UD 8. Comunicación

Concurrencia y Sistemas Distribuidos

Objetivos de la Unidad Didáctica

- Comprender los mecanismos de comunicación en un sistema distribuido: comunicación basada en mensajes.
  - Distinguir las características de los mecanismos de comunicación basada en mensajes
  - Conocer el funcionamiento de diferentes mecanismos de comunicación típicos
    - Mecanismo de comunicación ROI (invocación a objeto remoto)
    - Mecanismo de comunicación Java RMI
    - Mecanismo de comunicación Servicios Web RESTful
    - Mecanismo de comunicación Java Message Service (JMS)

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
- Servicios web
- Middlewares orientados a mensajería

Contenido

- Características de los mecanismos de comunicación
  - Utilización
  - Estructura y contenido de los mensajes
  - Direccionamiento
  - Sincronía
  - Persistencia
- Invocación a objeto remoto (ROI)
- Servicios web
- Middlewares orientados a mensajería

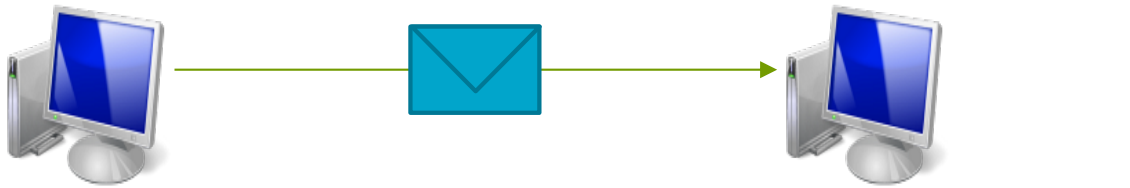
Características de los mecanismos de comunicación

- Mecanismos de comunicación:
  - Permiten la comunicación entre procesos que se ejecutan en ordenadores distintos
  - Ofrecidos por los middlewares de comunicación
  - Características:
    - Utilización
    - Estructura y contenido de los mensajes
    - Direccionamiento
    - Sincronía
    - Persistencia

Características de los mecanismos de comunicación

- **Utilización:**

A. Con primitivas básicas de comunicación
   - Operaciones de **envío** y **recepción**
   - Ejemplos: Sockets, colas de mensajes

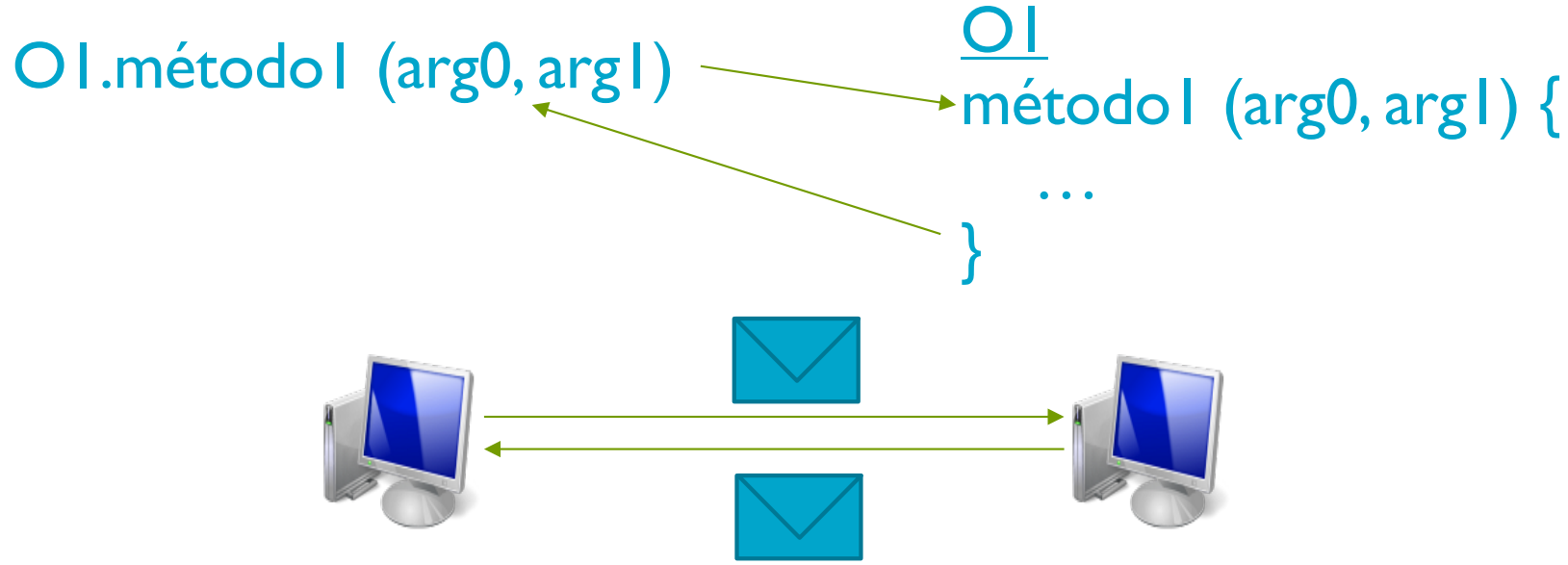


Enviar (mensaje) Recibir(mensaje)

Características de los mecanismos de comunicación

- Utilización:

B. Mediante construcciones del lenguaje de programación
  - Mayor nivel de abstracción
  - El envío y recepción de mensajes es transparente al programador
  - Ejemplos: Remote Procedure Call (RPC), Remote Object Invocation (ROI)



Características de los mecanismos de comunicación

- **Estructura de los mensajes**

A. No estructurados, solo contenido
   - Contenido de formato libre
   - Ejemplo: sockets

B. Estructurados en cabecera + contenido
   - La cabecera es un conjunto de campos, generalmente extensible
   - Contenido de formato libre
   - Ejemplo: colas de mensajes

C. Estructura transparente al programador
   - Determinada por el middleware de comunicaciones
   - Ejemplo: RPC/ROI

Características de los mecanismos de comunicación

- **Contenido de los mensajes**

A. Bytes
   - **Ventajas**: eficiente y compacto
   - **Inconvenientes**: difícil de procesar, la representación en binario no es igual entre arquitecturas y lenguajes de programación

B. Texto
   - **Ventajas**: independiente de la arquitectura y lenguaje de programación
   - **Inconvenientes**: menos eficiente
   - Normalmente se emplea algún lenguaje con amplia disponibilidad de bibliotecas para su procesamiento. Ejemplos:
     - Extensible Markup Language (XML)
     - JavaScript Object Notation (JSON)

Características de los mecanismos de comunicación

- **Direccionamiento**

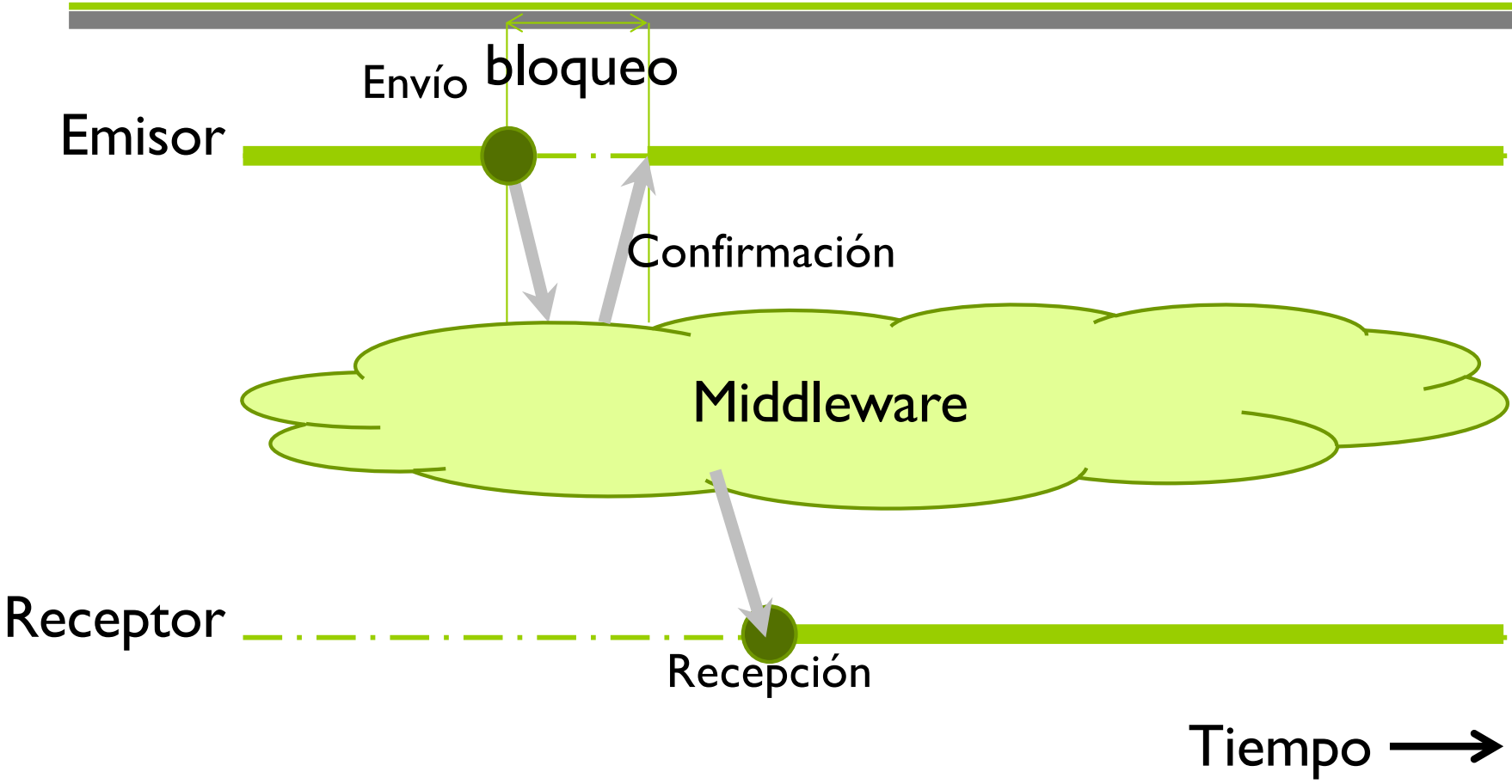
A. Direccionamiento directo:
   - El ordenador emisor envía los mensajes **directamente** al ordenador receptor
   - Ejemplos: sockets, servicios web, RPC/ROI

B. Direccionamiento indirecto:
   - El ordenador emisor envía los mensajes a un **intermediario** (bróker), que se encarga de hacerlos llegar al receptor
   - Ejemplos: colas de mensajes

Características de los mecanismos de comunicación

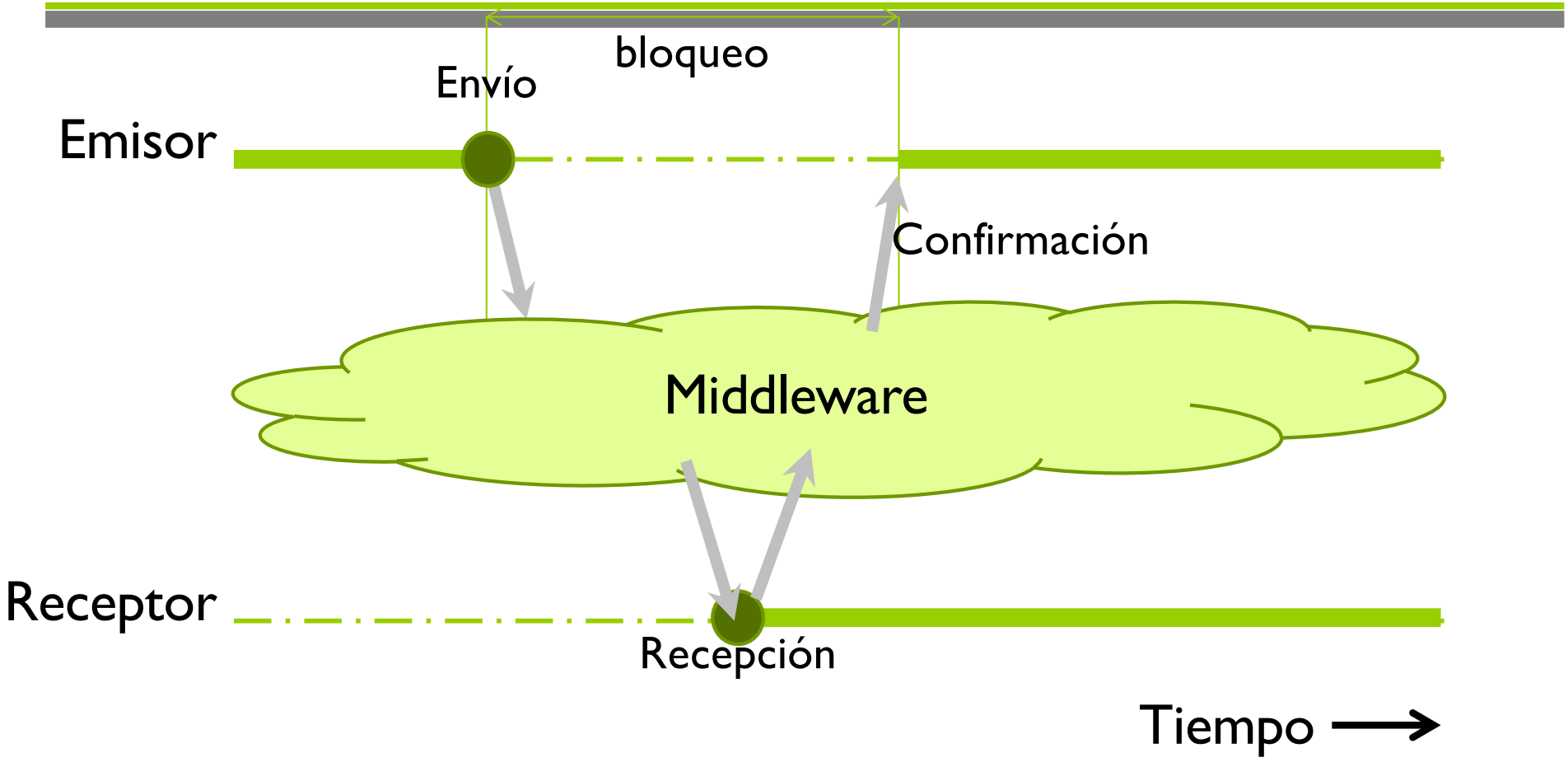
- **Sincronización**
  - A. Comunicación asincrónica: El middleware responde al emisor con la confirmación, tras almacenarlo en sus buffers
  - B. Comunicación sincrónica (entrega): El middleware responde cuando el receptor ha confirmado la entrega correcta del mensaje
  - C. Comunicación sincrónica (respuesta): El middleware responde al emisor tras recibir aviso del receptor de haber procesado el mensaje

Comunicación asincrónica



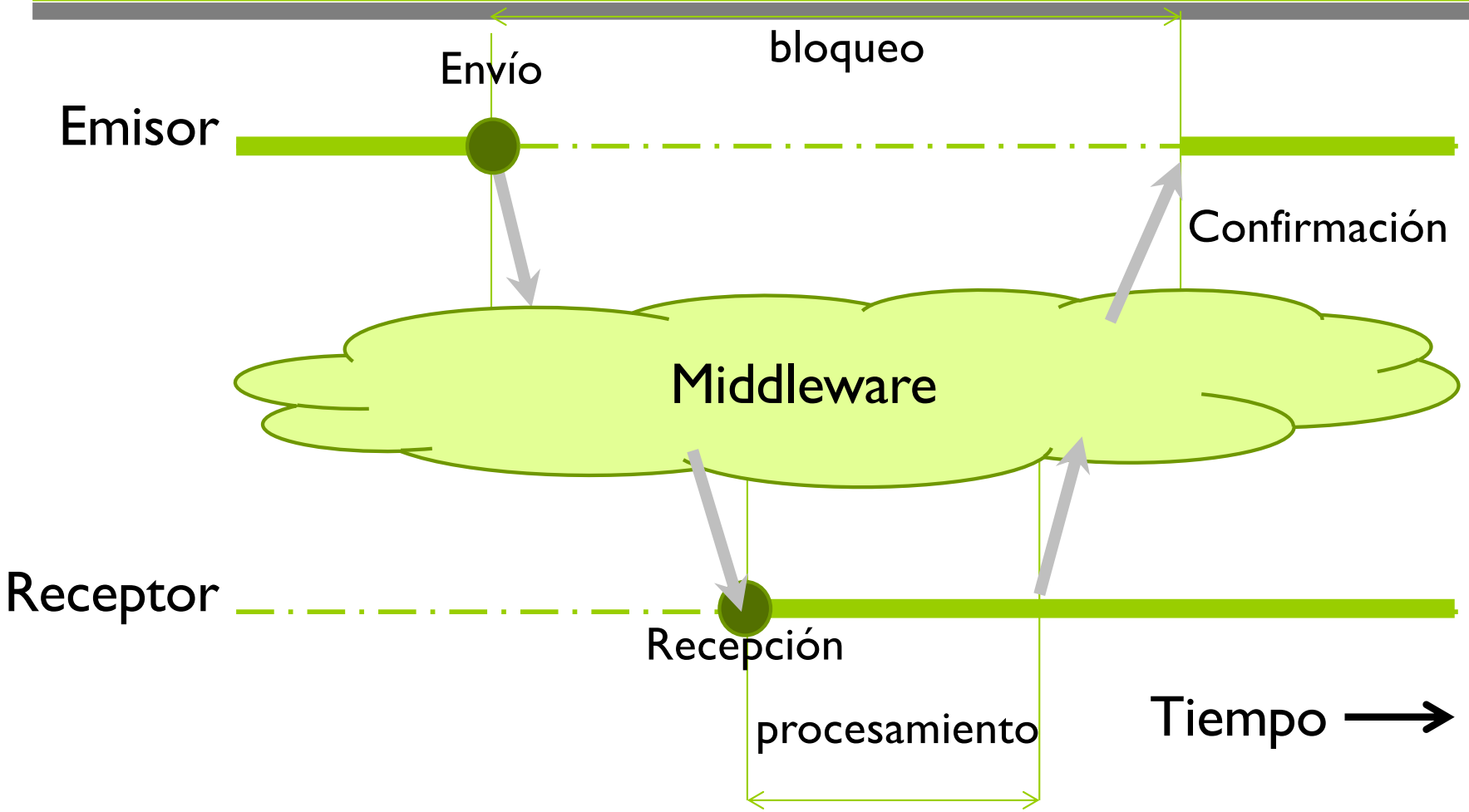
- El middleware responde al emisor con la confirmación, tras almacenarlo en sus buffers
- Ejemplos: sockets UDP, colas de mensajes

Comunicación sincrónica (entrega)



- El middleware responde cuando el receptor ha confirmado la entrega correcta del mensaje
- Ejemplos: sockets TCP, servicios web REST

Comunicación sincrónica (respuesta)

