# Final Sistemas Distribuidos

## Introducción

Definición: Sistema en el cual los componentes de hardware o software, localizados en computadoras unidos mediante red, comunican y coordinan sus acciones sólo mediante paso de mensajes. Ejemplos: internet, juegos online, redes corporativas, búsqueda web…

Características:

- Concurrencia de los componentes
- No hay un reloj global
- Fallos independientes de los componentes

Retos de los sistemas distribuidos:

- **Heterogeneidad:** se soluciona con middleware o código móvil.
- **Extensibilidad:** el sistema puede ser extendido y re-implementado en varias maneras. Un sistema es **abierto** si se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición del cliente. Algunas características de los sistemas abiertos son: interfaces públicas, mecanismos de comunicación estandarizados, software y hardware heterogéneo.
- **Seguridad:** Confidencialidad, Integridad, Disponibilidad
- **Escalabilidad:** eficiencia al aumentar tanto en número de usuarios como de recursos.
- **Manejo de fallos:** detección, enmascaramiento, tolerancia, recuperación y redundancia
- **Concurrencia:** recursos compartidos que pueden ser accedidos al mismo tiempo por múltiples usuarios.
- **Transparencia:** debe verse por el usuario final como una sola entidad.
- **QoS**

Modelos de sistemas distribuidos:

1. **Modelos físicos:**

Representación del hardware subyacente de un SD en términos de computadoras (y otros dispositivos) y su red de interconexión.

2. **Modelos arquitectónicos:**

Describen a los SD en términos de las tareas computacionales y tareas de comunicación realizadas por sus componentes. Estructura en términos de componentes separados y sus interrelaciones.

| Entidades comunicantes | |
|---|---|
| Perspectiva de sistema | Perspectiva de programación |
| Procesos (hilos), Nodos (ej. sensores) | Objetos, componentes, servicios WEB |

Paradigmas de comunicación:

Comunicación entre procesos (Sockets), Invocación remota (RMI,HTTP,RMC) y comunicación indirecta.

3. **Modelos fundamentales:**

Son 3 submodelos principales:

- Modelo de interacción: depende del desempeño de los canales de comunicación y de la imposibilidad de mantener una noción global del tiempo. El desempeño del modelo se mide en base a la latencia, el ancho de banda y el jitter.

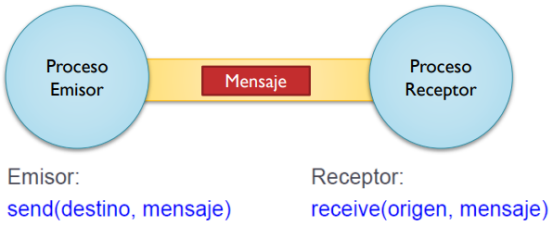
| SD síncronos | SD asíncronos |
|---|---|
| - Tiempo de ejecución de cada paso tiene límites inf y sup<br>- Cada mensaje se recibe en un tiempo limitado conocido<br>- Cada proceso tiene un reloj local (basado en un reloj real) | - Tiempo de ejecución sin límites<br>- La latencia puede ser arbitraria<br>- Valores de relojes locales arbitrarios. |

- Modelo de fallo: Fallos por omisión, de temporización y fallos arbitrarios (bizantinas).
- Modelo de seguridad: Proteger los objetos de los procesos y los canales de comunicación.

**Hilos:** Pequeñas secuencias de instrucciones que pueden ser ejecutadas por un sistema operativo. Es una tarea que puede ser ejecutada al mismo tiempo que otra.

