Ingeniería de Software

Diseño de Software
Clase 3

Sommerville 10 (9*) — Capítulos 17 y 18

Ingeniería de software basada en componentes

Ing de soft basada en componentes

- La ingeniería de software basada en componentes (CBSE) es un enfoque para el desarrollo de software que se basa en la reutilización de entidades llamadas 'componentes de software'.
- Surgió del fracaso del desarrollo orientado a objetos para apoyar la reutilización efectiva. Las clases de objetos individuales son demasiado detalladas y específicas.
- Los componentes son más abstractos que las clases de objetos y pueden considerarse proveedores de servicios independientes. Pueden existir como entidades independientes.

Pilares de CBSE

- Componentes independientes especificados por sus interfaces.
- Estándares de componentes para facilitar la integración de componentes.
- Middleware que proporciona soporte para la interoperabilidad de los componentes.
- Un proceso de desarrollo que está orientado a la reutilización.

CBSE y principios de diseño

- Los componentes son independientes, así que no interfieran entre sí;
- Las implementaciones de componentes están ocultas;
- La comunicación es a través de interfaces bien definidas;
- Un componente puede ser reemplazado por otro si se mantiene su interfaz;
- Las infraestructuras de componentes ofrecen una gama de servicios estándar.

Estándares de componentes

- Los estándares deben establecerse para que los componentes se puedan comunicar entre sí e interoperar.
- Desafortunadamente, se establecieron varios estándares de componentes competidores:
  - Enterprise Java Beans (EJBs) de Java
  - COM y .NET de Microsoft
  - CCM de CORBA
- En la práctica, estos estándares múltiples han dificultado la adopción de CBSE. Es imposible que los componentes desarrollados usando diferentes enfoques trabajen juntos.

Ing de software orientada a servicios

- Un servicio ejecutable es un tipo de componente independiente. Tiene una interfaz 'provee' pero no una interfaz 'requiere'.
- Desde el principio, los servicios se han basado en estándares para que no haya problemas en la comunicación entre los servicios ofrecidos por diferentes proveedores.
- El rendimiento del sistema puede ser más lento con los servicios, pero este enfoque reemplaza a CBSE en muchos sistemas.
- Cubierto en el Capítulo 19

Componentes

- Los componentes proporcionan un servicio sin importar dónde se está ejecutando el componente o su lenguaje de programación
  - Un componente es una entidad ejecutable independiente que puede estar compuesta por uno o más objetos ejecutables;
  - La interfaz del componente se publica y todas las interacciones se realizan a través de la interfaz publicada;
- Características
  - Estandarizado
  - Independiente – no tiene que ser compilado antes de usar.
  - Componible
  - Implementable
  - Documentado

Interfaces de los componentes

- Interfaces provistas
  - Definen los servicios que ofrece el componente.
- Interfaces requeridas
  - Definen qué servicios deben ofrecer otros componentes para que este componente opere correctamente.
  - Esto no compromete la independencia o la capacidad de despliegue de un componente porque la interfaz 'requiere' no define cómo se deben proporcionar estos servicios.

Componentes – Acceso y modelos

- Se accede a los componentes mediante llamadas a procedimientos remotos (RPC). Cada componente tiene un identificador único (generalmente una URL) y puede referenciarse desde cualquier computadora en red.
- Un modelo de componentes es una definición de estándares para la implementación, documentación y despliegue de componentes. Ejemplos: EJB (Enterprise Java Beans), COM + (modelo .NET), Corba.
- El modelo de componente especifica cómo se deben definir las interfaces y los elementos que deberían incluirse en una definición de interfaz.

Elementos de un modelo

- Interfaces

  El modelo especifica cómo se deben definir las interfaces y los elementos, como los nombres de operación, los parámetros y las excepciones, que deben incluirse en la definición de la interfaz.

- Uso

  Para que los componentes se distribuyan y accedan de forma remota, deben tener un nombre único o un identificador asociado a ellos. Esto tiene que ser globalmente único.

- Despliegue

  El modelo incluye una especificación de cómo los componentes se deben empaquetar para su despliegue como entidades ejecutables independientes.

