

APUNTES DE SISTEMAS DISTRIBUIDOS

Ingeniería del Software



Contenido

[**Tema 1: Sistemas Distribuidos (Introducción)** 2](#_Toc135325242)

[1. MOTIVOS 2](#_Toc135325243)

[2. PROPIEDADES 2](#_Toc135325244)

[3. HARDWARE DE LOS SISTEMAS DISTRIBUIDOS 3](#_Toc135325245)

[4. SOFTWARE DE LOS SISTEMAS DISTRIBUIDOS 3](#_Toc135325246)

[**Tema 2: Arquitectura de los Sistemas Distribuidos** 4](#_Toc135325247)

[1. TIPOS DE ARQUITECTURAS 4](#_Toc135325248)

[2. PARADIGMA CLIENTE-SERVIDOR 4](#_Toc135325249)

[3. TIPOS DE CLIENTES (LIGEROS vs PESADOS) 5](#_Toc135325250)

[4. SISTEMAS ENTRE IGUALES (peer-to-peer) 5](#_Toc135325251)

[5. SISTEMAS EDITOR-SUBSCRIPTOR 5](#_Toc135325252)

[**Tema 3: Comunicación entre Procesos** 6](#_Toc135325253)

[1. INTRODUCCIÓN 6](#_Toc135325254)

[2. MODELO OSI vs MODELO TCP/IP 6](#_Toc135325255)

[3. INTERFAZ DE SOCKETS 7](#_Toc135325256)

[**Tema 4: Llamadas a procedimientos remotos** 7](#_Toc135325257)

[1. INTRODUCCIÓN AL RPC 7](#_Toc135325258)

[2. FUNCIONAMIENTO DE RPC 7](#_Toc135325259)

[**Tema 5: Llamadas a Métodos remotos** 8](#_Toc135325260)

[1. INTRODUCCIÓN A RMI 8](#_Toc135325261)

[2. PROPIEDADES DE RMI 8](#_Toc135325262)

[**Tema 6: Net Remoting** 8](#_Toc135325263)

[1. INTRODUCCIÓN A NET REMOTING 8](#_Toc135325264)

## **Tema 1: Sistemas Distribuidos (Introducción)**

## MOTIVOS

Aumentar la capacidad de procesamiento subiendo la velocidad del procesador es muy costoso. Es mejor interconectar más máquinas, pero requiere compartir recursos con sistemas operativos distribuidos.

**PROGRESO HISTÓRICO (Evolución):**

1. Sistemas por lotes.
2. Tiempo compartido.
3. Teleprocesos.
4. Sistemas personales.
5. Sistemas en Red.
6. Sistemas Distribuidos.

Un **sistema distribuido** es un conjunto de ordenadores independientes que aparentan ser un único sistema. Poseen transparencia de ubicación local o remota.

El **Middleware** es la capa de abstracción intermedia para proporcionar una interfaz común. Sus ventajas y desventajas son:

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| Económico, fiable, flexible, alta disponibilidad, paralelismo, … | Costoso mantenimiento, bajo rendimiento individual, compleja gestión de recursos, … |

Un **sistema ubicuo** es un sistema que no se puede percibir. Ej.: Computador de abordo.

## PROPIEDADES

* **Transparencia:** los recursos parecen de una sola máquina.
* **Escalabilidad:** capacidad de crecer sin afectar al rendimiento.
* **Fiabilidad:** capacidad de realizar las funciones para las que se diseñó en todo momento.
* **Flexibilidad:** capacidad de soportar cambios y actualizaciones.
* **Eficiencia:** capacidad de repartir el trabajo para aumentar las prestaciones.

## HARDWARE DE LOS SISTEMAS DISTRIBUIDOS

**Clasificación de Flynn (núm. instrucciones y núm. datos):**

* **SISD:** 1 instrucción y 1 dato. Von Neumann clásico.
* **SIMD:** 1 instrucción y múltiples datos.
* **MISD:** múltiples instrucciones y 1 dato.
* **MIMD:** múltiples instrucciones y múltiples datos.

Los tipos de arquitectura de **MIMD** son Multiprocesadores y Multicomputadores.

## SOFTWARE DE LOS SISTEMAS DISTRIBUIDOS

Es complicado ofrecer un servicio único cuando los nodos cuentan con Hardware y Software distintos y están en diferentes partes del mundo. La solución es usar **Sistemas Abiertos**.