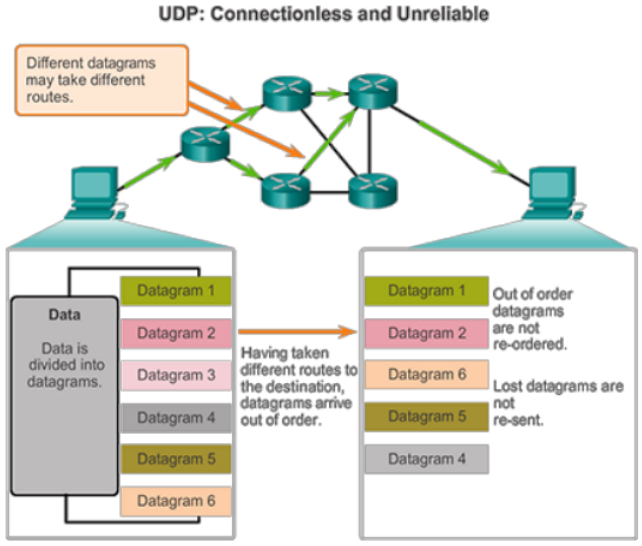
**Condición de carrera:** Se produce cuando dos subprocesos tienen acceso a una variable compartida al mismo tiempo y se ejecutan en un orden distinto de lo esperado.

# Comunicación entre procesos



La comunicación puede ser síncrona o asíncrona. En la síncrona, las operaciones de send() y receive() son bloqueantes. En la asíncrona, send() es no bloqueante, receive() puede o no serlo.

**Sockets:** Concepto abstracto que sirve para enviar y recibir datos en una red. Cada socket está asociado a un puerto local y una dirección IP en la computadora en la que está corriendo.




El proceso de recepción debe de especificar un arreglo de bytes en el cual se recibirá el mensaje. Se puede definir un tiempo límite de espera (timeout).

**TCP:** Comunicación por flujos orientado a conexión. La abstracción de flujos tiene las siguientes característica:

- La aplicación puede elegir la cantidad de datos que quiere escribir o leer del stream.
- Los mensajes perdidos son detectados vía acuse de recibo y son reenviados.
- Control de flujo, ajusta las velocidades de los procesos que leen y escriben.
- La duplicación y el ordenamiento son controladas vía identificadores.
- Los procesos establecen una conexión para comunicarse.
- Se utilizan colas para mandar los mensajes.
- Los servidores crean hilos por cada cliente.

**Comunicación Multicast:** comunicación en grupo, de un proceso a un grupo de procesos. Su utilidad está en la tolerancia a fallos en servicios replicados, descubrimiento de servicios, propagación de notificaciones y mejor rendimiento a través de datos replicados.
Los grupos son dinámicos y sólo está disponible vía UDP. Mensajes con TimeToLive.

**Representación externa de los datos:** Los datos en los programas tienen distintas estructuras, para transmitirse se tienen que convertir a una secuencia de bytes. Sin embargo, diferentes dispositivos tienen distintas formas de almacenar los datos. Entonces:

- Se utiliza una representación externa de datos para su transmisión.
- O se transmiten los datos en el formato del emisor y se indica cuál se usó.

El empaquetamiento es el proceso que toma una colección de datos y los ensambla de un modo adecuado para su transmisión.

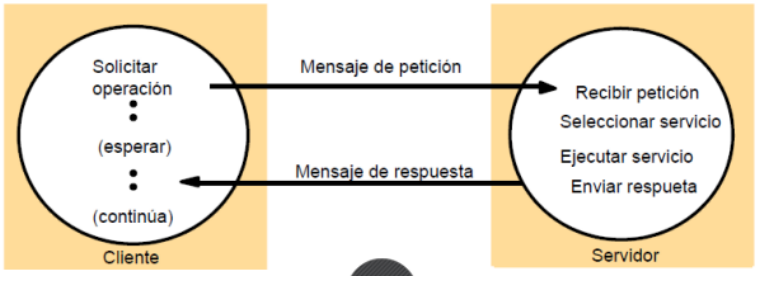
Tipos de representación externa de datos:

1. CORBA (Common OBject Request Broker Architecture)
2. Serialización de objetos en Java
3. XML (Extensible Markup Language)
4. Json (JavaScript Object Notation)

# Invocación Remota

**Protocolos de Request Reply (HTTP):**

**Llamada procedimientos remotos (RPC):**
Procedimientos en procesos en computadoras remotas pueden ser llamados como si fueran procedimientos locales. Ofrece una interfaz sencilla para construir aplicaciones distribuidas. Hay transparencia tanto de acceso como de ubicación.




