Diseño del Sistema

Índice

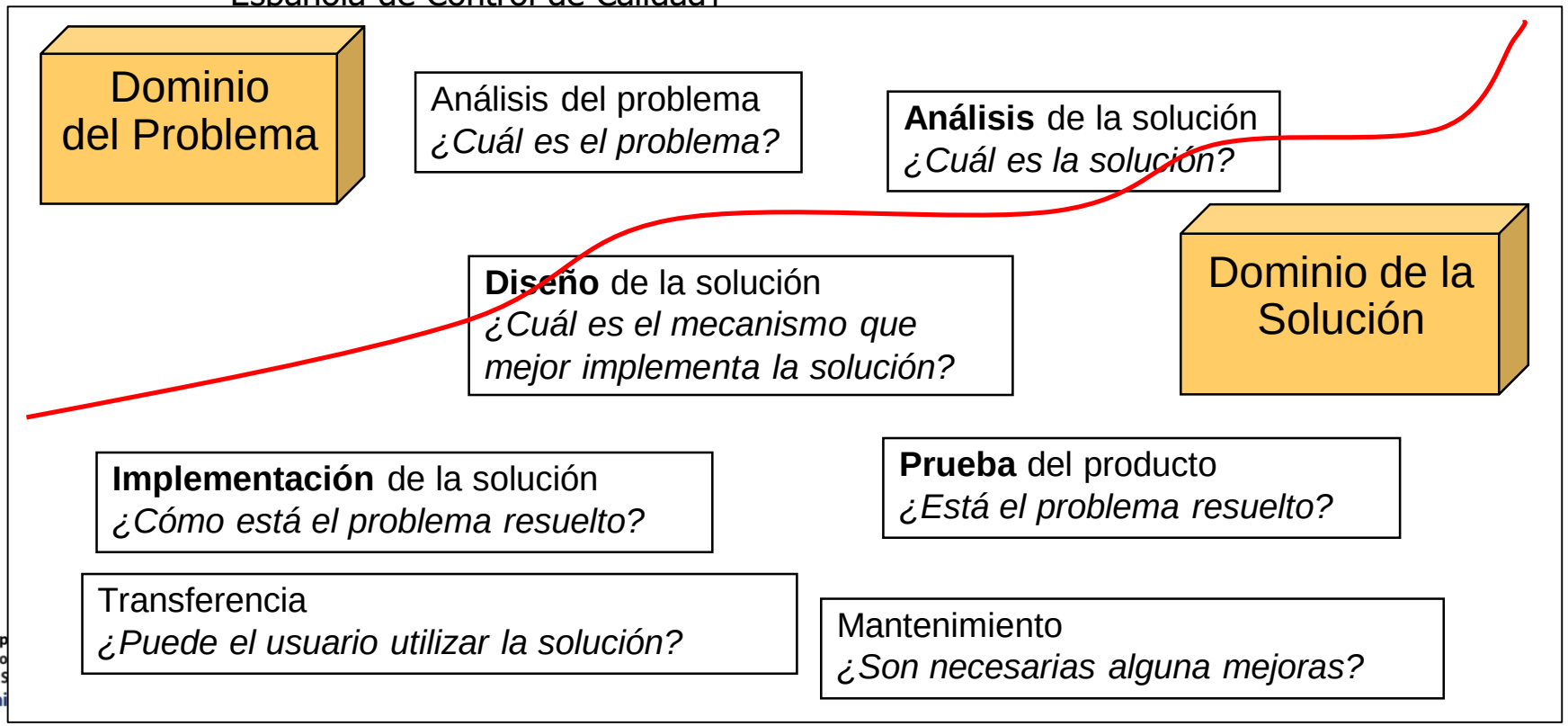
1. Introducción

Análisis	**Diseño**
El Análisis especifica lo QUÉ el sistema debe hacer.	El Diseño establece CÓMO alcanzar el objetivo.
Se centra en el dominio de la aplicación.	Se centra en el dominio de la solución.
Realiza una descripción del sistema desde un punto de vista "*lógico*".	Realiza una descripción del sistema desde un punto de vista "*físico*".

Diseño

- Una definición: "Es el proceso de definición de la arquitectura software: componentes, módulos, interfaces, procedimientos de prueba y datos de un sistema que se crean para satisfacer unos requisitos especificados"

[Glosario de Términos de Calidad e Ingeniería del Software de la Asociación Española de Control de Calidad]



¿Por qué es difícil el diseño?

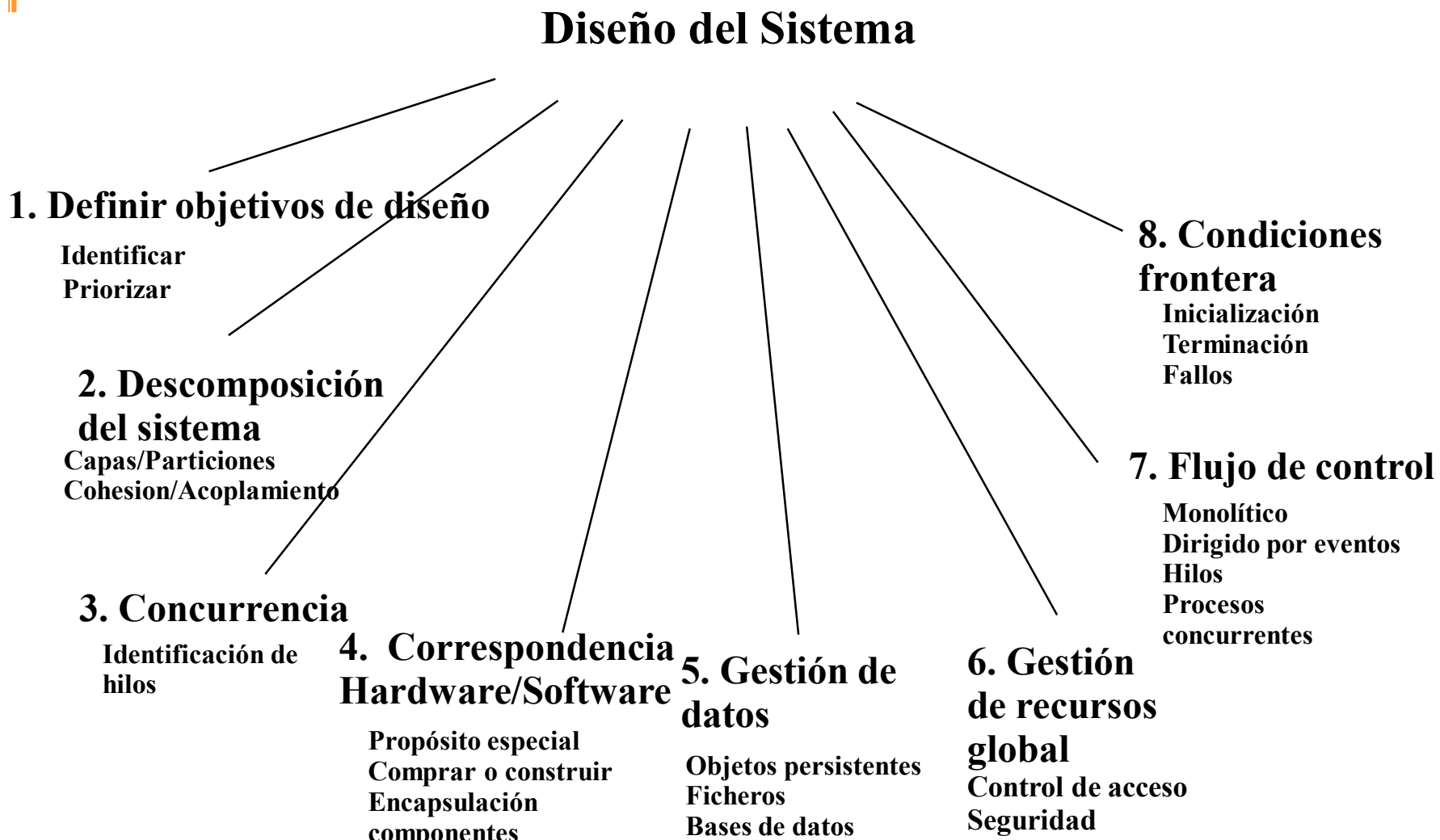
- El conocimiento del diseño es un objetivo en movimiento
- Las razones para decisiones de diseño cambian muy rápidamente
  - Tiempo de vida media del conocimiento en ingeniería del software: alrededor 3-5 años
  - Lo que enseñamos hoy puede estar desfasado en 3 años
  - El coste de hardware baja rápidamente

Diseño del sistema

- Normalmente el diseño se divide en:
  - Diseño del sistema
    - Definir los aspectos generales del sistema
  - Diseño de objetos
    - Definir los detalles, tomar decisiones de implementación
- Otras posibles clasificaciones

Diseño del sistema	Diseño arquitectural
	Diseño de Interfaz de Usuario (asignatura IPO)
Diseño de objetos	Diseño de estructuras de datos
	Diseño de algoritmos

Tareas del diseño del sistema



El diseño de sistemas no es algorítmico

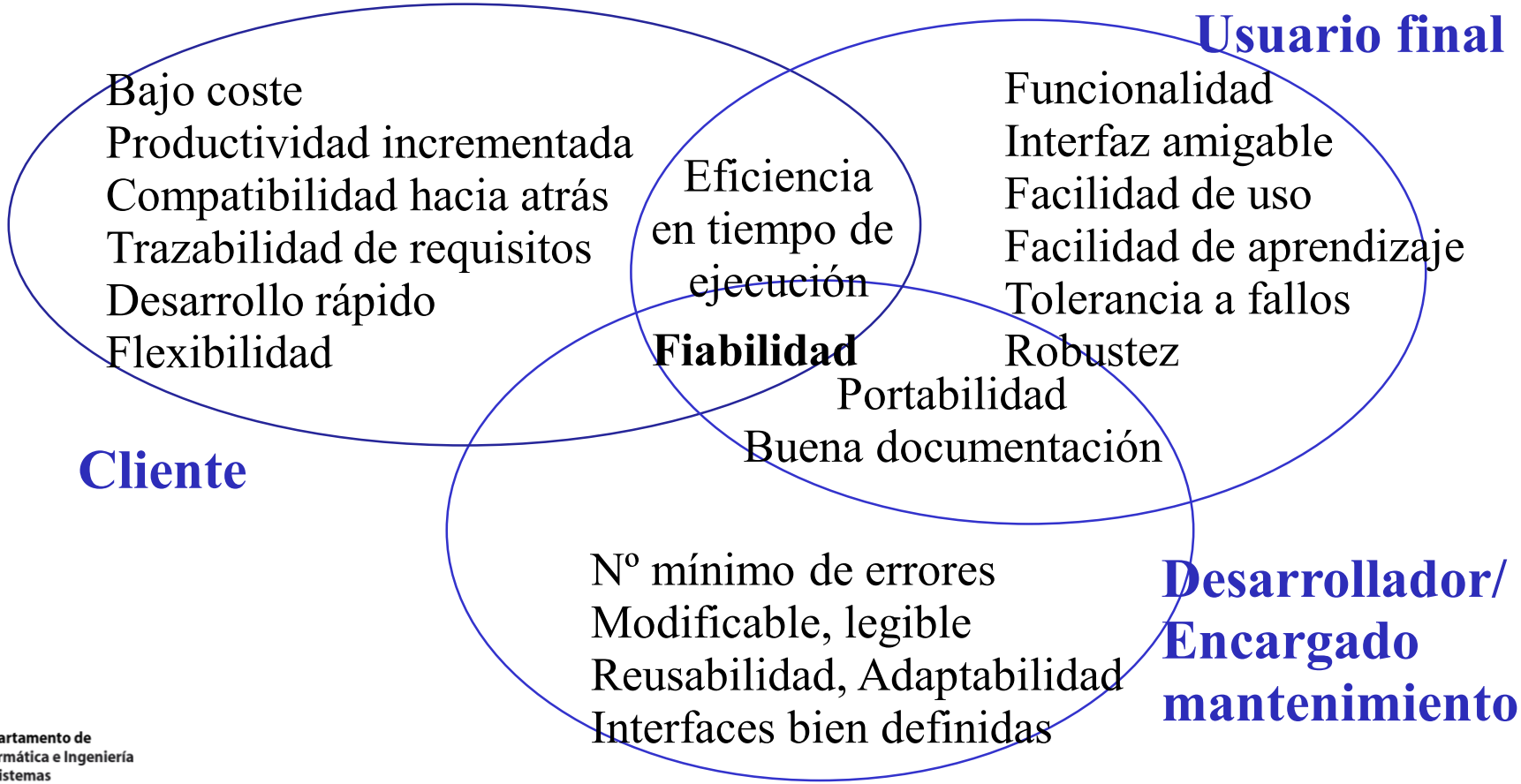
- Definir los objetivos de diseño
  - Se identifican y priorizan las cualidades/características del sistema que se deberían optimizar
- Diseñar la descomposición inicial en subsistemas ("divide y vencerás")
  - Descomponer en pequeñas partes en base a casos de uso y modelos del análisis (pasar de objetos a un nivel organizativo superior)
  - Utilizar estilos arquitecturales estándar como punto de partida
- Refinar la descomposición en subsistemas para abordar los objetivos de diseño
  - La descomposición inicial no suele satisfacer todos los objetivos
  - Se debe refinar hasta que satisfagan todos los objetivos

