Tema 1. Arquitectura Cliente/Servidor

SCS – Sistemas Cliente/Servidor
4º informática

http://ccia.ei.uvigo.es/docencia/SCS

27 de septiembre de 2009

1.1 Sistemas Distribuidos

1.1.1 Definición y características

(a) Definiciones

computación distribuida. "Modelo de computación adaptado a la ejecución de programas en sistemas distribuidos"

sistema distribuido. "Sistema informático compuesto por un conjunto de nodos de procesamiento comunicados y coordinados mediante una red que permite el intercambio de mensajes entre los mismos"

- Esquema

- *Middleware*: Abstracción que oculta las características específicas de cada nodo (hardware y S.O.)y proporciona un interfaz común que permite la interacción entre los componentes del sistema
  - Conjunto de componentes y tecnologías que se extienden a través de los distintos equipos formando una "capa" común que ofrece una interfaz que uniformiza las características de los equipos, los dota de coherencia y permite la comunicación entre ellos
  - Misión: ofrecer abstracción + dotar de transparencia
  - Ejemplos:
    - paso de mensajes mediate sockets
    - invocación remota de procedimientos
    - objetos distribuidos

(b) Características típicas

- **Finalidad:** hacer más accesibles los recursos disponibles en el sistema distribuido
- Características:
  - **Transparentes**: ocultan al desarrollador el hecho de que los recursos estén distribuidos
    - *transparencia de acceso*: ocultar las diferencias en la representación de los datos (formatos) y en el acceso a los recursos
    - *transparencia de ubicación*: ocultar los aspectos relativos a ubicación real de un recurso
    - *transparencia de migración*: ocultar el hecho de que un recurso pueda ser movido a otra ubicación
    - *transparencia de replicación*: ocultar que un recurso pueda estar replicado e distintas ubicaciones
    - *transparencia de concurrencia*: permitir que un recurso se comparta por distintos usuarios/aplicaciones de forma simultánea
    - *transparencia ante fallos*: ocultar los fallos y la recuperación de los mismos
  - **Abiertos**: soportan la interconexión con otros sistemas
    - Los servicios ofrecidos siguen reglas bien definidas y conocidas (protocolos) que describen su sintaxis (forma de usarlos) y su semántica (qué hacen/cúal es su resultado)
    - Son "independientes" de tecnologías y recursos concretos
    - Soportan la interoperabilidad y portabilidad de los recursos
  - **Escalables**: capaces de hacer frente al crecimiento de la demanda
    - Posibilidad de añadir recursos y usuarios al sistema sin afectar a la consistencia del sistema
    - **Técnicas:** balanceo de cargas, distribución y replicación (cachés) de datos, recursos, equipos, ...
    - Relacionado con tolerancia a fallos: fallos de nodos concretos no suponen que el sistema distribuido quede inoperativo

(c) Ventajas e inconvenientes

- **Ventajas**
  - Economía: compartición de recursos costosos $\Rightarrow$ ahorro de costes
  - Flexibilidad: independiencia de tecnologías/desarrollos, adaptabilidad
  - Fiabilidad: tolerancia a fallos $\longrightarrow$ recursos críticos pueden ser replicados
  - Escalabilidad: no limitado a recursos de un único equipo $\longrightarrow$ posibilidad de introducción de nuevos nodos
- **Inconvenientes**
  - Dificultad en el desarrollo del software
  - Limitaciones de las redes (ancho de banda, latencias, ...)
  - Problemas de seguridad: control de accesos, confidencialidad, integridad, etc

1.1.2 Arquitecturas de sistemas distribuidos

- En base a su **topología**:
  - **Arquitecturas centralizadas**: los componentes del sistema presentan diferentes roles
    - Paradigma cliente-servidor: dos tipos de elementos
      servidor que gestiona un recurso y atiende peticiones de clientes
    - Arquitecturas multicapa (*n-tier*): generalización del anterior
      los componentes pueden emitir peticiones y responder a las peticiones sobre los recursos concretos que gestionan (clientes y servidores a la vez)
  - **Arquitecturas descentralizadas**: todos los componentes tienen las mismas responsabilidades y funciones
    - Sistemas entre iguales (*peer-to-peer*)
- En base a su **estructura**:
  - **Basados en capas**: elementos del sistema organizados en capas "especializadas", donde la comunicación está limitada a componentes de capas contiguas conforme a un flujo preestablecido
  - **Basados en objetos (componentes)**: elementos del sistema son objetos autónomos que pueden intercambiar mensajes (llamadas a métodos) a través de la red de comunicaciones
  - **Basados en eventos**: no sigue esquema petición-respuesta
    elementos del sistema tienen acceso a un "bus" común donde se envían/recogen eventos a los que se responde
    Operaciones: suscripción, publicación, notificación

