3. Middlewares

Servicios Telemáticos Avanzados
4º Grado en Ingeniería en Tecnologías de Telecomunicación
Especialidad de Telemática

Indice

1 Middlewares

- La API de sockets BSD no maneja la problemática asociada a la representación de datos transmitidos por la red:
  - Big Endian/Little Endian de las máquinas
  - El tamaño de palabra de la arquitectura (32 o 64 bits)
  - La forma de estructuras de datos complejas, con diferencias en padding y alineamiento según la máquina
  - Falta de gestión de mensajes de control que permitieran por ejemplo mandar cierta tarea a una rutina en el equipo remoto
  - Necesario proceso de serialización: empaquetar los datos en forma adecuada para su transmisión por la red.
    - Marshaling: equivalente a serialización aplicada a objetos con estado.

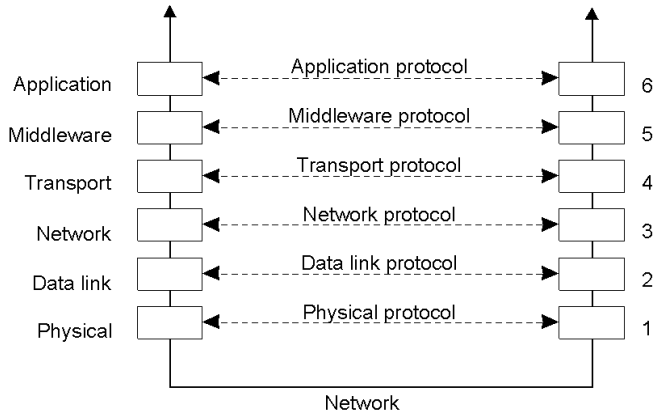
Middlewares

- *Middleware* es una capa software que se encuentra entre el sistema operativo (nivel transporte, sockets) y las aplicaciones
  - Ofrece una facilidad de comunicación que oculta el paso de mensajes a bajo nivel entre las redes de ordenadores
  - Oculta la heterogeneidad de máquinas
  - Implementa serialización y Marshaling de objetos
  - Permitir reutilizar facilidades
    - Por ejemplo, FTP y HTTP transfieren ficheros ¿Por qué no reutilizar esa parte?
- Facilidades típicas de un middleware:
  - Servicio de nombres: identificación de servicios
  - Ejecución remota
  - Acceso a datos distribuidos
  - Persistencia de datos
  - Transacciones: múltiples operaciones de manera atómica
  - Seguridad: encriptación, autenticación

Middlewares

- Desventajas
  - Pérdida de eficiencia en la transmisión de datos
    - Mayor carga de señalización (cabeceras de protocolos y paquetes de control)
    - Transferencias masivas de datos se deben implementar directamente sobre el API de sockets BSD (HTTP, FTP, etc.) en lugar de usando middlewares si se quieren hacer de forma eficiente
  - Pérdida de control
    - Sobre los paquetes transmitidos por la red y la información intercambiada

Protocolo Middleware

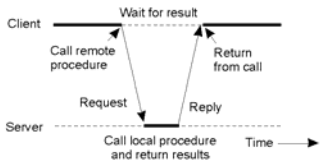


Modelos de Middleware

- Llamada a procedimientos remotos (RPC)
- Objetos distribuidos (ROI, RMI)
- Comunicación orientada a mensajes (MOC)
- Comunicación orientada a streams (streaming)

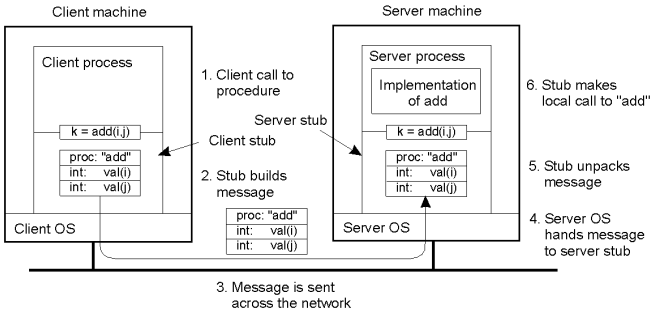
2 Remote Procedure Call (RPC)