Transformación del modelo del análisis en un modelo de diseño de sistemas

- Cómo resultado del análisis se ha descrito el sistema desde el punto de vista de los actores (comunicación entre clientes y desarrolladores)
  - Modelo de casos de uso
  - Requisitos no funcionales y restricciones
  - Modelo de objetos
  - Diagramas de secuencia
- Los resultados esperados del diseño de sistemas
  - Objetivos del diseño
  - Arquitectura del software
  - Casos de uso frontera

2. Definición de los objetivos de diseño

- A partir de los requisitos no funcionales y las restricciones, se identifican y priorizan las cualidades/características del sistema que se deberían optimizar



Decisiones de diseño típicas (sopesar alternativas)

- Funcionalidad vs usabilidad
- Coste vs robustez
- Eficiencia vs portabilidad
- Desarrollo rápido vs funcionalidad
- Coste vs reusabilidad
- Compatibilidad hacia atrás vs legibilidad

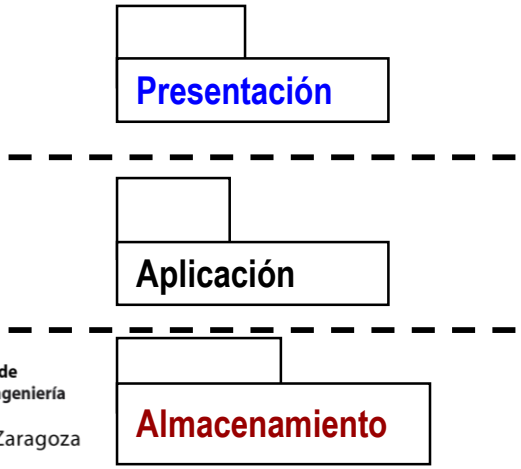
3. Descomposición en subsistemas

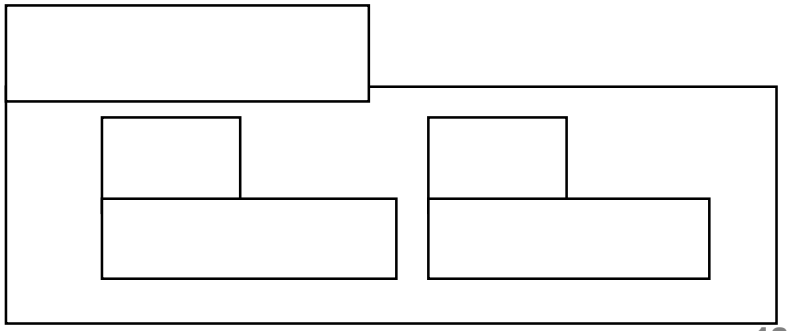
- 3.1. Subsistemas, servicios e interfaces de subsistema
- 3.2. Capas y particiones
- 3.3. Patrones arquitecturales

"La modularidad (descomposición en subsistemas) facilita la flexibilidad, mantenibilidad y reusabilidad."

3.1. Subsistemas, servicios, interfaces de susbsistema

- Subsistema
  - Colección de clases, asociaciones, operaciones, eventos y restricciones que están interrelacionadas
  - Origen de subsistemas: objetos y clases UML
- Cada subsistema se puede representar en forma de paquete UML
  - A su vez, los subsistemas se pueden organizar internamente en paquetes





Servicios

- Servicio:
  - Grupo de operaciones proporcionadas por un subsistema que comparten un propósito común
  - Origen de los servicios: casos de uso de subsistemas
  - Los servicios se definen en el diseño del sistema
- Ejemplo: Servicio de un subsistema de "Notificación":
  - *lookupChannel()*
  - *subscribeToChannel()*
  - *sendNotice()*
  - *unsubscribeFromChannel()*

Interfaces de subsistema

- Un servicio se especifica mediante una interfaz de subsistema:
  - Especifica la interacción y el flujo de información de/a las fronteras del subsistema, pero no dentro del subsistema
  - Deberían estar bien definidas y ser pequeñas
  - A menudo denominadas API (Application Programming Interface - Interfaz de Programación de Aplicación)
    - pero este término se debería usar en la implementación, no en el diseño del sistema
- Las interfaces de subsistema se definen durante el diseño de objetos

Criterios de selección de subsistemas

- Objetivo deseable: reducir complejidad cuando se producen cambios
- La cohesión mide la dependencia entre clases
  - Alta cohesión: Las clases en el subsistema realizan tareas similares y están relacionadas entre ellas (vía asociaciones)
  - Baja cohesión: muchas clases auxiliares y heterogéneas, no hay asociaciones
- El acoplamiento mide la dependencia entre subsistemas
  - Acoplamiento alto: cambios de un subsistema tendrán un gran impacto en otros subsistemas (cambio de modelo, recompilación masiva, etc.)
  - Acoplamiento bajo: Un cambio en un subsistema no afecta a cualquier otro subsistema
- Los subsistemas deberían tener la máxima cohesión y el mínimo acoplamiento como sea posible
  - Agrupar elementos que proporcionen servicios relacionados, con un grado "alto" de acoplamiento y colaboración (cohesión)
  - El acoplamiento y colaboración entre (elementos de) diferentes subsistemas debe de ser bajo

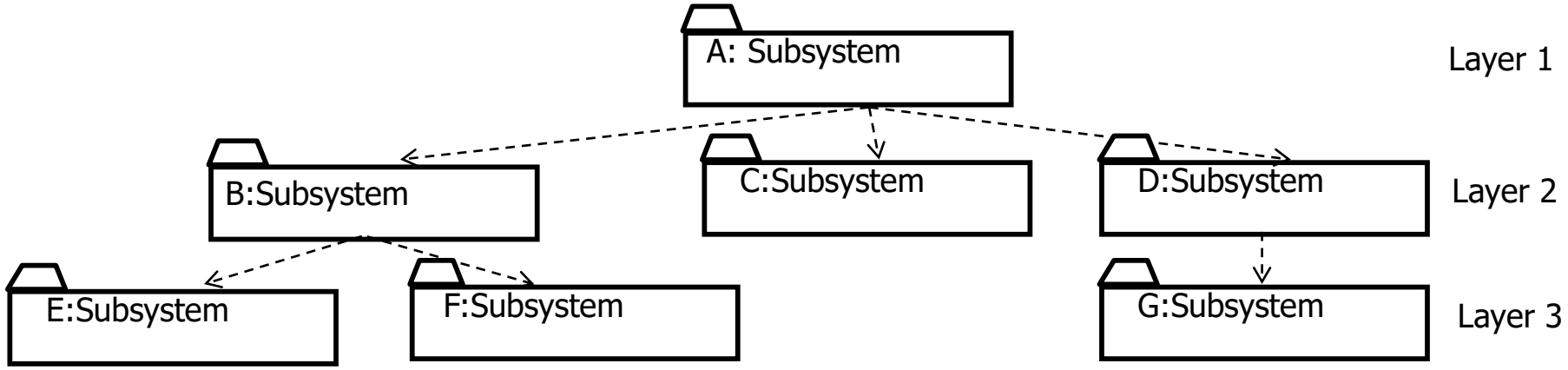
3.2. Capas y particiones

- La creación de Capas y Particiones son técnicas para conseguir un acoplamiento bajo de subsistemas
  - Un sistema grande se descompone en subsistemas usando tanto capas como particiones
- Capas
  - Dividen el espacio vertical del sistema
  - Una capa es un subsistema que proporciona servicios de subsistema a una capa superior (con mayor nivel de abstracción)
    - Una capa solo depende de las capas inferiores
    - Una capa no tiene conocimiento de las capas superiores
  - Representan el sistema desde un determinado punto de vista (nivel de detalle).
    - Ej: Sistema = Presentación → Aplicación → Almacenamiento

Capas y Particiones

- Particiones
  - Dividen verticalmente (el espacio horizontal de) un sistema en varios subsistemas independientes (o escasamente acoplados) que proporcionan servicios en el mismo nivel de abstracción
  - Dividen una capa en una serie de subsistemas en función de la naturaleza u objetivos de sus elementos
    - Ej: Capa de servicios = Seguridad + Comunicaciones + Informes

Descomposición de subsistemas en capas



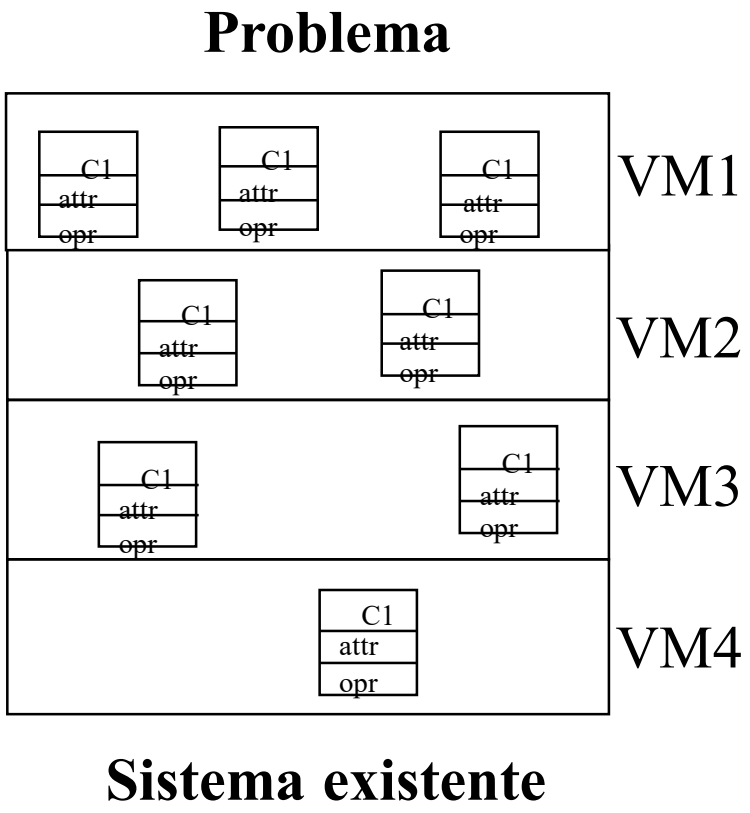
- Recomendaciones para descomposición de subsistemas:
  - No más de 7+/-2 subsistemas
    - Más subsistemas incrementan la cohesión pero también la complejidad (más servicios)
  - No más de 4+/-2 capas, usar 3 capas (bueno)

Relaciones entre subsistemas

- Relación entre capas
  - Capa A "llama a" capa B (tiempo de ejecución)
  - Capa A "depende de" capa B (dependencia "make", tiempo de compilación)
- Relación entre particiones
  - Los subsistemas tienen conocimiento mutuo pero no profundo entre ellos
  - La partición A "llama a" la partición B y la partición B "llama a" la partición A

