



INFORME DE GUÍA PRÁCTICA

I. PORTADA

Tema: Fundamentos de las Aplicaciones Distribuidas
Unidad de Organización Curricular: PROFESIONAL
Nivel y Paralelo: Sexto - A
Alumnos participantes: Carrasco Paredes Kevin Andres
Asignatura: Aplicaciones Distribuidas
Docente: Ing. José Rubén Caiza Caizabuano, Mg.

II. INFORME DE GUÍA PRÁCTICA

2.1 Objetivos

General:

Identificar los conceptos principales a los sistemas y aplicaciones distribuidas.

Específicos:

- Analizar los modelos arquitectónicos más relevantes en los sistemas distribuidos, como cliente-servidor, peer-to-peer, middleware y publicador-suscriptor, destacando sus características, ventajas y limitaciones.
- Evaluar la aplicabilidad de cada modelo arquitectónico en contextos actuales, como la computación en la nube y el Internet de las Cosas (IoT), considerando factores como escalabilidad, tolerancia a fallos y eficiencia.
- Diseñar una representación visual mediante una infografía, que sintetice los conceptos estudiados y facilite la comprensión de los modelos y su uso en el diseño de aplicaciones distribuidas.

2.2 Modalidad

Presencial

2.3 Tiempo de duración

Presenciales: 4

No presenciales: 0

2.4 Instrucciones

- El trabajo se desarrollará de manera individual.
- Leer el archivo Computación Distribuida: Fundamentos y Aplicaciones que se encuentra en la plataforma virtual.
- Seleccionar una herramienta que permita realizar infografías.
- Realizar un infografía a manera de resumen de la lectura realizada - Para la calificación se tomará en cuenta:
 - ortografía, redacción, organización y presentación.
 - Subir el documento en este espacio en formato PDF.

2.5 Listado de equipos, materiales y recursos

Listado de equipos y materiales generales empleados en la guía práctica:

-

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

- Plataformas educativas
- Simuladores y laboratorios virtuales
- Aplicaciones educativas
- Recursos audiovisuales
- Gamificación



Inteligencia Artificial
Otros (Especifique): _____

2.6 Actividades por desarrollar

- Leer el archivo Computación Distribuida: Fundamentos y Aplicaciones que se encuentra en la plataforma virtual.
- Seleccionar una herramienta que permita realizar infografías.
- Realizar una infografía a manera de resumen de la lectura realizada y ponerla en común con los compañeros en la siguiente clase.

2.7 Resultados obtenidos

Los sistemas distribuidos constituyen un pilar esencial en la computación moderna, permitiendo la integración de múltiples recursos a través de redes para ofrecer servicios unificados. Este informe analiza los principales modelos arquitectónicos utilizados en este tipo de sistemas, resaltando sus características, aplicaciones prácticas y relevancia en entornos actuales como la computación en la nube y el Internet de las Cosas (IoT).

Modelos Analizados

1. Cliente-Servidor:

Modelo clásico donde los clientes solicitan servicios que son proporcionados por uno o varios servidores. Su simplicidad facilita la administración, aunque enfrenta limitaciones en escalabilidad y tolerancia a fallos. [1]

2. Peer-to-Peer (P2P):

Todos los nodos actúan como clientes y servidores, promoviendo la descentralización y la resistencia a fallos. Es utilizado en sistemas como BitTorrent y blockchain.

3. Middleware:

Funciona como una capa intermedia que permite la comunicación entre componentes heterogéneos. Tecnologías como RPC y MOM aseguran la interoperabilidad en entornos complejos.

4. Publicador-Suscriptor:

Basado en la transmisión de eventos, permite un alto nivel de desacoplamiento entre componentes, ideal para sistemas en tiempo real como plataformas de streaming o notificaciones.

Metodología y Herramientas

Como complemento al análisis teórico, se elaboró una **infografía en Canva** que resume visualmente los conceptos tratados, facilitando la comprensión de las diferencias clave entre los modelos presentados.

2.8 Habilidades blandas empleadas en la práctica

- Liderazgo
- Trabajo en equipo
- Comunicación asertiva
- La empatía
- Pensamiento crítico
- Flexibilidad



- La resolución de conflictos
- Adaptabilidad
- Responsabilidad

2.9 Conclusiones

La elección del modelo arquitectónico adecuado depende de los objetivos del sistema, tales como escalabilidad, latencia o tolerancia a fallos. En la práctica, muchos sistemas adoptan enfoques híbridos, combinando varios modelos para lograr mayor flexibilidad. Además, la representación visual de estos conceptos resultó ser una herramienta eficaz para comunicar ideas complejas de forma clara y accesible.