- Permite llamar a procedimientos (líneas de código) localizados y ejecutados en otra máquina.
- Cuando una máquina A llama a un procedimiento en la máquina B, el proceso llamante en A es suspendido y tiene lugar la ejecución del procedimiento en B. Los resultados se retornan a A.
- El funcionamiento es transparente a la aplicación, incluso puede no saber que el procedimiento se está ejecutando en otra máquina.
  - La llamada en A se sustituye por un **client stub** que transforma la llamada en un mensaje que se envía al servidor. Se quedará bloqueado hasta que reciba la respuesta.
  - El mensaje que llega a B es atendido por un **server stub** que lo transforma en una llamada a un procedimiento local. El resultado se manda como mensaje de vuelta a A.



RPC

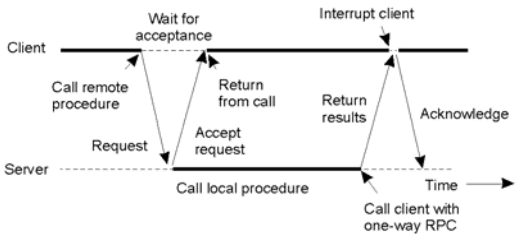
- Paso de parámetros
  - Mismo convenio de formatos y representación de datos independiente de la máquina, y unificación Big Endian y Little Endian.
  - Punteros o parámetros por referencia: se pasa copia de la zona de memoria al servidor y se copia el resultado de nuevo en la misma zona original del cliente.



RPC

- Los procedimientos (funciones, servicios) disponibles en el servidor se especifican mediante Interface Definition Language (IDL) que se utiliza en tiempo de compilación.

- Posibilidad de RPC asíncrono

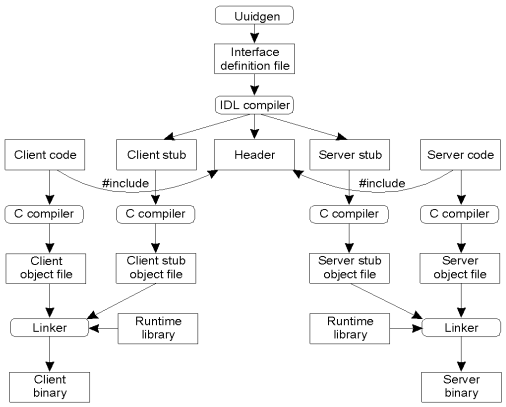


RPC

- Implementaciones RPC
  - DCE (Distributed Computing Environment), desarrollado por la Open Software Foundation (OSF) ahora llamada The Open Group.
  - SunRPC: NFS
- DCE-RPC
  - Inicialmente desarrollado para UNIX se encuentra para todas las plataformas.
  - Principalmente implementado a nivel de usuario; sólo el sistema distribuido de ficheros a nivel de kernel. Funcionalidades principales:
    - Servicio básico de ejecución remota de procedimientos
    - Servicio de ficheros distribuidos
    - Servicio directorio: localizar recursos como servidores, impresoras, datos…
    - Servicio de seguridad
    - Servicio de reloj distribuido: sincroniza los relojes

Pasos de desarrollo en DCE-RPC

- A la definición del interfaz IDL se le asigna un identificador único en base a la localización e instante de creación, que servirá luego para mandar peticiones al servidor correcto.



RPC en pseudocódigo

```
//your client code
result = function(parameters)

//client side stub
function(parameters) {
    address a = bind("function");
    socket s = connect(a);
    send(s,"function");
    send(s,parameters);
    receive(s,result); //blocking
    return result;
}
```