Las arquitecturas multicapa no son un concepto nuevo

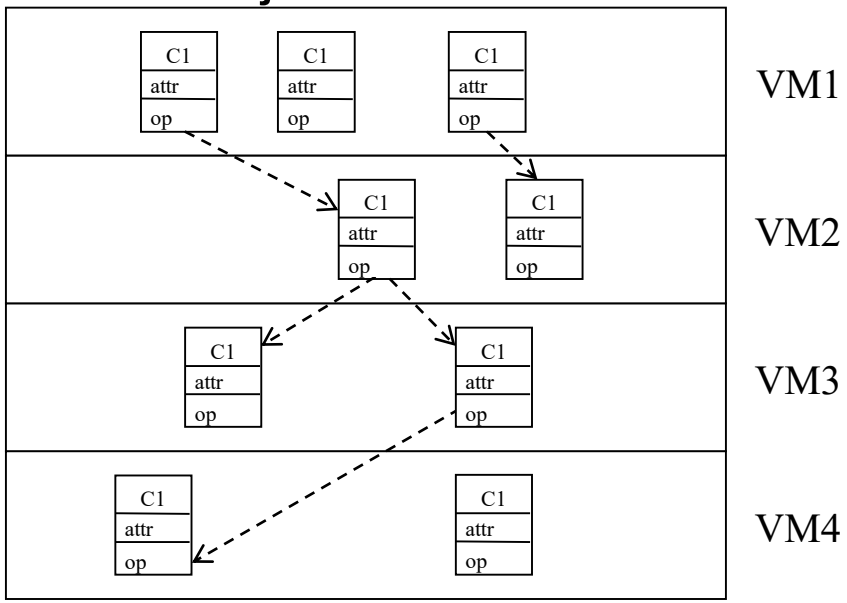
- Dijkstra: Sistema operativo T.H.E. (1965)
  - Un sistema debería desarrollarse mediante un conjunto ordenado de **máquinas virtuales**, cada una construida en base a las que están debajo.
  - Cada capa es un subsistema denominado "máquina virtual" que "proporciona servicios para" una máquina virtual superior



Visibilidad entre Capas

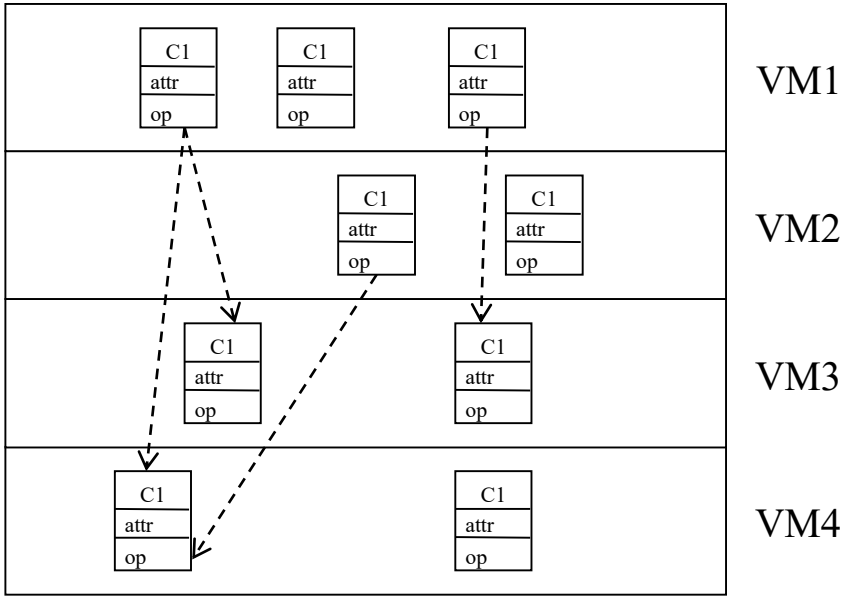
Arquitectura Cerrada

- Una capa solo puede invocar operaciones de la capa inmediatamente inferior
- Objetivo de diseño: Alta mantenibilidad, flexibilidad, portabilidad
- Ventaja: Robustez

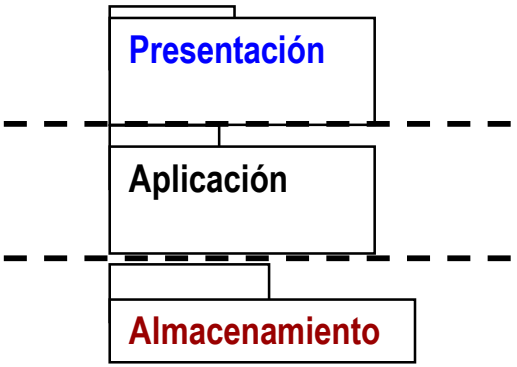


Arquitectura Abierta

- Una capa puede invocar operaciones de cualquier capa inferior
- Objetivo de diseño: eficiencia en tiempo de ejecución
- Ventaja: Eficiencia

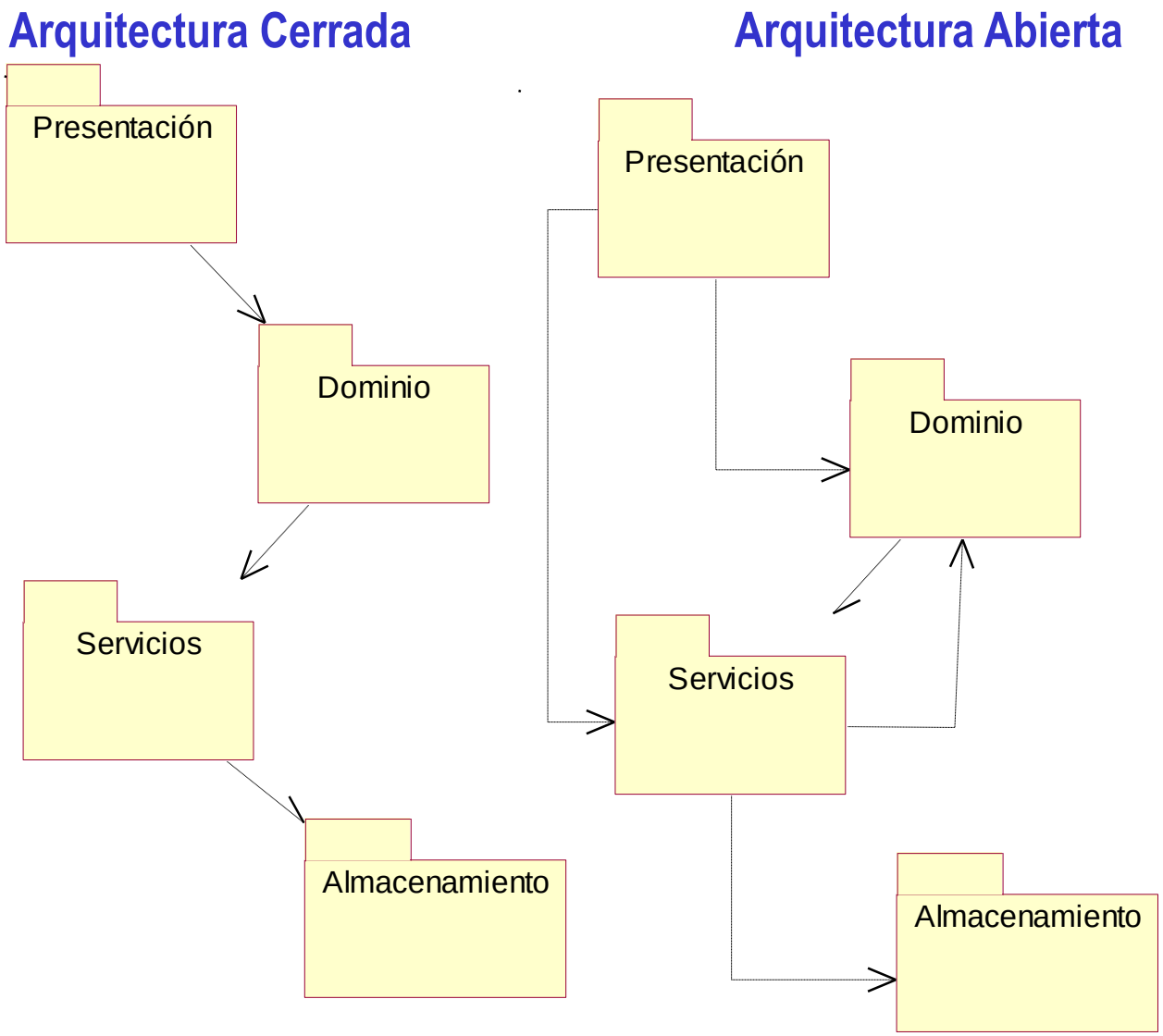


Ejemplos de arquitecturas multicapa



Descomposición de la capa de Aplicación en, al menos 2 capas:

- Clases propias del dominio del problema
- Clases que proporcionan servicios: seguridad, informes, acceso a BBDD,...



Arquitectura multicapa, ventajas e inconvenientes

Ventajas

- Los sistemas en capas son jerárquicos
  - la jerarquía reduce la complejidad (bajo acoplamiento)
- Separación de los componentes de la aplicación:
  - Permite reutilizarlos en otros sistemas (similares)
- Asignación de las diferentes capas a diferentes nodos y/o procesos:
  - Mejora el rendimiento del sistema y facilita la implantación
- Desarrollo independiente y especializado de las capas:
  - Desarrollo en paralelo

Inconvenientes

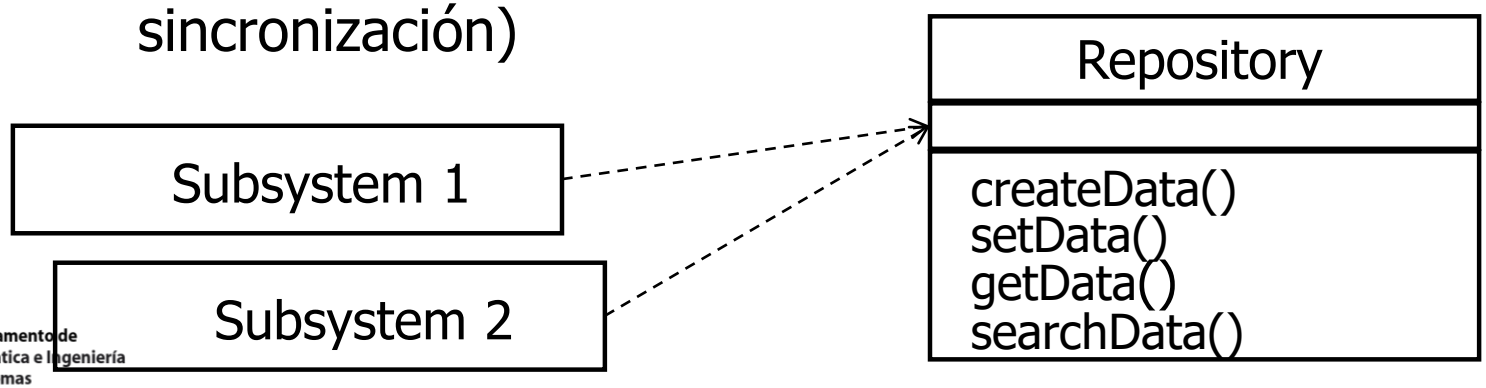
- Los sistemas en capas tienen a menudo el problema del “huevo y la gallina”
  - Ejemplo: Probar una capa A que depende de otra capa B (inferior) que no se ha implementado todavía

3.3. Patrones arquitecturales

- Conforme se incrementa la complejidad de los sistemas, se hace crítica la especificación de la descomposición del sistema
- Arquitectura del software
  - Concepto que incluye la descomposición del sistema, el flujo de control global, las políticas de manejo de errores y los protocolos de comunicación entre subsistemas
- Patrones para arquitectura de software
  - Depósito (Repository)
  - Modelo/Vista/Controlador (Model/View/Controller)
  - Cliente/Servidor
  - Par a Par (Peer-To-Peer)

