Desarrollo de Sistemas Distribuidos

Tema 4
Sistemas Cliente/Servidor y Peer-to-Peer

Contenidos

1. Introducción
2. Servicios
3. Modelos C/S y P2P
4. Modelo funcional y de comportamiento
5. Diseño

Introducción

- Interconectividad en computación distribuida implica resolverla entre:
  1. Máquinas
  2. Redes
  3. Aplicación (seguridad, verificación, manejo de fallos, etc)
- Uso de servicios de alto nivel reduce el código de la aplicación

Introducción

- **Servicio**:

  Una colección de atributos y comportamientos que pueden ser proporcionados por un recurso empresarial para su uso a través de interfaces bien definidas

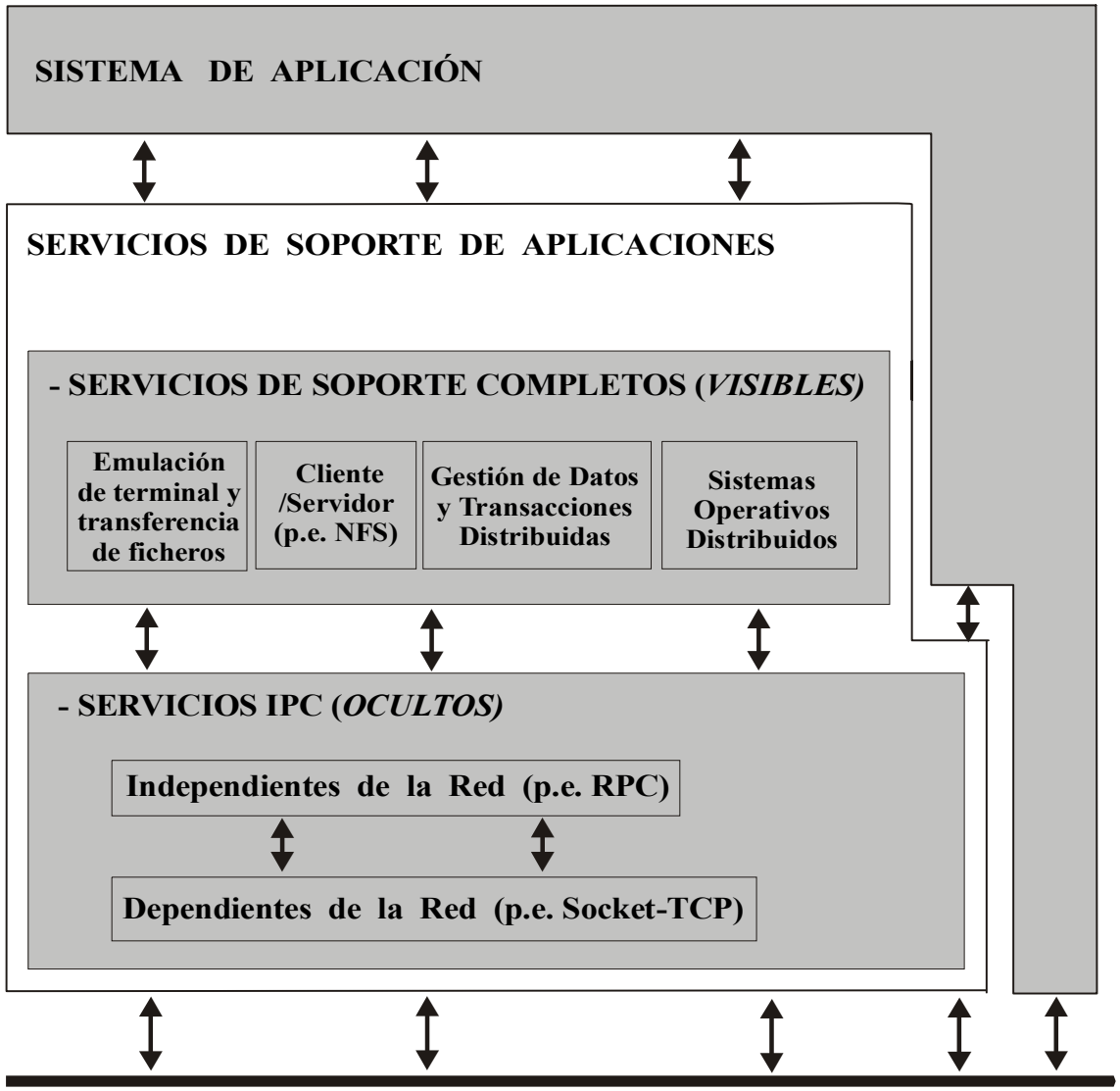
- El concepto de servicio resulta de separar el comportamiento "externo" e "interno" de un sistema, perspectivas:

  1. **Entorno**: Autónomo y tener un propósito claro
  2. **Desarrollador**: El comportamiento interno es lo que se requiere para realizar este servicio
  3. **Consumidores**: El comportamiento interno es generalmente irrelevante; interesados en la funcionalidad y calidad

Introducción

- Por tanto, es importante la **clasificación de los servicios**, por ejemplo, en TIC la infraestructura de servicios puede clasificarse como un conjunto de capacidades:
  1. Transporte en red
  2. Manejo de recursos de información (almacenamiento, recuperación, manipulación y visualización)
  3. Administración (fallos, configuraciones, contabilidad, seguridad, rendimiento, y gestión del ciclo de vida del servicio)

Introducción



Introducción

- Los **servicios completos** pueden ser utilizados directamente por los usuarios finales mediante órdenes y las aplicaciones se pueden construir en base a estos servicios ⇒ minimizan esfuerzo de desarrollo
- Los **servicios IPC** están generalmente disponibles sólo para desarrolladores de aplicaciones invocados a través de APIs ⇒ permiten más flexibilidad y eficiencia

Introducción

- Desde el punto de vista de la **gestión** se deben explotar los servicios de más alto nivel y así hacia abajo
- **Importante**: no es fácil interconectar aplicaciones que utilizan servicios a diferentes niveles

Servicios completos

- Modelos más importantes para interconexión entre aplicaciones:
  - Cliente/Servidor
  - Peer-to-Peer: cualquier proceso localizado remotamente puede iniciar una interacción
- Un modelo C/S se puede implementar sobre protocolos P2P, pero lo contrario no es necesariamente verdad
- Ambos modelos escalan bien tanto vertical como horizontalmente
- El procesamiento cooperativo distribuido usa C/S o P2P:
  - C/S y P2P resultan ser como subcategoría
  - Puede ser implementado con diferentes configuraciones

Servicios IPC

- Las aplicaciones distribuidas (C/S o P2P) pueden usar diferentes **paradigmas para el intercambio** de mensajes:

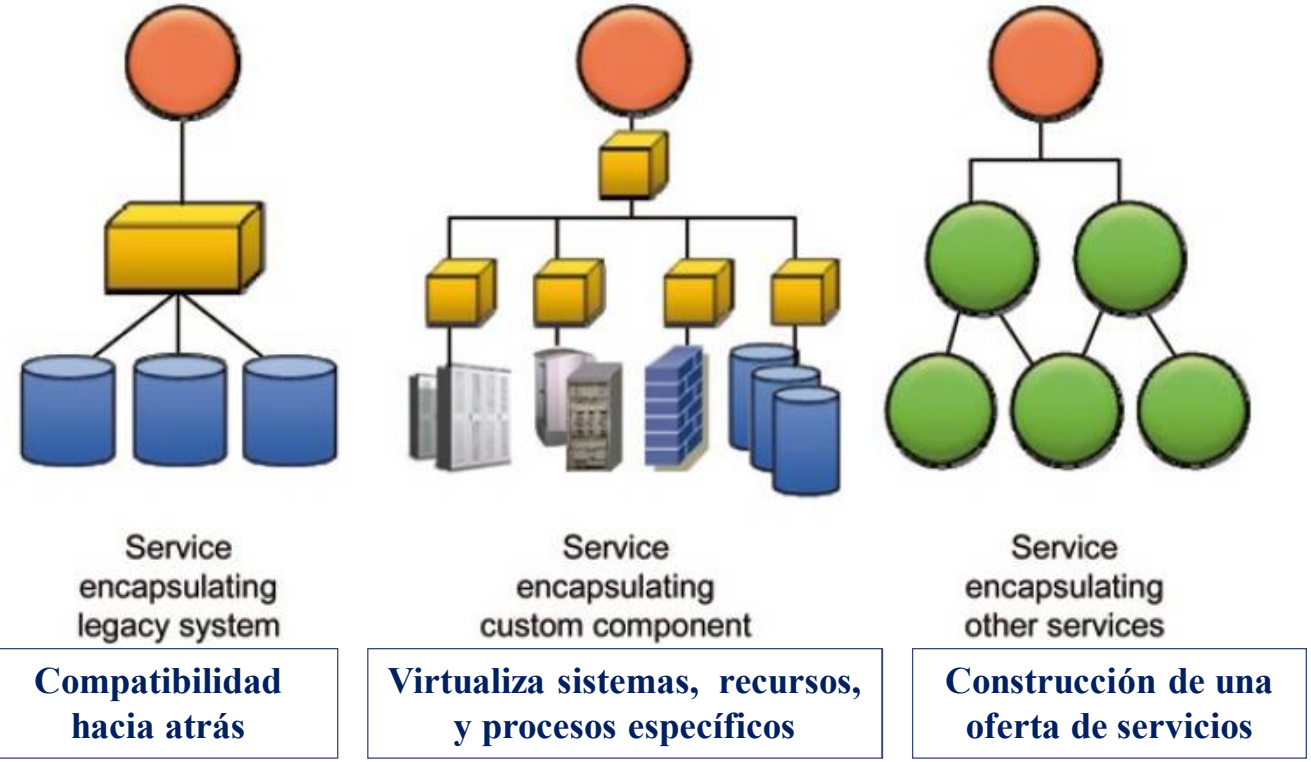
1. **Petición/respuesta**
   - Cada petición/respuesta como unidad separada
   - Cumple estrictamente el modelo C/S
2. **Conversacional**
   - Cada interacción no es autocontenida ni unidad independiente
   - Lo utilizan C/S y P2P
3. **Procesamiento de mensajes encolados**
   - El receptor almacena los mensajes de petición en una cola
   - Atiende los mensajes cuando está desocupado
   - Permite al emisor enviar peticiones asíncronas
   - Utilizado en sistemas de procesamiento de transacciones soportando C/S y P2P

Modelo Cliente/Servidor

- Principios principales de diseño de los servicios:
  1. **Abstracción**: Los contratos de servicios contienen sólo información esencial; La información se limita a lo que está en el contrato
  2. **Contrato estandarizado**: Servicios dentro del mismo inventario de servicios cumplen con los mismos estándares de diseño del contrato
  3. **Débil acoplamiento**: Los contratos de servicio imponen requisitos de acoplamiento de consumidores bajos y están disociados de su entorno
  4. **Reutilización**: Contienen y expresan lógica y pueden posicionarse como recursos reutilizables
  5. **Autonomía**: Ejercen un alto nivel de control sobre su entorno de ejecución de ejecución subyacente
  6. **Sin estado**: Minimizan el consumo de recursos relegando la gestión de la información de estado cuando es necesario
  7. **Descubrimiento**: Se complementan con metadatos para ser descubiertos e interpretados eficazmente
  8. **Componibles**: Son participantes eficaces de la composición, sin importar el tamaño y la complejidad de la composición

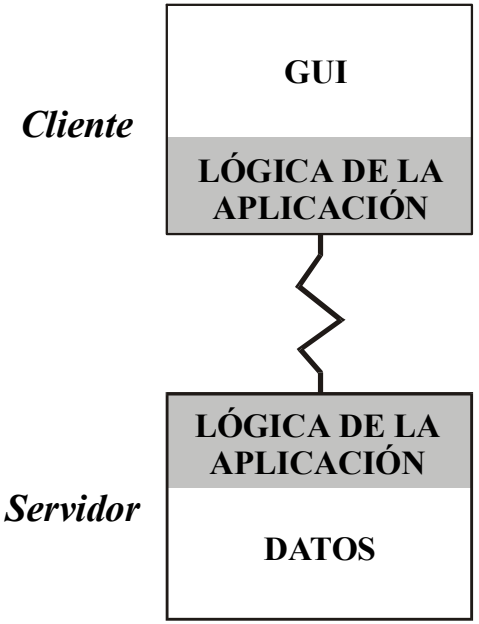
Modelo Cliente/Servidor

- SOA permite que el software esté disponible bajo demanda
- El principio de abstracción del servicio esconde los detalles subyacentes del servicio para permitir/preservar la relación de bajo acoplamiento
- Tres ejemplos de instancias de servicio y operación:

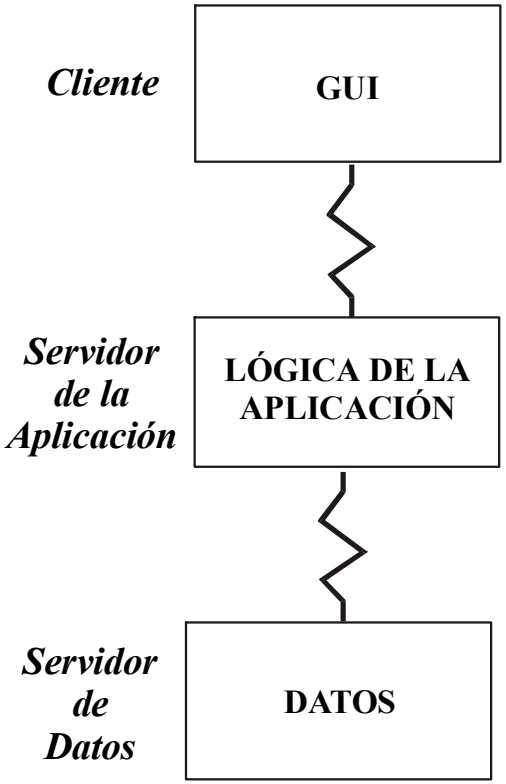


Modelo Cliente/Servidor

Modelo Monolítico de Aplicación (No es un modelo Cliente/Servidor real)