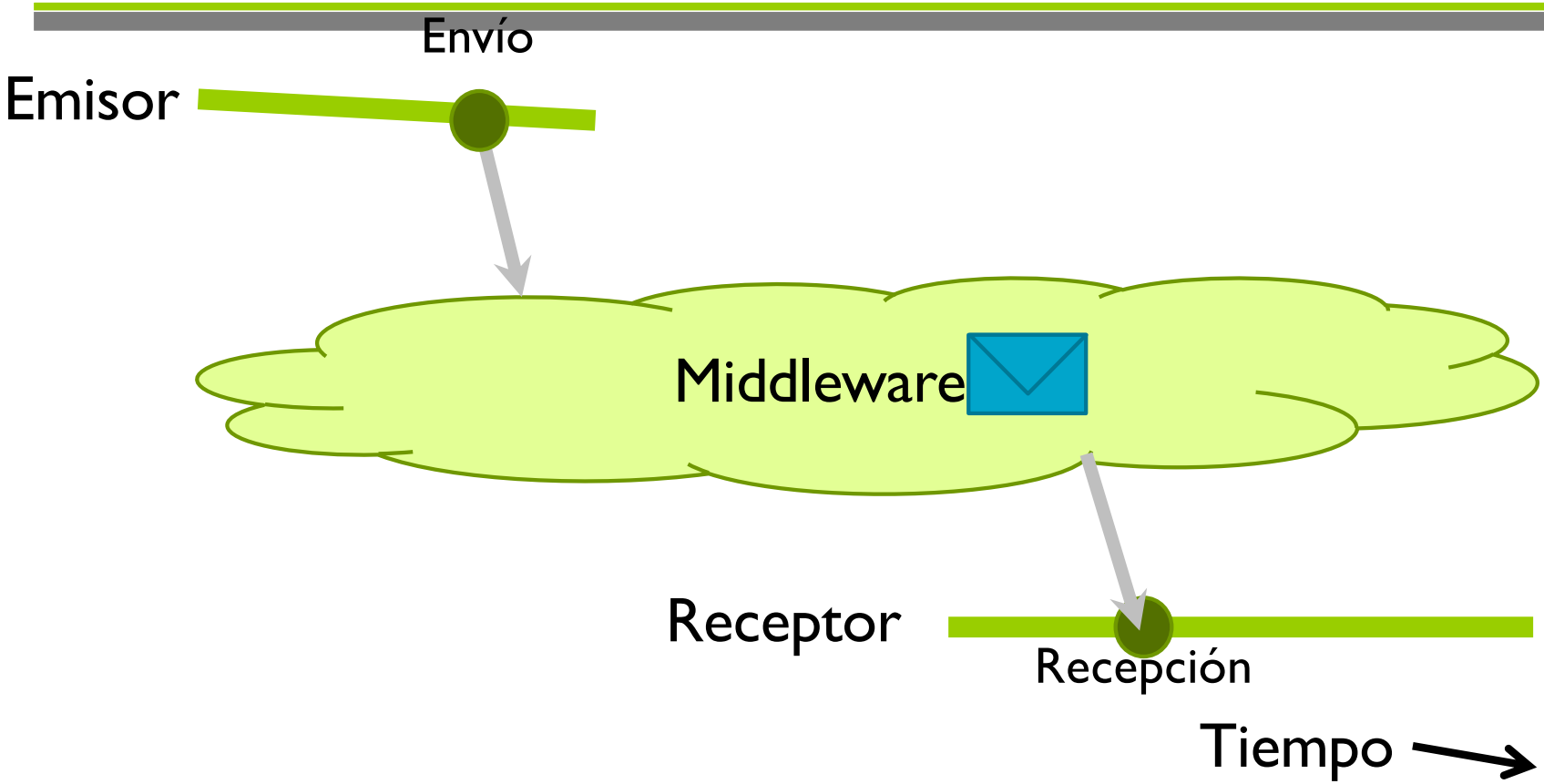


- El middleware responde al emisor tras recibir aviso del receptor de haber procesado el mensaje
- Ejemplos: RPC,ROI, servicios web SOAP

Características de los mecanismos de comunicación

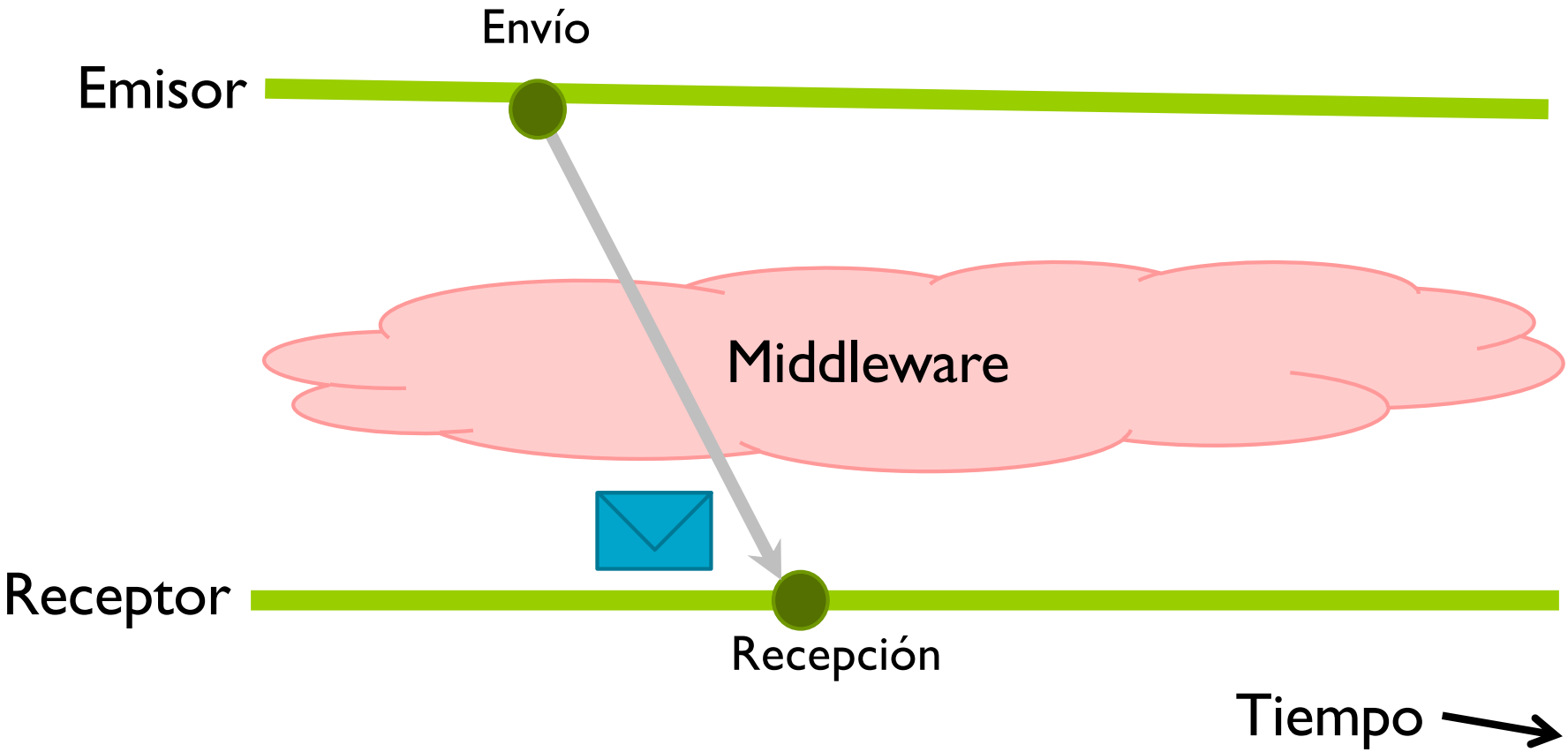
- **Persistencia**
  - A. Comunicación persistente: El middleware puede guardar los mensajes pendientes de entrega
  - B. Comunicación no persistente: El middleware no es capaz de mantener los mensajes que deben transmitirse

Comunicación persistente



- El middleware puede guardar los mensajes pendientes de entrega
- El receptor no tiene por qué estar en ejecución cuando se envía el mensaje
- El emisor puede finalizar su ejecución antes de que el mensaje se entregue
- Ejemplos: colas de mensajes

Comunicación no persistente



- El middleware no es capaz de mantener los mensajes que deben transmitirse
- Se requiere que el emisor y el receptor estén activos para que los mensajes lleguen a transmitirse.
- Ejemplos: sockets, servicios web, RPC/ROI

Características de los mecanismos de comunicación

- Las características mostradas son generalmente útiles para categorizar y comprender los diferentes mecanismos de comunicación existentes.
  - No obstante, la realidad es compleja y diversa, y en algunos casos hay mecanismos de comunicación en los que una caracterización precisa no es posible.

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
  - Conceptos generales
    - Elementos de una ROI
    - Pasos de una ROI
    - Paso de objetos como argumentos
    - Creación de objetos
    - RPC
  - Java RMI
- Servicios web
- Middlewares orientados a mensajería

Concepto de objeto remoto

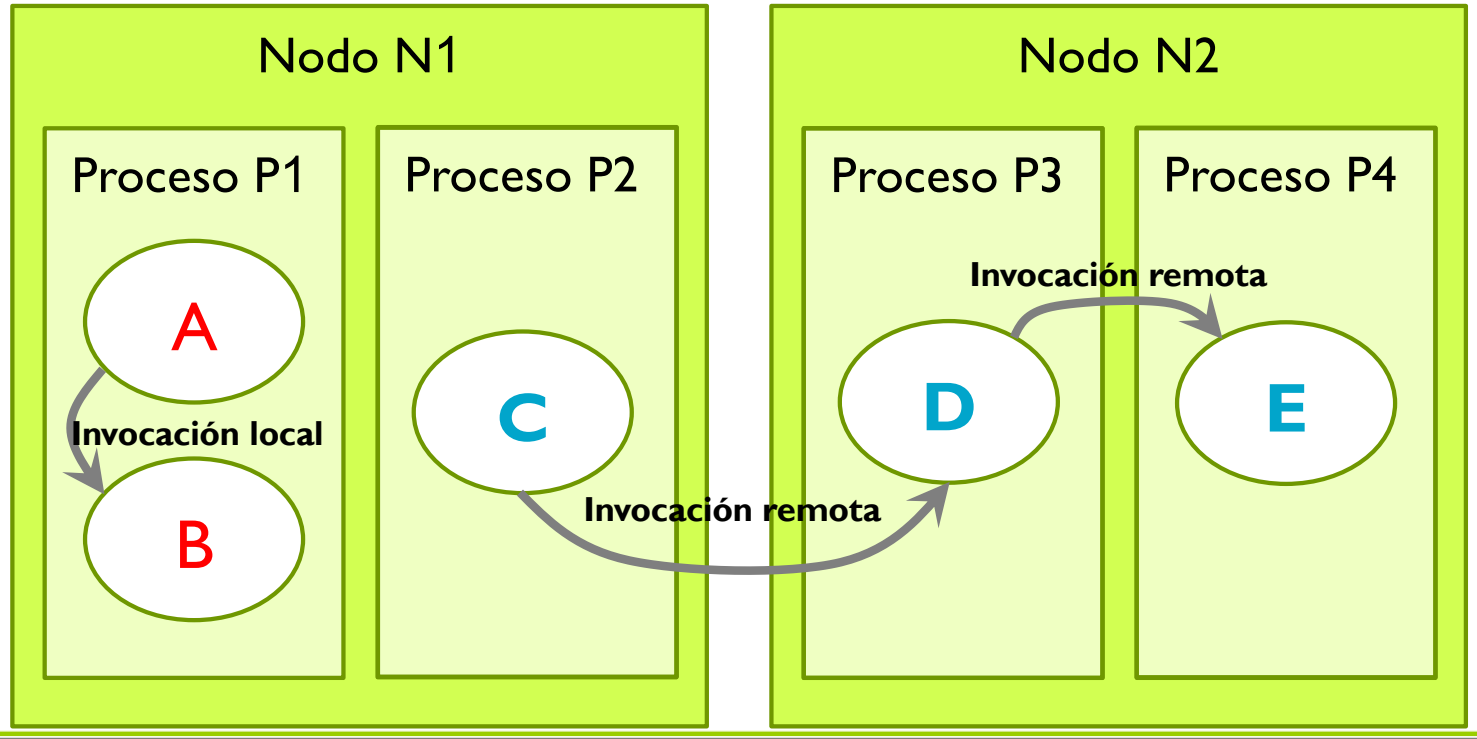
- **Objeto remoto**: un objeto que puede ser invocado desde otros espacios de direcciones.
  - Un objeto remoto se instancia en un servidor y puede responder a la invocación de clientes locales y remotos
- Las aplicaciones se organizan en colecciones dinámicas de objetos que pueden estar en diferentes nodos
  - Las aplicaciones y los objetos pueden invocar métodos de otros objetos de forma remota
  - Asumiremos que todo objeto está completamente contenido en un único nodo
    - Aunque existen modelos de objetos distribuidos que permiten dividir objetos en partes que son albergadas en diferentes nodos (p.ej. objetos fragmentados)

Ventajas del modelo de objetos distribuidos

- Aprovechar la expresividad, capacidad de abstracción y flexibilidad del **paradigma OO**
- Transparencia de **ubicación**
  - La sintaxis de invocación de los métodos de un objeto no depende del espacio de direcciones en el que reside dicho objeto
  - Podemos ubicar los objetos de acuerdo a distintos criterios: localidad de acceso, restricciones administrativas, seguridad, ...
- Podemos **reutilizar** aplicaciones "heredadas" (legacy) encapsulándolas en objetos (utilizando el patrón de diseño "Wrapper")
- **Escalabilidad**: los objetos pueden distribuirse sobre una red, teniendo en cuenta la demanda actual

Tipos de invocaciones

- **Invocación local**: los objetos invocador e invocado residen en el mismo proceso (p.ej. los **objetos A y B**)
- **Invocación remota (*ROI – Remote Object Invocation*)**: los objetos invocador e invocado residen en procesos diferentes, dentro del mismo nodo (p.ej. **D y E**) o en nodos distintos (p.ej. **C y D**)

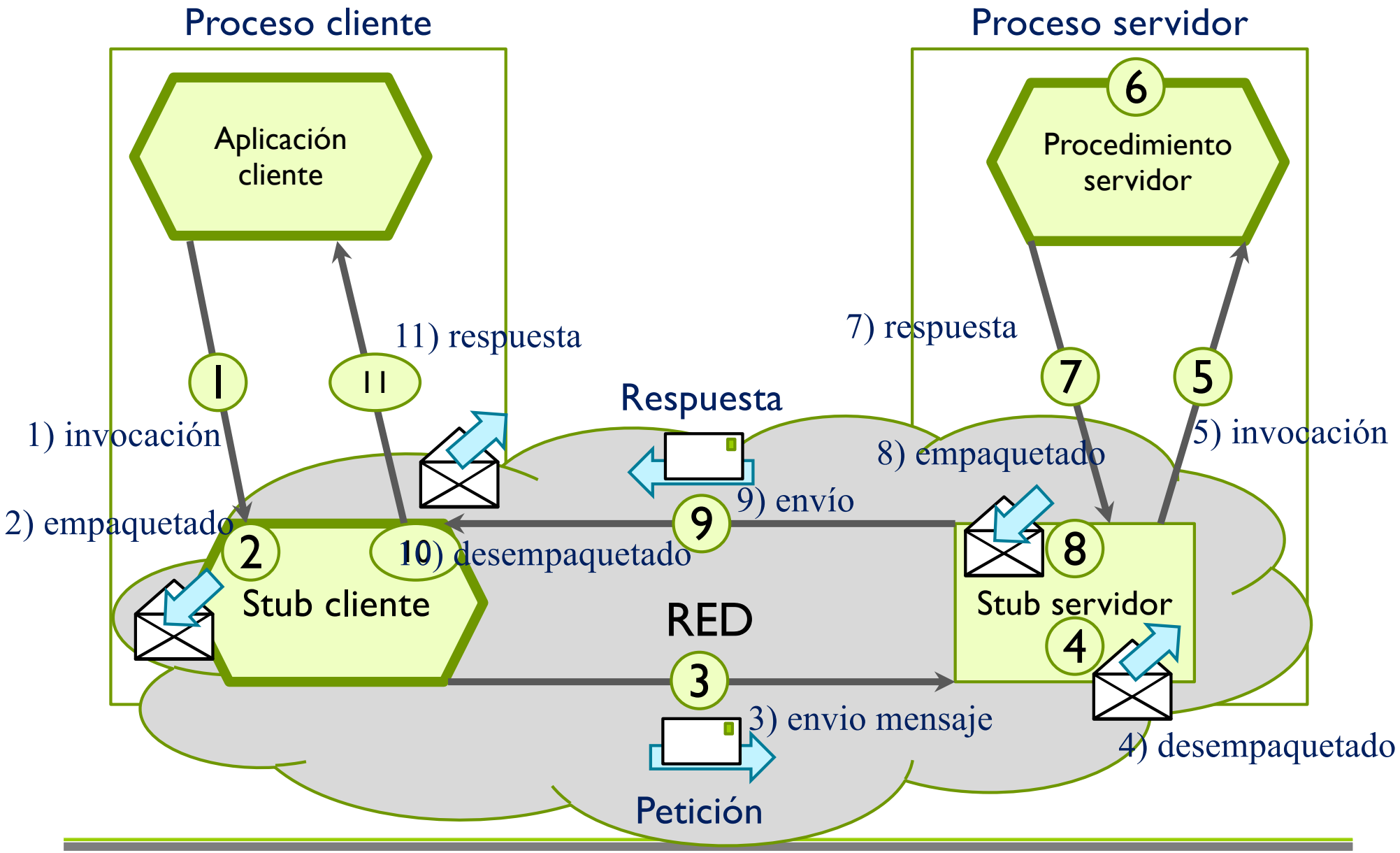


Antecedentes: Remote Procedure Call (RPC)

- RPC es el antecedente de ROI
- Comparte los mismos objetivos
- No contempla el concepto de objeto
- El ordenador remoto ofrece un catálogo de procedimientos
- Estos procedimientos pueden ser llamados de forma transparente (como si fueran locales) desde ordenadores cliente
  - El mecanismo es implementado por los denominados *stubs*: ***stub cliente*** y ***stub servidor***

Remote Procedure Call (RPC)

- Precursor del ROI
- Transparencia de ubicación
- Enmascara invocación a función remota como invoc. local



Remote Procedure Call (RPC)

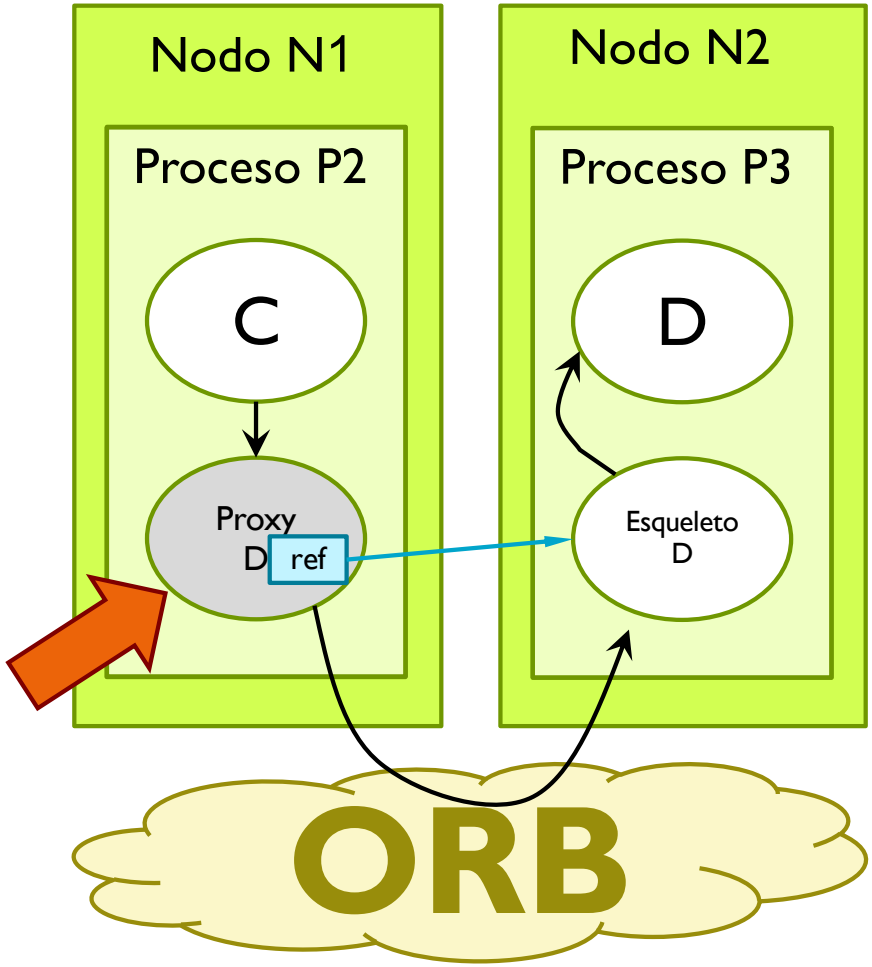
- Pasos:
  1. Invocación del procedimiento local
  2. Empaquetado de argumentos de entrada (marshalling) en un mensaje
  3. Envío del mensaje al servidor y espera de respuesta
  4. Desempaquetado del mensaje y extracción de argumentos de entrada
  5. Llamada al procedimiento
  6. Ejecución del procedimiento
  7. Retorno de control al *stub* servidor
  8. Empaquetado de argumentos de salida y resultado en un mensaje
  9. Envío del mensaje de respuesta
  10. Desempaquetado del mensaje y extracción de argumentos de salida y resultado
  11. Retorno de control al código que invocó el procedimiento local