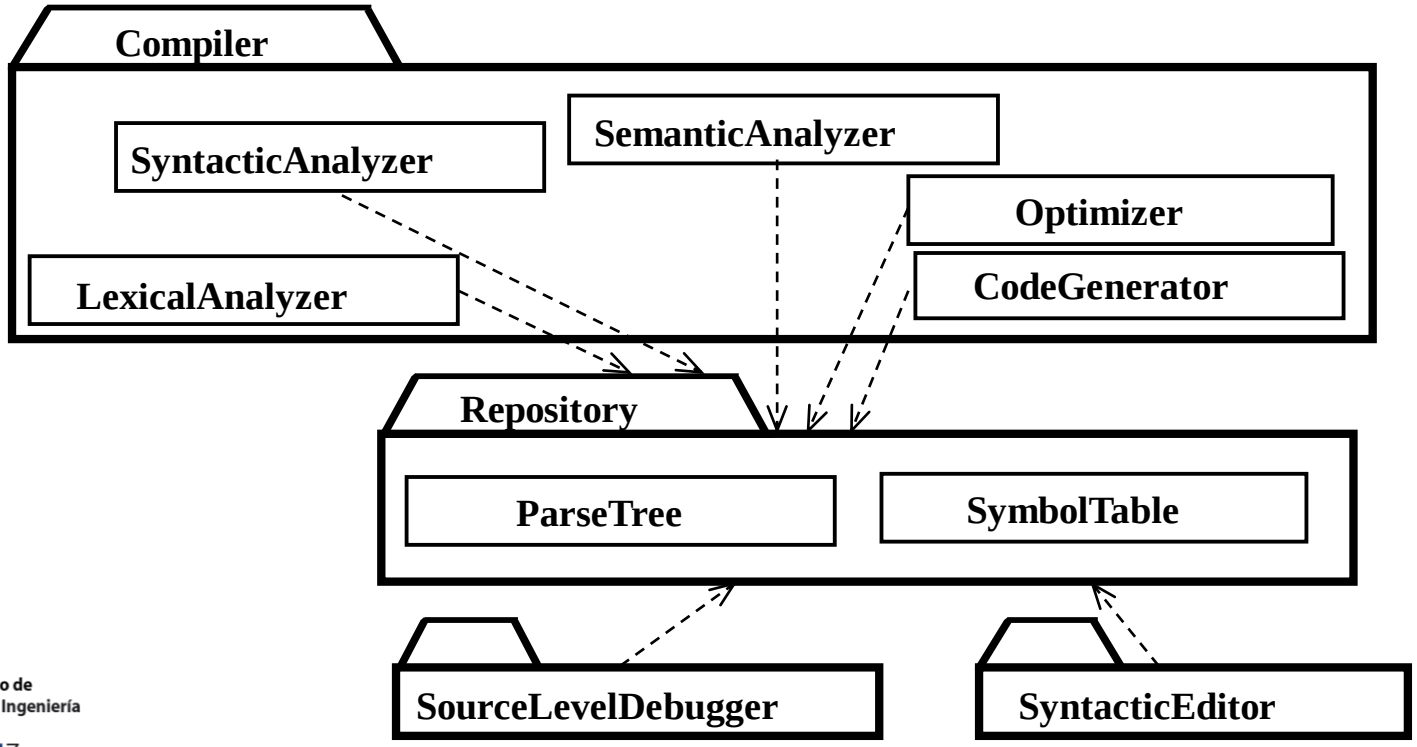
Arquitectura Depósito (Repositorio)

- Los subsistemas acceden y modifican datos de una sola estructura de datos llamada repositorio central
- Los subsistemas son relativamente independientes (poco acoplados): interactúan solo con el repositorio central
- Flujo de control
  - Dictado por el repositorio central (*triggers*)
  - O por los subsistemas (*locks*, semáforos, primitivas de sincronización)



Ejemplos de aplicación

- Sistemas de administración de bases de datos (sistemas de nómina, bancarios)
- Compiladores modernos

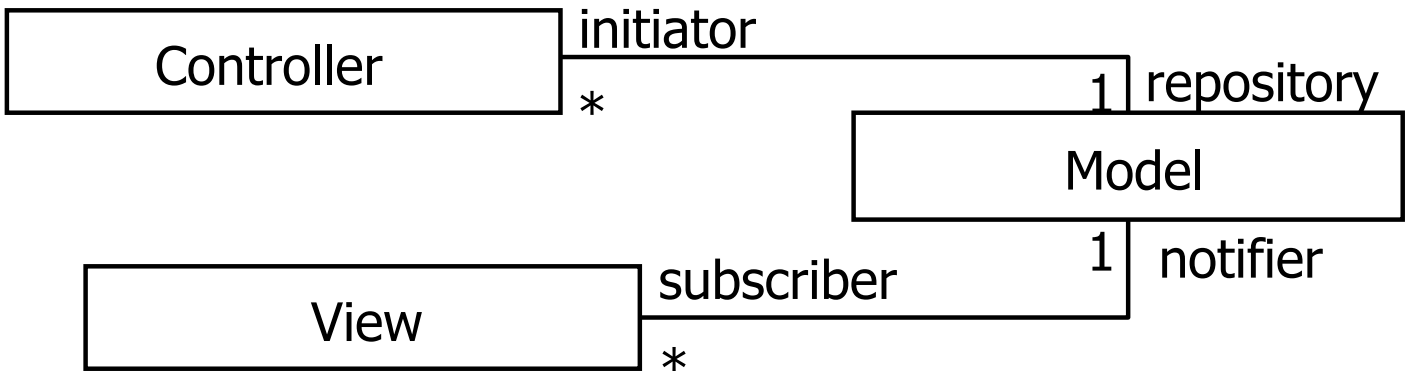


Modelo/Vista/Controlador (I)

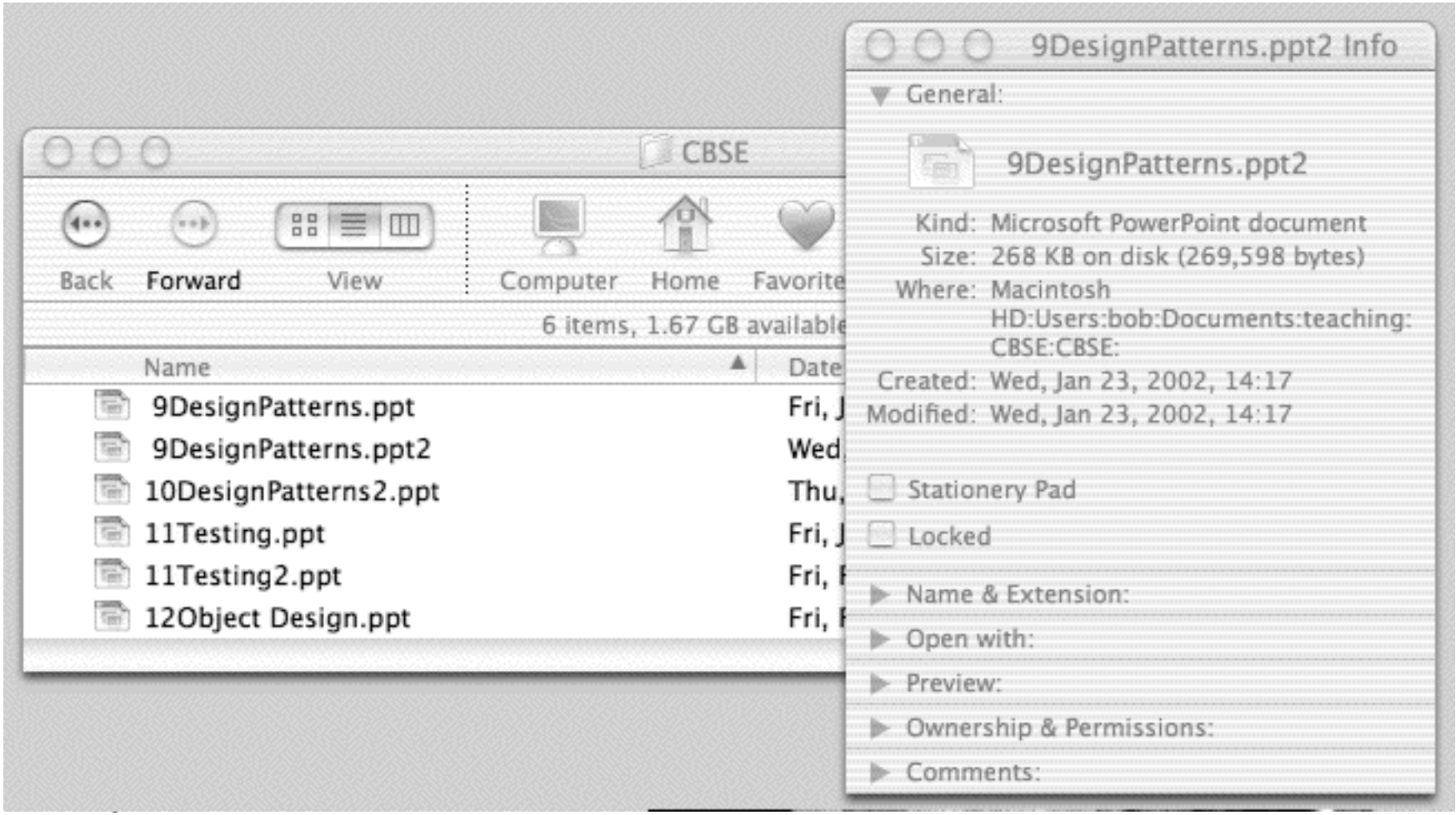
- Esta arquitectura facilita la creación de sistemas que necesitan mostrar los objetos del dominio (modelo) sincronizadamente desde varios componentes de la interfaz (ventanas, diálogos, …)
- En esta arquitectura los subsistemas se clasifican en 3 diferentes tipos
  - Subsistema Modelo: responsable del conocimiento en el dominio de la aplicación
  - Subsistema Vista: responsable de mostrar los objetos del dominio de la aplicación al usuario
  - Subsistema Controlador: responsable de la secuencia de interacciones con el usuario y notificar a las vistas sobre los cambios en el modelo

Modelo/Vista/Controlador

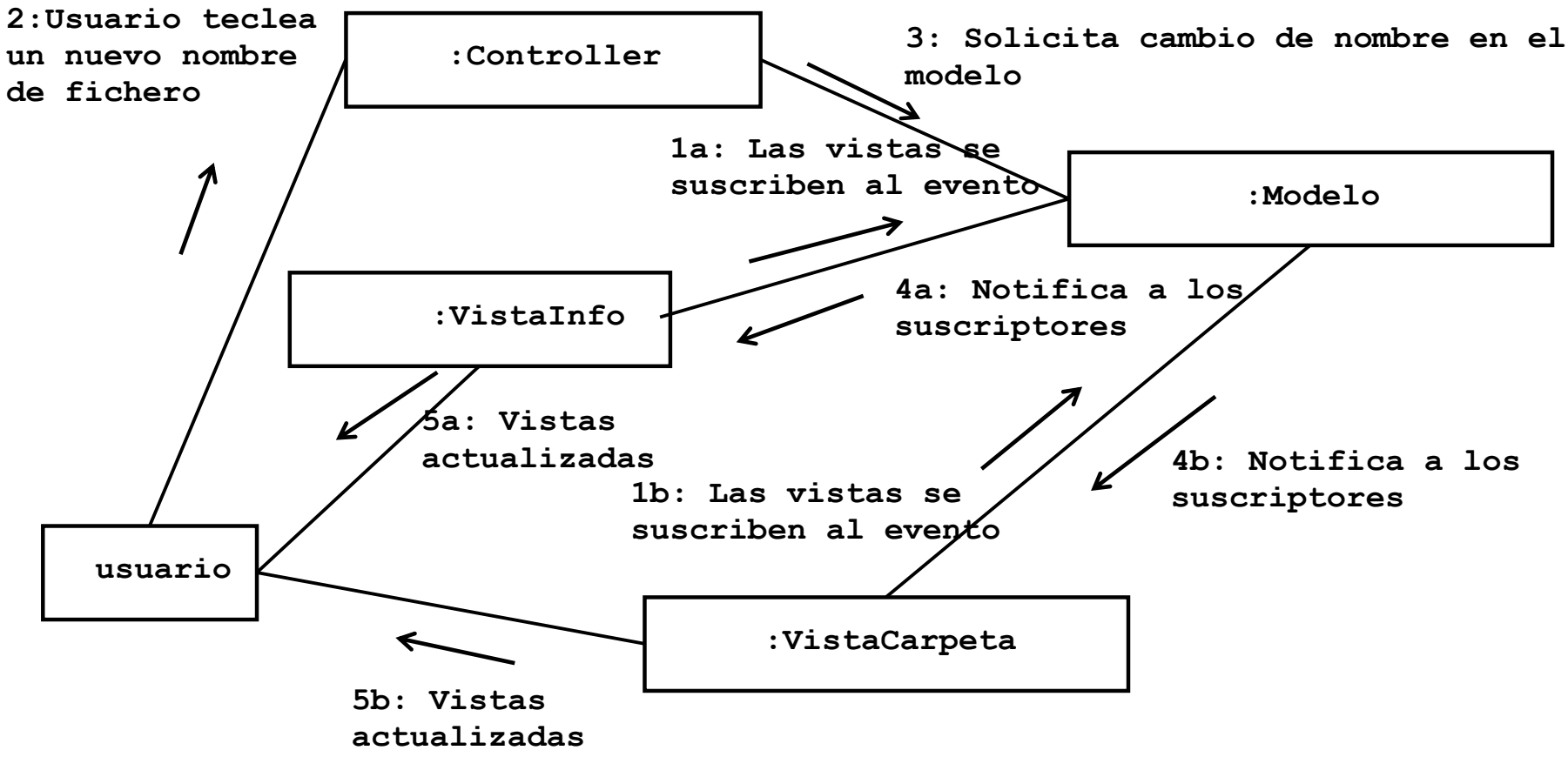
- MVC es un caso especial de la arquitectura depósito (repository) :
  - El subsistema Modelo implementa la estructura de datos central, el subsistema controlador dicta explícitamente el flujo de control