El estudio concluye que dominar los sistemas distribuidos requiere no solo una base técnica sólida, sino también habilidades para adaptarse a un entorno tecnológico en constante cambio y comunicar eficazmente soluciones a través de medios visuales y teóricos.

2.10 Recomendaciones

Se aconseja evitar bloques extensos de texto, emplear fuentes legibles y dejar suficiente espacio entre los elementos para mejorar la legibilidad. La claridad en la presentación de los conceptos es clave para que la infografía cumpla su propósito educativo.

2.11 Referencias bibliográficas

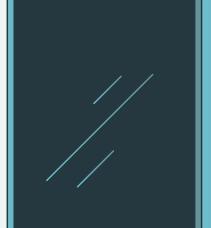
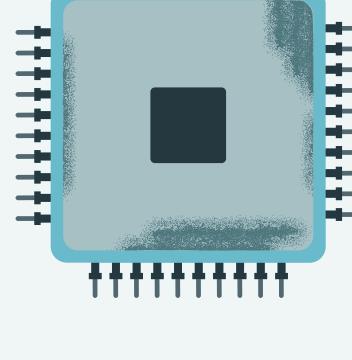
- [1] P. Bazán, 2017. [En línea]. Available:
https://sistemaseducaciononline.uta.edu.ec/course/format/tiles/mod_view.php?cmid=50161.
[Último acceso: 5 Abril 2025].

2.12 Anexos

INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

Definición

- Sistemas cuyos componentes (hardware/software) están en nodos de red y se comunican mediante mensajes, dando la ilusión de un sistema único.



Objetivos

- Transparencia: Ocultar la distribución al usuario.
- Escalabilidad: Capacidad de crecimiento en recursos y usuarios.

- Apertura: Adhesión a estándares para interoperabilidad.

- Accesibilidad: Garantizar acceso a recursos en todo momento.

Modelos Arquitectónicos

Cliente/Servidor

- Características:

- Roles definidos:

- Cliente: Inicia solicitudes y consume servicios.

- Servidor: Responde a solicitudes y gestiona recursos compartidos.

- Comunicación: Basada en el esquema petición-respuesta (sincrónica o asincrónica).

- Ejemplos: Servidores web (HTTP), bases de datos (SQL), correo electrónico (SMTP).

Ventajas:

- ✓ Fácil de implementar y escalar verticalmente (mejorando el servidor).

- ✓ Centraliza el control y la seguridad.

Desventajas:

- ✗ El servidor puede convertirse en un cuello de botella.

- ✗ Poca tolerancia a fallos (si el servidor falla, el sistema colapsa).



Peer-to-Peer P2P

- Características:

- Nodos igualitarios (peers): Todos pueden actuar como clientes y servidores.

- Comunicación descentralizada: No hay un nodo central; los recursos se distribuyen.

- Ejemplos: Redes de intercambio de archivos (BitTorrent), blockchain.

Ventajas:

- ✓ Alta tolerancia a fallos (no hay un punto único de fallo).

- ✓ Escalabilidad horizontal (más nodos = más capacidad).

Desventajas:

- ✗ Mayor complejidad en la coordinación y seguridad.

- ✗ Dificultad para garantizar consistencia en datos compartidos.

Basado en Capas

- Características:

- Organización jerárquica: Cada capa depende de la inferior (ej: modelo OSI).

- Abstracción: Las capas superiores no conocen los detalles de las inferiores.

- Ejemplos: Arquitectura web de 3 capas (presentación, lógica, datos).

Ventajas:

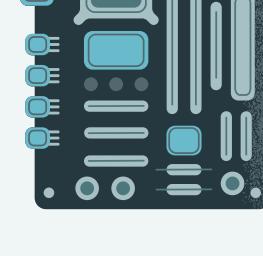
- ✓ Modularidad y facilidad de mantenimiento.

- ✓ Separación clara de responsabilidades.

Desventajas:

- ✗ Latencia adicional por el paso entre capas.