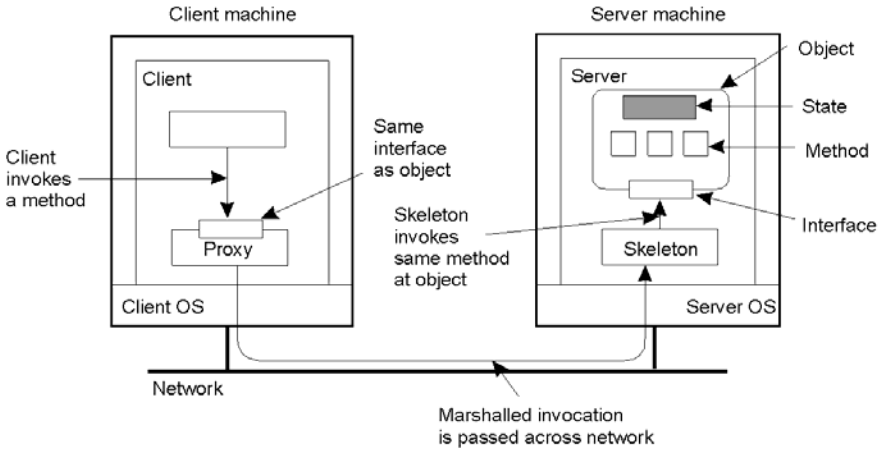
```
//rpc server main loop
void rpc_server() {
    register("function",address);
    while (true) {
        socket s = accept(); //blocking
        receive(s,id);
        if (id == "function")
            dispatch_function(s);
        close(s);
    }
}

//server side stub
void dispatch_function(socket s) {
    receive(s,parameters);
    result = function(parameters);
    send(s,result);
}
```


3 Remote Object/Method Invocation (ROI/RMI)

- Aplica la idea de RPC a la invocación de objetos remotos.
  - Un objeto encapsula su estado (datos), métodos (operaciones sobre esos datos) y ofrece estos métodos a través de un interfaz
  - Permite colocar el objeto en una máquina y el interfaz en otra
    - El objeto puede tener el estado distribuido o no entre varias máquinas: objeto distribuido
  - Equivalentes a los stub de las RPCs:
    - Proxy: equivalente al stub cliente
    - Skeleton: equivalente al stub servidor

ROI/RMI



ROI/RMI

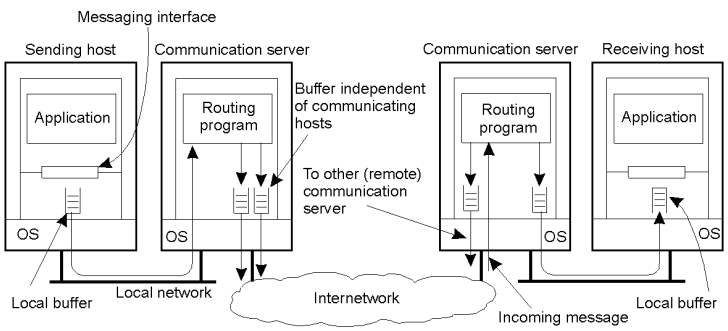
- Referencia al objeto:
  - (IP servidor, puerto servidor, identificador del objeto)
- Estrategias en la definición de los objetos:
  - Objetos en tiempo de compilación: Java, C++, etc. Exige toda la programación en el mismo lenguaje.
  - Objetos en tiempo de ejecución: permiten combinar cualquier lenguaje de programación haciendo uso de adaptadores de objetos (*wrapper* que provee el interfaz esperado).
- Tipos de objetos según su persistencia:
  - Persistentes: continúan existiendo en un sistema de almacenamiento secundario cuando no se encuentran en espacio de memoria del servidor.
  - Transitorios: sólo existen mientras se mantenga el servidor activo.

ROI/RMI

- ROI/RMI estático/dinámico:
  - Estático: se necesita conocer los interfaces de los objetos en tiempo de compilación. Ej: objeto.append(int)
  - Dinámico: en tiempo de ejecución se puede seleccionar el método a ejecutar en el objeto sin conocerlos a priori en tiempo de compilación. Ej: invoke(objeto,id(append),int)
- Implementaciones:
  - DCE-RMI: utiliza referencias a identificadores de procedimientos remotos (métodos) en un objeto de determinado servidor.
  - Java RMI: la más utilizada.
    - Los objetos distribuidos han sido integrados en el lenguaje.
  - CORBA: definiciones multilenguaje.