CARÁCTERÍSTICAS:

* **Interoperabilidad:** capacidad para mover información entre nodos.
* **Transportabilidad de aplicaciones:** capacidad de mover programas (no ejecutables) entre máquinas.
* **Transportabilidad de usuarios:** un usuario puede cambiar de nodo sin tener que conocer sus detalles.

COMUNICACIÓN:

* **Variables compartidas:** Memoria física compartida y modelo de Memoria compartida.
* **Buzones, pipes:** Memoria física compartida con paso de mensajes.
* **RMI:** Memoria física distribuida. Se comporta como si fuera compartida.
* **RPC:** Memoria física distribuida con paso de mensajes.

IMPLEMENTACIÓN DE LA COMUNICACIÓN:

* **Modelo Cliente-Servidor:** el cliente envía un mensaje al servidor solicitando un servicio, el servidor espera peticiones de los clientes y las satisface.
* **Comunicación de grupos:** permite enviar un mensaje a un conjunto de nodos que forman un grupo.

## **Tema 2: Arquitectura de los Sistemas Distribuidos**

## TIPOS DE ARQUITECTURAS

Una **arquitectura** es la estructura fundamental para interconectar hardware. Se clasifican por topología y por estructura:

* **Por topología:**
  + Centralizados: los componentes tienen roles.
  + Descentralizados: no hay roles, todos comparten obligaciones.
* **Por estructura:**
  + Capas: los elementos se especializan en capas.
  + Objetos: los elementos son objetos autónomos que invocan remotamente métodos en una red.
  + Eventos: el sistema tiene una cola donde se agregan las peticiones y se van desplazando en orden.

## PARADIGMA CLIENTE-SERVIDOR

Distribuye una **aplicación** entre dos o más **componentes** **especializados** cuya ejecución se distribuye en uno o más **equipos**.

Se definen dos tipos de **entidades diferenciadas** que se responsabilizan de acciones diferentes: Cliente y Servidor.

* **Cliente:** inicia el diálogo mediante el envío de peticiones.
* **Servidor:** presta servicios y responde las peticiones que recibe.
* **Servicio:** modelo de interacción. Basado en el diálogo Petición ↔ Respuesta.

El Cliente debe de conocer los Servicios que oferta el Servidor. El Servidor es un “Cuello de Botella”, es decir, tiene problemas de escalabilidad. Existen diferentes tipos de conexiones entre Cliente y Servidor:

* **Cliente-Servidor con Cache:** mejora la latencia al poder usar copias locales y reduce el uso de la red y los recursos, lo que ayuda a la escalabilidad.
* **Cliente-Servidor con Proxy:** agrega un intermediario entre el Cliente y el Servidor (Tubería). Se pueden usar tantos proxies como sean necesarios. Hay varios tipos:
  + Forward: proxy en la misma red que el cliente.
  + Reverse: proxy en la misma red que el servidor.
  + Open: proxy no está ni en la red del cliente ni del servidor.
* **Cliente-Servidor Multicapa:** los servidores son clientes de otros servidores. La funcionalidad se divide en varios niveles:
  + 2-Tier: Modelo tradicional.
  + 3-Tier: Extensión del modelo tradicional.
  + N-Tier: Extensión del 3-Tier.

## TIPOS DE CLIENTES (LIGEROS vs PESADOS)

Dependiendo de como se repartan el trabajo el cliente y el servidor:

* **Cliente ligero:** no implementan ninguna lógica, es una interfaz que recoge y presenta los datos. El servidor tiene más trabajo.
* **Cliente pesado:** implementan gran parte de la lógica, procesos o almacenamiento de datos. El servidor es sencillo y tiene poca responsabilidad.
* **Cliente híbrido:** la implementación de la lógica se reparte entre el cliente y el servidor.

## SISTEMAS ENTRE IGUALES (peer-to-peer)

Todos los nodos tienen la **misma funcionalidad**.

Se aprovechan los recursos de todas las máquinas y no hay cuellos de botella, los nodos pueden entrar y salir sin problemas y se autogestiona al no estar centralizado.

La administración es compleja y presenta fallos de seguridad debido a la arquitectura. Son útiles para almacenamiento distribuido, siempre y cuando compartan interfaz.