Soporte del Middleware

- Los modelos de componentes son la base del middleware que proporciona soporte para la ejecución de componentes.
- Las implementaciones del modelo de componentes proporcionan:
  - Servicios de plataforma que permiten que los componentes escritos según el modelo se comuniquen;
  - Servicios de soporte que son servicios independientes de la aplicación utilizados por diferentes componentes.
- Para utilizar los servicios proporcionados por un modelo, los componentes se implementan en un contenedor. Este es un conjunto de interfaces utilizadas para acceder a las implementaciones del servicio.

Desarrollo para el reuso

- Desarrollo de componentes o servicios que se reutilizarán.
- Los componentes desarrollados para una aplicación específica generalmente deben generalizarse para que sean reutilizables.
- Reutilización de componentes
  - Debería reflejar abstracciones de dominio estables;
  - Debe ocultar la representación del estado;
  - Debe ser lo más independiente posible;
  - Debe publicar excepciones a través de la interfaz del componente.
- Hay una compensación entre la reutilización y la usabilidad
- Cuanto más general es la interfaz, mayor es la capacidad de reutilización, pero es más compleja y, por lo tanto, menos utilizable.

Cambios para la reutilización

- Eliminar los métodos específicos de la aplicación.
- Cambiar los nombres para hacerlos generales.
- Agregar métodos para ampliar la cobertura.
- Hacer que el manejo de excepciones sea consistente.
- Agregar una interfaz de configuración para la adaptación del componente.
- Integrar los componentes necesarios para reducir las dependencias.

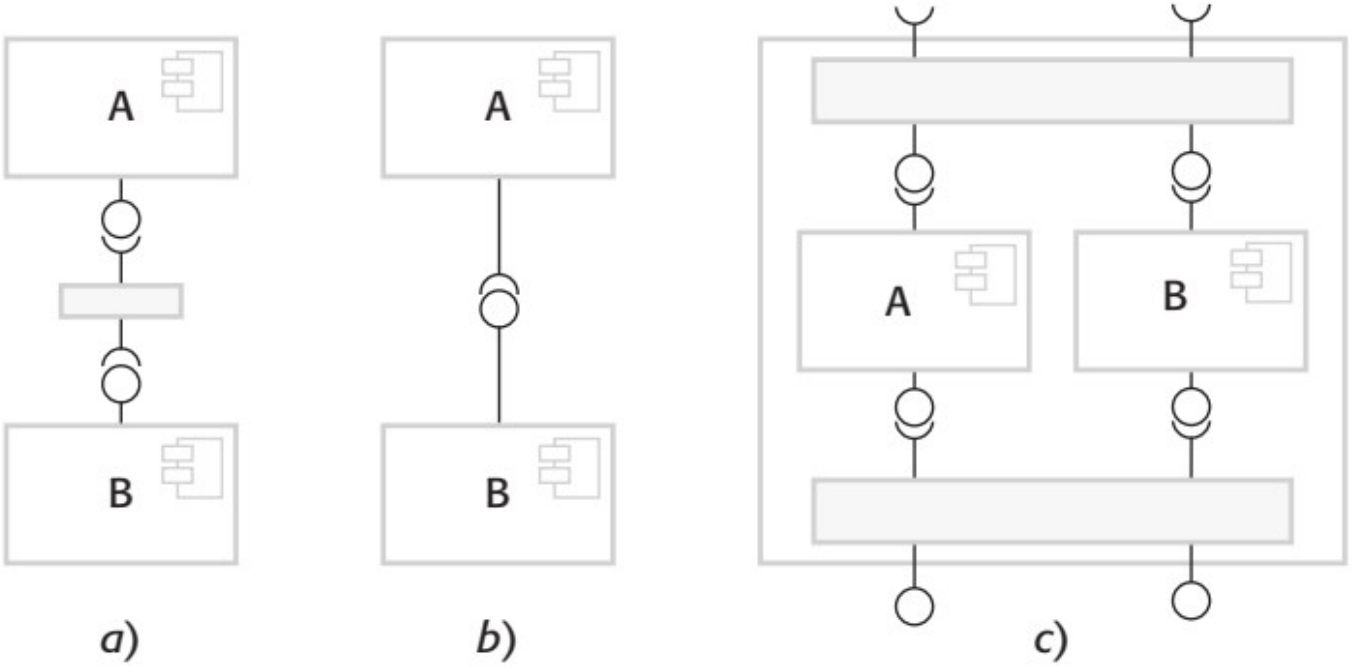
- → Sistemas legados/heredados: buenos candidatos para hacer componentes reusables, en general implica re-empaquetar funcionalidades mediante un adaptador (wrapper).