Ejemplo de un sistema de ficheros basado en el estilo arquitectural MVC

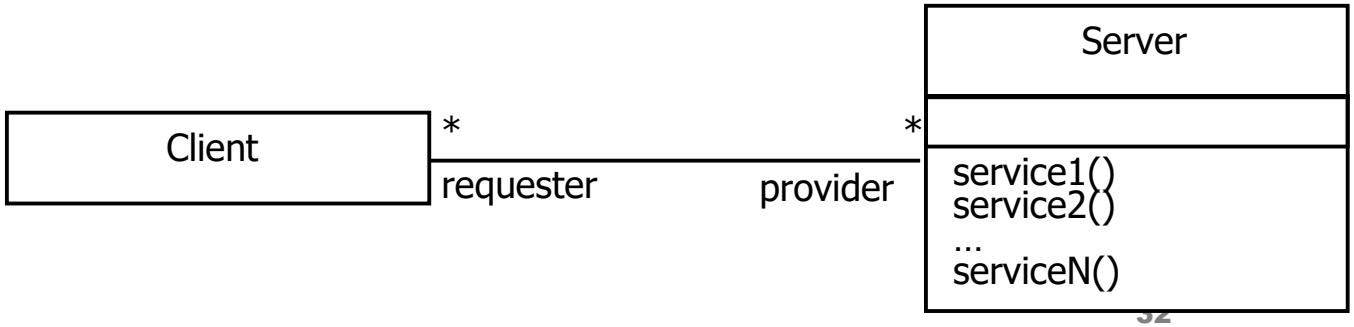


Secuencia de eventos (Colaboraciones)



Arquitectura Cliente/Servidor

- Un subsistema, el servidor, proporciona servicios a instancias de los demás subsistemas, llamados clientes
- Un cliente llama a un servidor, que realiza algún servicio y devuelve el resultado
  - El cliente conoce la *interfaz* del servidor (*su servicio*)
  - El servidor no necesita saber la interfaz del cliente
- Respuesta inmediata en general
- Los usuarios interaccionan sólo con el cliente
- Es un caso especial de la arquitectura depósito donde la estructura de datos es gestionada por un proceso
  - Además, un cliente puede invocar a varios servidores



Ejemplo de arquitectura Cliente/Servidor en Sistemas de Información

- *Front-end*: Aplicación de usuario (cliente)
  - Interfaz de usuario adaptada
  - Procesamiento de datos en *Front-end*
  - Inicialización de las llamadas a los procedimientos remotos del servidor
  - Acceso a servidores a través de la red
- *Back-end*: Manipulación y acceso a bases de datos (servidor)
  - Gestión de datos centralizada
  - Integridad y consistencia de datos
  - Seguridad en el acceso a datos
  - Gestión de operaciones concurrentes (acceso de múltiples usuarios)

Objetivos de diseño para sistemas Cliente/Servidor

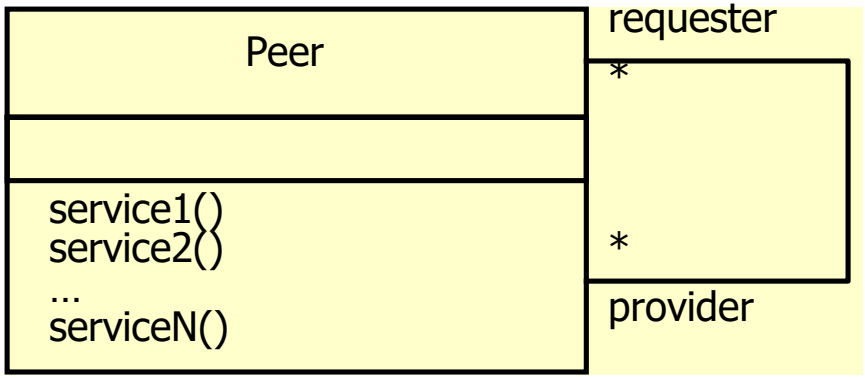
- Portabilidad de servicio
  - El servidor se puede instalar en una variedad de máquinas y sistemas operativos, funcionando en una gran variedad de entornos de red
- Transparencia, transparencia de localización
  - El servidor puede ser distribuido, pero proporciona un único servicio "lógico" al usuario
- Rendimiento
  - El cliente debería adaptarse para tareas interactivas muy intensivas
  - El servidor debería proporcionar operaciones intensivas en CPU
- Escalabilidad
  - El servidor debería tener capacidad adicional para gestionar un número más grande de clientes
- Flexibilidad
  - El sistema debería ser utilizable por distintas interfaces de usuario y dispositivos finales (ej. móviles, portátiles, equipos de escritorio)
- Fiabilidad
  - El sistema debería sobrevivir a problemas en enlaces de comunicaciones o nodos

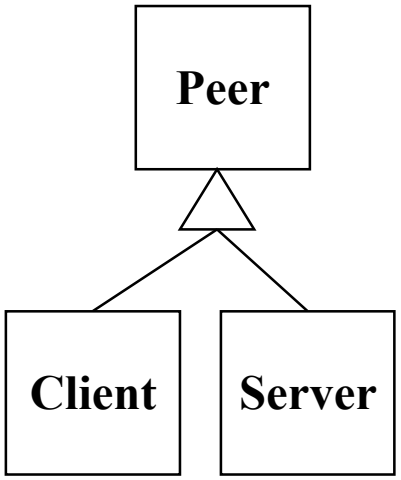
Problemas con arquitecturas Cliente/Servidor

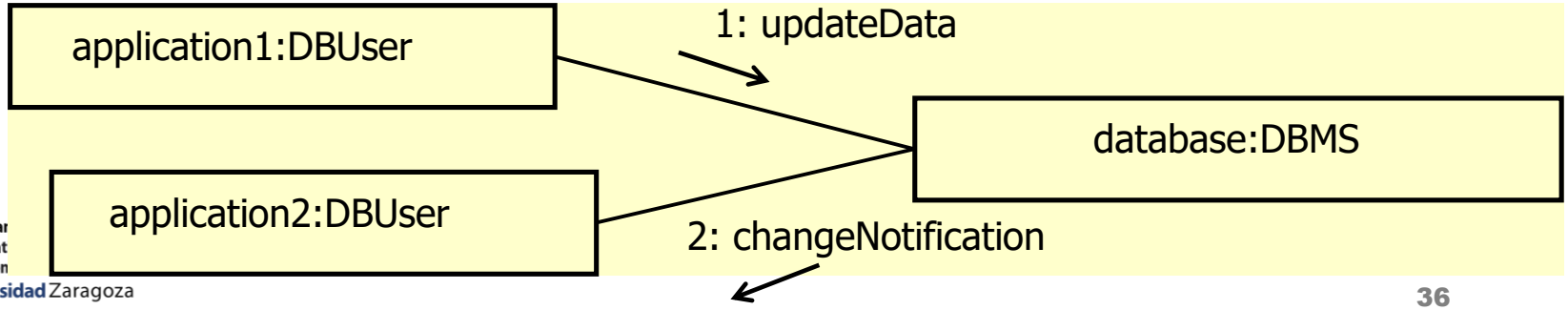
- Los sistemas en capas no proporcionan comunicaciones par a par
- La comunicación par a par se necesita a menudo
  - Ejemplo: una base de datos recibe consultas de una aplicación pero también envía notificaciones a las aplicaciones cuando han cambiado los datos

Arquitectura Par a Par (Peer-to-Peer)

- Generalización de la arquitectura cliente/servidor
- Los subsistemas pueden actuar como clientes y servidores
- Más difíciles de diseñar porque se pueden provocar situaciones de bloqueo (*deadlocks*)

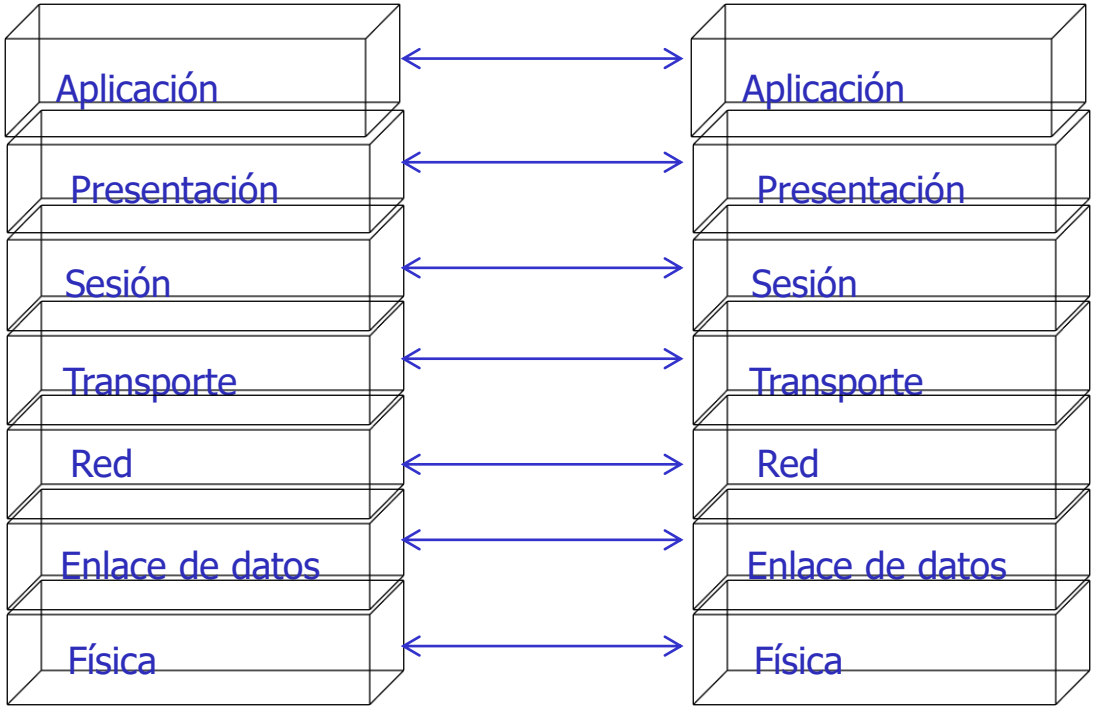






Oros ejemplos de arquitectura Par a Par

- Niveles OSI (Open Systems Interconnection) de ISO
  - Modelo de referencia que define 7 capas de protocolos de comunicación estrictos entre capas



- Sistemas de compartición de ficheros (ej: Bittorrent)

4. Refinar la arquitectura para conseguir los objetivos de diseño

- 4.1. Concurrencia
- 4.2. Correspondencia de subsistemas a componentes, artefactos y nodos
- 4.3. Gestión de datos persistentes
- 4.4. Definición de control de acceso
- 4.5. Diseño del flujo de control global
- 4.6. Identificación de condiciones frontera

4.1. Análisis de la concurrencia

- Identificar hilos concurrentes y abordar los aspectos de concurrencia
- Objetivo de diseño: tiempo de respuesta, rendimiento
- Hilos (Threads)
  - Un hilo de control es un camino a través de un conjunto de diagramas de estado en los cuales sólo hay un objeto activo a la vez
  - Un hilo permanece dentro de un diagrama de estados hasta que el objeto envía un evento a otro objeto y espera a otro evento
  - División del hilo: el objeto hace un envío no bloqueante de un evento

Concurrencia (continuación)

- Concurrencia
  - 2 objetos son inherentemente concurrentes si pueden recibir eventos a la vez sin interactuar
  - Los objetos inherentemente concurrentes deberían asignarse a diferentes hilos de control
  - Los objetos con actividad exclusiva mutua deberían agruparse en un único hilo de control
- Preguntas
  - ¿Qué objetos de un modelo son independientes?
  - ¿Qué clases de hilos de control son identificables?
  - ¿Proporciona acceso el sistema a múltiples usuarios?
  - ¿Puede una única petición al sistema descomponerse en múltiples peticiones? ¿Se pueden gestionar estas peticiones en paralelo?