Tiene 2 participantes: Un cliente activo que envía una RPC al servidor. Un servidor pasivo (esclavo) que calcula un resultado y lo devuelve al cliente.

**Stubs:** Responsables de convertir los parámetros de la aplicación cliente/servidor durante una RPC (se generan automáticamente por el software de RPC).

Protocolo básico:

- Proceso cliente: Se conecta a un servidor binding, invoca una llamada al procedimiento remoto. El stub del cliente se encarga de empaquetar los parámetros y construir los mensajes, enviar los mensajes al servidor y bloquearse hasta esperar la respuesta.
- Proceso servidor: Registra las RPC, implementa los procedimientos y envía la respuesta al cliente. El stub del servidor se encarga de recibir la petición del cliente, desempaquetar los parámetros, invocar el procedimiento de manera local y bloquearse hasta esperar la respuesta.

Se utiliza una programación con **interfaces**, la cuál es compartida por cliente y servidor. El formato de los procedimientos remotos se especifica por medio de un IDL. Cada interfaz específica: Nombre de servicio, nombre de los procedimientos, parámetros de los procedimientos y tipos de datos de los argumentos.

El enlace **binding** es una asociación entre cliente y servidor, implica localizar al servidor que ofrece un determinado servicio. El servidor debe registrar su dirección en un servicio de nombres.

Fallos en las RPC: El cliente podría no ser capaz de localizar al servidor, pérdidas de mensajes, el servidor falla después de recibir una petición o el cliente falla después de mandar una.

**Invocación de métodos remotos (RMI):**
Es un modelo equivalente a las llamadas a procedimientos remotos. Modelo orientado a objetos sobre aplicaciones distribuidas. Los objetos proporcionan métodos, los cuales dan acceso a los servicios. Comparación de RMI sobre Sockets:
Ventajas:

- Los programas RMI son más sencillos de diseñar.
- Servidor RMI concurrente.

Inconvenientes:

- Sockets tienen menos sobrecarga
- “RMI sólo para plataformas Java”

Tanto RMI como RPC tienen interfaces, están basados en protocolos de petición-respuesta y son transparentes. Pero RMI está orientado hacía POO, y tienen referencias a los objetos como parámetros.

# Comunicación indirecta

Comunicación entre entidades en un SD a través de un intermediario sin acoplamiento directo entre emisores y receptores. Esto facilita la adaptación a cambios, pero introduce retrasos. Hay **desacoplamiento espacia**l, porque el emisor no necesita conocer al receptor y viceversa; y hay **desacoplamiento temporal** porque los participantes pueden tener líneas de existencias independientes. Se utiliza principalmente en Sds donde se anticipan **cambios**: Ambientes altamente volátiles o diseminación de eventos donde los usuarios pueden cambiar y/o son desconocidos.

**Memoria distribuida compartida**: abstracción utilizada para el intercambio de datos entre equipos que no comparten memoria física. Los datos almacenados se consideran persistentes y no hay pasos de mensajes entre procesos.

**Comunicación grupal:** El mensaje se manda a un grupo y se difunde a todos los miembros del mismo. El emisor no conoce a los receptores y los miembros pueden unirse y abandonar el grupo. Es una abstracción respecto a multicast.
Puede haber grupos abiertos y cerrados, con y sin sobrelapamiento. Hay fiabilidad, es decir garantías en la entrega que los miembros se ponen de acuerdo sobre los mensajes que reciben y en su orden. Se define mediante integridad, validez y acuerdo. Hay distintos tipos de ordenamiento: Total (todos los mensajes son entregados en el mismo orden en todos los

nodos), FIFO (Mensajes de la misma fuente llegan en el orden que fueron enviados) y causal (generaliza la entrega FIFO a un escenario en el que los mensajes se entregan en el orden causalidad, causal implica FIFO).