Modelos de 2-Etapas *("Two-Tiers")* P.e. X-Windows, NFS, Navegadores Web



Modelos de 3-Etapas *("Three-Tiers")* P.e. Aplicaciones de gestión

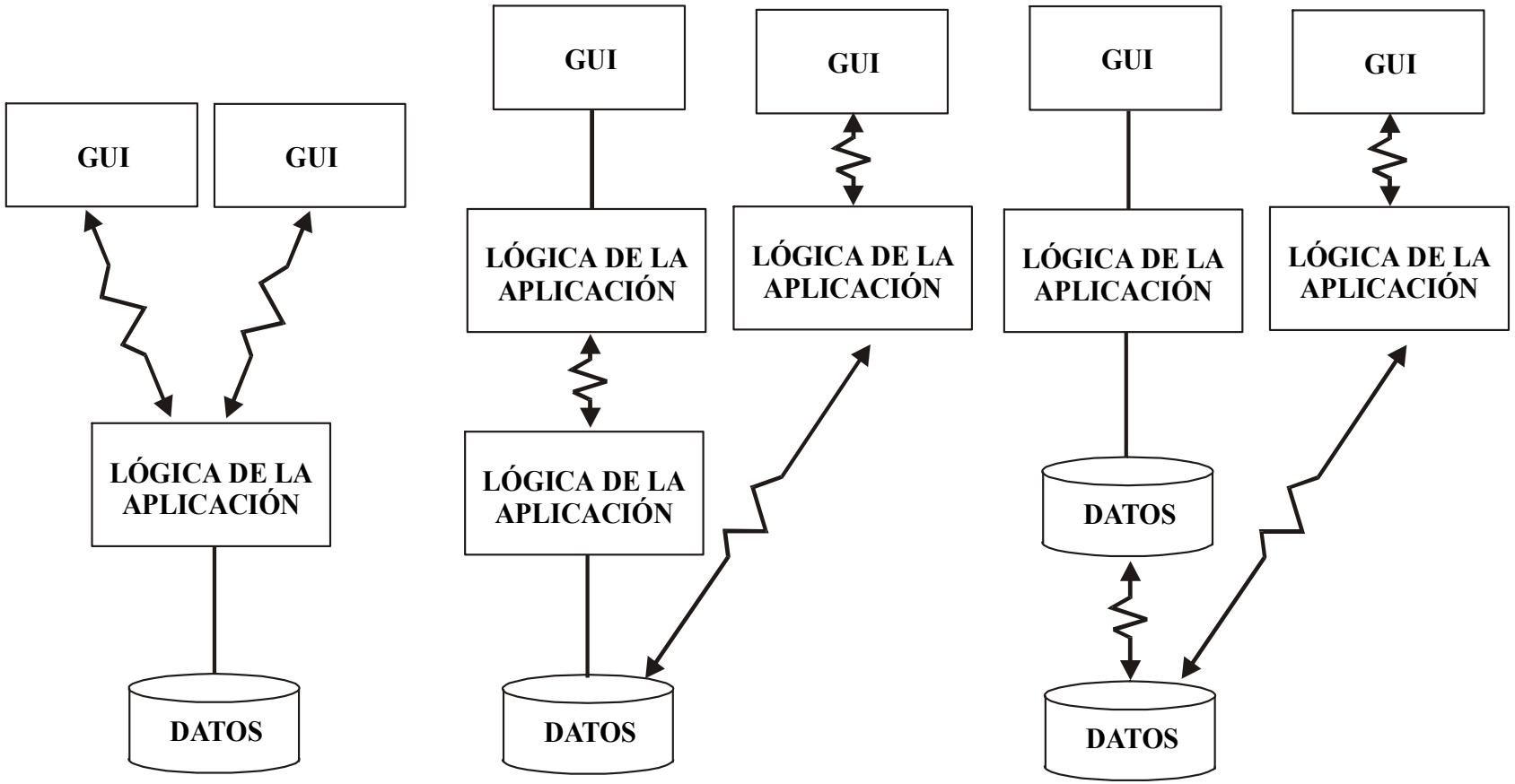
Modelo Cliente/Servidor

- Modelos de 2 etapas (canónicos) pueden distribuir autoridad/responsabilidad/inteligencia. Configuraciones:
  1. Grandes Servidores ("*Fat Servers*"):
     - Facilidad en actualización, manejo, depuración y mantenimiento
     - Mayor encapsulación y compatibilidad entre clientes y servidores
     - Menor número de transferencias en la red
  2. Grandes Clientes ("*Fat Clients*"):
     - Servidor más estable
     - Mayor flexibilidad de programación, ampliación, extensión.
  3. Intermedios

Modelo Cliente/Servidor

- Modelos de ***n*** etapas:
  - Más avanzado y flexible al dar mayor autonomía
  - Más robusto debido a las partes independientes
  - Adecuado para sistemas con datos distribuidos
  - **Inconveniente**: es difícil construir sistemas fiables y eficientes con más de 3 etapas