Invocación a Objeto Remoto (ROI)

- Enmascara invocación a objeto remoto como invocación local
- Similar a RPC (proporciona transparencia de ubicación) pero aplicado al modelo de objetos (paradigma OO)
  - **Aplicación = colección de objetos repartidos en distintos nodos**
  - Podemos invocar métodos de otros objetos de forma remota

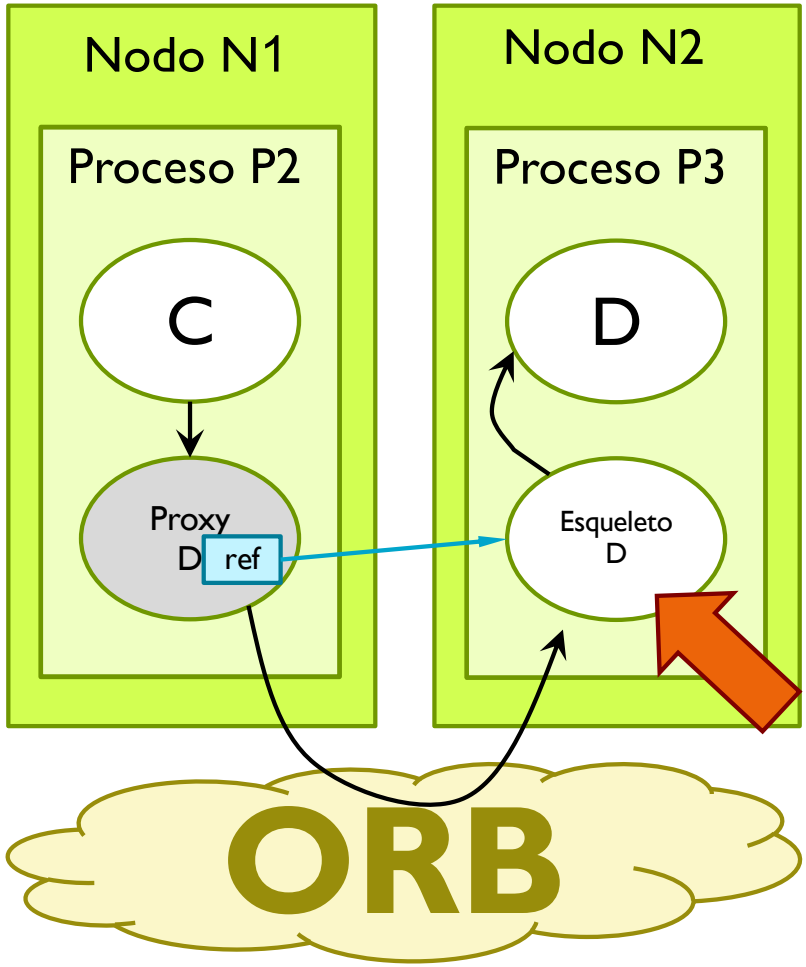
Elementos que intervienen en una ROI

- **Proxy**
  - Ofrece la **misma interfaz** que el objeto remoto
  - Contiene una **referencia** al objeto remoto
    - Proporciona acceso al objeto remoto y a su interfaz
  - Se crea en tiempo de ejecución cuando se accede por primera vez al objeto remoto



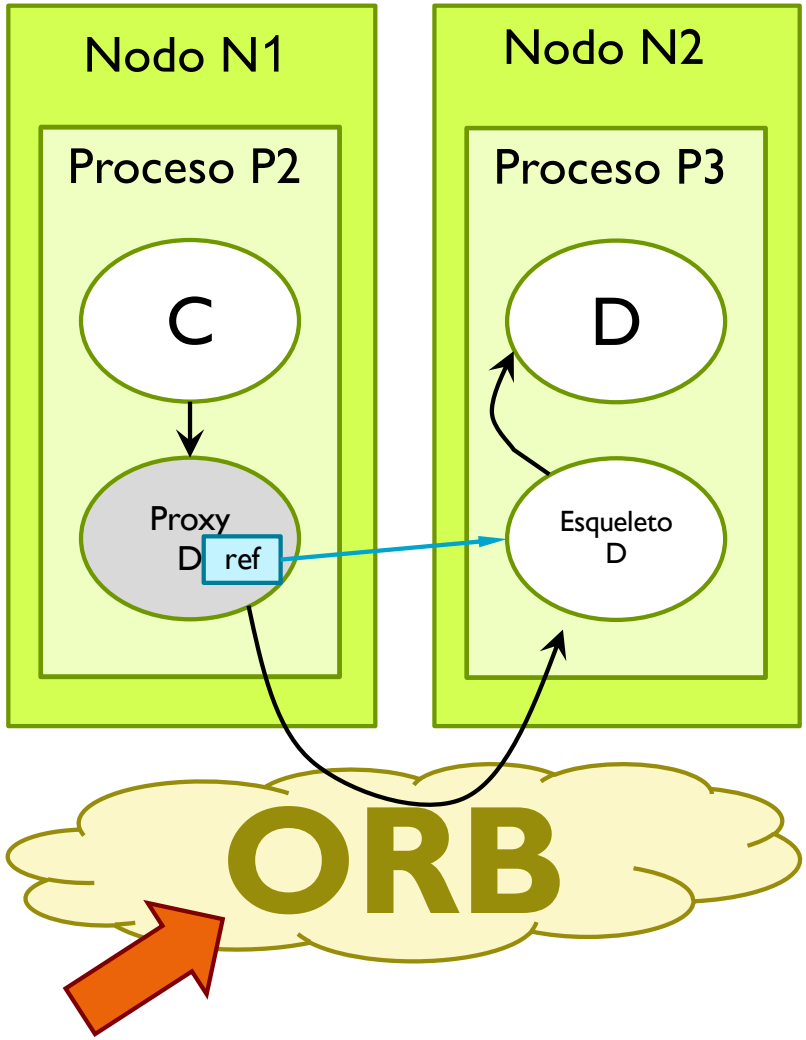
Elementos que intervienen en una ROI

- **Esqueleto**
  - **Recibe las peticiones** de los clientes
  - Realiza las **verdaderas llamadas** a los métodos del objeto remoto
  - Se crea en tiempo de ejecución cuando se crea el objeto remoto



Elementos que intervienen en una ROI

- **Object Request Broker (ORB)**
  - Componente principal de un middleware orientado a objetos
  - Se encarga de:
    - **Identificar y localizar** los objetos
    - **Realizar las invocaciones** remotas de los proxies a los esqueletos
    - **Gestionar el ciclo de vida** de los objetos (creación, registro, activación y eliminación)

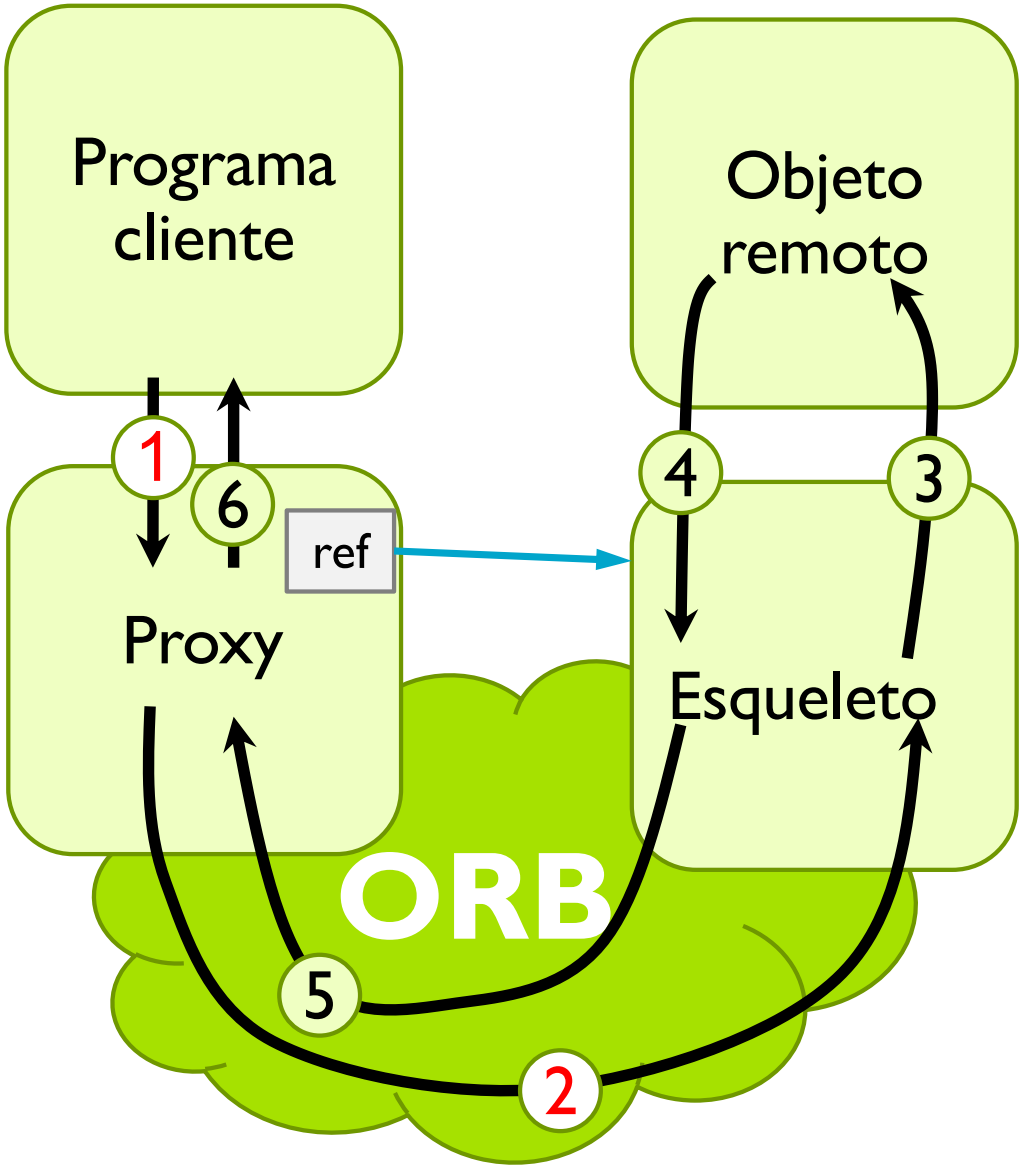


Pasos de una ROI

Pasos 1 y 2

1. El proceso cliente invoca el método del proxy local relacionado con el objeto remoto

2. El proxy empaqueta los argumentos y, utilizando la referencia al objeto, llama al ORB. El ORB gestiona la invocación, haciendo que el mensaje llegue al esqueleto

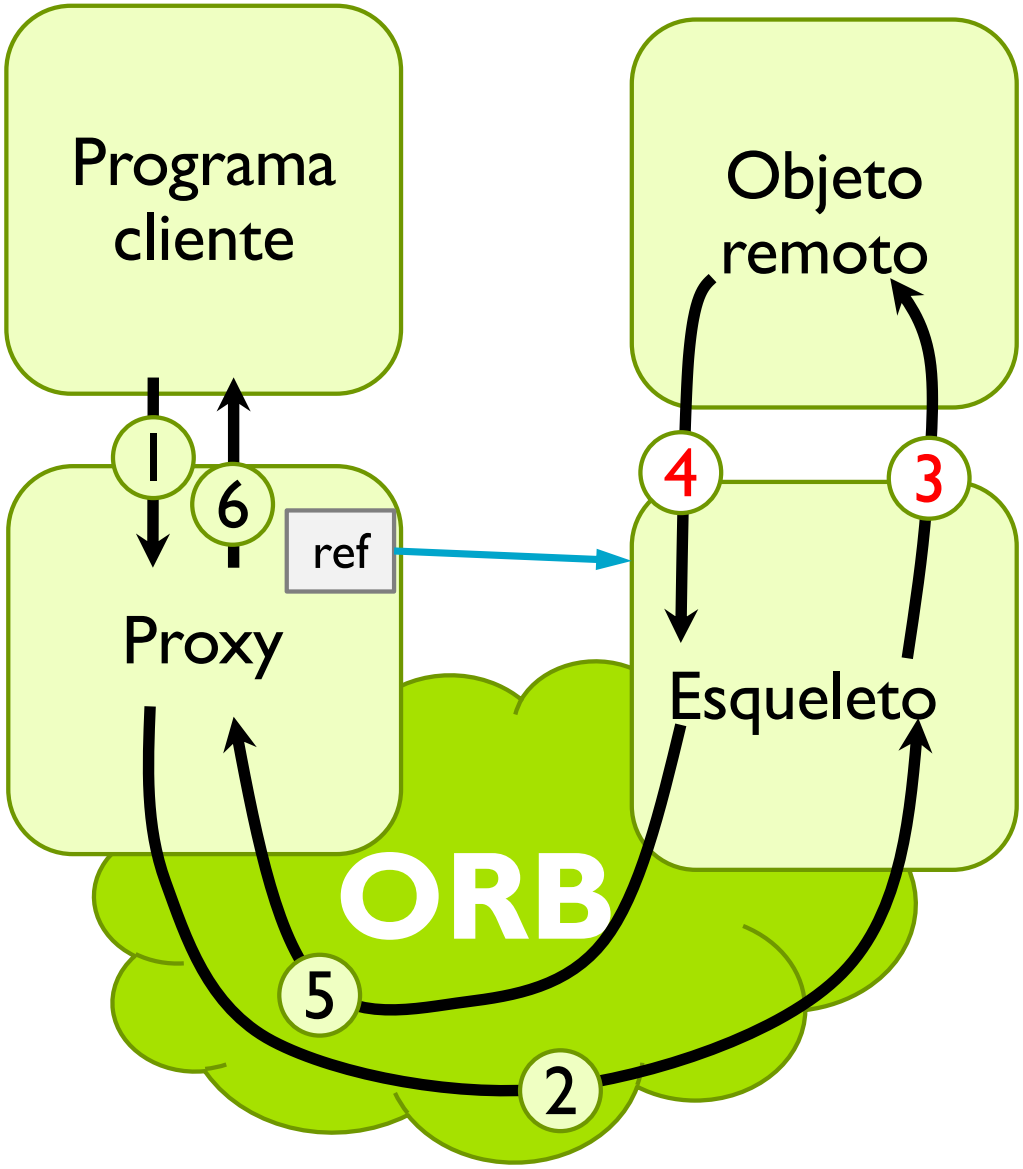


Pasos de una ROI

Pasos 3 y 4

3. El esqueleto desempaqueta los argumentos e invoca el método solicitado, quedando a la espera de que finalice

4. El método llamado finaliza y se desbloquea el esqueleto.

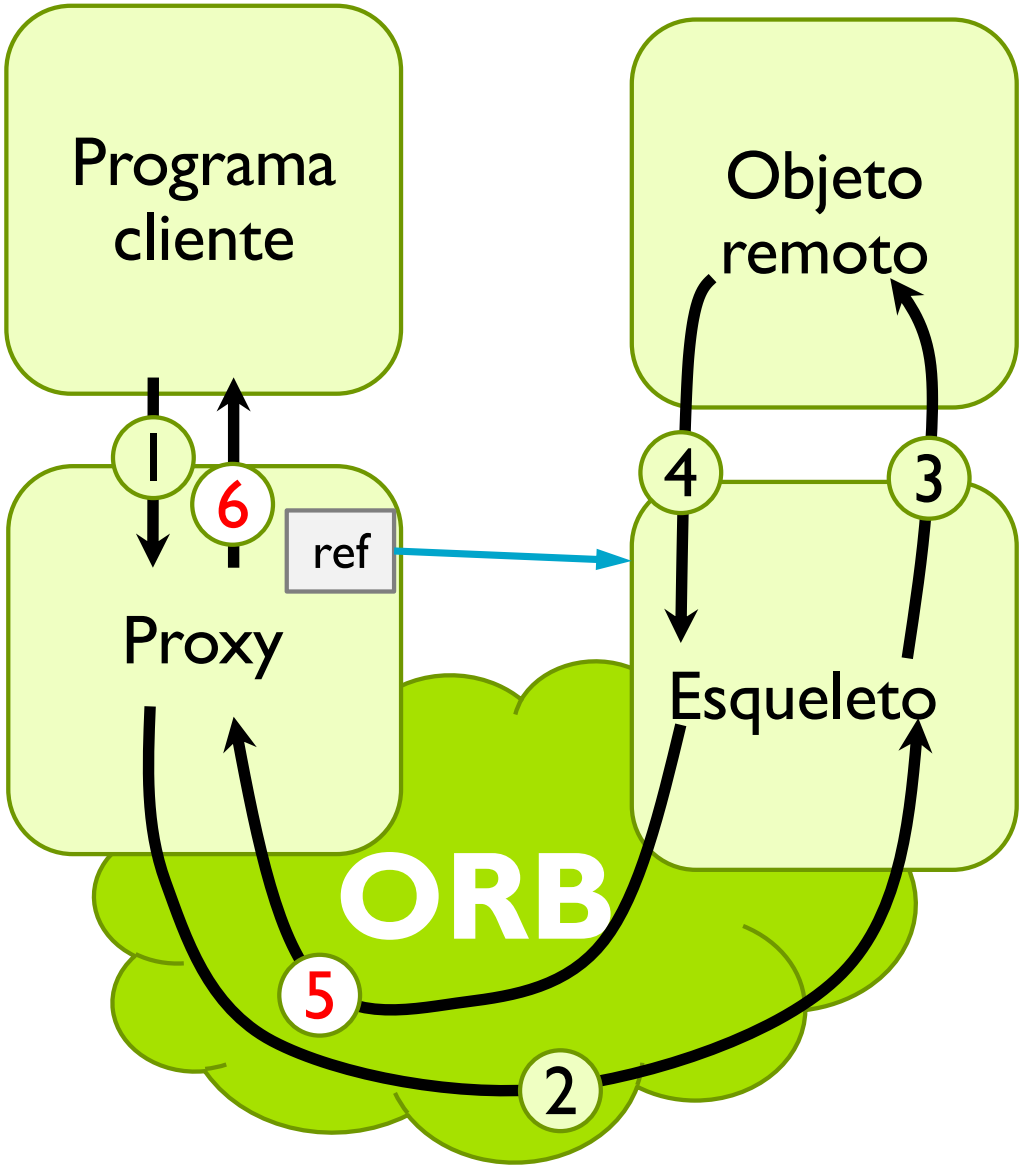


Pasos de una ROI

Pasos 5 y 6

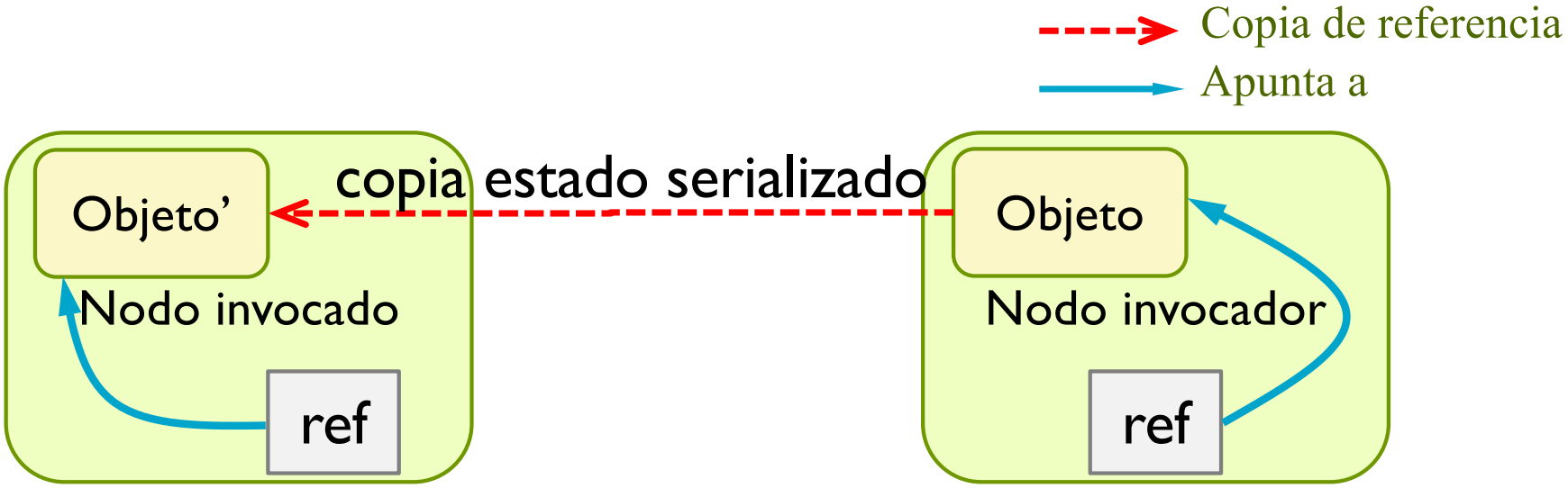
5. El esqueleto empaqueta los resultados y llama al ORB, el cual hace llegar el mensaje al proxy