**Comunicación publicación-inscripción:** SDs basados en eventos. Los editores publican eventos estructurados a un servicio. Los suscriptores expresan interés por un tipo particular de eventos. El servicio casa las suscripciones con los eventos publicados y asegura el envío correcto de notificaciones de eventos. Un evento se envía a un conjunto de suscriptores.
Propiedades de este tipo de comunicación:

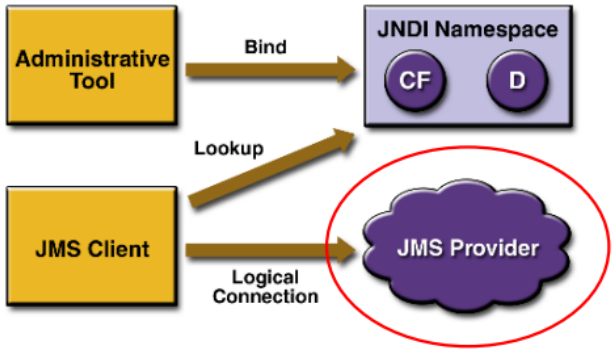
- **Heterogeneidad:** Facilita que componentes variados trabajen juntos por medio de interfaces de recepción de eventos.
- **Asincronía:** Las notificaciones se envían asíncronamente a los suscriptores, así se evita que estén acoplados.

El editor puede publicar y notificar eventos, y anunciar filtros. El suscriptor se puede suscribir y cancelar la suscripción con filtros. Los filtros pueden estar basados en:

- Canales: los sucesos se publican en un canal, y los suscriptores se suscriben al canal.
- Asunto o tema: es uno de los campos del suceso o evento, y la suscripción se define en base a este dato.
- Contenido: condición lógica o query sobre los valores de los atributos del evento.
- Tipos: tipos de eventos compatibles con un tipo de filtro.

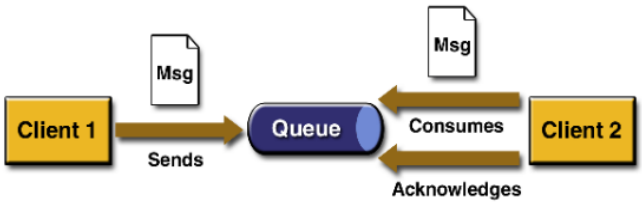
**Colas de mensajes:** Este concepto fundamenta la indirección. Proporciona desacoplamiento espacial y temporal. Se usa para integrar aplicaciones o como base de sistemas de procesamiento de transacciones.
Modelo de programación sencillo:
Los emisores mandan mensajes a las colas y los receptores leen mensajes de la cola.
Hay 3 estilos de receptores: recepción bloqueante, recepción no bloqueante y notificación. En el último se manda un evento cuando llega un mensaje.
Las colas tienen un orden FIFO o por prioridades, se pueden seleccionar los mensajes por sus características ya que tienen un destino metadatos y cuerpo.

Los mensajes en colas son persistentes, es decir que se almacenan indefinidamente, hasta que se lean. Se garantiza entonces un envío fiable, pero no cuando se hace.

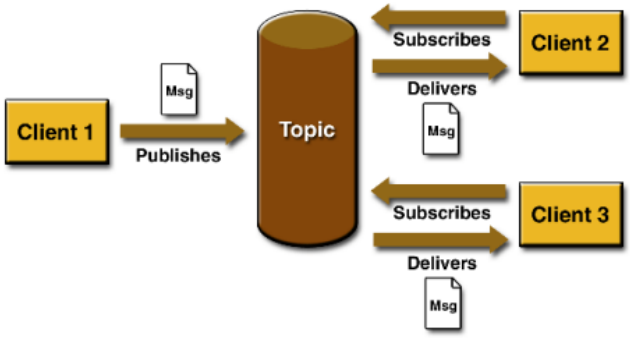



Java Message Service:

El proveedor JMS implementa las interfaces, los clientes producen y consumen mensajes. JNDI es Java Naming and Directory Interface. Las herramientas administrativas vínculan los destinos con las fábricas de conexión usando el directorio JNDI.




