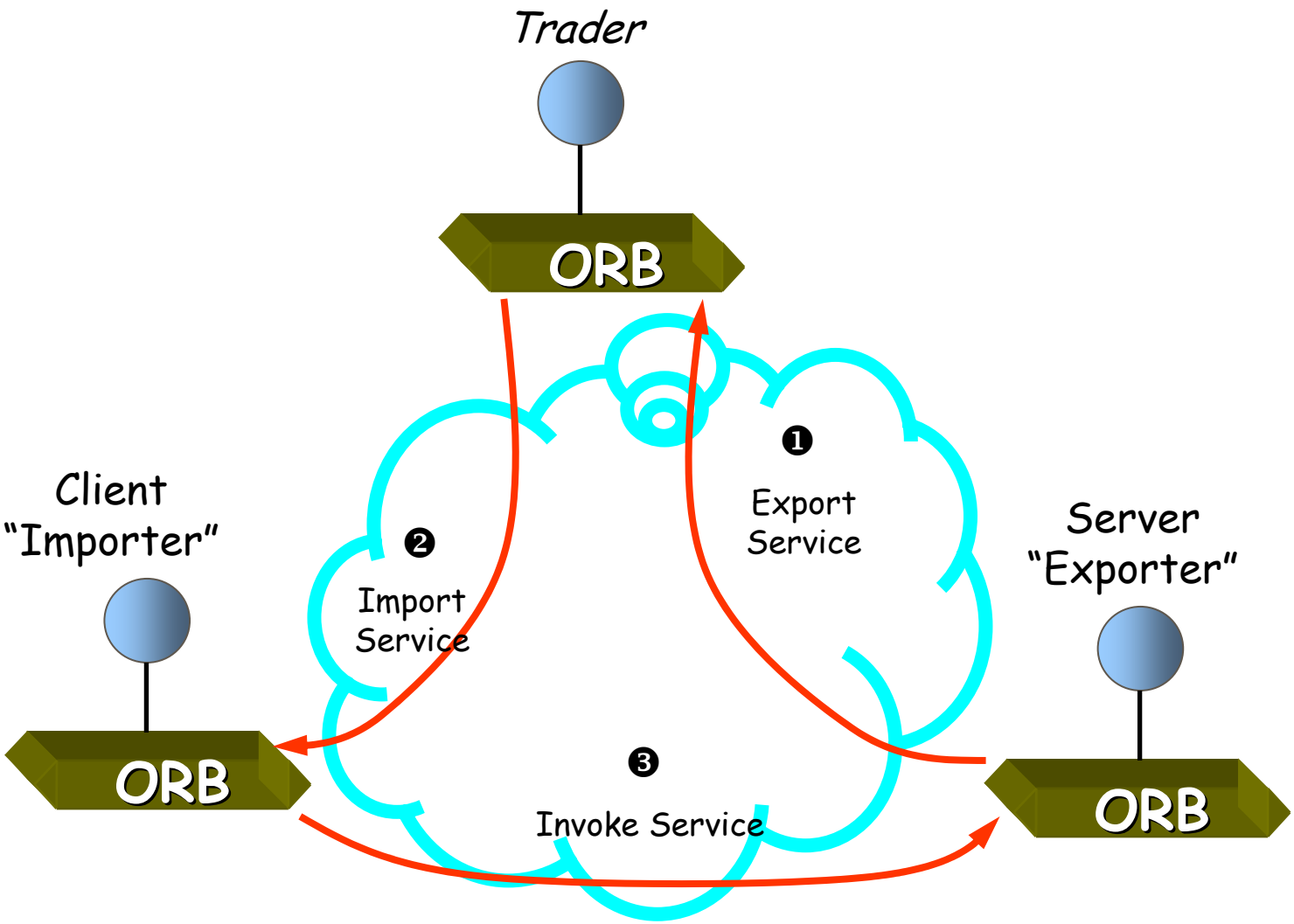
Servicio de Tiempo (ST)

- Concepto
  - Permite ordenar en el tiempo eventos
    - Consigue el tiempo actual
    - Averigua el orden en que ocurren los eventos
    - Genera eventos basados en temporizadores y alarmas
    - Calcula el intervalo entre dos eventos
  - El mecanismo de sincronización
    - El ST tiene un agente en cada máquina (*Time Clerk*)
    - Introduce factores de inexactitud para compensar desplazamientos
    - Introduce objetos (*Time Server*) que responde a consultas sobre el tiempo
  - Unidad de tiempo utilizada
    - UCT Time: *Universal Time Coordinated*
    - Representación universal del tiempo (X/Open)
    - Se define en unidades de 100 nanosegundos

Servicio de Negociación

- Concepto
  - Servicio de páginas amarillas
    - Descubre los objetos basándose en los servicios que provee
  - Mecanismo de exportación - importación
  - Los Negociadores (*Traders*) se pueden federar
    - Pueden anunciar sus servicios
    - Pueden propagar sus peticiones
- Criterios de búsqueda
  - Políticas
    - Cómo emprender la búsqueda (sobre qué *traders*, en qué orden,...)
  - Restricciones
    - Criterio de selección
  - Preferencias
    - Orden en el cual devolver los objetos encontrados

Servicio de Negociación
Cómo funciona



Algunas Referencias

– ***Orfali, R., Harkey D. Y Edwards J. (1997). Instant CORBA.*** *New York: John Wiley &Sons inc. ISBN 0-471-18333-4.*

– ***Object Management Group (1995).*** *CORBAServices: Common Object Services Specification. 95-12-30. Diciembre 1995.*

– ***Orfali, R. y Harkey D. (1997). Client/Server Programing with JAVA and CORBA****. New York: John Wiley &Sons inc. ISBN 0-471-16351-1.*