1.2 Paradigma Cliente/Servidor

- Patrón arquitectónico para el desarrollo de sistemas distribuidos.
  - Distribuye una aplicación entre 2 o más componentes especializados cuya ejecución se distribuye entre 1 o más equipos.
  - Define dos tipos de entidades diferenciadas (asimétricas) que se responsabilizan de acciones diferentes: clientes y servidores
    - Especifica 2 tipos de procesos con roles diferenciados
  - Define un modelo de interacción basado en el concepto de servicio implementado sobre un diálogo petición-respuesta
    - Cliente inicia el diálogo mediante el envío de peticiones
    - Servidor presta el servicio y responde las peticiones recibidas
  - Especifica el modo en que se sincronizan los procesos
    - **Cliente** (parte activa)
      - ◇ demanda servicios a los servidores
      - ◇ se asume que cada petición deberá obtener respuesta
      - ◇ diseñado para soportar la interacción con el usuario final
    - **Servidor** (parte pasiva)
      - ◇ espera las peticiones de los clientes
      - ◇ procesa esas peticiones y envía una respuesta
      - ◇ diseño orientado a maximizar la eficiencia
- Posibilidad de aplicar el patrón cliente-servidor en múltiples niveles de abstracción dentro de un mismo sistema distribuidos (arquitecturas multinivel [n-*tier*])

1.3 Componentes de los Sistemas Cliente/Servidor

(a) Características de los clientes

- Componente del sistema que interactúa con el usuario
- No comparte sus recursos con otros clientes (en general)
- No suelen tener restricciones especiales respecto a rendimiento, fiabilidad y escalabilidad
  - no suele requerir equipos de altas prestaciones
  - fallo en un cliente no afecta al resto del sistema
- Debe dar soporte a restricciones relativas a ergonomía (facilidad de uso) y seguridad (evitar comprometer los demás componentes)

(b) Características de los servidores

- Componente del sistema que presta servicios a los clientes
- Gestiona y comparte sus recursos con los clientes a los que sirve
- Suele tener restricciones especiales respecto a redimiento, fiabilidad, escalabilidad y seguridad
  - capacidad suficiente para atender múltiples clientes
  - fallos en el servidor son críticos e invalidan el sistema
  - el núm. de clientes (peticiones) puede ser muy variable y aumentar si así se requiere
  - evitar comprometer la seguridad de los recursos o datos gestionados y de los clientes

(c) Características del *middleware*

- Componente del sistema que da unidad y abstrae las peculiaridades de las plataformas (hardware y S.O.) de clientes y servidor
- Gestiona los aspecto de bajo nivel para ofrecer un interfaz común y coherente para el desarrollo de clientes y servidores (**abstracción**)
- Misiones principales:
  - dar soporte al envío/recepción de mensajes
  - adaptación del formato de la información intercambiada (*marshaling*/aplanamiento)
  - localización y acceso transparente a recursos/servicios: nombrado, direccionamiento
  - soporte al paradigma de abstracción: *stubs/skeletons* en RPC, RMI, CORBA, etc
  - otros servicios: seguridad, replicación, control concurrencia, ...
- Combina e integra:
  - servicios de bajo nivel (S.O.): seguridad, autorización/permisos, ficheros distribuidos, ...
  - servicios de red: librerías, pila de transporte TCP/IP,...
  - servicios específicos de la abstracción que ofrece: acceso a datos, *portmapper*, ORB, ...
- Aproximaciones: (de menor a mayor nivel de abstracción)
  - *middleware* de paso de mensajes: interfaz de *sockets* (esquema petición-respuesta)
  - *middleware* de invocación de métodos remota: RPC (llamadas a funciones)
  - *moddleware* de objetos distribuidos: RMI, CORBA (interacción entre objetos)

1.4 Modelos y tipologías

Esquema abstracto de aplicaciones distribuidas genéricas (capas)

Se corresponden con las funciones típicas en un sistema

- **Capa de presentación** (interfaz de usuario)
  - interacciona con el usuario, presenta los datos y recibe las entadas
- **Capa de aplicación/negocio** (lógica de aplicación)
  - responsable de las tareas propias de la aplicación concreta
  - implementa la lógica de la aplicación y aplica las reglas de negocio sobre los datos y las entradas de usuario
- **Capa de datos** (almacenamiento y acceso a datos)
  - responsable de la gestión y almacenamiento permanente de los datos

Cada tipo de sistema cliente-servidor distribuye esas capas de modo distinto entre los componentes cliente y servidor