Implementación de la concurrencia

- Los sistemas concurrentes se pueden implementar sobre cualquier sistema que proporcione
  - Concurrencia física
    - Vía hardware
  - Concurrencia lógica
    - Vía software
    - Problema de *Scheduling* (ej: Sistemas Operativos)

4.2. Correspondencia de subsistemas a componentes, artefactos y nodos

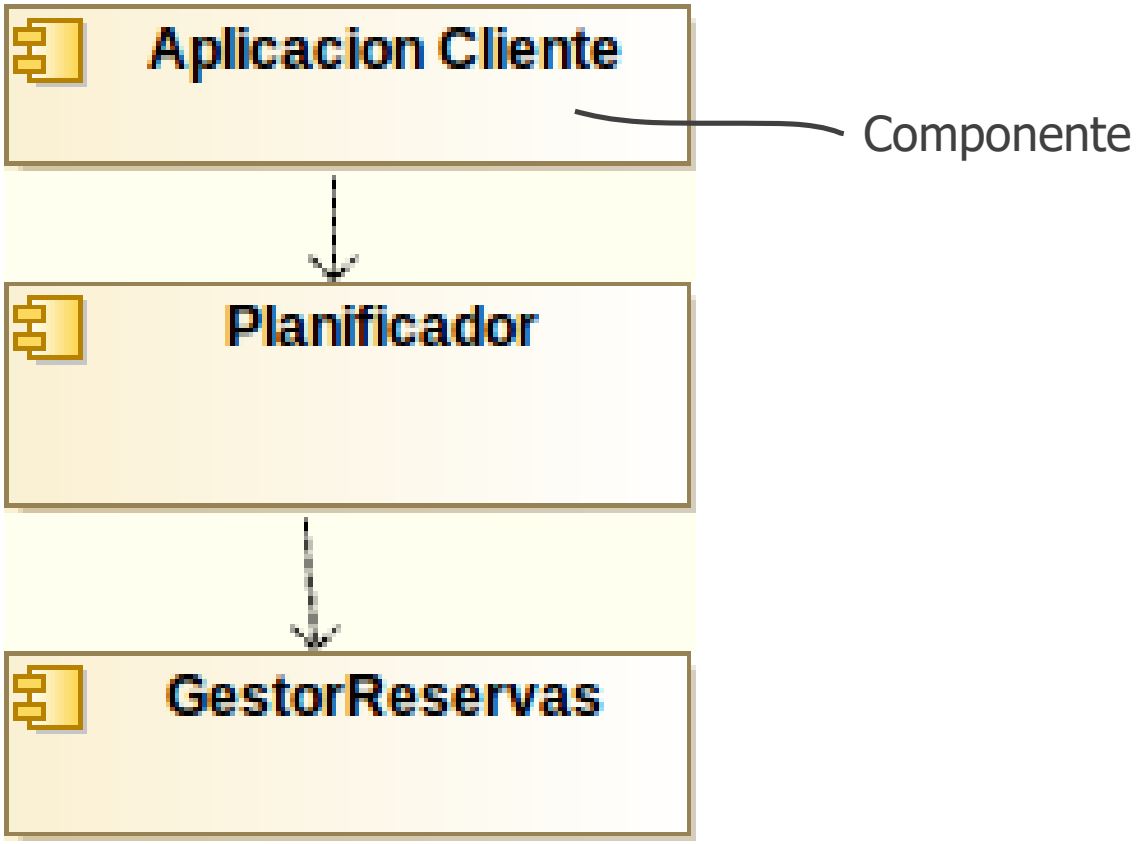
- Antes de tomar decisiones sobre la plataforma física, conviene pensar en una **vista lógica** basada en componentes
- Un componente es una parte reemplazable de un sistema que conforma y proporciona la implementación de un conjunto de interfaces
- En nuestro contexto nos permite representar la construcción de un subsistema como un elemento lógico reemplazable

Diagrama de componentes

- Un grafo de componentes conectados por relaciones de dependencia
- Otros elementos del diagrama
  - Interfaces proporcionadas
    - Interfaces facilitadas por un componente a otros componentes
  - Interfaz requerida
    - Interfaz requerida por un componente cuando solicita servicios de otros componentes
  - Puertos
    - Una ventana explícita dentro de un componente encapsulado

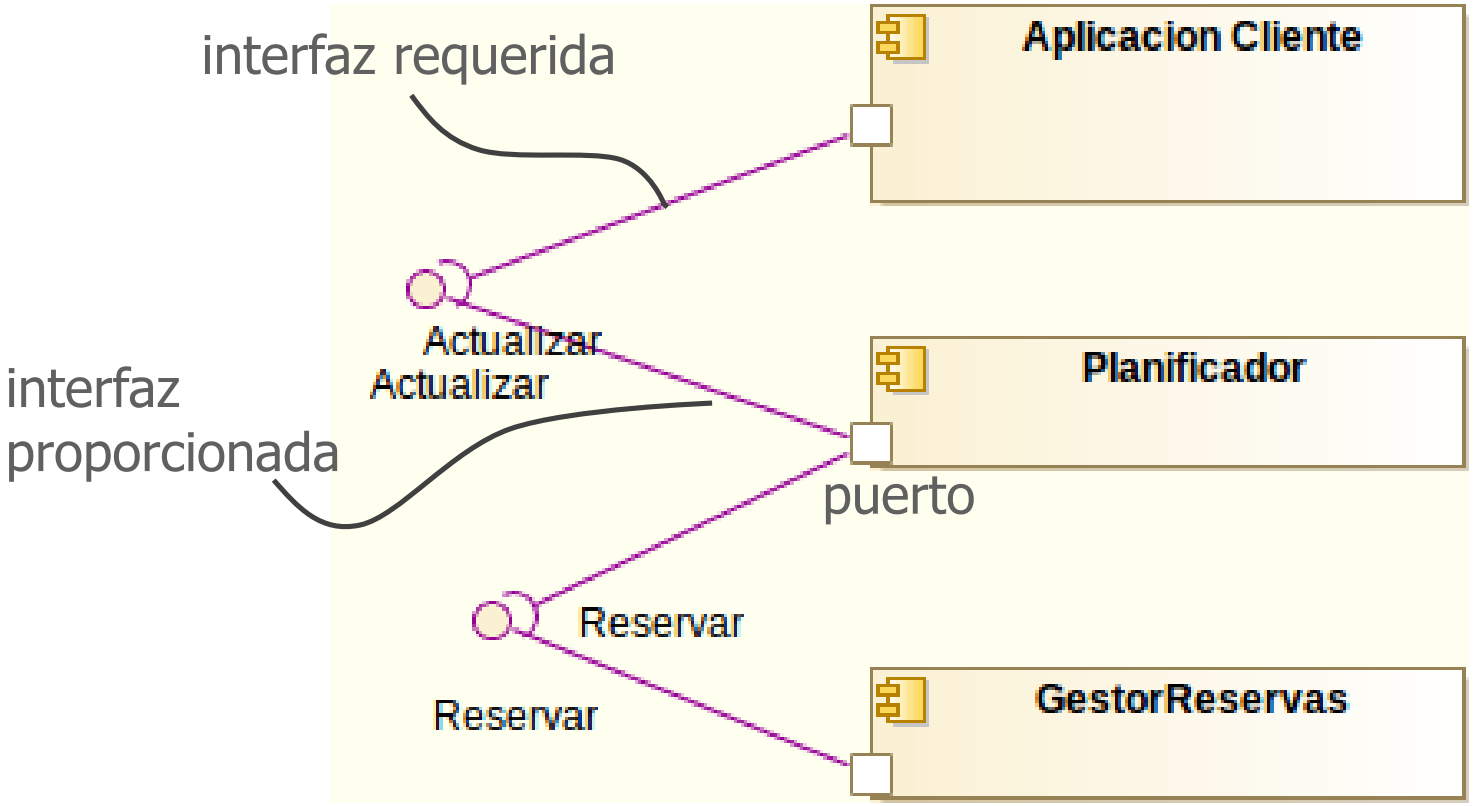
Ejemplo de una agencia de viajes (I)

- Diagrama de componentes con detalle de dependencias



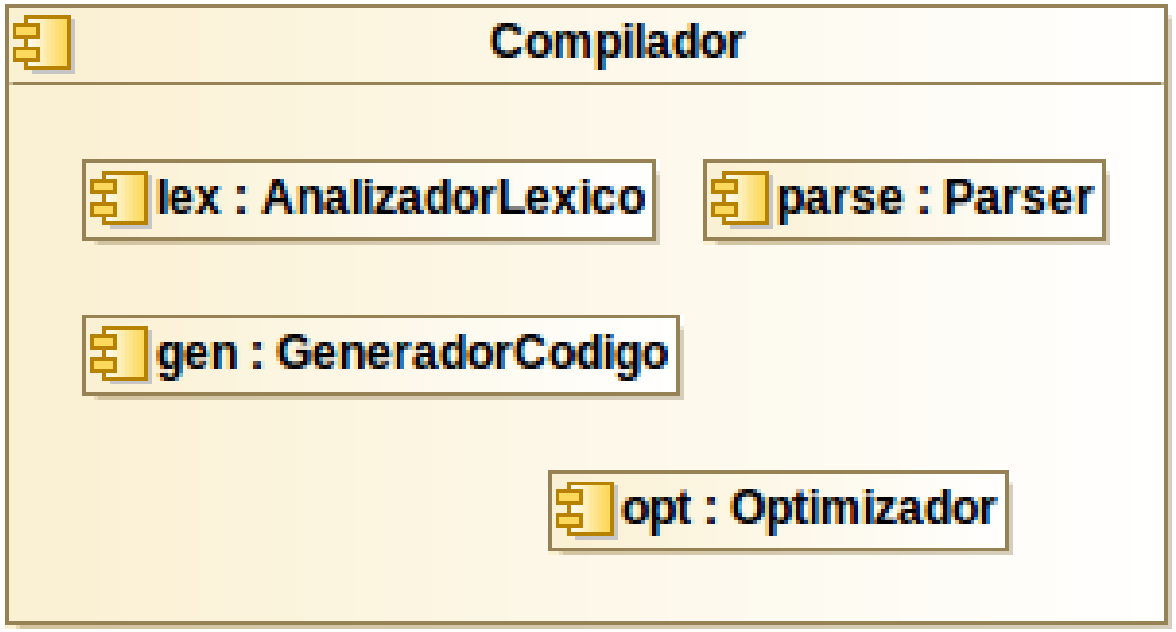
Ejemplo de una agencia de viajes (II)

- Diagrama de componentes con detalle de interfaces proporcionadas, interfaces requeridas y puertos



Otros elementos del diagrama de componentes

- Estructura interna
  - Los componentes más grandes utilizan componentes menores como bloques de construcción
  - Dentro de un componente se pueden mostrar subcomponentes



Decisiones sobre la plataforma física

- ¿Cómo mapeamos el modelo de objetos sobre el hardware y software elegidos?
  - Mapeo de objetos a la realidad:
    - Procesador
    - Memoria
    - Entrada/Salida
  - Mapeo de asociaciones a la realidad:
    - Describir la conectividad para sistemas distribuidos

Selección de una configuración hardware y una plataforma

- Aspectos de procesador:
  - ¿El nivel de uso de la CPU es demasiado exigente para un único procesador?
  - ¿Podemos acelerar distribuyendo tareas en varios procesadores?
  - ¿Cuántos procesadores se requieren para mantener la carga de trabajo?
- Aspectos de memoria:
  - ¿Hay suficiente memoria para guardar en el buffer las explosiones de peticiones?
- Aspectos de entrada/salida:
  - ¿Necesitas una pieza extra de hardware para mantener la tasa de generación de datos?
  - ¿El tiempo de respuesta sobrepasa el ancho de banda de comunicación disponible?