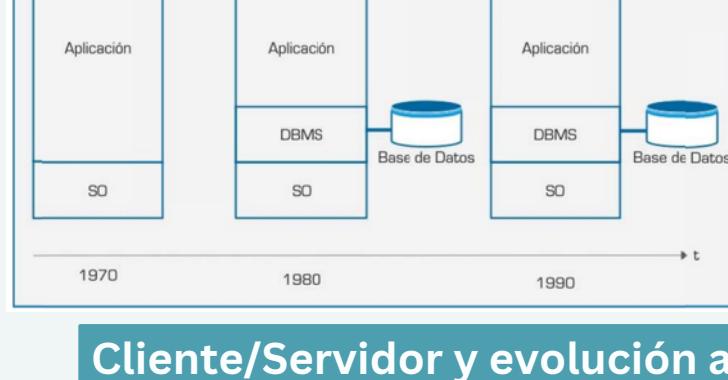
- ✗ Rigididad en cambios estructurales.



EVOLUCIÓN DE LOS SISTEMAS DISTRIBUIDOS

Sistemas Monolíticos

- Sistemas concebidos como una única pieza funcional, donde un solo código maneja la interfaz de usuario, el acceso a datos y la lógica algorítmica.
- Características:
 - Desarrollados en un único lenguaje, ejecutados en una sola computadora.
 - Comunicación mediante terminales de caracteres (años '70).
- Evolución: Mejoras con programación estructurada y modular, aunque aún no distribuidos.



Cliente/Servidor y evolución a n-capas

- Arquitectura Cliente/Servidor (2 capas):
 - Cliente: Interfaz de usuario (lenguajes visuales como Delphi).
 - Servidor: Acceso a datos (DBMS).
 - Desventajas: Desbalanceo de carga y falta de escalabilidad.
- Evolución a n-capas:
 - Incorporación de una capa media (lógica de negocio), exemplificada en la tecnología web.
 - Ejemplo: Web Server Dinámico (3 capas: navegador, servidor web, servidor de base de datos).



Sistemas Distribuidos con Objetos

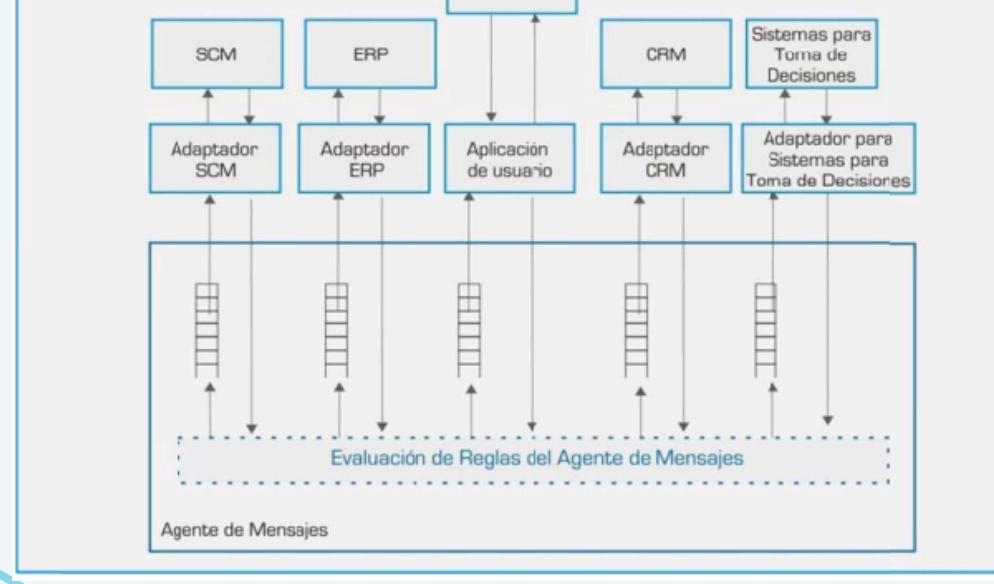
- Objetos distribuidos: Extienden la POO con características como transaccionalidad, seguridad y persistencia.
- Estándar CORBA:
 - Facilita comunicación entre objetos en distintas plataformas.
 - Incluye IDL (lenguaje de definición de interfaces) y ORB (bus de objetos).