Mensajes punto a punto: cada mensaje solo tiene un consumidor, no hay dependencias de tiempo entre emisor y receptor y el receptor confirma la lectura del mensaje.




Comunicación publicación-inscripción: Cada mensaje puede tener múltiples consumidores. Productores y consumidores tienen dependencias de tiempo controladas por la suscripción.

# Programación Web

**JavaScript:** Es un lenguaje script de la WEB, interpretado en tiempo de ejecución (en lugar de ser compilado). Utilizado del lado del cliente, se puede insertar en páginas HTML y es compatible con múltiples navegadores.

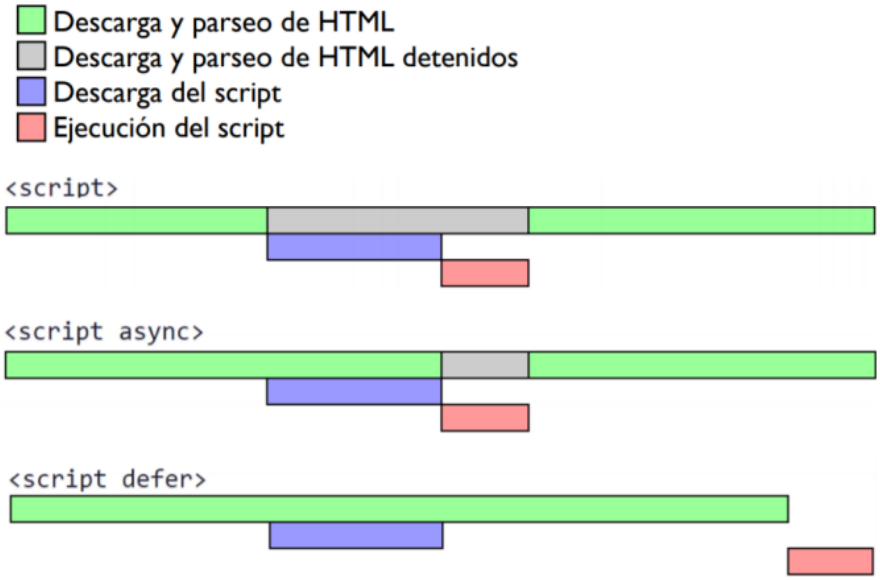
```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <script>
            document.write("<p>This is my first line using JavaScript</p>");
        </script>
    </body>
</html>
```

Se pueden insertar scripts en el body o head.

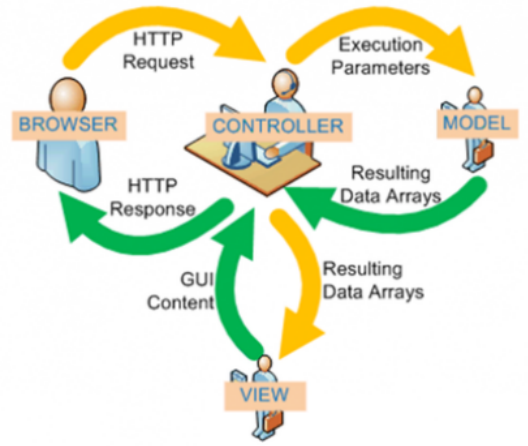
Hay diferentes scopes en JavaScript para definir variables:

| variable | global |
|---|---|
| var variable | función |
| let variable | Bloque {} |

Hay diferentes órdenes de ejecución de scripts:




**Django:** Es un web framework escrito en Python. Utiliza un modelo vista controlador. El controlador es un intermediario lógico entre los datos y la interfaz.




Un sitio web en Django está conformado por una o múltiples apps. Cada app tiene un modelo (datos) y páginas web (vista).