6. El proxy desempaqueta los resultados y los devuelve al proceso cliente



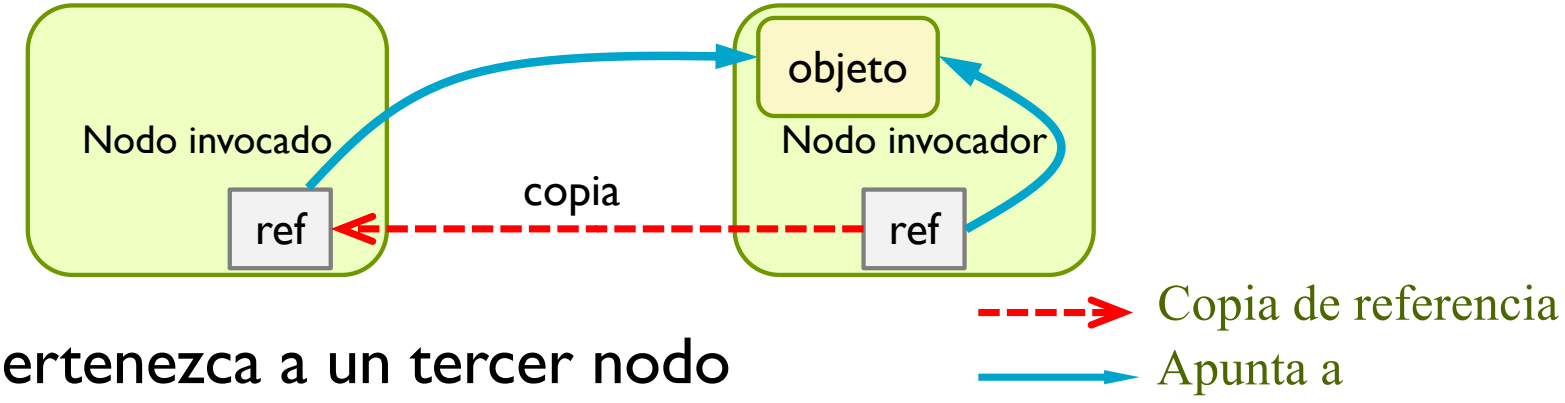
Paso de objetos como argumentos

- **Paso por valor:**
  - El estado del objeto origen se empaqueta, mediante un proceso denominado ***serialización***
  - El objeto serializado se transmite al nodo destino, donde a partir de dicha información se crea un nuevo objeto copia del original
  - Ambos objetos evolucionan por separado

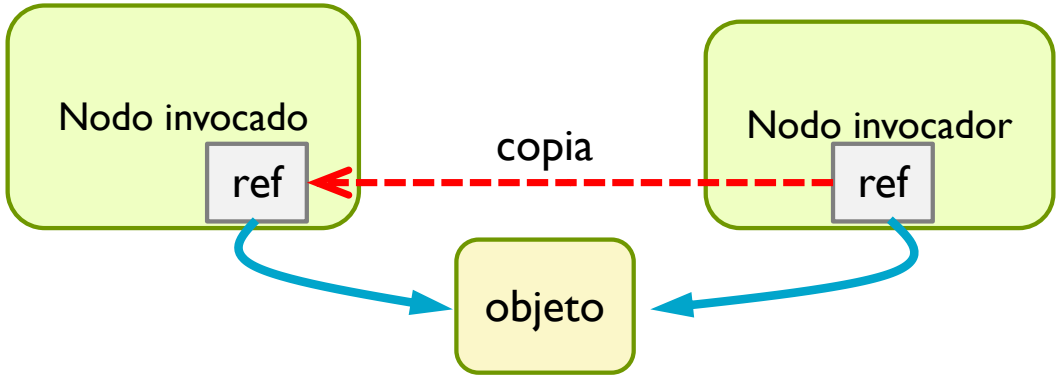


Paso de objetos como argumentos

- **Paso por referencia:**
  - Para pasar un objeto por referencia basta con copiar la referencia del nodo invocador al nodo invocado
  - No importa que el objeto pertenezca al nodo invocador



  - O que pertenezca a un tercer nodo



Creación de objetos

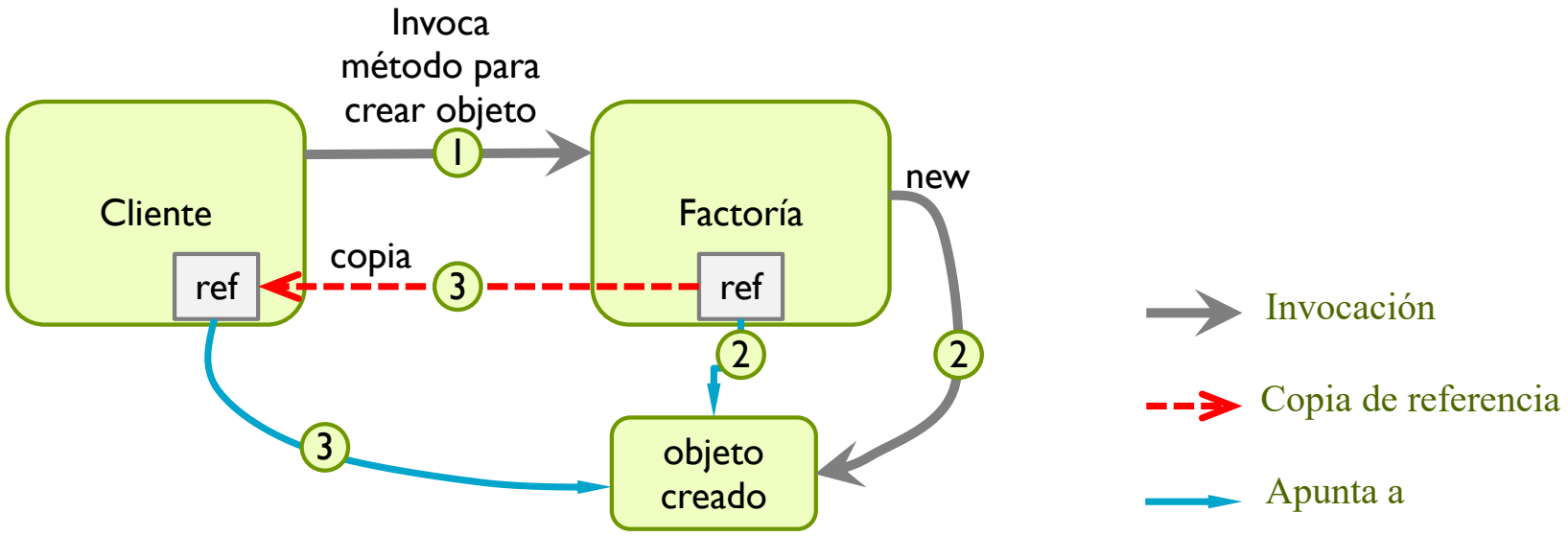
- En ROI, la creación de objetos (y su registro en el ORB) puede realizarse mediante dos procedimientos distintos:
  - Por iniciativa del cliente
    - El cliente solicita a una factoría crear el objeto.
    - Una **factoría** es un objeto servidor que crea objetos de un determinado tipo
  - Por iniciativa del servidor
    - Un proceso servidor crea un objeto y lo registra en el ORB, obteniendo una referencia al objeto
    - El proceso servidor registra la referencia en el **servidor de nombres**, para que otros clientes puedan buscarlo.

Creación de objetos: a iniciativa del cliente

1. El cliente solicita a una **factoría** (un servidor que crea objetos de un determinado tipo) crear el objeto

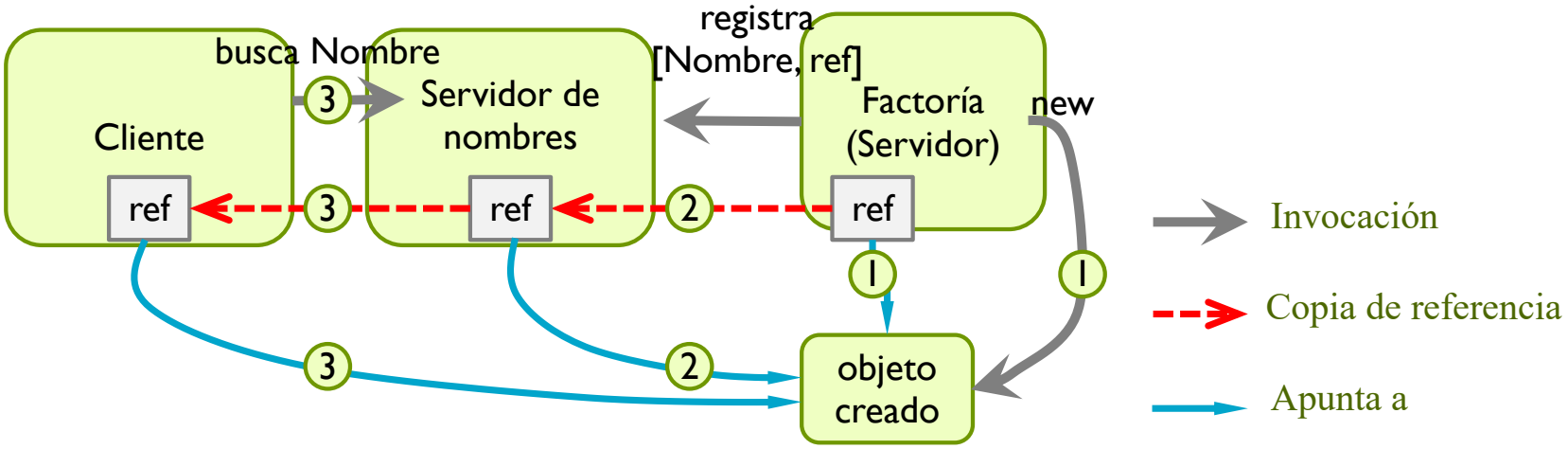
   Asumimos que el cliente dispone de un proxy para invocar a la factoría y habrá un método que devolverá como resultado una referencia al objeto que se creará

2. La factoría crea el objeto solicitado y lo registra en el ORB, obteniendo una referencia al objeto y su esqueleto
3. La factoría devuelve al cliente una copia de su referencia



Creación de objetos: a iniciativa del servidor

1. Un proceso, que se convertirá en servidor, crea un objeto y lo registra en el ORB, creándose así la primera referencia al objeto.
2. El proceso servidor usa la ***referencia*** para registrarla en un **servidor de nombres**, proporcionando una cadena de texto como ***nombre lógico*** del objeto.
3. Cualquier otro proceso que conozca el nombre lógico utilizado para registrar el objeto puede contactar con el **servidor de nombres** y obtener una referencia a dicho objeto



Contenido

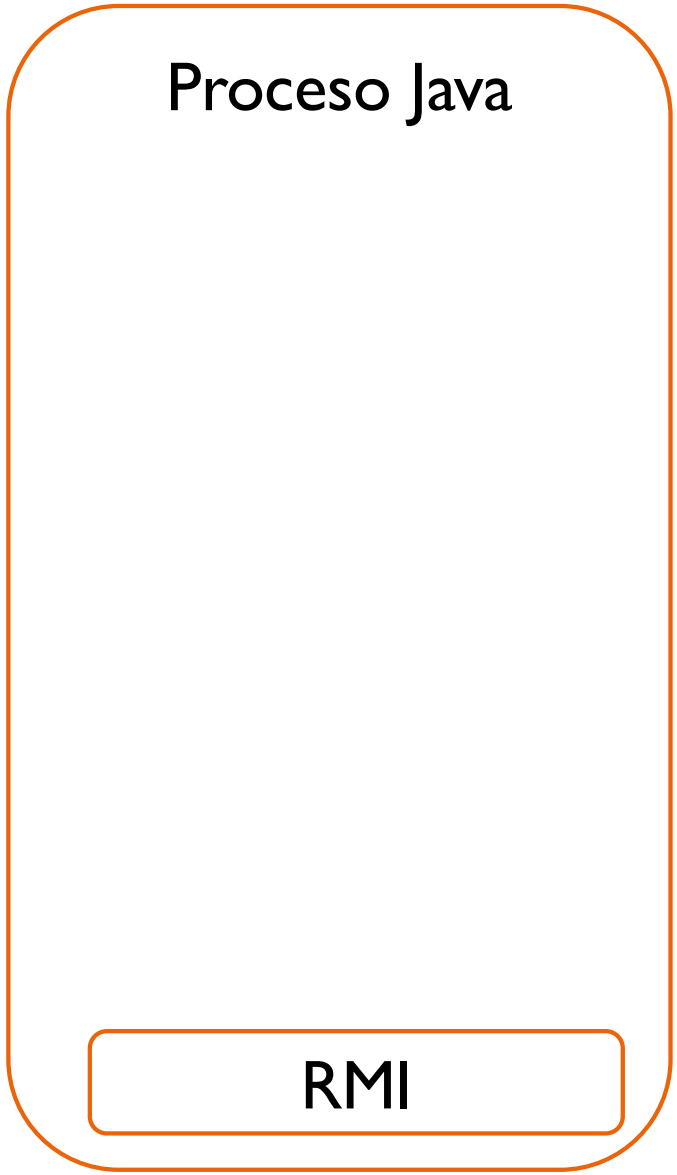
- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
  - Conceptos generales
  - Java RMI
    - ¿Qué es Java RMI?
    - El servidor de nombres de Java RMI
    - Desarrollo de una aplicación Java RMI
    - Paso de objetos como argumentos
- Servicios web
- Middlewares orientados a mensajería

¿Qué es Java RMI?

- **Java RMI** (*Remote Method Invocation*) es un **middleware** de comunicación orientado a **objetos** que proporciona una solución para un lenguaje OO específico (Java) que da soporte a la portabilidad
  - No es multi-lenguaje, pero es **multi-plataforma**
  - Permite **invocar** métodos de objetos Java de otra JVM, y **pasar objetos** Java como argumentos cuando se invocan dichos métodos

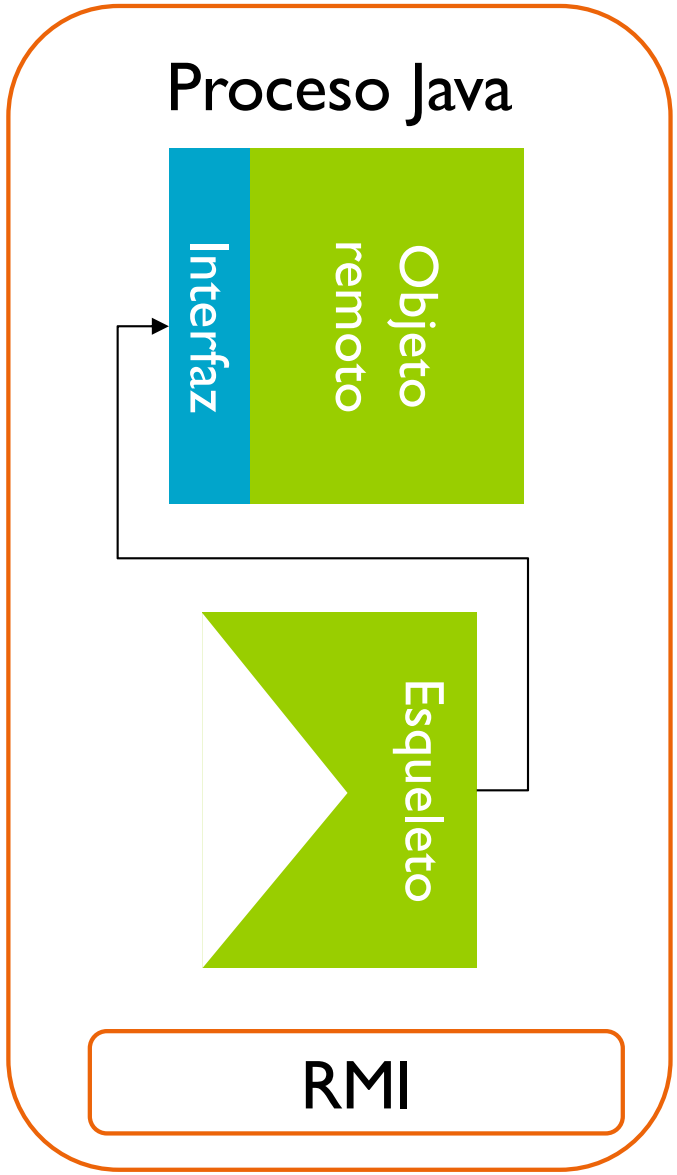
¿Qué es Java RMI?

