Arquitecturas Distribuidas

TEMA 5. Otras arquitecturas distribuidas

I. Invocación remota y middleware
II. Objetos distribuidos y CORBA
III. Otros entornos de objetos distribuidos
IV. Web Services

I. Invocación remota y middleware

1. Revisión de conceptos de comunicación entre procesos (Tema 1)
2. Llamada a procedimiento remoto
    1. Paso de parámetros
    2. Protocolo de petición-respuesta
    3. Semánticas ante fallos
    4. Interfaz común y generación de delegados

Comunicación entre procesos (I)

- Proceso: instancia de un programa en ejecución
- Hilo *(thread)*: tarea que se ejecuta concurrentemente (en paralelo) dentro un proceso
- Los hilos comparten memoria con el resto de hilos de un proceso
- Distintos procesos dentro de una misma máquina no comparten memoria
- *¿Cómo comparten datos distintos procesos?*
  - *Intercambio de mensajes*
- Comunicación entre procesos: técnicas para el intercambio de información (mensajes) entre procesos (local o remotamente)

Comunicación entre procesos (II)

- Procesos pueden estar en la misma máquina (com. local) o en distintas máquinas (com. remota)
- Mecanismos para la comunicación entre procesos:
  - Archivos
  - Señales
  - Tuberías
  - Colas de mensajes
  - Sockets
  - Otros
- Operaciones básicas de paso de mensajes: *enviar* y *recibir*.
- Características de la comunicación entre procesos:
  - Comunicación síncrona o asíncrona.
  - Destinos de los mensajes (dirección IP, puerto)
  - Fiabilidad y ordenación

Interfaz *Socket*

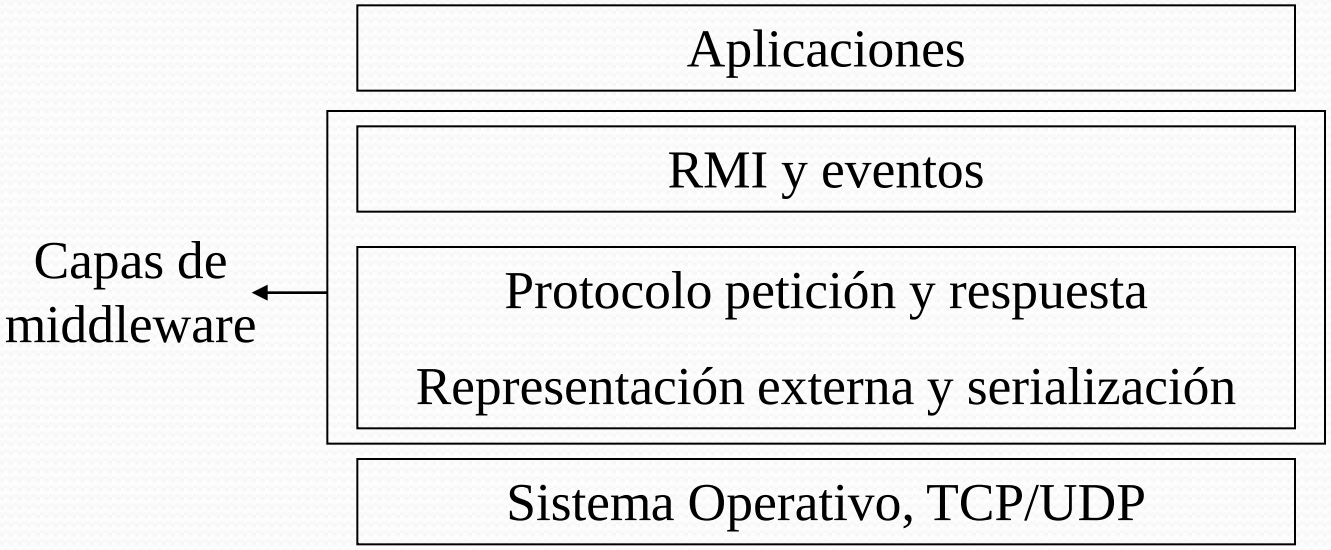
- Interfaz estándar que abstrae las tareas de intercomunicación entre procesos
- *Socket()* crea un conector y devuelve un descriptor de la conexión. Acepta 3 parámetros:
  - Dominio: PF_INET, PF_INET6, PF_UNIX
  - Tipo: Stream (fiable OC), datagrama (no fiable, NOC), otros
  - Protocolo: por defecto TCP para stream y UDP para datagrama
- Operaciones:
  - Servidor: socket*(), bind(), listen(), accept(), write(), read(), close().*
  - Cliente: *socket(), connect(), write(), read(), close()*
  - Permiten el transporte de datos sobre canales fiables o no fiables
- Normalmente la implementación de las operaciones la proporciona el propio S.O.