Desarrollo con reuso

- CBSE con proceso de reutilización tiene que encontrar e integrar componentes reutilizables.
- Al reutilizar los componentes, es esencial negociar entre los requisitos ideales y los servicios realmente proporcionados por los componentes disponibles.
- Esto involucra:
  - Desarrollar requisitos de esquema;
  - Búsqueda de componentes y luego modificación de requisitos según la funcionalidad disponible.
  - Buscando de nuevo para encontrar si hay mejores componentes que cumplan con los requisitos revisados.
  - Componiendo componentes para crear el sistema.

Composición de componentes

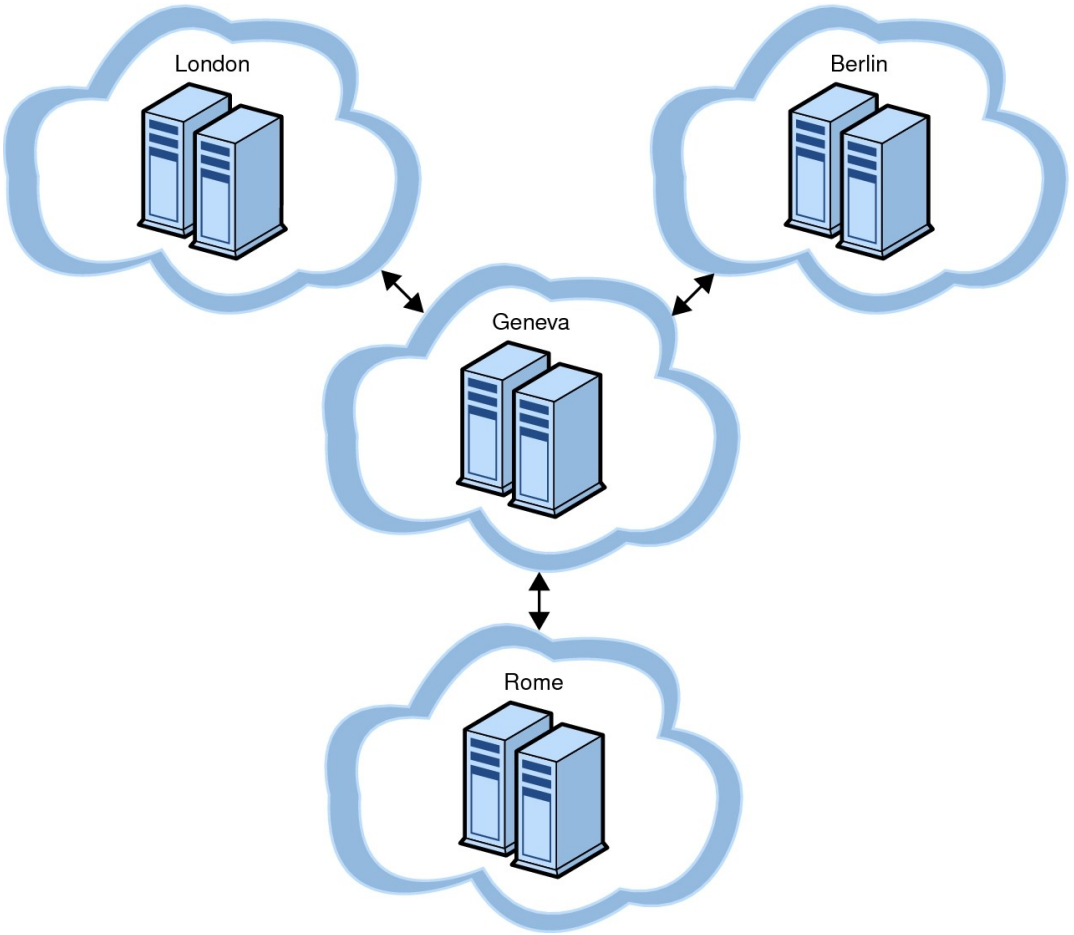
- Secuencial — los componentes se ejecutan en secuencia.
- Jerárquica — un componente llama a los servicios de otro.
- Aditiva — las interfaces de dos componentes se juntan para crear un nuevo componente.