Arquitectura Orientada a Servicios (SOA)



- Enfoque que organiza capacidades distribuidas como servicios reutilizables.
- Componentes clave:
 - Proveedor, consumidor, capacidad, servicio y descripción del servicio.
- Ventajas: Flexibilidad, reutilización y alineación con necesidades de negocio.



LA DISTRIBUCIÓN Y LOS SISTEMAS OPERATIVOS

¿Qué es un sistema operativo?

Software que gestiona recursos hardware/software y facilita la interacción usuario-máquina.

- **Evolución:** De sistemas centralizados (mainframes) a entornos distribuidos, donde múltiples nodos aparecen como una única unidad.
- **Rol en sistemas distribuidos:** Administrar recursos compartidos, garantizar transparencia y sincronización en nodos heterogéneos.



Hitos

- **Mainframes:** Sistemas centralizados (IBM 360/370) con altos niveles de seguridad y robustez.
- **Sistemas abiertos:** Surgimiento de Unix (AIX, HP-UX) y software libre, promoviendo interoperabilidad y colaboración.
- **Revolución cliente-servidor:** Separación lógica de responsabilidades entre clientes (interfaz) y servidores (datos).

Cloud y Grid Computing

- **Grid Computing:** Integración de recursos distribuidos para fines científicos (ej: clusters geográficos).
- **Cloud Computing:** Servicios bajo demanda (IaaS, PaaS, SaaS) con virtualización como pilar.
- **Middleware:** Software que abstrae heterogeneidad, actuando como "sistema operativo distribuido".

Virtualización

- **Concepto:** Creación de entornos virtuales aislados sobre hardware físico.
- **Hipervisores:**
 - **Tipo 1:** Ejecución directa sobre hardware (ej: VMware ESXi).
 - **Tipo 2:** Ejecución sobre un SO host (ej: VirtualBox).
- **Ventajas:** Aislamiento, eficiencia energética, migración fácil de máquinas virtuales.



Servicios en Sistemas Distribuidos

- **Básicos:**
 - **Comunicación:** Protocolos para intercambio de mensajes en redes heterogéneas.
 - **Sincronización:** Reloj lógico/físico para ordenar eventos.
 - **Gestión de procesos:** Modelos como estaciones de trabajo o pilas de procesadores.
 - **Memoria compartida:** Acceso transparente a memoria remota mediante caché.
- **Extendidos:**
 - **Nombres y localización:** DNS para identificar recursos.
 - **Seguridad:** Autenticación y cifrado en comunicaciones.

Sistemas Operativos de Red

- **Funciones:** Gestionar acceso a recursos compartidos (archivos, impresoras).
- **Modelos:**
 - **Acceso remoto:** Datos residen en servidor (ej: SSH).
 - **Carga/descarga:** Datos se copian localmente (ej: FTP).

CLIENTES Y SERVIDORES EN SISTEMAS DISTRIBUIDOS

Roles y Funciones del Servidor

- **Funciones clave:**
 - Espera pasiva de solicitudes.
 - Ejecución concurrente de múltiples peticiones.
 - Priorización de clientes (ej: servicios interactivos sobre batch).
 - Tolerancia a fallas y escalabilidad.
- **Tipos de servidores:**
 - Archivos: Envío de registros.
 - Bases de datos: Procesamiento de consultas SQL.
 - Transacciones: Ejecución atómica de procedimientos remotos.
 - Grupos de trabajo: Manejo de información semi-estructurada.
 - Objetos: Comunicación mediante ORB (Object Request Broker).



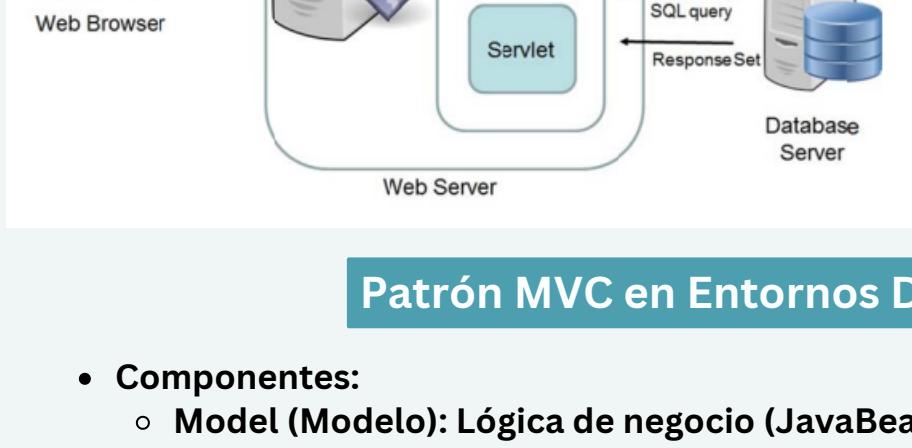
Roles y Funciones del Cliente

- **Responsabilidades:** Iniciar solicitudes, procesar respuestas y mostrar resultados al usuario.
- **Tecnologías:** Lenguajes de cuarta generación (Visual Basic, Delphi) para interfaces gráficas.