- El componente RMI se incorpora de forma automática a un proceso Java cuando se utiliza su API
  - Escucha peticiones que llegan en un puerto TCP

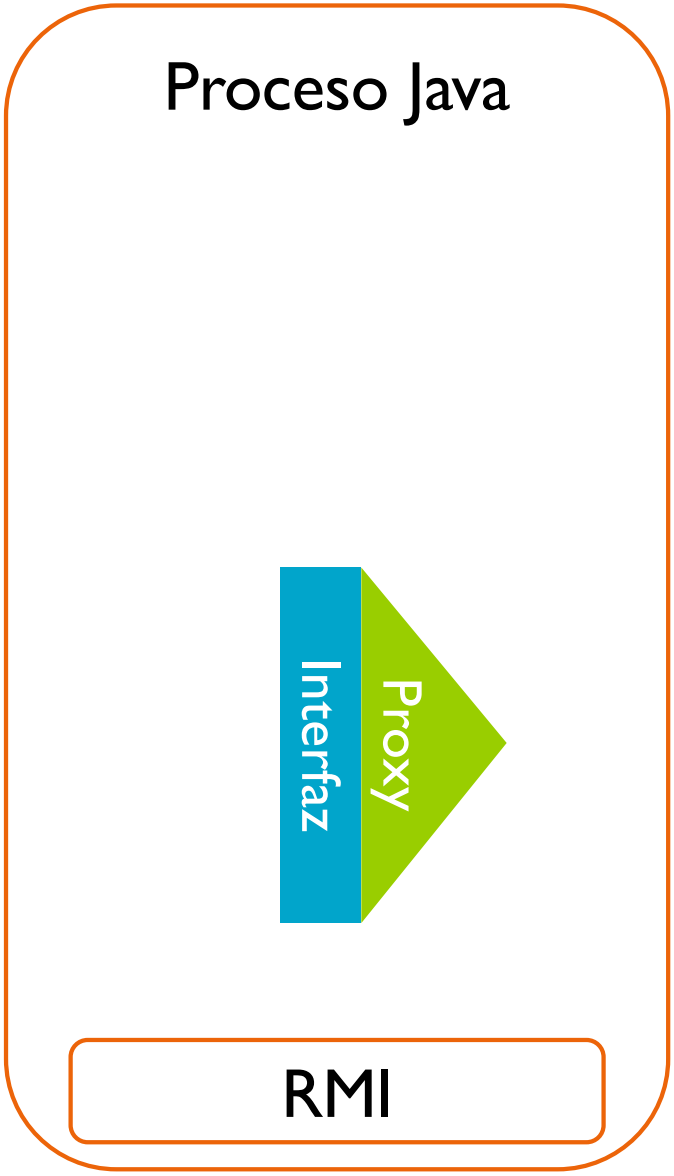


¿Qué es Java RMI?

- Un objeto es invocable de forma remota
  - si implementa una interfaz que extiende la interfaz Remote
- Para cada uno de estos objetos remotos, RMI crea dinámicamente un objeto (no accesible por el usuario) llamado esqueleto



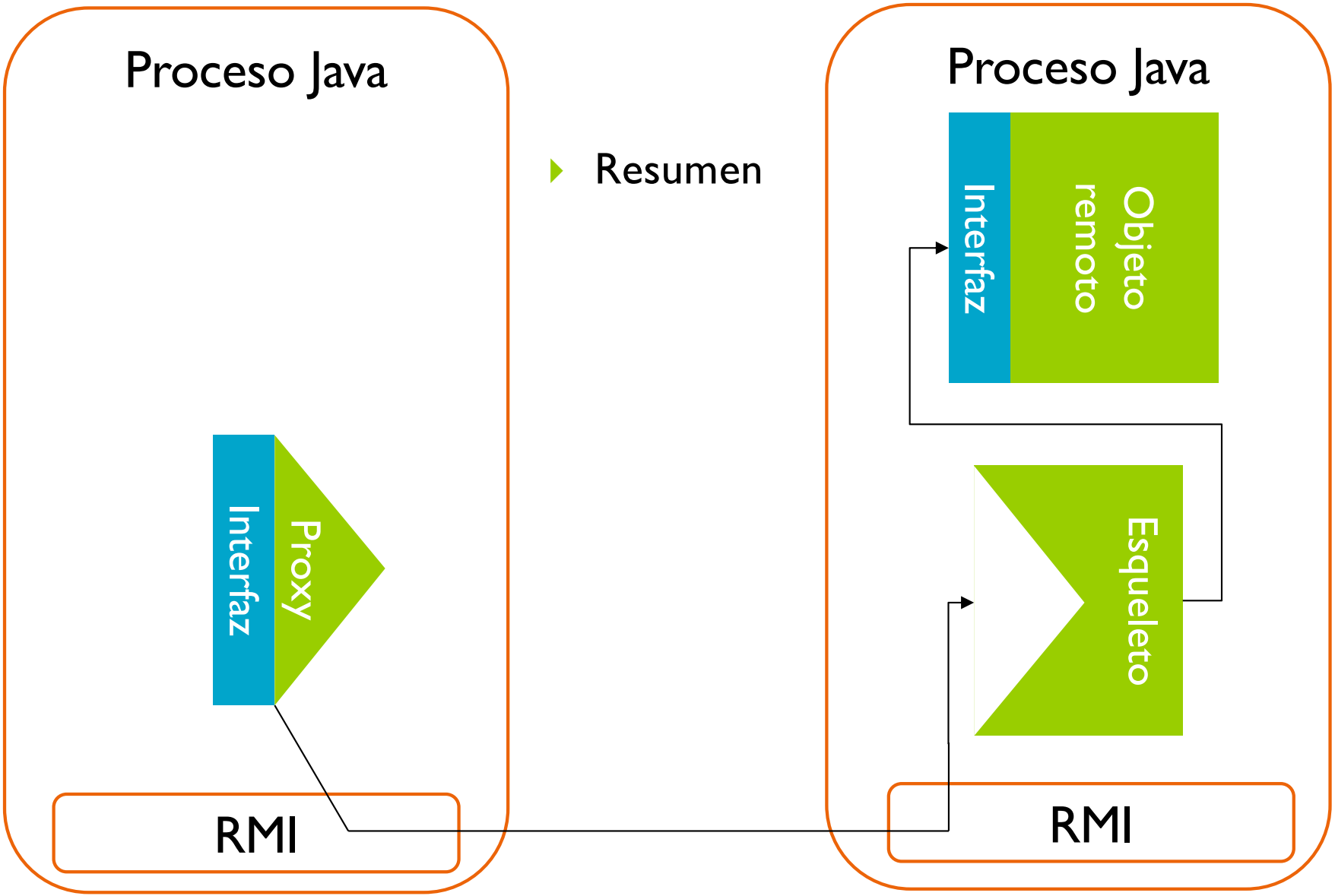
¿Qué es Java RMI?



- Para invocar a un objeto remoto desde otro proceso se utiliza un objeto Proxy
- Su interfaz es idéntica a la del objeto remoto
- Contiene una referencia al objeto remoto
  - Dirección IP + puerto + qué objeto
  - Permite localizar al esqueleto del objeto remoto

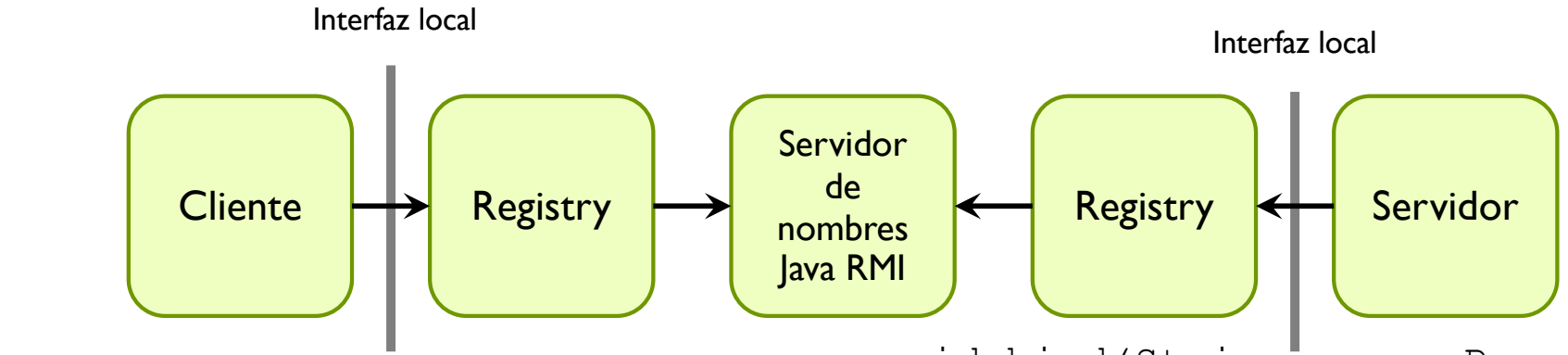
¿Qué es Java RMI?

- Resumen



El servidor de nombres de Java RMI

- El servidor de nombres almacena, para cada objeto:
  - nombre simbólico + referencia
- Puede residir en cualquier nodo y es accedido desde el cliente o el servidor usando la interfaz local llamada **`Registry`**
- En las distribuciones Oracle de Java, el servidor de nombres se lanza usando la orden **`rmiregistry`**



Desarrollo de una aplicación Java RMI → Aplicación **NO RMI**

```
public interface Hola {
  String saluda();
}

class ImplHola implements Hola {
  ImplHola() {..} // constructor
  public String saluda() {return "Hola a todos";}
}

...
Hola h = new ImplHola();
System.out.println(h.saluda());
```

Desarrollo de una aplicación Java RMI → Reglas para programar objetos remotos en Java

- **Interfaz de objeto remoto**
  - La interfaz del objeto remoto debe **extender** la interfaz **`java.rmi.Remote`**
  - Los métodos de dicha interfaz deben indicar que puede generarse la **excepción** **`RemoteException`**
  - A partir de la definición de la interfaz, el compilador de Java genera proxies y esqueletos

```
public interface Hola extends Remote {
  String saluda() throws RemoteException;
}
```

Desarrollo de una aplicación Java RMI → Reglas para programar objetos remotos en Java

- **Clase de objetos remotos**
  - La clase de los objetos remotos debe:
    - **implementar** la interfaz remota
    - **extender** `java.rmi.server.UnicastRemoteObject`
  - Esto permite registrar los objetos en el ORB de Java

```
class ImpleHola extends UnicastRemoteObject implements Hola {
  ImpleHola() throws RemoteException {..} // constructor
  public String saluda() throws RemoteException {
    return "Hola a todos";
  }
}
```

Desarrollo de una aplicación Java RMI → Aplicaciones servidor y cliente

// SERVIDOR

```
...
Registry reg = LocateRegistry.getRegistry(host, port);
reg.rebind("objetoHola", new ImpleHola());
System.out.println("Servidor Hola preparado");
```

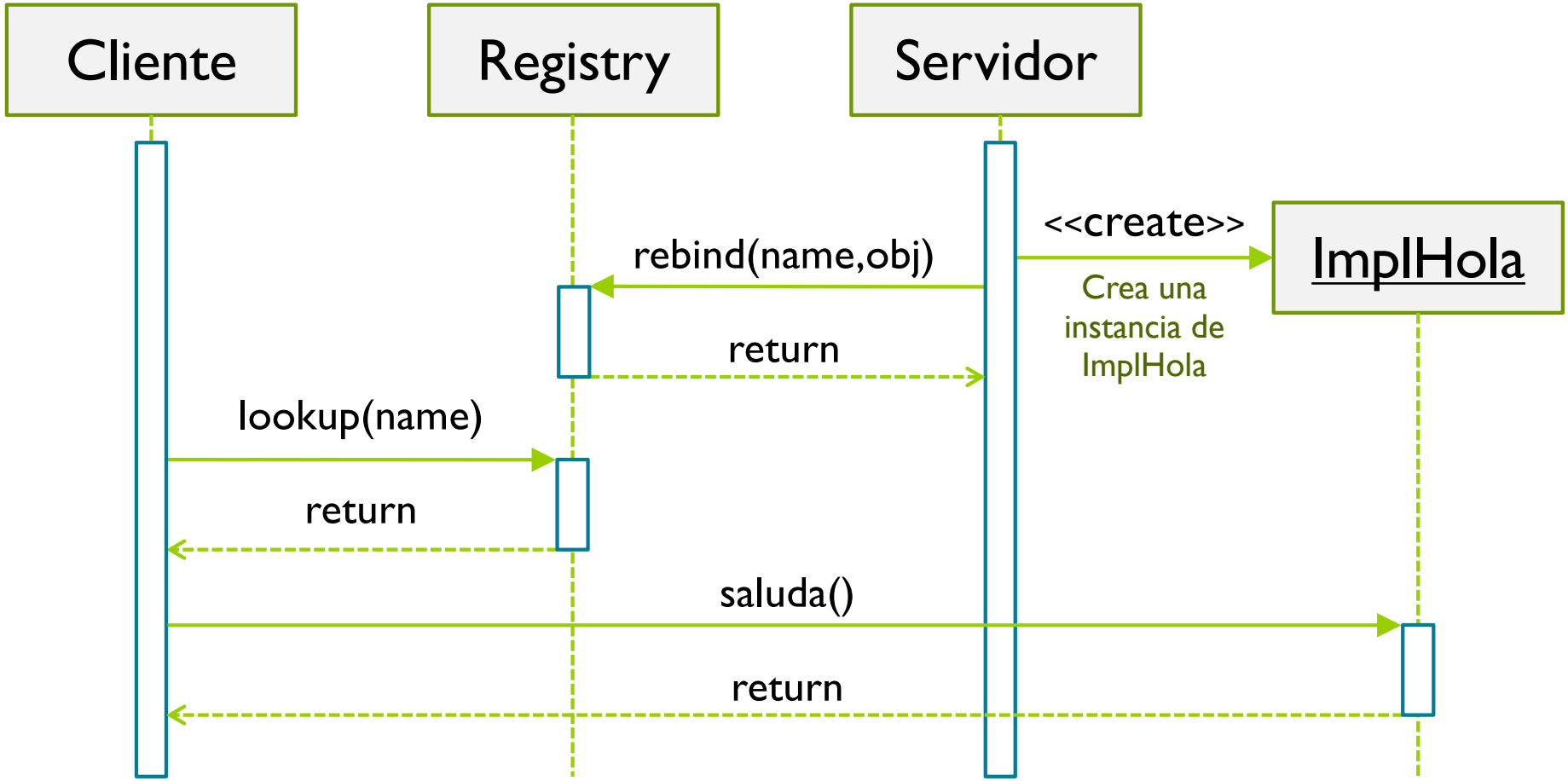
// CLIENTE

```
...
Registry reg = LocateRegistry.getRegistry(host, port);
Hola h = (Hola) reg.lookup("objetoHola");
System.out.println(h.saluda());
```

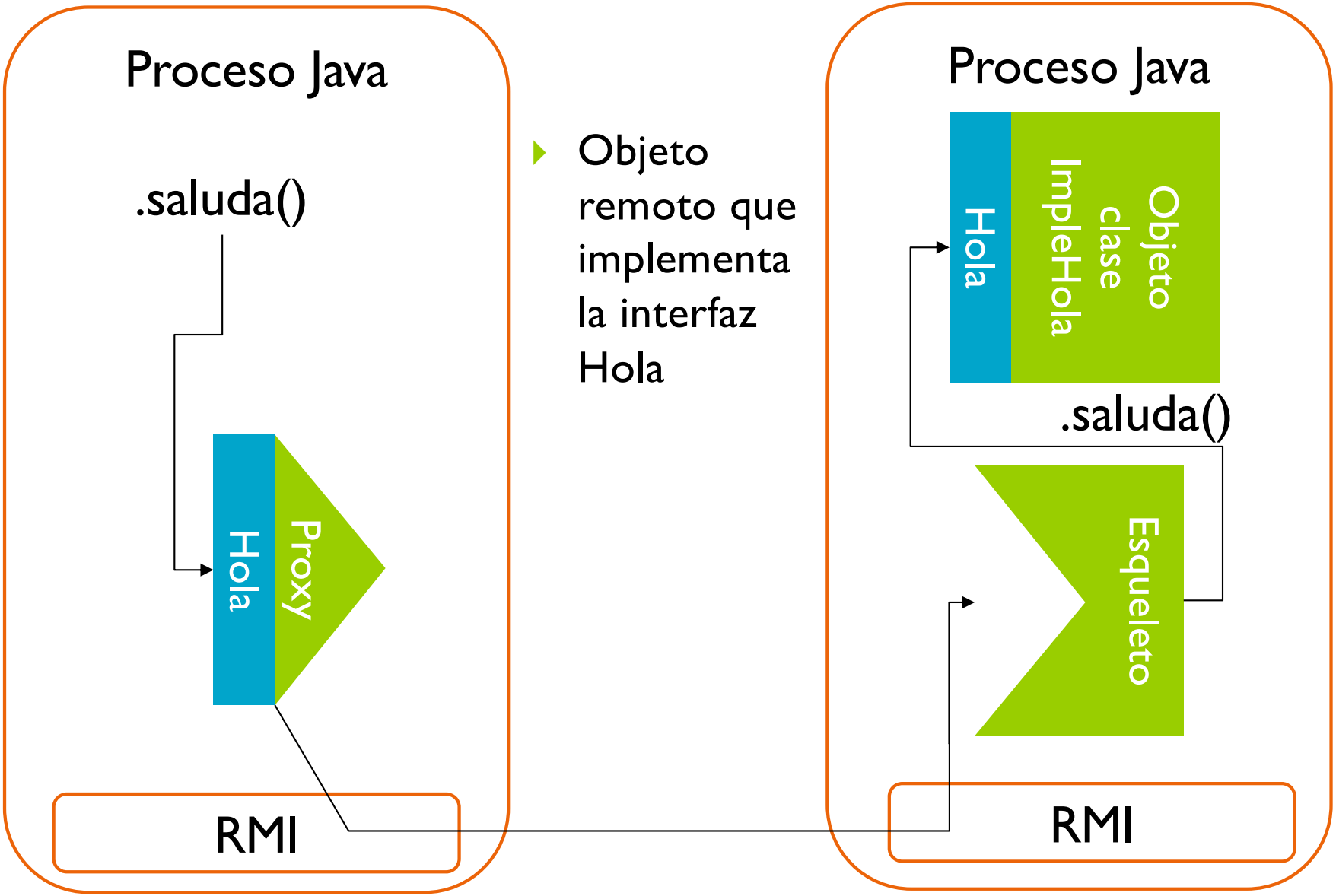
// La dirección (host, port) del Registry se ha facilitado como argumentos

Desarrollo de una aplicación Java RMI

- **Pasos en la ejecución de este ejemplo:**



Desarrollo de una aplicación Java RMI



Paso de objetos como argumentos en Java RMI

- Cuando se invoca un método, podemos pasar objetos como argumentos
  - Si el objeto que se pasa como argumento implementa la interfaz **Remote**
    - Se pasa por referencia
    - El objeto es compartido por las referencias previas y la nueva
  - Si el objeto que se pasa como argumento **NO** implementa la interfaz **Remote**
    - Se **serializa** y se pasa por valor
    - Se crea un objeto en la máquina virtual destino totalmente independiente al original

Características de la comunicación en Java RMI

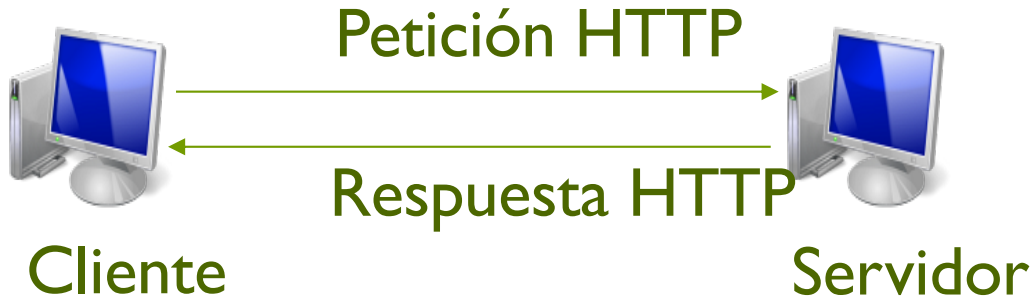
- Utilización:
  - Las llamadas a métodos remotos hacen transparente al programador el uso de las primitivas básicas de comunicación
- Estructura y contenido de los mensajes
  - Determinados por el compilador de Java, transparente al programador
- Direccionamiento
  - Directo al ordenador donde reside el objeto remoto
- Sincronización
  - Sincrónica en la respuesta. Se espera a que el método remoto termine
- Persistencia
  - No persistente. El objeto remoto debe estar activo.