Composición de componentes

- Problemas de adaptabilidad se solucionan con adaptadores (wrappers).
- Para resolver posibles conflictos se puede pensar:
  1) ¿Qué composición es más efectiva para entregar los requerimientos funcionales del sistema?
  2) ¿Qué composición facilitará la adaptación del componente cuando cambien los requerimientos?
  3) ¿Cuáles serán las propiedades emergentes del sistema compuesto? (rendimiento y confiabilidad).
- Utilizar el principio de separación de intereses — cada componente tiene un papel claramente definido y que los roles no se traslapen.

Ingeniería de software distribuido



Sistemas distribuidos

- Prácticamente todos los grandes sistemas de sw son distribuidos.
  - "... una colección de computadoras independientes que aparece para el usuario como un único sistema coherente".
- El procesamiento de información se distribuye en varias computadoras en lugar de limitarse a una sola máquina.
- Características:
  - Compartir recursos → compartir recursos de hardware y software.
  - Apertura → uso de equipos y software de diferentes proveedores.
  - Concurrencia → procesamiento concurrente para mejorar rendimiento.
  - Escalabilidad → mayor rendimiento al agregar nuevos recursos.
  - Tolerancia a fallas → la capacidad de continuar en funcionamiento después de que se ha producido un error.

Sistemas distribuidos

- Los sistemas distribuidos son más complejos que los sistemas que se ejecutan en un solo procesador.
- La complejidad surge porque las diferentes partes del sistema se manejan de manera independiente como lo hace la red.
- No hay una sola autoridad a cargo del sistema, por lo que el control descendente es imposible.

Aspectos de diseño

- Transparencia
  - ¿En qué medida debería aparecer el sistema distribuido al usuario como un sistema único?
  - Idealmente queremos transparencia pero en la práctica es casi imposible por la gestión independiente y las demoras en la red. A veces es mejor advertir a los usuarios.
  - Recursos direccionados de forma lógica y no física
- Apertura
  - ¿Debe diseñarse un sistema utilizando protocolos estándar que respalden la interoperabilidad?
  - Desarrollo basado en estándares generalmente aceptados.
  - Implica desarrollo independiente en cualquier lenguaje.

Aspectos de diseño

- Escalabilidad
  - ¿Cómo se puede construir el sistema para que sea escalable?
  - Capacidad para ofrecer un servicio de alta calidad a medida que aumentan las demandas sobre el sistema
  - Tamaño (agregar más recursos), Distribución (dispersar geográficamente los componentes), Capacidad de administración.
  - Ampliar (scaling-up) → sistema más poderoso; escalar (scaling-out) → más instancias del sistema.
- Seguridad
  - ¿Cómo se pueden definir e implementar políticas de seguridad utilizables?
  - La cantidad de formas en que el sistema puede ser atacado aumenta significativamente, en comparación con los sistemas centralizados.
  - Porque diferentes organizaciones pueden poseer partes del sistema.

Aspectos de diseño

- Calidad del servicio
  - ¿Cómo se especifica la calidad del servicio?
  - La calidad de servicio (QoS) refleja la capacidad del sistema de entregar sus servicios de manera confiable y con un tiempo de respuesta y rendimiento aceptable para sus usuarios.
  - Es importante cuando el sistema maneja datos de tiempo crítico tales como transmisiones de sonido o video.
- Gestión de fallas
  - ¿Cómo se pueden detectar, contener y reparar las fallas del sistema?
  - En un sistema distribuido, es inevitable que se produzcan fallas, por lo que el sistema debe diseñarse para ser resistente a estas fallas.
  - Si un componente del sistema ha fallado tratar de continuar entregando tantos servicios como sea posible y si se puede recupearse.