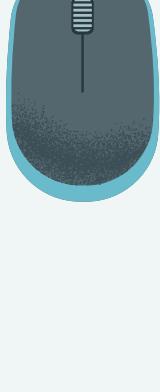
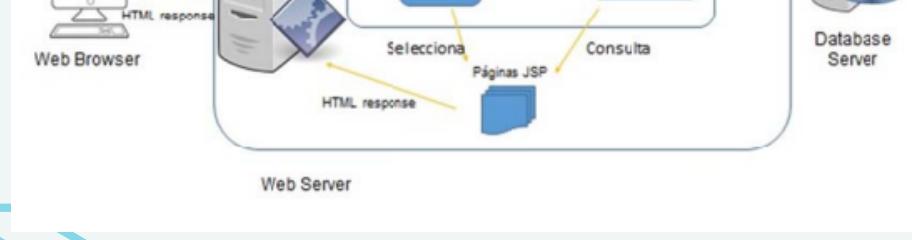
Cliente/Servidor Web con Java

- **Web dinámico:** Extensión de servidores web para ejecutar código (ej: JSP, PHP).
- **Enfoques:**
 - **Nativo:** Modificación del núcleo del servidor (ej: Java EE).
 - **Interfaz:** Uso de APIs como CGI (Common Gateway Interface).
- **Arquitectura Java EE:**
 - **Servlets:** Programas Java en el servidor que generan HTML.
 - **JSP:** Páginas HTML con código Java incrustado, traducidas a servlets.



Patrón MVC en Entornos Distribuidos

- **Componentes:**
 - **Model (Modelo):** Lógica de negocio (JavaBeans, EJB).
 - **View (Vista):** Interfaz (JSP).
 - **Controller (Controlador):** Gestiona flujo (Servlets).
- **Ventajas:** Separación clara de responsabilidades.
- **Desafíos:** Carga en el servidor y duplicación de lógica en clientes no web.



ARQUITECTURAS DISTRIBUIDAS

Arquitecturas de n-Niveles y Tecnología CGI

- **Modelo de 3 niveles:**
 - Cliente: Interfaz gráfica (GUI).
 - Nivel medio: Lógica de negocio separada.
 - Servidor de datos: Base de datos.
 - Ventajas: Mayor seguridad, encapsulamiento, y escalabilidad frente al modelo de 2 niveles (Tabla comparativa).
- **Tecnología CGI (Common Gateway Interface):**
 - Mecanismo para ejecutar programas externos en servidores web mediante APIs.
 - Ventajas: Lenguaje-agnóstico (Perl, C, etc.), aislamiento de fallos.
 - Desventajas: Lentitud, falta de escalabilidad, y mezcla de lógica/presentación.



Arquitectura Orientada a Servicios (SOA)

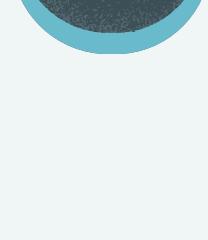
- **Definición (OASIS):** Paradigma para organizar capacidades distribuidas bajo estándares.
- **Características clave:**
 - Reuso: Federación de aplicaciones heterogéneas.
 - Interoperabilidad: Independencia de plataformas.
 - Flexibilidad: Alineamiento con procesos de negocio.
- **Componentes:**
 - Servicios: Unidades funcionales (ej: "proceso de compra").
 - ESB (Enterprise Service Bus): Infraestructura para comunicación segura y transformación de mensajes.
- **Modelo de Interacción:**
 - Publicar-Ligar-Ejecutar: Registro de servicios → Descubrimiento → Invocación



Web Services como Implementación de SOA

- **Estándares:**
 - WSDL: Describe servicios en XML.
 - SOAP: Protocolo para intercambio estructurado.
 - UDDI: Registro global de servicios.
- **Diferencias con CORBA:**
 - Menor acoplamiento y mayor meta-information en servicios.