Conectividad en Sistemas Distribuidos

- Si la arquitectura es distribuida, necesitamos describir la red de comunicaciones
- Aspectos a detallar:
  - ¿Cuál es el medio de transmisión?
    - Ethernet (topología en árbol, estrella, matriz, anillo …), inalámbrico
  - ¿Cuál es el protocolo de comunicación apropiado entre subsistemas?
    - Protocolos a nivel de transporte: TCP, UDP
    - Protocolos a nivel de aplicación: HTTP, SSH, SMTP, …, RMI, JDBC
    - Depende del ancho de banda requerido, latencia, fiabilidad deseada, calidad de servicio (QoS) deseada
  - ¿La interacción debería ser asíncrona, síncrona o bloqueante?

Diagramas de despliegue (I)

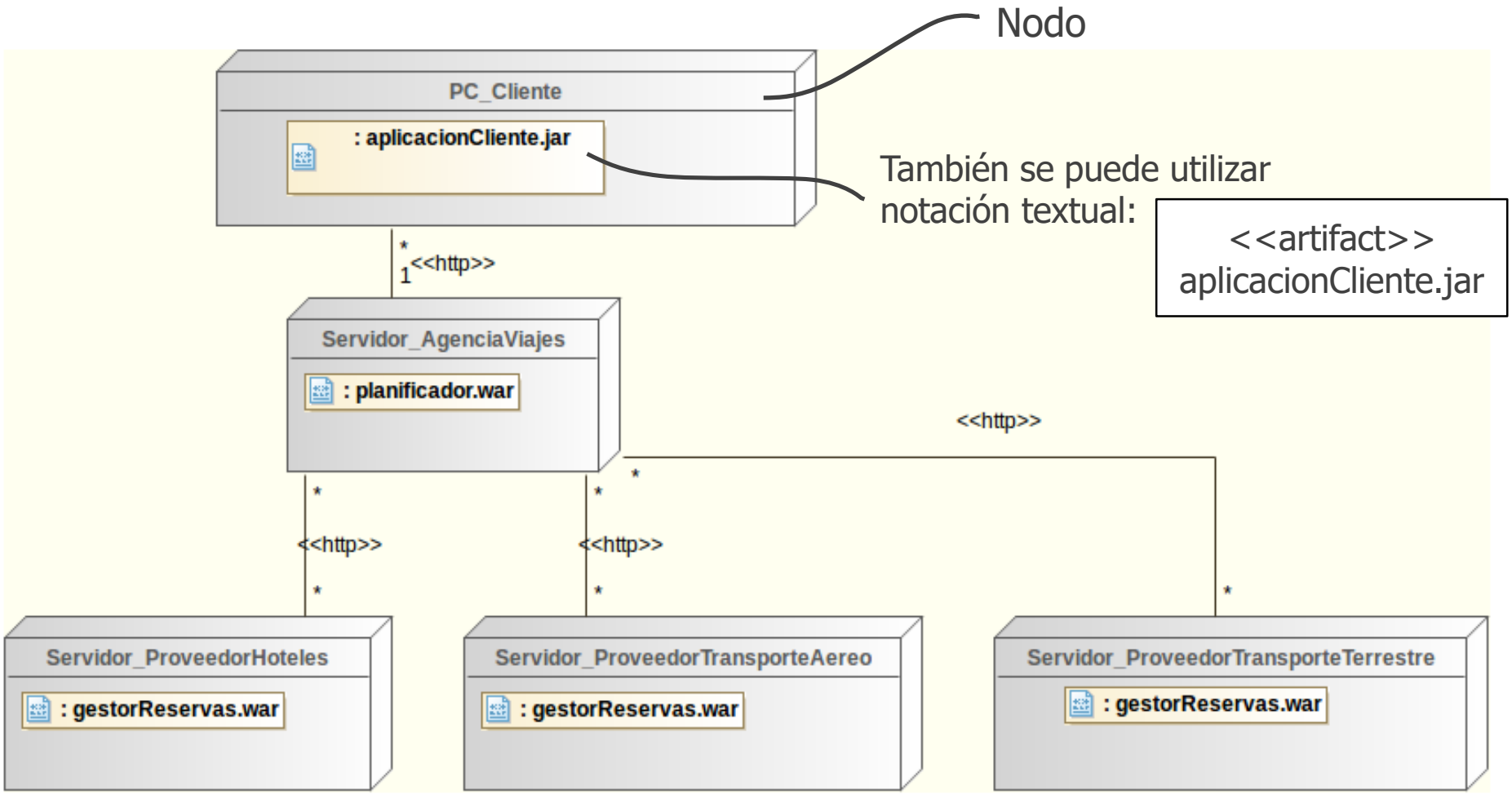
- Los diagramas de despliegue son útiles para mostrar la **vista física** del sistema después de haber tomado las decisiones de diseño
  - Descomposición en subsistemas
  - Concurrencia
  - Mapeo Hardware/Software
- Un diagrama de despliegue es un grafo de nodos conectados por asociaciones de comunicación
  - Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional (memoria + capacidad de procesamiento)
  - Los nodos se muestran como cajas 3-D
  - Las conexiones entre nodos se pueden estereotipar para modelar nuevos tipos de conexiones (<<RS-232>>, <<Ethernet-10T>>)

Diagramas de despliegue (II)

- Dentro de un nodo se despliegan y ejecutan artefactos
- Un artefacto es una parte física y reemplazable de un sistema que existe a nivel de la plataforma de implementación
  - Pertenecen al mundo material de los bits
  - Modelan elementos físicos tales como ejecutables, librerías, tablas, archivos o documentos
    - Normalmente representan el empaquetamiento físico de elementos lógicos como componentes, clases, interfaces
  - Se muestran gráficamente con un rectángulo con la palabra clave <<artifact>>

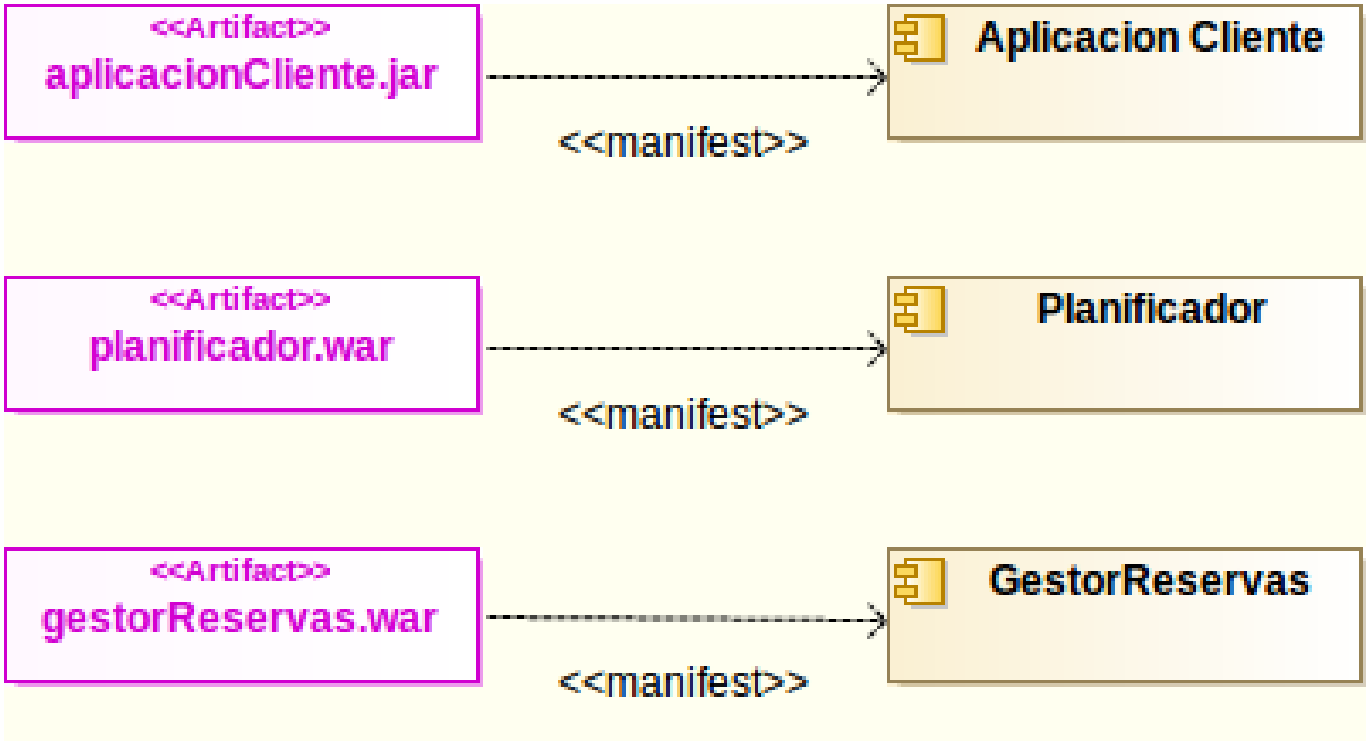
Ejemplo de una agencia de viajes (III)

- Diagrama de despliegue



Ejemplo de una agencia de viajes (IV)

- La relación entre un artefacto y los componentes (elementos) que implementa se puede mostrar con una dependencia de tipo *manifest*



4.3 Gestión de datos persistentes

- Algunos objetos de los modelos necesitan ser persistentes. Posibles candidatos:
  - Objetos del dominio del problema identificados durante el análisis
  - Pero también objetos identificados en el diseño: preferencias de GUI, usuarios del sistema
- Opciones

Ficheros	Bases de datos
• Baratos, simples, almacenamiento permanente • Bajo nivel (leer, escribir) • Las aplicaciones deben añadir código para proporcionar el nivel de abstracción apropiado	• Potentes, fácil de portar • Soporta múltiples escritores y lectores

¿Ficheros o bases de datos?

- ¿Cuándo deberías elegir un fichero?
  - ¿Son los datos voluminosos (ej: formatos raster de imágenes)?
  - ¿Tienes grandes cantidades de datos brutos (core dump, trazas de eventos)?
  - ¿Necesitas mantener los datos por un periodo de tiempo breve?
  - ¿Es baja la densidad de información (archivos históricos, logs históricos)?
- ¿Cuándo deberías elegir una base de datos?
  - ¿son los datos accedidos por muchos usuarios a un nivel fino de detalles?
  - ¿Deben ser portados los datos a través de múltiples plataformas (sistemas heterogéneos)?
  - ¿Necesitan acceder a los datos múltiples programas de aplicación?
  - ¿Necesita la gestión de los datos una gran infraestructura?