Llamada a procedimiento remoto (LPR)

- Extensión del modelo de programación para aplicarlo a programas distribuidos: invocación de operaciones en distintos procesos en la misma o distintas máquinas.
  - Un proceso invoca una función que se ejecuta en otro proceso
- Llamada a procedimiento remoto (*Remote Procedure Call*, RPC) = Invocación a un método remoto (*Remote Method Invocation, RMI),* cuando se utiliza POO.
  - Un objeto instanciado en un proceso invoca a un método de un objeto instanciado en otro proceso (local o remoto).
  - La capa RMI permite estas operaciones de manera *transparente* al programador => **Abstracción de las operaciones realizadas**
  - Necesita *Protocolo de aplicación+representación externa de datos + interfaz de descripción común*

Middleware

- Capa software que abstrae las operaciones de paso de mensajes entre procesos, permitiendo la invocación remota y notificación de eventos
  - Hay otras definiciones de *middleware* más genéricas



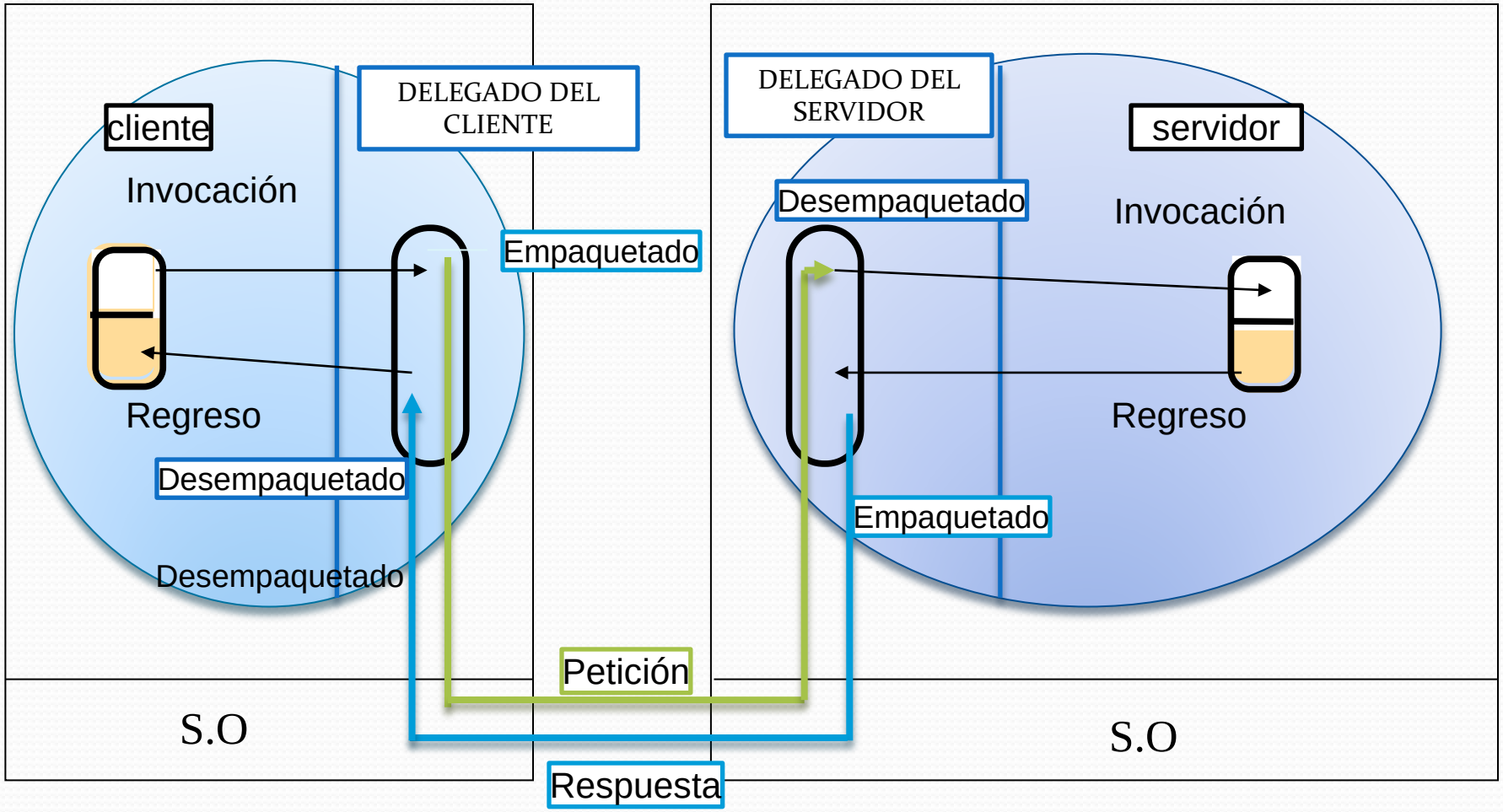
Llamada a procedimiento local

- Mecanismo de invocación local de una función o procedimiento
  - double c = modulo(int elementos, double* v)
- Implica el paso de parámetros
  - Paso por valor o por referencia, depende del lenguaje de programación
- El código de la función así como los parámetros se colocan en la pila de memoria dinámica
- La ejecución del flujo de programa se suspende hasta que la función devuelve el control.

Llamada a procedimiento remoto

- Se pretende que la RPC sea lo más parecida posible a la operación local
  - Transparencia en la operación
  - El programador no tiene por qué saber ni siquiera en qué máquina se ejecutó la operación (en la práctica no es tan transparente)
  - double c = modulo(int elementos, double* v)
- La función *modulo* no es la misma que antes sino una *función delegada* que abstrae (oculta) el proceso de intercambio de mensajes entre procesos
- Es necesario además resolver una serie de problemas que no se dan en una llamada a procedimiento local