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
- Servicios web
  - Conceptos generales
  - Servicios web RESTful
- Middlewares orientados a mensajería

Servicios web

- Sistema software diseñado para proporcionar **interacciones ordenador-ordenador** utilizando la red *[W3C Web Services Glossary]*
- Normalmente basado en una arquitectura **cliente-servidor** (petición-respuesta) implementada sobre el protocolo **HTTP**
- No se solicitan páginas web, sino consultas y acciones



Servicios web

- Existen numerosas variantes. Las más representativas son
  - Servicios web basados en SOAP y WSDL
    - Generalmente conocidos simplemente como "servicios web"
    - SOAP (Simple Object Access Protocol): especificación XML de la información que se intercambia
    - WSDL (Web Services Description Language): especificación XML que describe la funcionalidad ofrecida por un servicio web
  - Servicios web RESTful
    - Alternativa más simple y flexible, lo que ha hecho que tengan gran aceptación en la actualidad, desbancando en gran medida a los servicios web "clásicos"
    - JSON (JavaScript Object Notation) es el formato más habitual para el intercambio de información
    - No ser requiere ningún lenguaje de descripción de funcionalidad. No obstante, empiezan a emplearse, siendo OAS (OpenAPI Specification) el más utilizado actualmente

Servicios web

- Alternativas para utilizar servicios web
  - Construcción y procesado directo del contenido de los mensajes HTTP.
  - Generación automática de código proporcionada por frameworks de desarrollo y despliegue de servicios web, tanto para el lado cliente como el lado servidor, a partir de la definición del servicio
    - Dado el mecanismo de petición-respuesta y la generación automática de código, los servicios web podrían considerarse como una forma de RPC

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
- Servicios web
  - Conceptos generales
  - Servicios web RESTful
    - ¿Qué es un servicio web RESTful?
    - Un ejemplo: Google Drive Web API
- Middlewares orientados a mensajería

Servicios web RESTful (I)

- Constituyen una de las tecnologías **más importantes** para el desarrollo de aplicaciones web
- Están **disponibles** en la inmensa mayoría de lenguajes de programación y *frameworks* de desarrollo
  - Existen APIs REST para acceso a múltiples aplicaciones distribuidas. Ejemplo: Google Drive, Instagram, Gmail, ..
- Gran parte de su éxito se debe a su **sencillez** de uso

Servicios web RESTful (II)

- Estrictamente no es un estándar, sino un conjunto de convenciones de uso (estilo arquitectónico **REST = Representational State Transfer**)
  - **Datos y funcionalidad** se consideran **recursos** → se accede a ellos mediante URIs
  - Se actúa sobre recursos utilizando un conjunto de **operaciones simples** y bien definidas (basadas en HTTP)
  - Se emplea la arquitectura cliente/servidor
  - Diseñado para utilizar un protocolo de comunicación **sin estado**
    - Son servicios sin estado (**deterministas**), y basados en caching
    - Un cliente puede almacenar información para no tener que consultar constantemente al servidor
  - REST + Servicios Web = **Servicios Web RESTful**

Referencias a recursos en REST

- **Identificación de recursos a través de la URI:** proporcionan un espacio de dirección global para el descubrimiento de recursos y servicios.
- Ejemplos:
  - `http://administracion.upv.es/alumno`
    - La colección de todos los alumnos de la universidad
  - `http://administracion.upv.es/alumno/123456789F`
    - Un alumno cuyo identificador es 123456789F

Representación de recursos en REST

- Representación de recursos libre
  - Las más comunes son XML y JSON

XML	JSON

<Person>
<ID>123456789F</ID>
<Name>Agustín Espinosa</Name>
 	
<Email>agessa@alumno.upv.es</Email>
</Person>
	{
	"ID": "123456789F",
	"Name": "Agustín Espinosa",
	"Email": "agessa@alumno.upv.es"
	}

Operaciones en REST

- Métodos HTTP para indicar al servidor el tipo de operación

Método	Operación en el servidor
GET	Leer un recurso
POST	Crear un recurso
PUT	Modificar un recurso
DELETE	Borrar un recurso
…	…

Códigos de estado en REST

- En la respuesta se indica cómo ha finalizado la llamada al servicio utilizando códigos de estado HTTP
- Ejemplos:

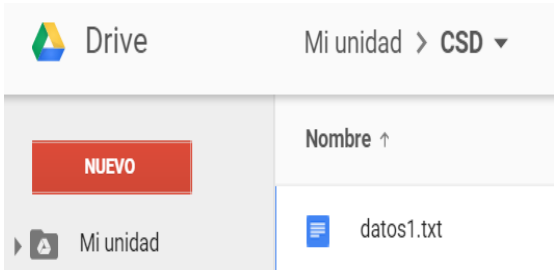
Código HTTP	Significado habitual
200	Todo correcto
201	Recurso creado
400	Solicitud incorrecta
404	Recurso no encontrado
500	Fallo en el proveedor del servicio

Parámetros en REST

- Los URIs pueden incluir parámetros para:
  - Realizar consultas
    - http://administracion.upv.es/alumno?apellido=Espinosa
  - Paginar las respuestas
    - http://administracion.upv.es/alumno?pagina=4&tampagina=50
  - Facilitar información de autenticación
    - http://administracion.upv.es/alumno?key=01234567-89ab-cdef-0123-456789abcdef
  - …

RESTful.- Ejemplos

- HTTP GET http://administracion.upv.es/alumno?apellido=Espinosa
  - consulta: argumento "apellido" con valor Espinosa
- HTTP GET http://administracion.upv.es/alumno?key=01234567-89abcf-0123
  - consulta todos los datos de alumnos, facilita info. autenticación (arg. key)
- HTTP POST http://www.appdomain.com/users
  - crea el recurso indicado (ej. crea colección)
- HTTP PUT http://www.appdomain.com/users/123
  - actualiza el recurso (ej. actualiza item de una colección)
- HTTP DELETE http://www.appdomain.com/users/123
  - elimina item de una colección



Ejemplo: Google Drive Web API

Petición HTTP

```
GET
https://www.googleapis.com/drive/v2/files?q=title+%3D
+'datos1.txt'&
fields=items(id%2Ctitle)&key={YOUR_API_KEY}
```

Respuesta HTTP

```
200 OK
{
 "items": [
  {
   "id":
"1Buza8URDmLc4vbb2EP_mXkktRmRtelVMPaOjkSWtzq4",
   "title": "datos1.txt"
  }
 ]
}
```

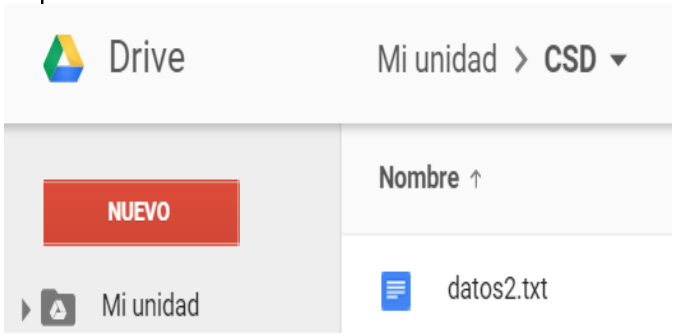
Petición HTTP

```
PUT
https://www.googleapis.com/drive/v2/files/1Buza8URDmL
c4vbb2EP_mXkktRmRtelVMPaOjkSWtzq4?fields=title&key={Y
OUR_API_KEY}

{
 "title": "datos2.txt"
}
```

Respuesta HTTP

```
200 OK
 {
 "title": "datos2.txt"
}
```



Características de la comunicación en REST

- REST es un mecanismo de difícil categorización, en función del enfoque que se emplee para analizarlo:
  1. Desde el punto de vista del mecanismo subyacente, sus características serían aquellas asociadas a la comunicación basada en el protocolo HTTP, y por tanto, en el uso de sockets TCP.
  2. En cambio, desde el punto de vista del uso, es básicamente un mecanismo de petición-respuesta, similar en muchos aspectos a RPC y ROI.

Características de la comunicación en REST

- Utilización:
  - Enfoque 1: Primitivas básicas de envío y recepción (API de sockets)
  - Enfoque 2: API de alto nivel proporcionada por el proveedor del servicio
- Estructura y contenido de los mensajes
  - Enfoque 1: Contenido codificado con HTTP, XML, JSON
  - Enfoque 2: Contenido oculto por la API del proveedor
- Direccionamiento
  - Directo mediante peticiones al ordenador que alberga el servicio
- Sincronización
  - Enfoque 1: Sincrónica en la entrega
  - Enfoque 2: Sincrónica en la respuesta
- Persistencia
  - No persistente

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
- Servicios web
- Middlewares orientados a mensajería
  - Conceptos generales
  - Java Message Service

Middlewares orientados a mensajería (MOMs)

- Middlewares que ofrecen comunicación basada en **mensajes**.
- Los emisores envían mensajes no directamente al receptor, sino a un elemento intermedio denominado generalmente **cola**.
  - Una vez depositado el mensaje en la cola, el emisor prosigue con su ejecución, sin esperar a que el receptor recoja el mensaje
  - El receptor recoge el mensaje en cualquier momento (inmediatamente cuando se deposita o más tarde)
  - La comunicación es por tanto **asíncrona**.

Middlewares orientados a mensajería (MOMs)

- La mayoría de estos sistemas están basados en un **bróker** de comunicaciones:
  - Un proceso servidor
  - Gestiona totalmente las colas:
    - Creación y borrado
    - Atiende los envíos de emisores para depositar los mensajes en la cola indicada.
    - Atiende las peticiones de receptores para recoger mensajes de la cola indicada
    - Mantiene los mensajes en las colas pendientes de entrega, aunque emisores y receptores no estén en ejecución, incluso ni siquiera el propio bróker, ofreciendo por tanto persistencia.
- Excepción notable: ZeroMQ
  - No requiere la existencia del bróker

Middlewares orientados a mensajería (MOMs)

- Principales ventajas
  - Permiten componentes del sistema distribuido altamente desacoplados.
  - Pueden ofrecer un alto grado de disponibilidad mediante réplicas activas de brokers
  - Es posible conseguir un alto grado de seguridad al ser sistemas centralizados
- Principales inconvenientes
  - Menor rendimiento, al introducir el bróker como elemento intermedio en las comunicaciones
  - Difícil escalabilidad, de nuevo por la existencia del bróker
  - Falta de estandarización. Existen protocolos abiertos pero muchas de las implementaciones más utilizadas son propietarias.

Contenido

- Características de los mecanismos de comunicación
- Invocación a objeto remoto (ROI)
- Servicios web
- Middlewares orientados a mensajería
  - Conceptos generales
  - Java Message Service
    - Componentes
    - Modelo de programación
    - Ejemplo

Java Message Service 2.0 (JMS)

- Java Message Service (JMS) es una **API Java** que permite a las aplicaciones enviar y recibir mensajes
  - Ejemplo de MOM sobre Java
- Ofrece una comunicación **débilmente acoplada**
  - El **emisor** envía mensajes a un destino y el **receptor** recibe mensajes de dicho destino
  - Emisor y receptor no necesitan conocerse entre sí, solo deben estar de acuerdo en el formato del contenido de los mensajes.

Java Message Service 2.0 (JMS)

- Resulta interesante utilizar JMS cuando:
  - No se quiere que los componentes de una aplicación dependan de conocer las interfaces de otros componentes
  - No es necesario que todos los componentes estén simultáneamente en ejecución
  - Los componentes, tras enviar un mensaje, no necesitan recibir una respuesta inmediata para poder continuar operando con normalidad

Java Message Service 2.0 (JMS) → Componentes

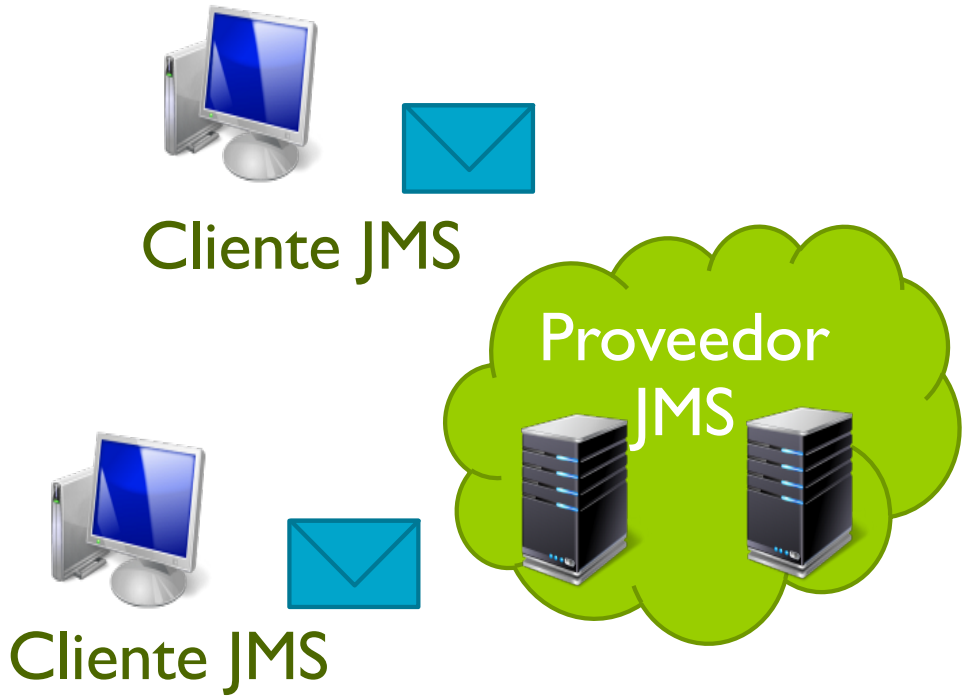
- **Componentes de JMS**
  - **Proveedores JMS**
    - Sistema de mensajería que implementa las interfaces de JMS y proporciona herramientas administrativas y de control
  - **Clientes JMS**
    - Programa o componente escrito en Java que produce o consume mensajes
  - **Mensajes**
    - Objetos que comunican información entre clientes JMS
  - **Objetos administrados**
    - **Factorías de conexiones**
    - **Destinos**: colas y temas (*topics*)

- **Proveedor JMS**
  - Es un sistema de mensajería que implementa las interfaces de JMS y proporciona herramientas administrativas y de control
  - Incluido en los servidores de la plataforma Java Enterprise Edition (JEE):
    - GlassFish, Oracle WebLogic Server, IBM WebSphere Application Server, Apache Geronimo, etc.
  - También disponible como servidores independientes (solo JMS)
    - Apache ActiveMQ, JBoss Messaging, SwiftMQ, Open Message Queue, etc.

Java Message Service 2.0 (JMS) → Componentes (II)

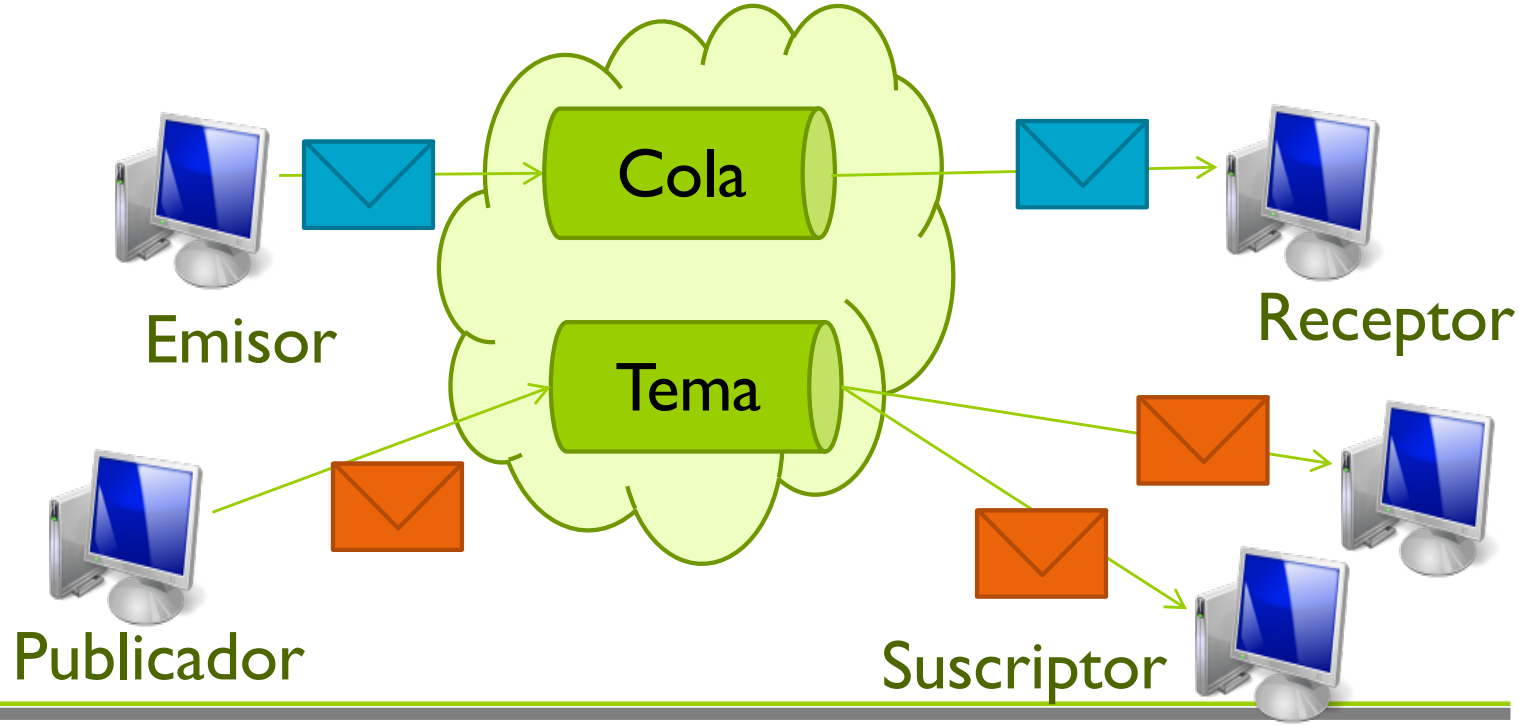
- **Mensajes:** con tipo y estructura definida

Cabecera
Campos fijos, definidos por JMS (identificador, instante de envío, etc.)
Propiedades
Extensión de la cabecera. Campos definidos por el programa
Cuerpo
Tipos: vacío, texto, bytes, objeto serializado, diccionario, etc.