## SISTEMAS EDITOR-SUBSCRIPTOR

No tiene nada que ver con el sistema Cliente-Servidor.

Los **suscriptores** se suscriben por intereses y lo **editores** generan eventos. Estos eventos son enviados a los suscriptores. Hay varios tipos de suscriptores:

* **Push:** el suscriptor recibe eventos.
* **Pull:** el suscriptor pregunta si hay eventos.
* **Híbrido:** el suscriptor recibe los eventos, pero tiene que solicitarlos previamente.

## **Tema 3: Comunicación entre Procesos**

## INTRODUCCIÓN

La **comunicación** permite que los procesos colaboren, se realiza mediante el paso de mensajes. Todos los mecanismos cuentan con las siguientes primitivas:

* **Enviar** datos a otro proceso
* **Recibir** datos del emisor.
* **Iniciar conexión:** emite una petición de conexión (Cliente).
* **Esperar conexión:** escucha a la espera de conexiones entrantes (Servidor).
* **Aceptar conexión:** acepta las conexiones entrantes (Servidor).

Los tres últimos solo se realizan en los sistemas basados en conexión.

Tipos de primitivas:

* **Bloqueantes:** el proceso que la invoca espera hasta que la primitiva termine completamente.
* **No Bloqueantes:** el proceso que la invoca continua sin esperar a que la primitiva termine completamente.

Tipos de comunicación:

* **Comunicación síncrona:** enviar y recibir son bloqueantes. La sincronización está garantizada. Fácil de usar, pero mal rendimiento.
* **Comunicación asíncrona:** enviar es no bloqueante. Mejora el rendimiento y evita bloqueos indefinidos, aunque Recibir suele bloquearse.

Es recomendable poner las operaciones bloqueantes en un hilo aparte.

## MODELO OSI vs MODELO TCP/IP

|  |  |
| --- | --- |
| **Modelo OSI** | **Modelo TCP/IP** |
| **Aplicación**  **Presentación**  **Sesión**  Transporte  Red  **Enlace**  **Física** | **Aplicación**  Transporte  Red  **Acceso a la Red** |

## INTERFAZ DE SOCKETS

API a nivel de transporte que permite la conexión de procesos en un punto común. Es parte del Kernel de un SO.

* **Protocolo TCP:** fiable, orientado a la conexión (Stream).
* **Protocolo UDP:** no fiable, no orientado a la conexión sino a la velocidad de conexión (Datagrama).

Un **Socket** consta de una dirección IP y un puerto. No son compartidos y la implementación suele ser asíncrona.

## **Tema 4: Llamadas a procedimientos remotos**

## INTRODUCCIÓN AL RPC

La llamada a procedimientos remotos (RPC) se implementa sobre Streams, abstrae los detalles de la conexión y es un entorno que genera código de apoyo automáticamente. Permite:

* Aplanamiento de los datos.
* Desplazamiento de los datos.
* Formato de los datos.
* Gestión de diálogos por el socket.

## FUNCIONAMIENTO DE RPC

**Cliente:**

Cuando el proceso llama a una función:

Se envía un mensaje con ID de función y los argumentos.

Al recibir la respuesta desempaqueta el valor devuelto y lo pasa al proceso.

**Servidor:**

Cuando recibe el ID de función y los argumentos:

Invoca a la función con ID correspondiente.

Envía un mensaje al cliente con la respuesta.

## **Tema 5: Llamadas a Métodos remotos**

## INTRODUCCIÓN A RMI

**RMI** sigue el mismo objetivo que RPC, pero integra la orientación a objetos.

Permite llamadas a métodos de objetos remotos y paso de referencias a objetos remotos de forma transparente.

## PROPIEDADES DE RMI

Los clientes solo interactúan con las interfaces del servicio, nunca con las clases que lo implementan. A estas referencias sólo se las puede castear.

Los objetos locales solamente funcionan en la JVM que los creó, por eso los parámetros y los valores devueltos se pasan clonándolos binariamente y sus referencias también. Toda la serialización es transparente.

## **Tema 6: Net Remoting**

## INTRODUCCIÓN A NET REMOTING

Es una solución para que los objetos de distintos dominios de aplicación puedan comunicarse, aunque estén en máquinas distintas.