  - ¿Qué ocurre si las máquinas no son idénticas?
  - Se pueden producir diversos tipos de errores durante el intercambio de mensajes que en una llamada a procedimiento local no se producen, ¿qué mecanismos podemos utilizar para recuperarlos o evitarlos?

Esquema LPR


cliente
DELEGADO DEL CLIENTE
DELEGADO DEL SERVIDOR
servidor
Invocación
Empaquetado
Desempaquetado
Invocación
Regreso
Desempaquetado
Empaquetado
Regreso
Desempaquetado
Petición
S.O
S.O
Respuesta

Pasos de una llamada a procedimiento remoto

1. El objeto cliente llama al delegado de modo convencional (operación local)
2. El delegado empaqueta el mensaje en la forma adecuada
3. El delegado invoca el protocolo de petición-respuesta
4. El protocolo de petición-respuesta utiliza la pila de comunicaciones del SO para enviar el mensaje
5. El SO remoto recibe el mensaje mediante su pila de comunicaciones
6. El delegado del servidor recibe el mensaje y lo desempaqueta
7. El delegado del servidor invoca la operación en el objeto servidor
8. El objeto servidor realiza la operación y devuelve un resultado
9. El delegado del servidor empaqueta el mensaje e invoca la respuesta del protocolo petición-respuesta
10. La respuesta se envía utilizando la pila de com. del SO.
11. El delegado del cliente recibe la respuesta
12. El delegado desempaqueta la respuesta y se la pasa al objeto cliente

Paso de parámetros en LPR

- El paso de parámetros debe ser también transparente
- Los tipos primitivos no se representan de la misma forma en todas las máquinas
  - Enteros dependen de la arquitectura (32, 64 bits)
  - Ordenación de enteros (little endian, big-endian)
  - Codificación de caracteres (ASCII, UNICODE)
- **Empaquetado y desempaquetado** consiste en ensamblarlos de modo adecuado para la transmisión de un mensaje
  - Formato externo acordado de antemano (“forma canónica”)
  - Ineficiente si las máquinas son iguales. Sol: se envía un campo adicional para indicar si se envía en formato máquina o forma canónica