Modelos de interacción

- Interacción procesal
- Una computadora llama a un servicio conocido ofrecido por otra computadora y espera una respuesta.
- Implementación RPC (remote procedure calls), solicitudes como si el otro fuera un componente loca (solución middleware). En Java RMI (remote method invocations).
- Interacción basada en mensajes
- La computadora que envía envía información sobre lo que se requiere a otra computadora. No hay necesidad de esperar una respuesta.
- Se crean mensajes que se mandan a traves del middleware. En este enfoque no es necesario conocer al otro.

Middleware

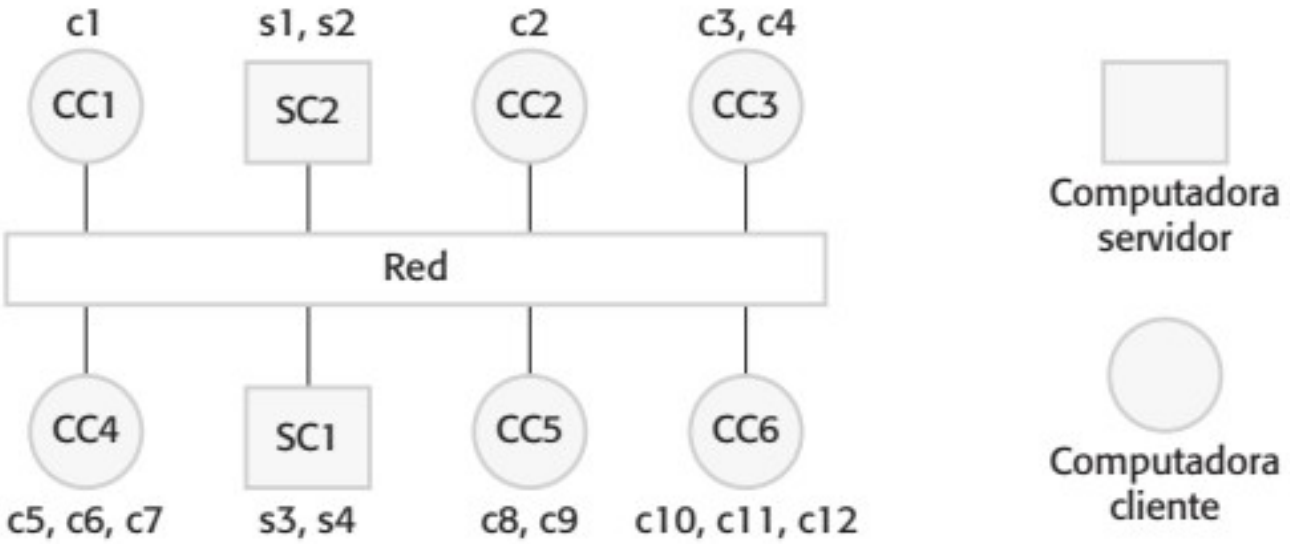
- Los componentes en un sistema distribuido pueden implementarse en diferentes lenguajes de programación y pueden ejecutarse en tipos de procesador completamente diferentes. Los modelos de datos, la representación de información y los protocolos de comunicación pueden ser diferentes.
- Middleware es un software que puede administrar estas diversas partes y garantizar que puedan comunicarse e intercambiar datos.

Soporte de middleware

- Soporte de interacción — coordina las interacciones entre los diferentes componentes en el sistema
  - No es necesario que los componentes conozcan las ubicaciones físicas de otros componentes.
- Provisión de servicios comunes — proporciona implementaciones reutilizables de servicios que pueden ser requeridas por varios componentes en el sistema distribuido.
  - Los componentes pueden interactuar fácilmente y proporcionar servicios al usuario de manera consistente.

Computación cliente-servidor

- Los sistemas distribuidos a los que se accede a través de Internet normalmente están organizados como sistemas cliente-servidor.
- La computadora remota proporciona servicios, como el acceso a páginas web, que están disponibles para clientes externos.