1.4.1 Clientes ligeros vs. clientes pesados

Clasificación dependiendo de las responsabilidades asignadas al cliente

- **Cliente ligero** (*thin client*)
  - No implementa ningún aspecto de la lógica de aplicación
  - Simplemente actúa como intermediario entre usuario y servidor
    - recoge entradas (opcionalmente, las valida) y las envía al servidor
    - presenta datos y resultados del servidor
  - Normalmente, requisitos mínimos respecto a recursos hardware
  - Aumenta la complejidad del servidor (tendrá mayores responsabilidades)
  - Ejemplo: clientes basados en navegadores web (JSP, ASP, ...)
    - Capa de presentación repartida entre servidor (genera HTML "al vuelo") y cliente (navegador que lo presenta)
    - En últimos años surgen clientes ligeros ricos (tecnologías AJAX, RIA [*Rich Internet Application*])
      - ◇ clientes basados en navegadores web + soporte de interacciones complejas (Javascript, carga XML asíncrona, ...)
- **Cliente pesado** (*fat client*)
  - Implementa la mayor parte de la lógica de aplicación
  - Realiza procesamiento sobre datos de usuario antes de comunicar con servidor
  - Requiere equipos con capacidad de proceso y/o almacenamiento de datos
  - Servidor sencillo (responsabilidades mínimas, gestión datos)
  - Ejemplo: aplicación cliente contra servidor de base de datos
- **Cliente híbrido**
  - Implementación de la lógica de aplicación repartida entre cliente y servidor
  - Ejemplo: aplicación cliente contra servidor de base de datos con procedimientos almacenados

Comparación tipos de clientes

1.4.2 Arquitecturas *2-tier*, *3-tier* y *n-tier*

Clasific. en función de la ubicación física de las distintas funcionalidades

- **Modelo tradicional: *2-tier*** (cliente-servidor en 2 niveles)
  - Un único servidor atiende a múltiples clientes
  - Problemas
    - **escasa escalabilidad** en servidores con lógica de negocio compleja o con grandes bases de datos (difícil replicación, etc)
    - **rigidez**: modificaciones en la lógica de aplicación suponen grandes cambios en la totalidad de clientes
    - **difícil evolución** del servidor
  - Limitación principal: alto acoplamiento/dependencia del cliente respecto del servidor
  - Clientes ligeros, pesados o híbridos
- **Modelo *3-tier*** (cliente-servidor en 3 niveles)
  - Extensión del modelo tradicional que pretende aumentar el desacoplamiento entre servidor y clientes
  - Introduce un nivel intermedio (separa servidor en 2 componentes)
    - cliente dedicado casi exclusivamente a interfaz de usuario
    - servidor de datos comparte con servidor del nivel intermedio la lógica de la aplicación
    - el reparto preciso depende del modelo concreto seguido
  - Clientes ligeros o híbridos
- **Modelos *n-tier* ó *multi-tier*** (cliente-servidor en n niveles)
  - Generalización del modelo 3-tier (añade nuevas capas)
  - La lógica de aplicación se reparte en diferentes capas/niveles ubicadas entre el cliente y los datos
  - Las capas intermedias se proporcionan servicios entre si.
    - cada nivel se comunica sólo con los niveles contiguos a través de interfaces bien definidos
    - capa k ofrece servicios a capa $k-1$ y demanda servicios de capa $k+1$
  - Estructura típica en sistemas basados en componentes distribuidos (objetos distribuidos)
  - Clientes ligeros o híbridos

- Beneficios de las arquitecturas multinivel
  - Elementos críticos de la lógica de negocio ubicados en nivel medio
    - más cercanos a la capa de datos $\longrightarrow$ eficiencia de acceso
    - sólo los datos realmente necesarios acaban llegando al cliente
  - Mayor flexibilidad y modularidad
  - Escalabilidad: facilita añadir recursos para soportar mayor núm. de clientes
  - Extensibilidad: facilita añadir nuevas funcionalidades al sistema sin afectar a los clientes existentes
  - Seguridad: facilidad para propagar autenticación y permisos a través de las distintas capas
  - Facilidades de desarrollo y administración:
    - reusabilidad de componentes
    - aislamiento frente a cambios en otras capas
    - independencia frente a cambios en base de datos
- Desventajas de las arquitecturas multinivel
  - Complejidad: mayor núm. de elementos hardware y software a definir, gestionar y mantener
    - interacciones complejas entre componentes
    - dificultad para detectar, asilar y coregir fallos
  - Coste de comunicaciones: mayor latencia y consumo de ancho de banda (atravesar capas distribuidas por la red)
  - Coste de mantenimiento: al crecer las capas aumenta el coste y la dificultad de instalación y mantenimiento