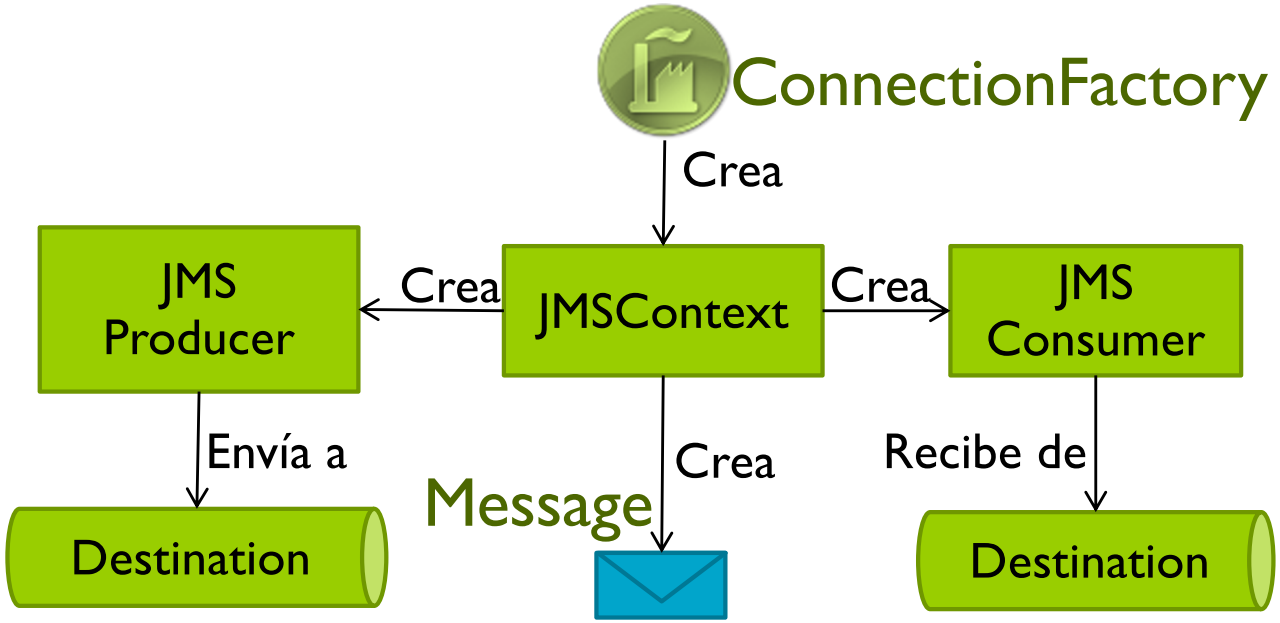
- **Objetos administrados: Factorías de conexiones**
  - Creadas mediantes las herramientas administrativas del proveedor JMS
  - Se utilizan para crear las conexiones de los clientes al sistema de mensajería

- **Objetos administrados: Destinos**
  - Creados mediantes las herramientas administrativas del proveedor JMS
  - Tipos:
    - **Colas**: entrega a un solo cliente
    - **Temas** (*Topics*): entrega a múltiples clientes (difusión)



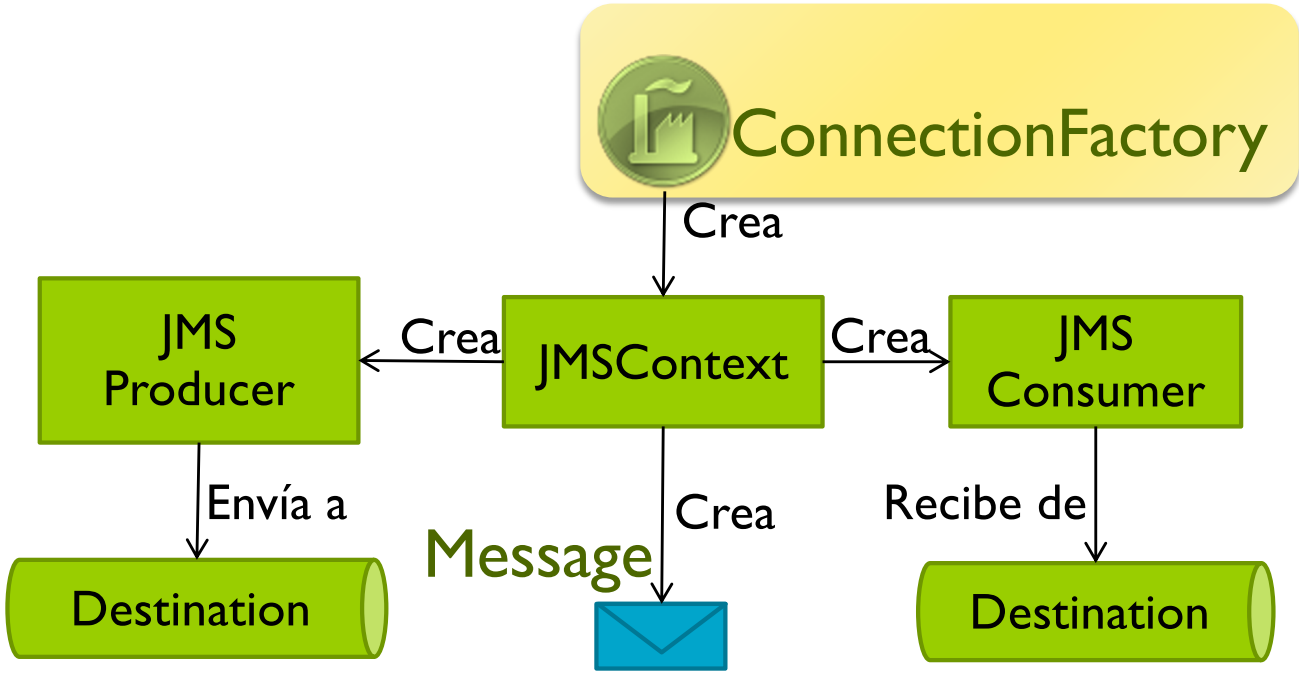
JMS: Modelo de programación

- **Elementos del modelo de programación**
  - Interface ConnectionFactory
  - Interface JMSContext
  - Interface JMSProducer
  - Interface JMSConsumer
  - Interface Destination
  - Interface Message



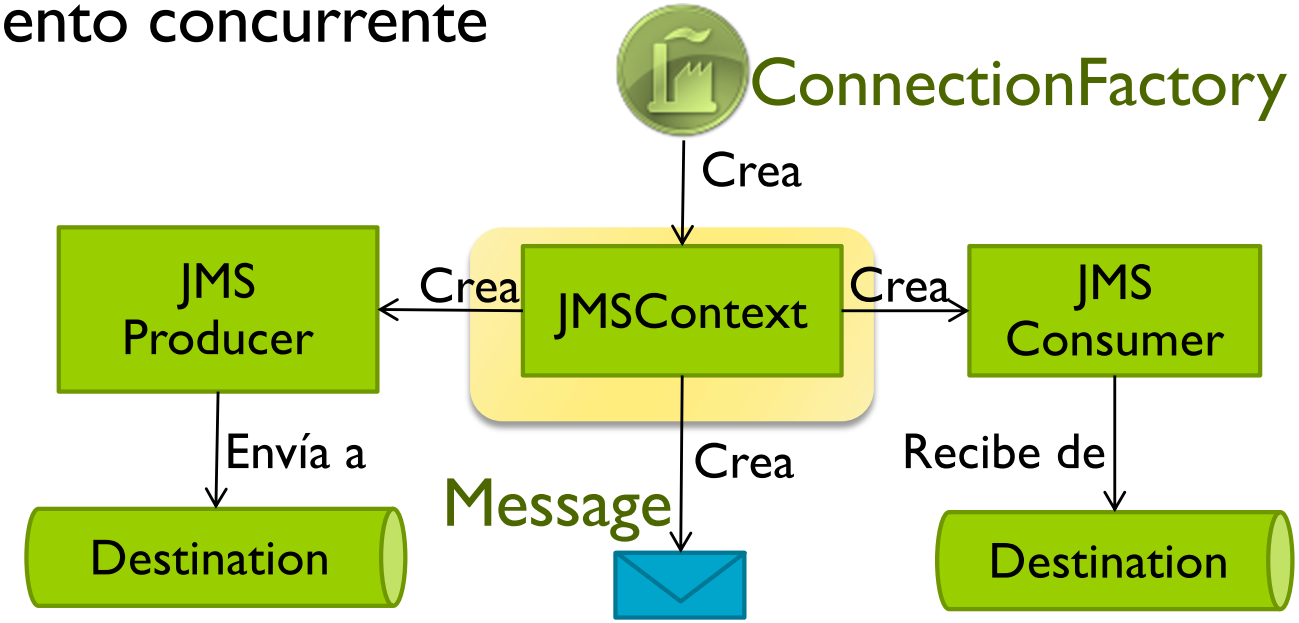
JMS: Modelo de programación (I)

- **Interface ConnectionFactory**
  - Los objetos que la implementan:
    - Vinculan la aplicación con un objeto administrado
    - Crean conexiones con el proveedor JMS



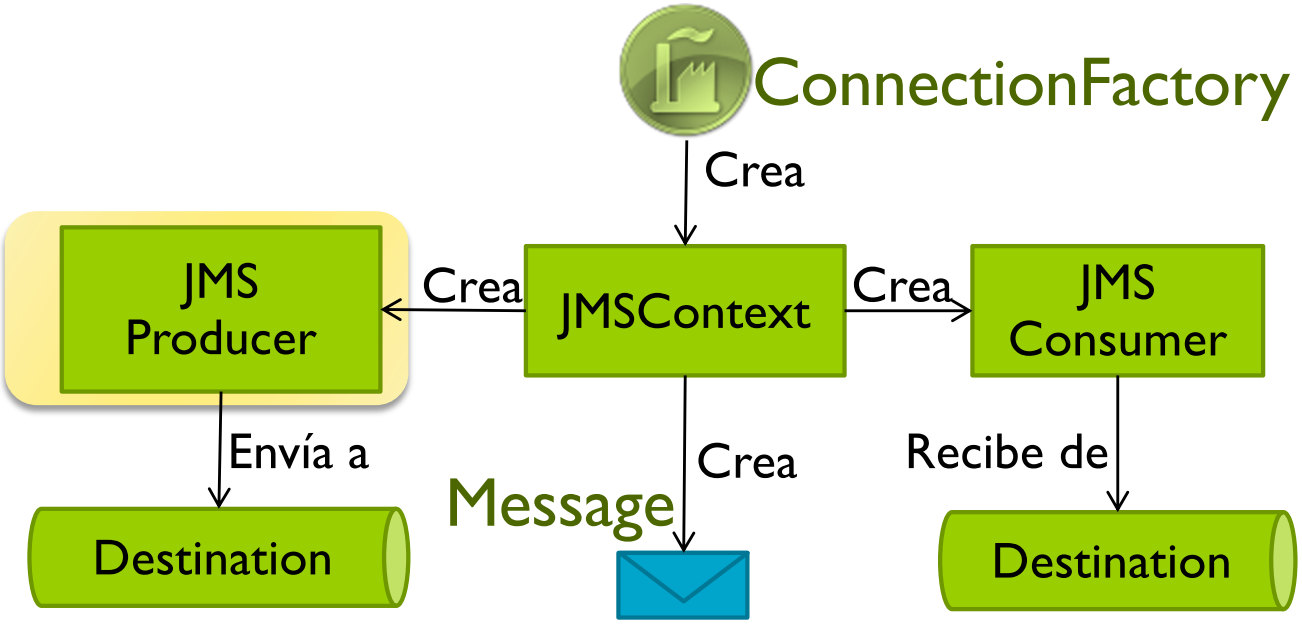
JMS: Modelo de programación (II)

- **Interface JMSContext**
  - Los objetos que la implementan:
    - Mantienen una conexión con el proveedor JMS
    - Son utilizables por un solo hilo de ejecución
    - Procesan envíos y recepciones de forma secuencial
    - Una aplicación puede usar varios objetos JMSContext si requiere procesamiento concurrente



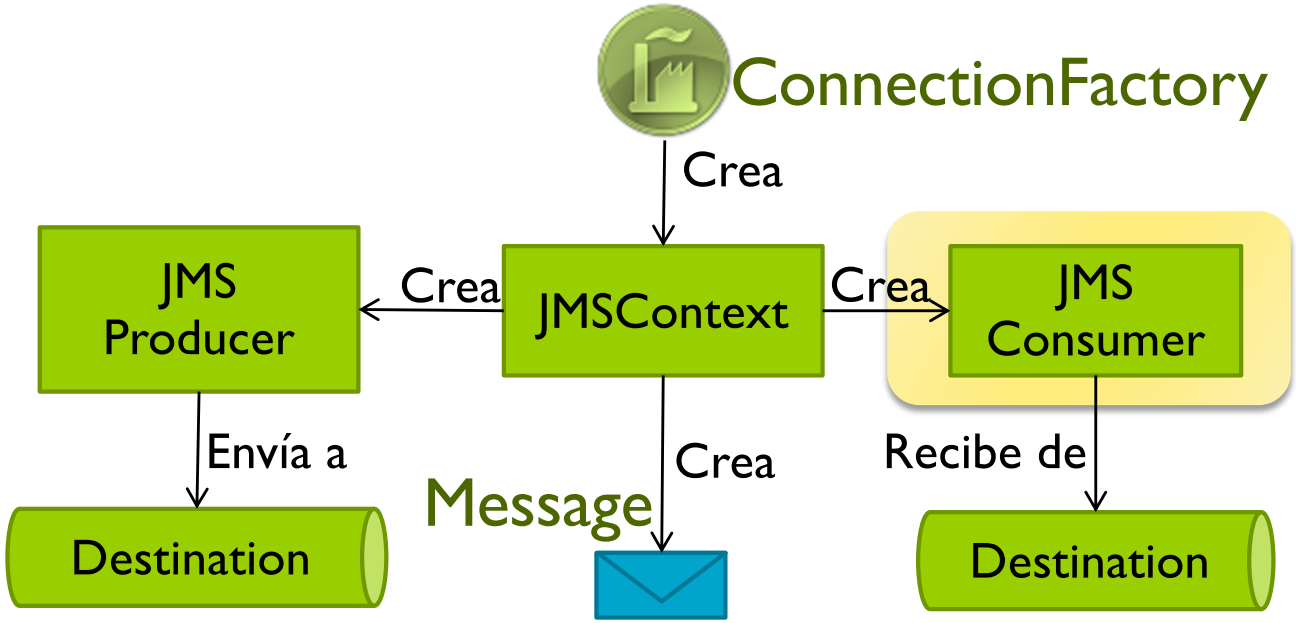
JMS: Modelo de programación (III)

- **Interface JMSProducer**
  - Los objetos que la implementan:
    - Permiten **enviar** mensajes a colas y temas



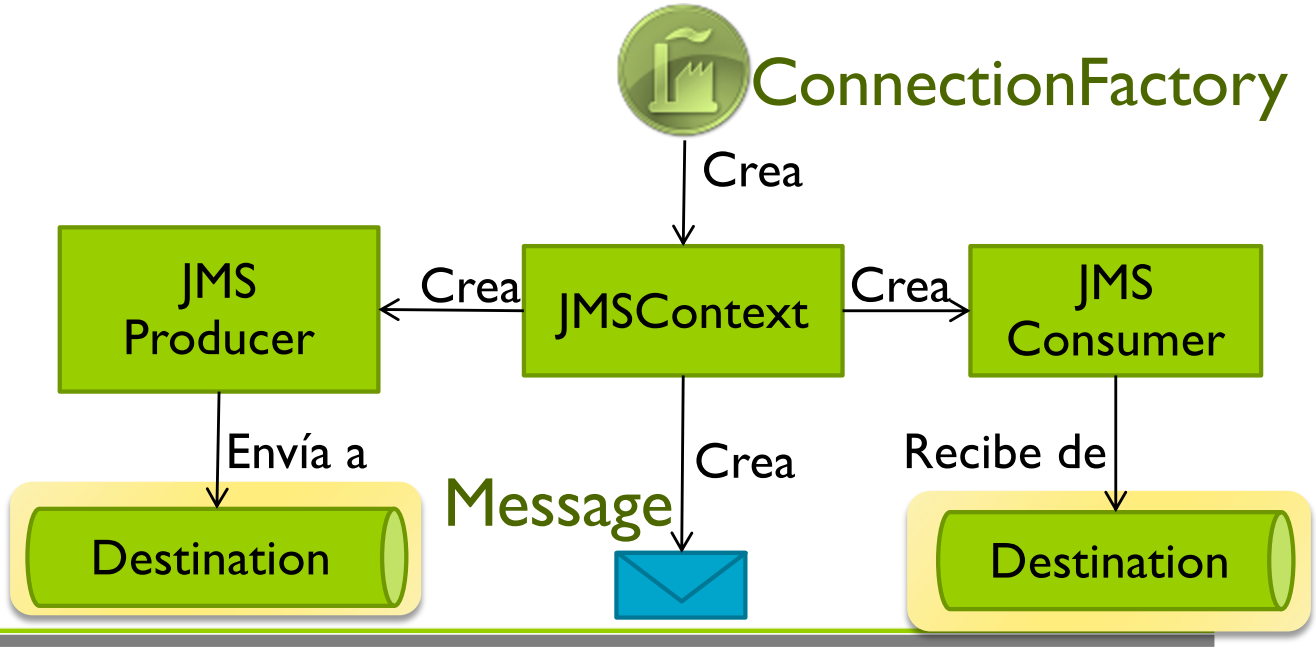
JMS: Modelo de programación (IV)

- **Interface JMSConsumer**
  - Los objetos que la implementan:
    - Permiten **recibir** mensajes de colas y temas



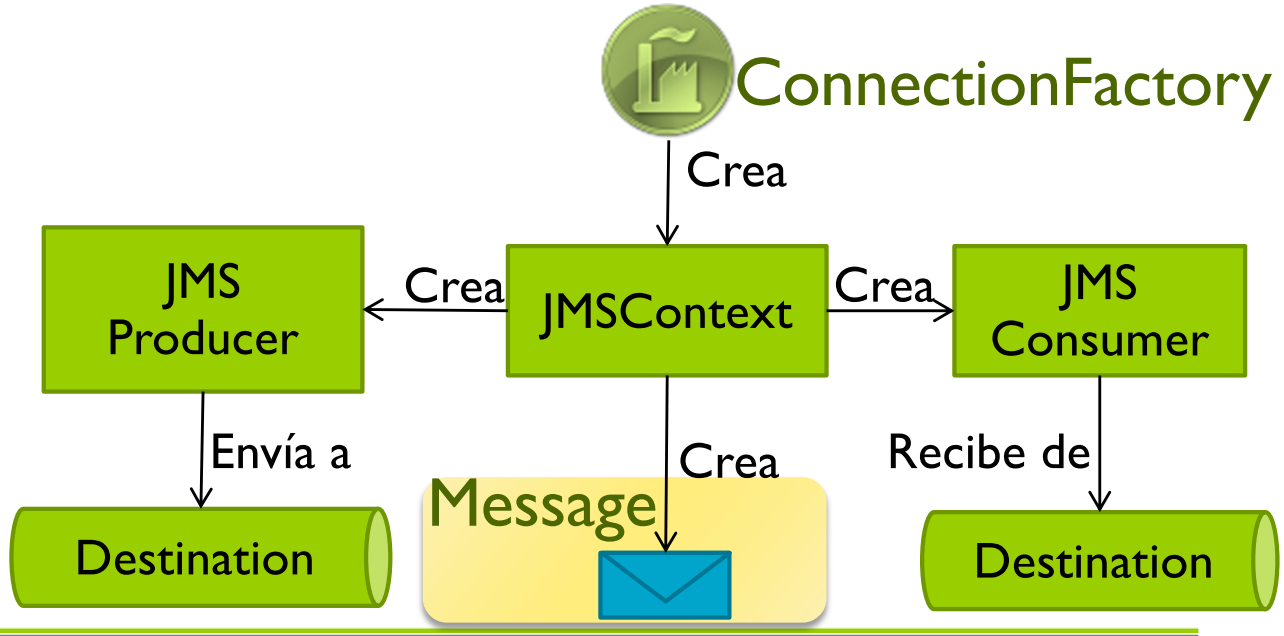
JMS: Modelo de programación (V)

- **Interface Destination**
  - Los objetos que la implementan:
    - Vinculan la aplicación con un objeto administrado
    - Encapsulan una dirección específica del proveedor JMS
  - Subinterfaces:
    - **Queue**
    - **Topic**



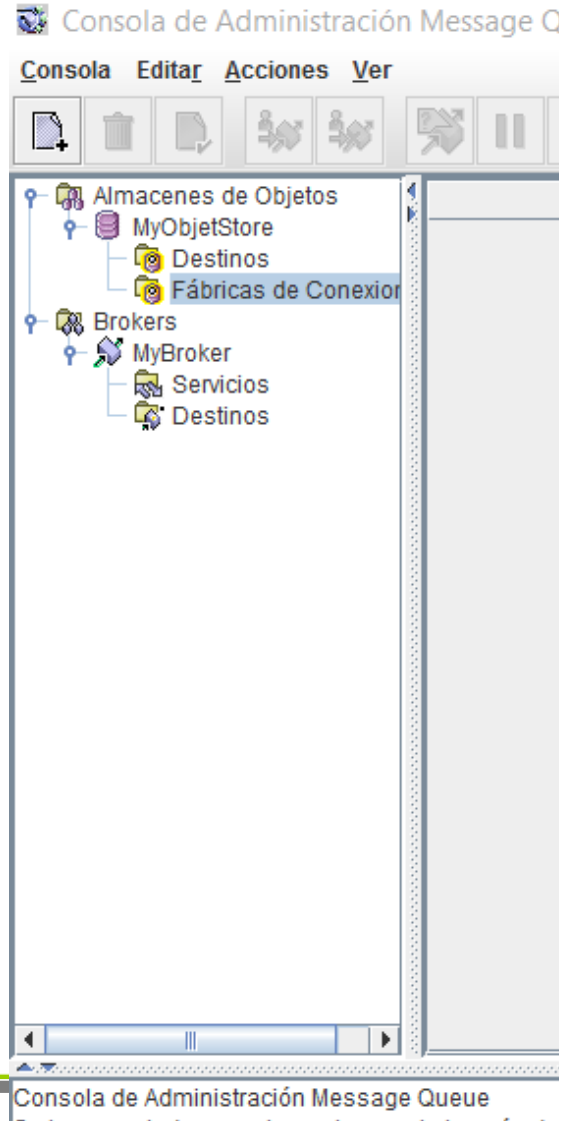
JMS: Modelo de programación (VI)

- **Interface Message**
  - Los objetos que la implementan:
    - Son los mensajes que se envían y reciben en JMS
  - Subinterfaces:
    - TextMessage
    - BytesMessage
    - ObjectMessage
    - MapMessage
    - etc.