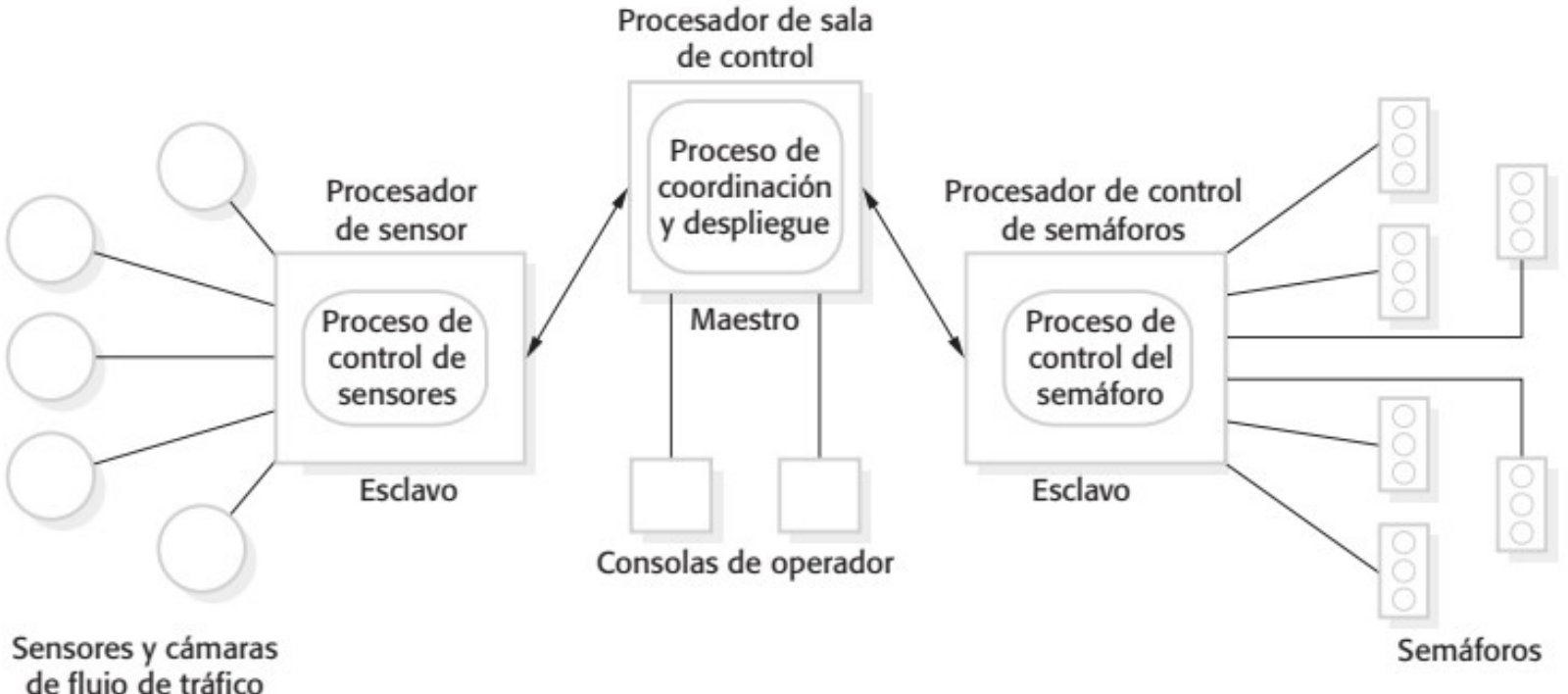


Patrones arquitectónicos para sistemas distribuidos

- Maestro-esclavo
- Cliente-servidor de dos niveles
- Cliente-servidor multinivel
- Componentes distribuidos
- Entre pares (peer-to-peer)

→ Maestro-esclavo

- Se usan comúnmente en sistemas en tiempo real.
- El proceso 'maestro' generalmente es responsable del cálculo, la coordinación y las comunicaciones, y controla los procesos 'esclavos'.
- Los procesos 'esclavos' están dedicados a acciones específicas, como la adquisición de datos de una matriz de sensores.



→ Cliente-servidor de dos niveles

- El sistema se implementa como un solo servidor lógico más un número indefinido de clientes que usan ese servidor.
- Cliente ligero → la capa de presentación se implementa en el cliente y todas las demás capas (gestión de datos, procesamiento de aplicaciones y base de datos) se implementan en un servidor.