# Servicios WEB

Las aplicaciones Web tienen varios problemas: diversas tecnologías como applets, desarrollos muy ad-hoc.
La idea de los servicios Web es adaptar el modelo de programación web (débilmente acoplado) para su uso en aplicaciones no basadas en un navegador. El objetivo principal es ofrecer una

plataforma para construir aplicaciones distribuidas utilizando un software que enmascara la heterogeneidad.

**Servicios Web RESTful:** REpresentational State Transfer No es un estándar, usa métodos de HTTP. Cada operación es identificada por un único URL y los mensajes no requieren ser empaquetados en “sobres”, lo que lo hace más rápido que SOAP. Los servicios web RESTful regresan datos en XML, JSON, HTML y texto plano. En cambio SOAP solo regresa en XML.

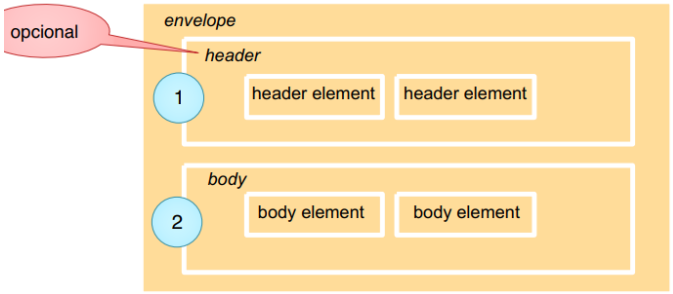
**Servicios Web SOAP:** Estandarización controlada por W3C. Es una colección de protocolos y estándares abiertos que sirven para intercambiar datos entre aplicaciones. Estos servicios pueden estar escritos en distintos lenguajes de programación, ejecutarse en distintos sistemas operativos y arquitecturas, son desarrollados de manera independiente y son independientes de la aplicación que los usa.
Ventajas:

- Interoperabilidad entre aplicaciones.
- Independencia entre el servicio web y el cliente.
- Uso de estándares.
- Al ejecutar “comúnmente” HTTP, pueden atravesar firewalls sin necesidad de cambiar las reglas de filtrado.

Desventajas:

- Bajo rendimiento comparado con otros modelos como RMI o corba.
- Pueden esquivar firewalls.

**SOAP Simple Object Access Protocol:** Empaqueta la información y la transmite entre el cliente y el proveedor del servicio. Permite intercambiar mensajes basados en XML sobre redes de computadoras. Define un esquema para poder usar XML como representación del contenido de los mensajes. Usa HTTP, UDP, TCP y SMTP.
SOAP específica cómo representar los mensajes en XML, cómo procesar los elementos de los lenguajes, cómo combinar mensajes para reproducir un modelo de petición-respuesta y cómo utilizar el protocolo de aplicación para enviar mensajes.
Un mensaje SOAP se transporta en un sobre:




El sobre define el documento XML como un mensaje SOAP.
El encabezado es opcional, incluye información de control como: Identificador de transacción, identificador de mensajes, nombre de usuario, clave pública…

El cuerpo es obligatorio. Incluye el mensaje y referencia al esquema XML que describe el servicio. En la comunicación cliente-servidor, el cuerpo contiene una petición o respuesta.

**Interfaz:** Una interfaz de servicio web consta de un conjunto de operaciones que pueden ser accedidas por un cliente en internet. El conjunto de operaciones  pueden ser ofrecidas por programas, objetos, bases de datos…

**WSDL Web Services Description Language** es un lenguaje de descripción de interfaz para servicios web en XML. Describe el servicio web y cómo acceder a él.

El portType define las operaciones del servicio web y los mensajes que están involucrados. 4 tipos de operaciones:

- Sólo ida: el punto final recibe un mensaje.
- Notificación: el punto final envía un mensaje.
- Solicitud-respuesta: el punto final recibe un mensaje y envía uno correlacionado.
- Petición-respuesta: el punto final envía el mensaje y recibe uno correlacionado.

La vinculación o binding define el formato del mensaje y el protocolo.