Servicios completos

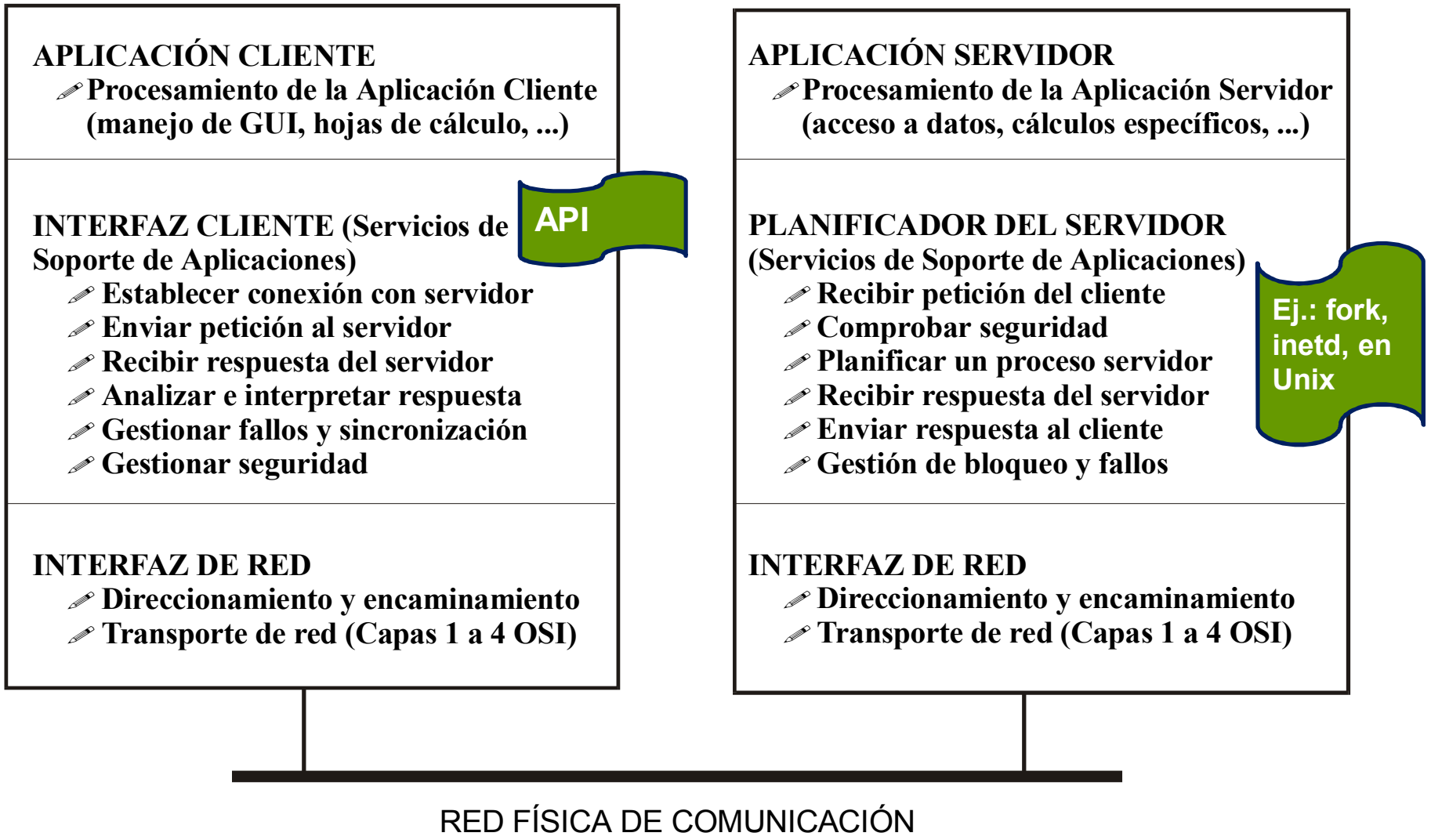


Distribución del Interfaz de Usuario

Distribución de la Lógica del Programa

Distribución de los Datos

Modelo Funcional



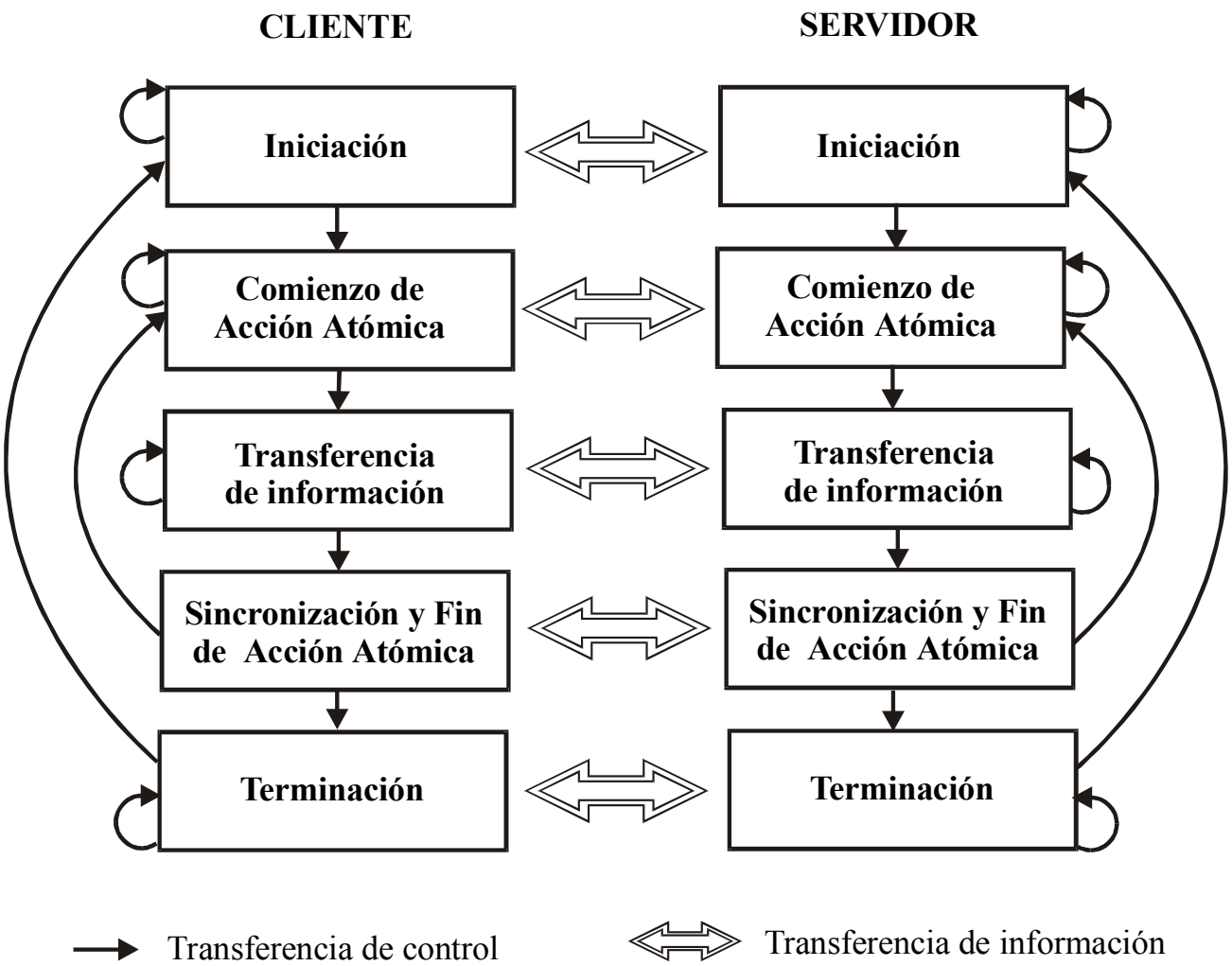
Cuestiones Técnicas

- El cliente cambiará si la interfaz cambia (suele ser **API**)
- La **planificación** de peticiones del servidor depende del SO (*fork*, demonios, …), e.g. INETD para servidores de Unix
- Por **interoperabilidad** clientes y servidores deben utilizar los mismo protocolos/servicios (e.g. diferentes implementaciones de RPC dan problemas)
- **Manejo de fallos** no es trivial, se utilizan técnicas (e.g. transacciones)
- Responsabilidad del planificador y servidores de la **seguridad** de los datos

Cuestiones Técnicas

- Hay que evaluar el **rendimiento** del sistema:
  - Utilizar **simuladores** para analizar:
    - retardos en la red
    - tiempos de procesamiento en nodos (¿nuevos servidores o hacerlos reentrantes?), …
  - **Minimizar mensajes** como principal consideración:
    - Patrones de tráfico de red impredecibles
    - Tiempos de respuesta difíciles de calcular por encaminamientos
    - Problemas de escalabilidad debidos a protocolos y algoritmos (seguridad, recuperación de fallos, …)

Modelo Comportamiento



Diseño

- El **planificador de servidores**:
  - Escucha en los canales de comunicación e invoca a los procesos apropiados
  - Se pueden construir con: facilidades del SO, gestores de transacciones o entornos de soporte a aplicaciones especializadas
- **Interacción C/S**:
  - Puede seguir cualquiera de los paradigmas de comunicación vistos y su elección depende del tipo de la aplicación (e.g. RPC no es adecuado para colas de mensajes)
  - Se deben minimizar interacciones

Diseño

- **Procesos servidores**:
  - Eficientes en términos de utilización de recursos y algoritmos utilizados
  - Se clasifican:
    - **Único paso**: el propio planificador realiza el servicio utilizando pocos recursos eficientemente
    - **Múltiples pasos:** centralizados, con planificadores o no, replicados, distribuidos, jerarquizados, compartición de tareas, ...
  - Hacen uso de otros servicios (p.e., DNS, NFS,...)
  - Entornos y lenguajes de programación