Tecnologías Web Services: SOAP vs REST



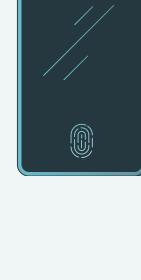
- **SOAP:**
 - Basado en operaciones WSDL.
 - Usa XML encapsulado en mensajes SOAP.
 - Soporta múltiples protocolos (HTTP, JMS).
- **REST:**
 - Operaciones estándar HTTP (GET, PUT, DELETE).
 - Enfoque en recursos (no en operaciones).
 - Más ligero y flexible (Tabla comparativa).

	REST	SOAP
Formato del mensaje	XML	XML dentro de SOAP
Definición del interfaz	No es necesario	WSDL
Transporte	HTTP	HTTP, JMS, FTP, etc.

SERVIDORES DE BASE DE DATOS Y LA DISTRIBUCIÓN

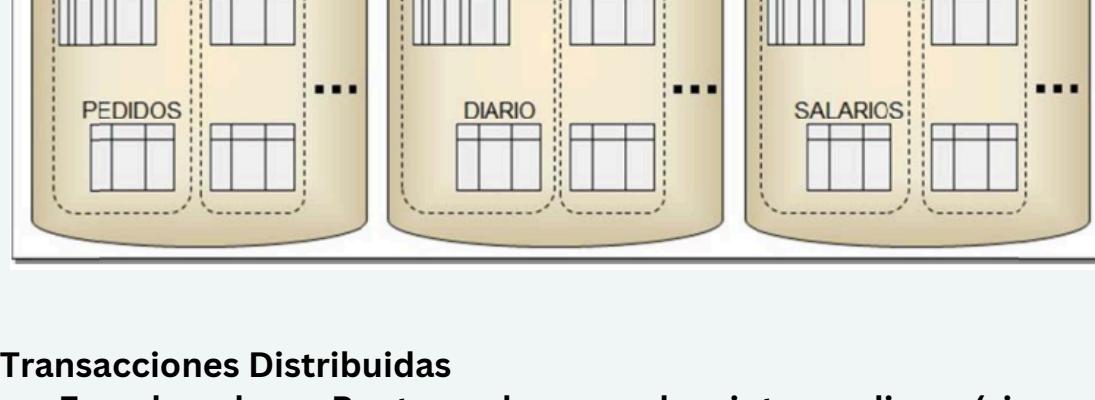
Fundamentos de SQL y Bases de Datos Relacionales

- **SQL:** Lenguaje estándar para manipulación, definición y control de datos, basado en teoría de conjuntos.
- **Roles:**
 - Consulta interactiva.
 - Programación embebida (ej: en APIs como X/Open).
 - Administración de datos (tablas, índices, seguridad).
- **Servidores SQL:**
 - Ejecutan comandos SQL.
 - Gestionan concurrencia, seguridad, y transacciones.
 - Optimizan planes de ejecución.



Arquitecturas de Servidores SQL

- **Por almacenamiento:**
 - Única base de datos: Todos los objetos en un dispositivo (ventaja: eficiencia; desventaja: copias de respaldo lentas).
 - Múltiples bases de datos: Objetos distribuidos en dispositivos (ventaja: administración flexible; desventaja: duplicación de datos).
- **Por procesamiento:**
 - Proceso por cliente: Asigna un proceso por solicitud (seguro pero consume recursos).
 - Multihilada: Usa hilos dentro de un proceso (eficiente pero vulnerable a fallos).
 - Híbrida: Balancea carga mediante un dispatcher (equilibrio entre rendimiento y seguridad).



- **Transacciones Distribuidas**
 - Encadenadas: Puntos de guarda intermedios (sin cumplir durabilidad ante fallos).
 - Anidadas: Jerarquía de sub-transacciones (commit depende del padre).
- **Monitores de Transacciones**
 - TP-Lite: Integrado en motores de bases de datos (ej: procedimientos almacenados). Limitado a recursos del vendedor.
 - TP-Heavy: Monitores independientes (ej: CICS, Tuxedo).
 - Gestionan transacciones globales en recursos heterogéneos.
 - Balancean carga y recuperan fallos.