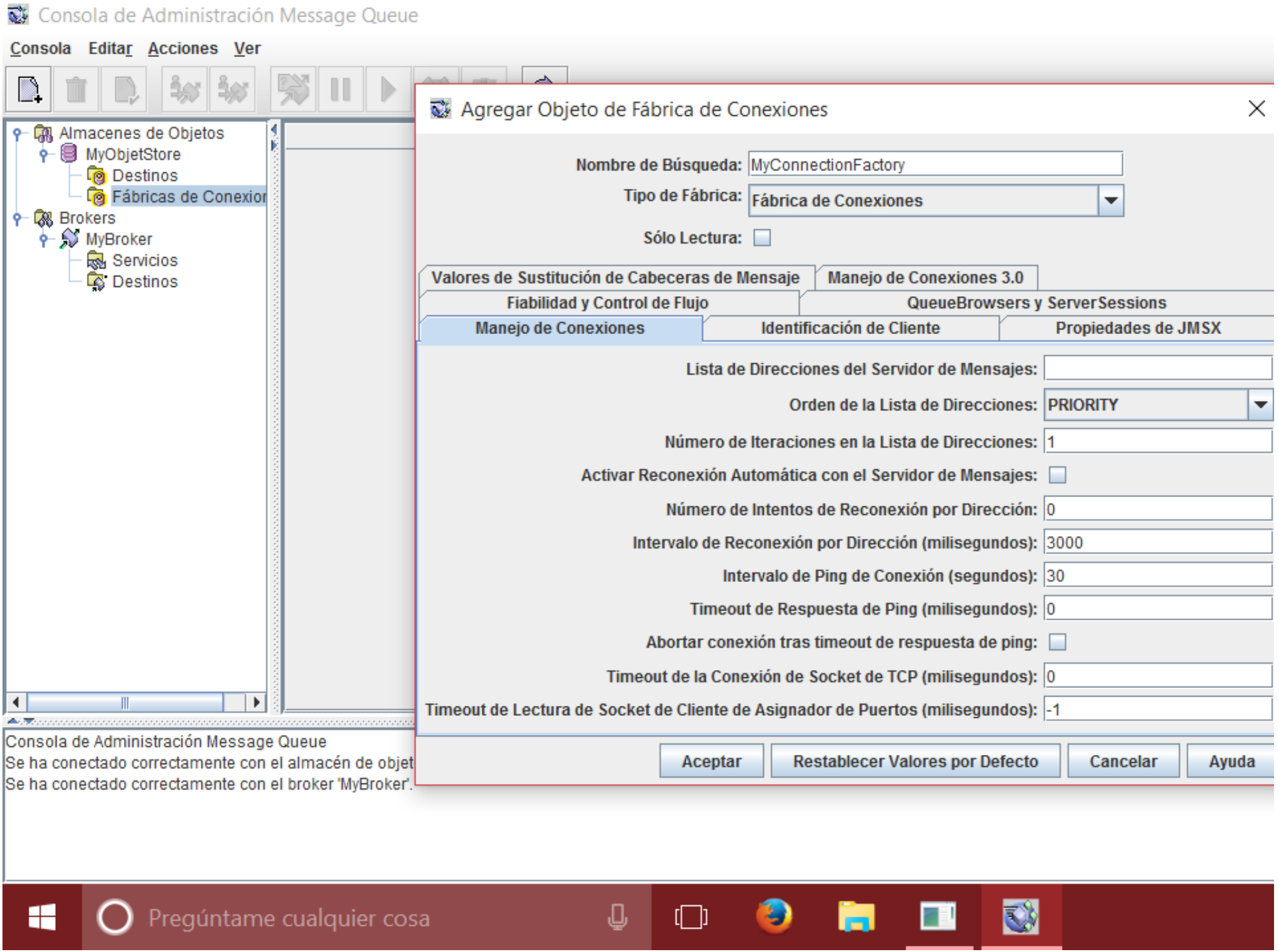


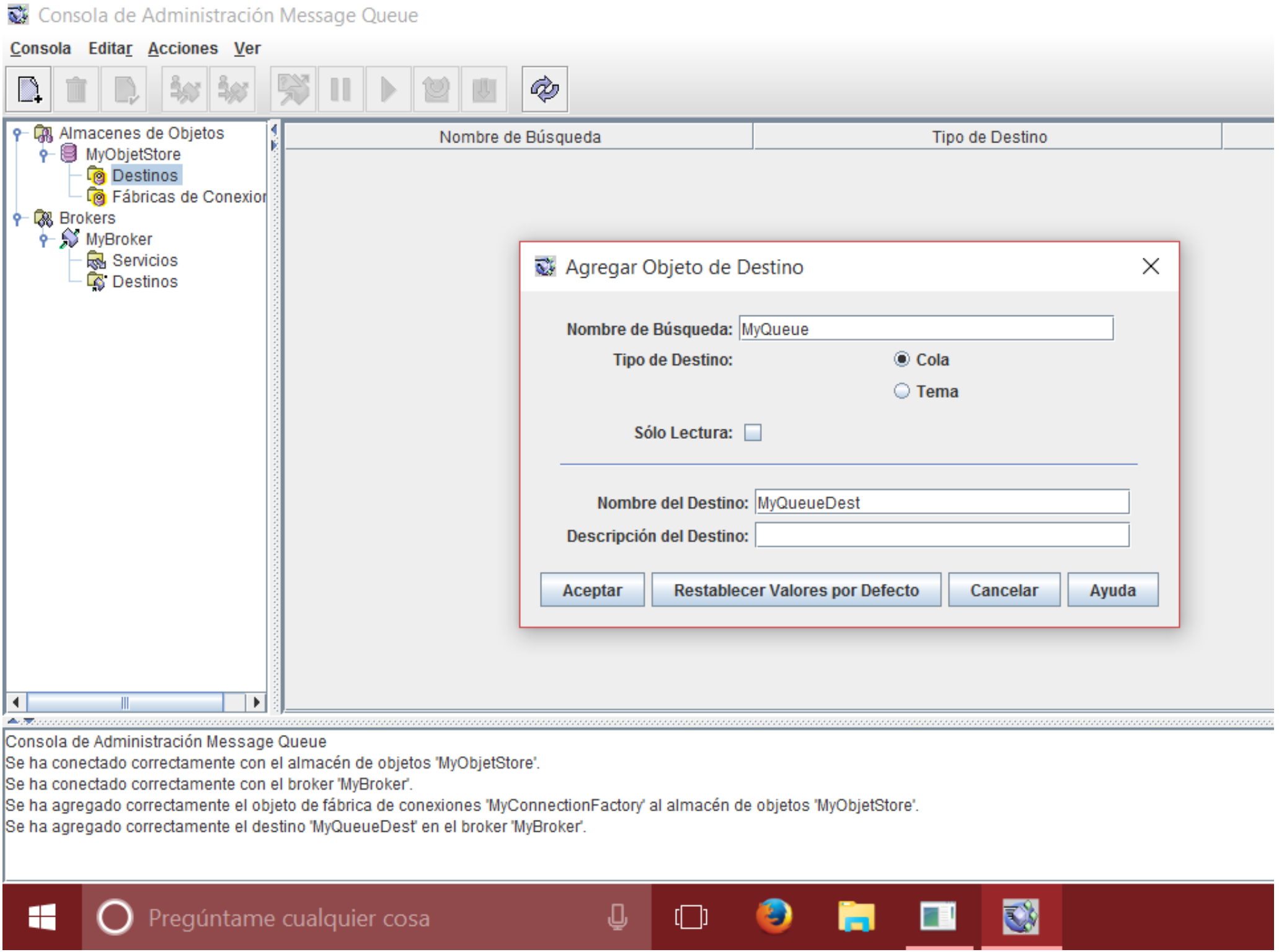
Java Message Service 2.0 (JMS) → Ejemplo

- Crear una cola de forma que un cliente envía un mensaje y otro lo recibe
- Proveedor JMS
  - Utilizamos Open Message Queue
  - Creamos destinos y factorías de conexiones
- Con la factoría se crea un contexto JMS, que crea al productor y al consumidor



- Productor
  - Envía un mensaje de texto a la cola "MyQueue" y termina
- Consumidor
  - Recibe mensajes de la cola "MyQueue"
  - Muestra cada mensaje
  - Finaliza tras 10 segundos sin recibir mensajes


Consola de Administración Message Queue
Consola Editar Acciones Ver
Almacenes de Objetos
MyObjetStore
Destinos
Fábricas de Conexior
Brokers
MyBroker
Servicios
Destinos
Agregar Objeto de Fábrica de Conexiones
Nombre de Búsqueda: MyConnectionFactory
Tipo de Fábrica: Fábrica de Conexiones
Sólo Lectura:
Valores de Sustitución de Cabeceras de Mensaje
Manejo de Conexiones 3.0
Fiabilidad y Control de Flujo
QueueBrowsers y ServerSessions
Manejo de Conexiones
Identificación de Cliente
Propiedades de JMSX
Lista de Direcciones del Servidor de Mensajes:
Orden de la Lista de Direcciones: PRIORITY
Número de Iteraciones en la Lista de Direcciones: 1
Activar Reconexión Automática con el Servidor de Mensajes:
Número de Intentos de Reconexión por Dirección: 0
Intervalo de Reconexión por Dirección (milisegundos): 3000
Intervalo de Ping de Conexión (segundos): 30
Timeout de Respuesta de Ping (milisegundos): 0
Abortar conexión tras timeout de respuesta de ping:
Timeout de la Conexión de Socket de TCP (milisegundos): 0
Timeout de Lectura de Socket de Cliente de Asignador de Puertos (milisegundos): -1
Aceptar
Restablecer Valores por Defecto
Cancelar
Ayuda
Consola de Administración Message Queue
Se ha conectado correctamente con el almacén de objet
Se ha conectado correctamente con el broker 'MyBroker'.
Pregúntame cualquier cosa


Consola de Administración Message Queue
Consola Editar Acciones Ver
Almacenes de Objetos
MyObjetStore
Destinos
Fábricas de Conexior
Brokers
MyBroker
Servicios
Destinos
Nombre de Búsqueda
Tipo de Destino
Agregar Objeto de Destino
Nombre de Búsqueda: MyQueue
Tipo de Destino:
Cola
Tema
Sólo Lectura:
Nombre del Destino: MyQueueDest
Descripción del Destino:
Aceptar
Restablecer Valores por Defecto
Cancelar
Ayuda
Consola de Administración Message Queue
Se ha conectado correctamente con el almacén de objetos 'MyObjetStore'.
Se ha conectado correctamente con el broker 'MyBroker'.
Se ha agregado correctamente el objeto de fábrica de conexiones 'MyConnectionFactory' al almacén de objetos 'MyObjetStore'.
Se ha agregado correctamente el destino 'MyQueueDest' en el broker 'MyBroker'.
Pregúntame cualquier cosa

Java Message Service 2.0 (JMS) → Ejemplo

Producer

```
import javax.jms.*; import javax.naming.*; // Interfaces JMS y JNDI
public class Producer {
    public static void main(String[] args) {
        try {
           Context jndiCtx = new InitialContext(); // Inicialización JNDI
           // Acceso a objetos administrados
           ConnectionFactory connectionFactory =
             (javax.jms.ConnectionFactory) jndiCtx.lookup("MyConnectionFactory");
           Queue queue = (javax.jms.Queue) jndiCtx.lookup("MyQueue");
            // Crea contexto a partir de la factoría de conexiones
           JMSContext context = connectionFactory.createContext();
           // Crea productor de mensajes a partir del contexto
           JMSProducer producer = context.createProducer();
           // Crea mensaje a partir del contexto
           TextMessage message = context.createTextMessage();
           // Construye cuerpo del mensaje (tipo texto)
           message.setText("This is a message");
           // Crea y asocia al mensaje una propiedad
           message.setBooleanProperty("Important", true);
           // Envia mediante el productor de mensajes el mensaje a la cola.
           // Espera a que el proveedor JMS lo guarde, pero no a que lo reciba el cliente
           producer.send(queue, message);

        } catch (JMSException | JMSRuntimeException | NamingException e) { ...
```

Java Message Service 2.0 (JMS) → Ejemplo

Consumer

```
import javax.jms.*; import javax.naming.*; // Interfaces JMS y JNDI
public class Consumer {
    public static void main(String[] args) {
        try {
           Context jndiCtx = new InitialContext();  // Inicialización JNDI
           // Acceso a objetos administrados
           ConnectionFactory connectionFactory =
               (javax.jms.ConnectionFactory) jndiCtx.lookup("MyConnectionFactory");
           Queue queue = (javax.jms.Queue) jndiCtx.lookup("MyQueue");
            // Crea contexto a partir de la factoría de conexiones
           JMSContext context = connectionFactory.createContext();
           // Crea consumidor de mensajes a partir del contexto
           // asociado a una cola particular
           JMSConsumer consumer = context.createConsumer(queue);
            // Espera un mensaje durante 10 segundos máximo
           while (true) {
              Message m = consumer.receive(10000);
              if (m != null) {
                  System.out.println(m);
              } else {
                  break; // Espera máxima alcanzada
              }
           }
        } catch (JMSRuntimeException | NamingException e) { ...
```

Java Message Service 2.0 (JMS) → Ejemplo

```
Text:   This is a message
Class:
com.sun.messaging.jmq.jmsclient.TextMessageImpl
getJMSMessageID():      ID:9-192.168.137.1(f9:0:96:1b:b8:68)-
58313-1427645760159
getJMSTimestamp():      1427645760159
getJMSCorrelationID():  null
JMSReplyTo:             null
JMSDestination:         PhysicalQueue
getJMSDeliveryMode():   PERSISTENT
getJMSRedelivered():    false
getJMSType():           null
getJMSExpiration():     0
getJMSDeliveryTime():   0
getJMSPriority():       4
Properties:             {Important=true, JMSXDeliveryCount=1}
```

Cuerpo

Cabecera

Propiedades

Características de la comunicación JMS

- Utilización
  - Primitivas básicas de envío y recepción
- Estructura y contenido de los mensajes
  - Cabecera, propiedades, cuerpo (con diferentes tipos)
- Direccionamiento
  - Indirecto a través del proveedor JMS
- Sincronización
  - Asincrónica. El emisor sigue cuando entrega el mensaje al proveedor JMS
- Persistencia
  - Persistente. Incluso si el proveedor JMS se detiene, ya que este lo guarda en almacenamiento secundario.

Resultados de aprendizaje de la Unidad Didáctica

- Al finalizar esta unidad, el alumno deberá ser capaz de:
  - Caracterizar los mecanismos de comunicación mediante mensajes, describiendo sus características más relevantes.
  - Caracterizar el mecanismo de invocación a objeto remoto (ROI) y el uso de referencias a objeto.
  - Caracterizar el mecanismo de Java RMI y detallar los pasos en el desarrollo de una aplicación Java RMI
  - Caracterizar los servicios web RestFul. Representar de forma adecuada los recursos en REST. Describir las operaciones en REST.
  - Caracterizar las colas de mensajes y el mecanismo de comunicación de Java Message Service.

Bibliografía

- Llamada a procedimiento remoto (RPC)
  - Bruce J. Nelson. **Remote Procedure Call**. Tesis doctoral, Carnegie Mellon Univ., 1981.
  - Andrew D. Birrel y Bruce J. Nelson. **Implementing remote procedure calls**. ACM Trans. Comput. Syst., 2(1):39–59, febrero 1984.
- Invocación a objeto remoto (ROI)
  - Marc Shapiro. **Structure and encapsulation in distributed systems: The proxy principle**. In 6th Intnl. Conf. on Distrib. Comput. Sys. (ICDCS), pags. 198–204, Cambridge, Massachusetts, EE.UU., mayo 1986.
- Java RMI
  - Oracle Corp. **Remote Method Invocation home**. Disponible en: *http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, noviembre 2012.*
  - Oracle Corp. **The JavaTM tutorials: Rmi**. Disponible en: *http://docs.oracle.com/javase/tutorial/rmi/index.html, noviembre 2012.*

Bibliografía

Servicios Web RESTful

- **Tutorial "Developing RESTful APIs with JAX-RS"**. Java Brains. Koushik Kothagal. Disponible en *http://javabrains.koushik.org/courses/javaee_jaxrs*

- **"Simplemente REST"** By Gabriel Fagúndez. TechMeetUp. 23 Noviembre 2013. Disponible en *https://www.youtube.com/watch?v=NXtMM7Wmn8M*

- Rafael Navarro Marset. REST vs Web Services. **Modelado, Diseño e Implementación de Servicios Web 2006-07**. ELP-DSIC-UPV. *http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf*

- The Java EE 6 Tutorial. Chapter 18 - **Introduction to Web Services.** Oracle. January 2013. *https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf*

Bibliografía

Java Message Service

- **Introducción a Java Message Service**. Departamento de Ciencia de la Computación e Inteligencia Artificial. Universidad de Alicante.
  http://expertojava.ua.es/j2ee/publico/mens-2010-11/sesion01-apuntes.html

- **Introducing the Java Message Service**. Willy Farrel. ibm.com/developerWorks
  http://www.ibm.com/developerworks/java/tutorials/j-jms/j-jms-updated.html

- **Java Message Service Concepts**. Java Platform, Enterprise Edition: The Java EE Tutorial. Chapter 45. Oracle Java Documentation.
  http://docs.oracle.com/javaee/7/tutorial/partmessaging.htm#GFIRP3

- **Package javax.jms** *http://docs.oracle.com/javaee/7/api/javax/jms/package-summary.htm*