Sistemas de gestión de Bases de Datos

- Contiene mecanismos
  - para describir datos,
  - manejar almacenamiento persistente
  - y proporcionar mecanismos de *backup*
- Proporciona acceso concurrente a los datos almacenados
- Contiene información acerca de los datos ("meta-data")
  - Esquemas de datos (data schema)
- Distintos tipos de modelos
  - Relacionales, OO, NoSQL, …

Bases de datos relacionales

- Basadas en el álgebra relacional
- Los datos se presentan en tablas bidimensionales
  - Las tablas tienen un número específico de columnas y un número arbitrario de filas
  - Claves primarias: Combinación de atributos que identifican unívocamente una fila en una tabla
    - Cada tabla debería tener una única clave primaria
  - Clave ajena: Referencia a una clave primaria en otra tabla
- SQL es el lenguaje estándar para definir y manipular tablas
- Las bases de datos comerciales más importantes soportan restricciones
  - Integridad referencial, por ejemplo, significa que las referencias a entradas en otras tablas realmente existen

Bases de datos Orientadas a Objeto

- Soportan todos los conceptos fundamentales de modelado de objetos
  - Clases, Atributos, Métodos, Asociaciones, Herencia
- Mapeando el modelo de objetos a una base de datos OO
  - Determinar qué objetos son persistentes
  - Realizar análisis normal de requisitos y diseño de objetos
  - Crear índices de atributos para reducir los cuellos de botella de rendimiento
  - Realizar el mapeo (específico de un producto disponible comercialmente). Ejemplo:
    - En ObjectStore, implementar clases y asociaciones preparando declaraciones C++ para cada clase y cada asociación en el modelo de objetos

Otras Bases de Datos

- Bases de datos NoSQL (Not Only SQL)
  - Bases de datos donde se manejan grandes volúmenes de datos poco estructurados
  - Algunos ejemplos:
    - Orientadas a documentos:
      - eXist DB (documentos XML)
    - Orientadas a grafos:
      - OpenLink Virtuoso (grafos RDF)
    - Orientadas a pares clave-valor:
      - MongoDB (los registros de pares clave-valor se guardan en formato binario derivado de JSON)

Aspectos a considerar para seleccionar una base de datos (I)

- Espacio de almacenamiento
  - Las bases de datos requieren cerca del triple de espacio de almacenamiento que los datos reales
  - ¿Deberían estar los datos distribuidos?
  - ¿Debería ser la base de datos extensible?
- Tiempo de respuesta
  - Limitado por E/S o comunicaciones (bases de datos distribuidas)
  - También se ve afectado por el tiempo de CPU, bloqueos, y volcados por pantalla frecuentes
  - ¿Con qué frecuencia se accede a la base de datos?
  - ¿Cuál es la tasa de peticiones esperada? ¿En el peor caso?
  - ¿Cuál es el tamaño de las peticiones en un caso típico y en el peor caso?

Aspectos a considerar para seleccionar una base de datos (II)

- Modos de bloqueo
  - Bloqueo pesimista:
    - Bloquear antes de acceder al objeto y liberar cuando el acceso al objeto está completo
  - Bloqueo optimista:
    - Las lecturas y escrituras pueden ocurrir libremente.
    - Cuando la actividad se ha completado, la base de datos chequea si ha ocurrido un conflicto. Si ha ocurrido, todo el trabajo se ha perdido
- Administración
  - Las grandes bases de datos requieren personal especialmente formado para establecer políticas de seguridad, manejar el espacio de disco, preparar backups, monitorizar el rendimiento, ajustar la optimización

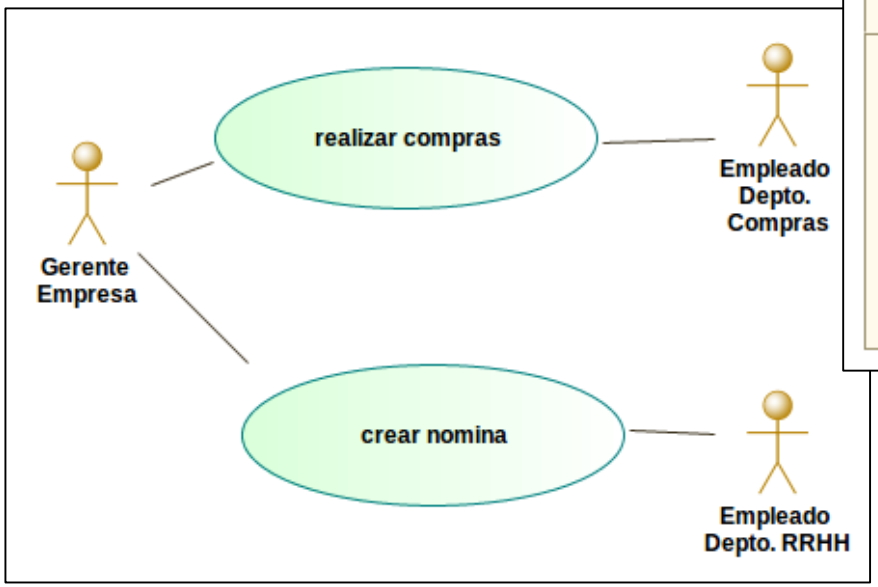
4.4. Definición del control de acceso

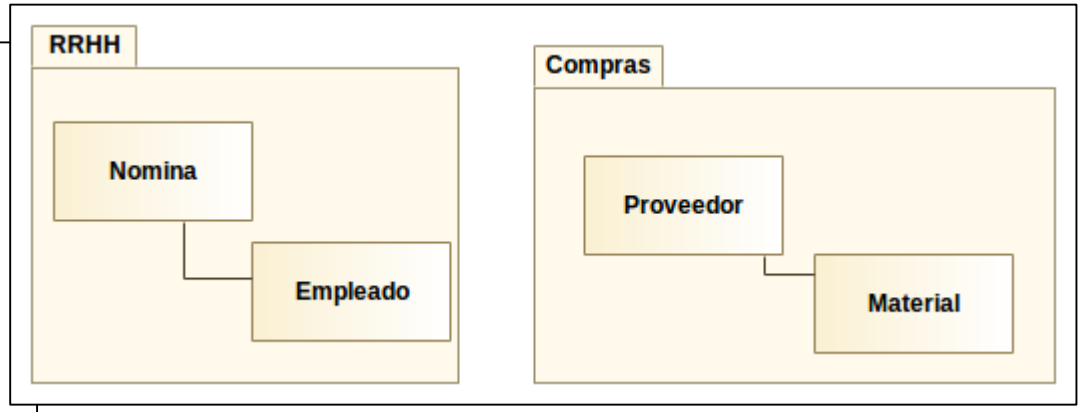
- En los sistemas multiusuario, actores diferentes tienen acceso a funcionalidades y datos diferentes
  - Durante el análisis modelamos estas distinciones asociando diferentes casos de uso a diferentes actores
  - Durante el diseño del sistema modelamos el acceso examinando el modelo de objetos y determinando cuáles de estos objetos están compartidos entre actores y determinando la manera en que los actores pueden controlar el acceso
    - Dependiendo de los requisitos de seguridad, definimos cómo se autentifican los actores ante el sistema y la manera en que deben cifrarse datos seleccionados en el sistema

Matriz de acceso

- Modelar el acceso a clases con una matriz
  - La filas representan a los actores del sistema
  - Las columnas representan a las clases cuyo acceso controlamos
- Derecho de acceso
  - Es una entrada (clase, actor) en la matriz de acceso
  - En esta entrada se listan las operaciones que un actor puede ejecutar sobre las instancias de la clase

Ejemplo de matriz de acceso





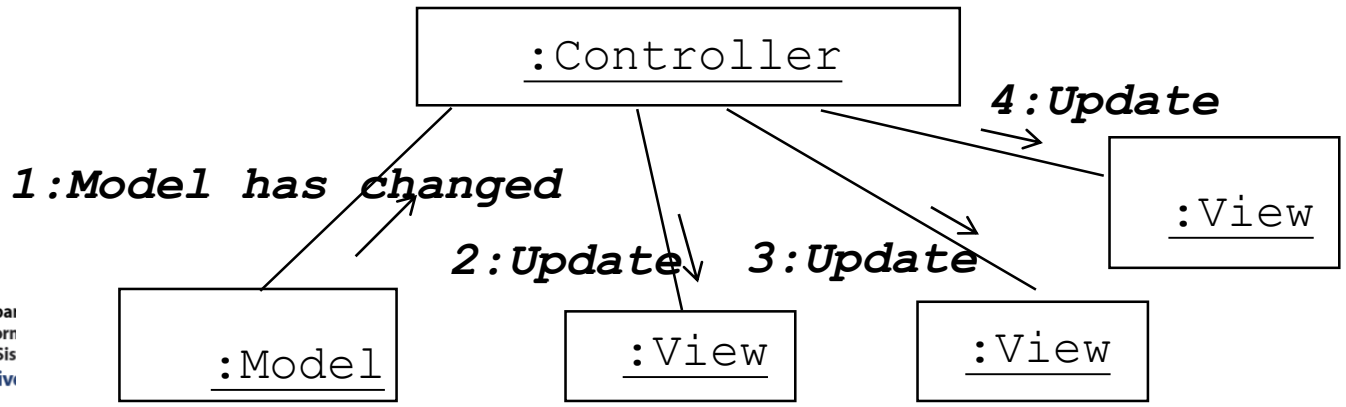
Actor\Clase	Empleado	Nomina	Proveedor	Material
Empleado Depto. Compras	Ninguna operación	Ninguna operación	Todas operaciones	Todas operaciones
Empleado Depto. RRHH	Todas operaciones	Todas operaciones	Ninguna operación	Ninguna operación
Gerente Empresa	Operaciones de consulta	Operaciones de consulta	Operaciones de consulta	Operaciones de consulta

4.5. Diseño del flujo de control global

- Flujo de control
  - Ordenamiento de las acciones en un sistema
- Elegir entre
  - Control implícito (no procedimental, lenguajes declarativos)
    - Sistemas basados en reglas
    - Programación lógica
  - Control explícito (lenguajes centralizados)
    - Centralizado
    - o descentralizado

Control centralizado

- Un objeto de control o subsistema controla todo
- Control dirigido por procedimientos
  - El control reside dentro del código del programa. Ejemplo: programa principal llamando a procedimientos de subsistemas.
  - Simple, fácil de construir, duro de mantener (costes altos de recompilación)
- Control dirigido por eventos
  - El control reside dentro de un despachador llamando a funciones vía callbacks
  - Muy flexible, bueno para el diseño de interfaces gráficas de usuario, fácil de extender



Model-View-Controller

(Vista pasiva)

Control descentralizado

- No hay un único objeto al control, el control es distribuido y reside en varios objetos independientes
- Posible aceleración mapeando los objetos a diferentes procesadores
  - Aunque se incrementa sobrecarga de comunicación
- Ejemplo: sistemas basados en paso de mensajes

Diseños centralizados vs descentralizados

- ¿El diseño debería ser centralizado o descentralizado?
  - Tomar los diagramas de secuencia y los objetos de control del modelo del análisis
  - Comprueba la participación de objetos de control en los diagramas de secuencia
    - Si el diagrama de secuencia se parece más un tenedor: diseño centralizado
    - Si el diagrama de secuencia se parece más a una escalera: diseño descentralizado
- Comparativa

	Centralizado	Descentralizado
Ventajas	El cambio en la estructura de control es muy fácil	Se ajusta perfectamente al desarrollo OO
Inconvenientes	El único objeto de control es un cuello de botella de rendimiento posible	La responsabilidad está repartida

4.6. Identificación de condiciones frontera

- La mayor parte del diseño se centra en el comportamiento del sistema en marcha
- Sin embargo, se necesita también examinar las condiciones frontera (iniciar, apagar, excepciones)
- Se definen casos de uso frontera (de administración) donde el actor es frecuentemente el administrador
  - Casos de uso de inicialización
    - Describen cómo pasa el sistema de un estado no iniciado a un estado de preparado
  - Casos de uso de terminación
    - Describen qué recursos se liberan y qué sistemas son notificados de la terminación
  - Casos de uso de fallo
    - Muchas causas posibles: Bugs, errores, problemas externos (abastecimiento de energía)
    - Un buen diseño de sistema prevé fallos fatales

Ejemplo

- Durante el diseño del sistema se encuentra un nuevo subsistema: el servidor
- Se deben definir nuevos casos de uso para este subsistema