Serialización de objetos

- Un objeto es un tipo de datos estructurado: contiene tipos primitivos y otros objetos
- La serialización consiste en representar de forma adecuada un objeto o conjunto de objetos para que pueda ser almacenado en disco o enviado en un mensaje.
  - Se escribe la información de la clase además de los tipos de los campos y los nombres de los campos
  - Si un objeto contiene referencias a otros objetos, éstos deben serializarse también de manera que puedan reconstruirse en el destino
- El paso de parámetros en las LPR se hace siempre por valor, no tiene sentido pasar referencias.
  - El delegado se encarga de restaurar la referencia

Protocolo petición-respuesta

- Protocolo de **nivel de aplicación** para implementar LPR o RMI
- Se implementa sobre el nivel de transporte (TCP habitualmente)
- HTTP y SOAP (Tema 5-IV) son ejemplos de protocolos petición-respuesta que se pueden utilizar
- Los mensajes incluyen:
  - Tipo
  - Identificador de la petición
  - Referencia al objeto remoto
  - Referencia al método remoto
  - Argumentos (parámetros)

Semánticas ante fallos

- Las operaciones remotas se ocultan al programador, en principio.
- Pero, ¿qué ocurre con los fallos? ¿Se ocultan y se intentan resolver? ¿Se deja al programador esta tarea?
- La naturaleza de las LPR implica una serie de fallos que no ocurren en invocaciones locales:
  1. Cliente no puede localizar al servidor
  2. Pérdida del mensaje de solicitud
  3. Pérdida del mensaje de respuesta
  4. Caída del servidor antes de la recepción de la respuesta
  5. Caída del cliente después de enviar una solicitud

Semánticas ante fallos

- Cliente no puede localizar al servidor
  - Delegado del cliente informa del error
  - Habitualmente se lanza una excepción
    - Programador debe capturarla: pérdida de la transparencia
    - Algunos lenguajes de programación no admiten excepciones
- Pérdida del mensaje de solicitud
  - **Retransmisión**
  - Número límite de retransmisiones: lanzar excepción
- Pérdida del mensaje de respuesta
  - Cliente puede retransmitir la petición
  - Idempotencia: "la operación se puede repetir sin alterar el resultado"
  - Las operaciones no idempotentes no admiten la retransmisión simple. Es necesario incluir un **número de secuencia** para evitar operaciones duplicadas.
  - Y un **buffer con los resultados**. Si el servidor recibe un solicitud duplicada no la ejecuta. Sólo devuelve el resultado.

Semánticas ante fallos

- Caídas del servidor
  - El servidor realiza 3 operaciones: recibe, ejecuta, responde
  - Si se cae antes de la ejecución se puede **retransmitir la petición**
  - Si se cae durante la ejecución se puede lanzar una excepción en el cliente informando de un regreso incorrecto.
  - En ambos casos no hay respuesta
  - ¿Cómo sabe el cliente dónde falló? No puede saberlo
- Soluciones: Filosofía (semántica) de la LPR
  - Reintentar hasta conseguir respuesta: Semántica AL-MENOS-UNA-VEZ
    - Mecanismos: retransmisión sin filtrado de duplicados, re-ejecutar método
  - Darse por vencido de inmediato: Semántica COMO-MUCHO-UNA-VEZ
    - Mecanismos: retransmisión con filtrado de duplicados y retransmisión de respuesta
  - No garantizar nada: Semántica QUIZAS
    - Mecanismos: sin retransmisiones
- Normalmente los sistemas LPR implementan una de estas semánticas. El programador debe implementar adicionalmente los mecanismos que considere necesarios para asegurar la fiabilidad necesaria