  — Simple de manejar por clientes, soporta manejo de sistemas legados, fuerte carga en red y servidor.
- Cliente pesado → parte o todo el procesamiento de la aplicación se lleva a cabo en el cliente. La administración de datos y las funciones de la base de datos se implementan en el servidor.

  — Adecuado cuando sabemos capacidades de los clientes y se pueden utilizar, más difícil de gestionar y se debe instalar el sw en los clientes.
- Las fronteras son borrosas, javascript ha permitido clientes más pesados sin gestión adicional, pocos cliente finos actualmente.

→ Cliente-servidor multinivel

- Las diferentes capas del sistema, a saber, presentación, administración de datos, procesamiento de aplicaciones y base de datos, son procesos separados que pueden ejecutarse en diferentes procesadores.
- Son más escalables.
- Se usan cuando hay varias bases de datos (se agrega servidor de integración).
- El procesamiento puede distribuirse entre la lógica de la aplicación y los servidores de gestión de datos → respuesta más rápida.

→ Componentes distribuidos

- No hay distinción entre clientes y servidores.
- Cada entidad distribuible es un componente que proporciona servicios a otros componentes y recibe servicios de otros componentes.
- La comunicación de componentes se realiza a través de un sistema de middleware.

→ Componentes distribuidos

- (+) Permite al diseñador retrasar las decisiones sobre dónde y cómo se deben proporcionar los servicios.
- (+) Es una arquitectura muy abierta que permite agregar nuevos recursos según sea necesario.
- (+) El sistema es flexible y escalable.
- (-) Es posible reconfigurar el sistema de forma dinámica según sea necesario.
- (-) Son más complejos de diseñar que los sistemas cliente-servidor.
- (-) El middleware estandarizado nunca ha sido aceptado por la comunidad.
- → las arquitecturas orientadas a servicios están reemplazando las arquitecturas de componentes distribuidos en muchas situaciones.

→ Entre pares (peer-to-peer, P2P)

- Son sistemas descentralizados donde los cálculos pueden ser llevados a cabo por cualquier nodo en la red.
- El sistema general está diseñado para aprovechar la capacidad de cómputo y el almacenamiento de una gran cantidad de computadoras en red.
- La mayoría de los sistemas p2p han sido sistemas personales, pero el uso comercial de esta tecnología es cada vez mayor.
- Cuando → sistema de cómputo intensivo y podemos distribuir procesamiento, sistema de intercambio de información sin necesidad de gestión de información centralizada.
- Arq descentralizadas (todos pares, más redundante, más robusto) o semicentralizadas (uno o más servidores, menos carga de red).

Software como servicio (SaaS)

- El software como servicio (SaaS) implica alojar el software de forma remota y proporcionar acceso a él a través de Internet.
- Se implementa en un servidor (o más comúnmente en varios servidores) y se accede a él a través de un navegador web. No se implementa en una PC local.
- El software es propiedad y está administrado por un proveedor de software, en lugar de las organizaciones que usan el software.
- Los usuarios pueden pagar por el software de acuerdo con la cantidad de uso que hacen de él o mediante una suscripción anual o mensual.

SaaS y SOA

- SaaS es una forma de proporcionar funcionalidad en un servidor remoto, con acceso de clientes mediante un navegador Web. El servidor conserva los datos y el estado del usuario durante una sesión de interacción. Por lo regular, las transacciones son largas.
- SOA (Arquitecturas orientadas a servicios) es un enfoque a la estructuración de un sistema de software como un conjunto de servicios independientes, sin estado. Éstos pueden proporcionarse mediante múltiples proveedores y distribuirse. Por lo general, las transacciones son transacciones cortas.
- SaaS es una forma de entregar funcionalidad de aplicación a los usuarios, mientras que SOA es una tecnología de implementación para sistemas de aplicación.

Software como servicio

- El software como servicio (SaaS) implica alojar el software de forma remota y proporcionar acceso a él a través de Internet.
- Se implementa en un servidor (o más comúnmente en varios servidores) y se accede a él a través de un navegador web. No se implementa en una PC local.
- El software es propiedad y está administrado por un proveedor de software, en lugar de las organizaciones que usan el software.
- Los usuarios pueden pagar por el software de acuerdo con la cantidad de uso que hacen de él o mediante una suscripción anual o mensual.