4 Comunicación orientada a mensajes (MOC)

- RPCs y RMIs son síncronos por naturaleza.
- Comunicación orientada a mensajes:
  - Los mensajes se pueden almacenar en los servidores de comunicación intermedios o en el destino.
  - Útil en escenarios con conectividad intermitente o de mala calidad

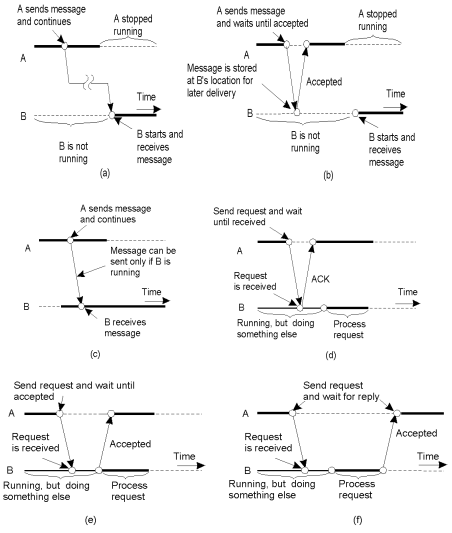


MOC

- Tipos de comunicación con mensajes:
  - Servicio <u>persistente</u>: los mensajes se almacenan hasta que se puedan entregar al destino. Símil correo electrónico.
    - Necesario para aplicaciones que interconectan redes dispersas a gran escala.
    - Soluciones más sencillas de recuperación de errores (el retardo no es crítico).
  - Servicio <u>transitorio</u>: los mensajes se almacenan sólo mientras la aplicación emisora y receptora estén activas.
  - <u>Asíncrona</u>: el emisor continua inmediatamente después de haber enviado el mensaje.
  - <u>Síncrona</u>: el emisor se bloquea hasta recibir confirmación de recepción por el destino o incluso hasta la recepción de la respuesta.
    - <u>Basada en recibo</u>: el emisor espera a que el mensaje llegue al receptor y se almacene en su buffer.
    - <u>Basado en envío</u>: el emisor espera a que el mensaje se entregue al servidor del receptor y éste lo acepte.
    - <u>Basada en respuesta</u>: el emisor espera a la respuesta tras el procesado en el receptor.

MOC

a) Asíncrono persistente
b) Síncrono persistente basado en recibo
c) Asíncrono transitorio
d) Síncrono transitorio basado en recibo
e) Síncrono transitorio basado en envío
f) Síncrono transitorio basado en respuesta



MOC

- MOC transitorio, implementaciones:
  - Message-Passing Interface (MPI)
    - Es el estándar de facto para comunicación entre nodos corriendo una aplicación paralelizada.
    - Consiste en una librería disponible para casi todos los lenguajes de programación.
    - No soporta recuperación de situaciones críticas como la caída de procesos o de la propia red.
    - Soporta todos los modos transitorios de comunicación excepto síncrono transitorio basado en recibo.
    - La comunicación MPI se produce entre un grupo de procesos. Origen/destino identificado por (groupID, processID)

MOC

- MOC persistente:
  - Habitualmente denominados message-queuing systems o Message-Oriented Middleware (MOM).
  - Proporcionan capacidad de almacenamiento a medio plazo de mensajes, evitando que el emisor y receptor estén activos durante la transmisión.
  - La transmisión puede llegar a costar minutos en lugar de centenares de milisegundos de los transitorios.
  - Arquitectura basada:
    - Colas: en emisor, receptor y saltos intermedios. Al colocar un mensaje en la cola del emisor se especifica la cola destino.
    - Relays: reenvían mensajes hasta el destino (routers).
    - Brokers: se encargan de la conversión de formatos de los mensajes, cuando es diferente en emisor y receptor.
  - Primitivas de manejo de colas:
    - Put/Get/Poll/Notify(callback function)
  - Implementación: IBM MQSeries para mainframes.

5 Comunicación orientada a streams (streaming)

- Stream: información continua con requerimientos temporales y de ordenación en los datos encapsulados.
  - Ejemplo: señal PCM voz 8bits 8KHz -> 64Kbps
- Tipos de transmisión:
  - Síncrona: los datos se envían a la máxima tasa para que lleguen al destino antes de cuando sean requeridos. Ej: predescarga de contenidos pregrabados.
  - Isosíncrona: los datos lleguen al destino a determinada tasa dentro de ciertos umbrales, de manera que el buffer en recepción es menor. Ej: voz sobre IP (paquetes de 200bytes cada 20ms).
- Un stream puede ser complejo y estar compuesto por varios substreams. Ej: pista video y audio.
- La dependencia temporal exige recursos de calidad de servicio (QoS) en la red: tasa, retardo, y jitter.