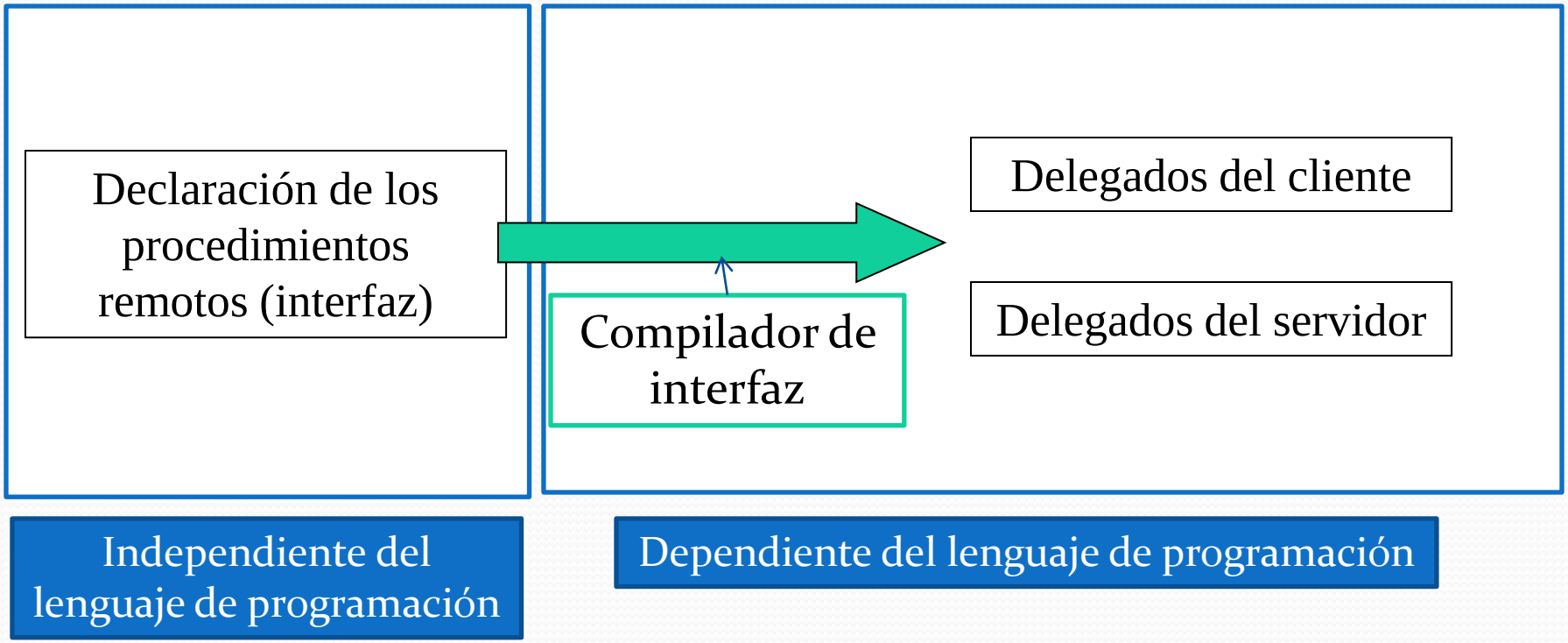
Semánticas ante fallos

- Caídas del cliente
  - El objeto remoto queda en ejecución: desperdicio de CPU y bloqueo de recursos
  - Normalmente el servidor gestiona el “tiempo de vida de los objetos” de alguna forma
- Conclusión: las operaciones remotas no pueden ser completamente transparentes. El programador debe saber si se invoca un objeto local o remoto.

Implementación de RMI

- Se utilizan distintos componentes/módulos funcionales
  - Delegado del cliente o *stub* o *proxy*: módulo de comunicación + módulo de referencia remota
  - Delegado del servidor o *skeleton*: módulo de comunicación + módulo de referencia remota
- Generación de delegados:
  - Se generan automáticamente mediante un programa especial
  - Se parte de una declaración de procedimientos remotos (interfaz) **independiente del lenguaje de programación**
  - Un compilador especial genera a partir de la interfaz los delegados del cliente y servidor **en un lenguaje de programación concreto**

Generación de delegados



Referencias y bibliografía

- Libros:
  - G. Colouris et alt., “Distributed Systems: Concepts and Design”, 3ª edd., Addison-Wesley, 2001, Tema 4,5,11
  - I. Taylor y A. Harrison, “From P2P and Grids to Services on the Web”, 2º Ed., Springer