# Maestría en Ingeniería

## Curso de Arquitectura de Software

### Sesión 5

*Fernando Barraza A.*
*fbarraza@javerianacali.edu.co*

# Sesión 5

- Objetivo: Dar a conocer al estudiante el concepto de estilos, vistas y patrones, y los más comúnmente utilizados en las AS.
- Temas:
  - Estilos y tipos de vistas
  - Concepto de patrón de AS
  - Estilos de capas
  - Estilos de Componentes y Conectores
  - Otros estilos comunes.

# Estilos (styles)

- Sirven para sintetizar estructuras de soluciones
- Un estilo de arquitectura establece
  - elementos y relaciones,
  - una serie de restricciones de cómo pueden usarse.
- La elección de un estilo determina los elementos y las restricciones que habrá que documentar

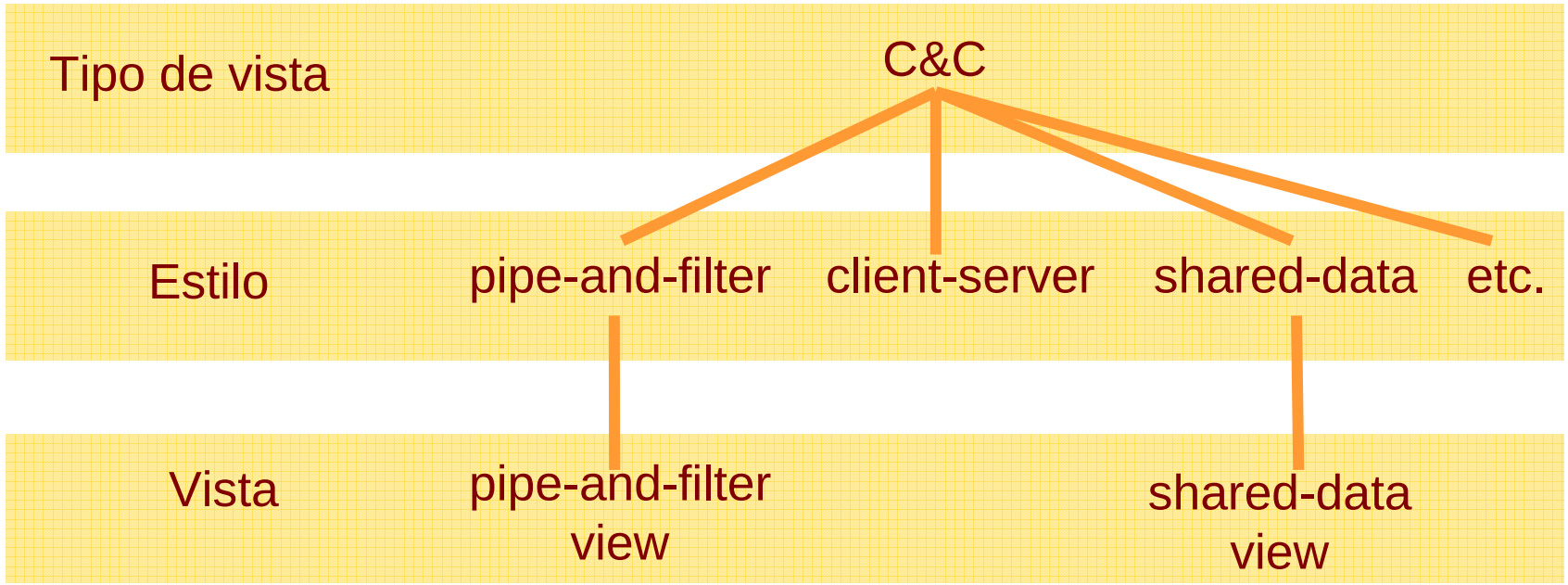
# Estilos (2)

- Definen los patrones posibles de las aplicaciones
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales

# Algunos Estilos y sus Tipos de Vistas


- Estilo de capas → tipo módulo.
  - El sistema tendrá ciertas características de modificabilidad, portabilidad, independientemente del contenido de los módulos.
- Estilo cliente-servidor → tipo C&C.
  - Existen clientes, servidores y protocolos,
  - características como facilidad para crear nuevos clientes.

# Vistas, Estilos y Tipos de Vistas




- Los estilos son especializaciones de los tipos de vistas, y las vistas son instancias de un estilo para un sistema particular.

# Mezcla de Estilos

- Distintas partes del sistema pueden tener distintos estilos:
  - hay elementos puente entre las distintas vistas de cada estilo,
  - generalmente mediante distintas interfaces que le permiten interactuar con uno u otro estilo.
- Un mismo sistema puede verse desde distintos puntos de vista como distintos estilos.

# Patterns


- Christopher Alexander, 1977
- Un patrón es una solución a un problema en un contexto
- Un patrón codifica conocimiento específico acumulado por la experiencia en un dominio
- Un sistema bien estructurado está lleno de patrones

# *Patterns* - Alexander

- “Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.”

**The Timeless Way of Building, Christopher Alexander – 1979.**

# Elementos de un patrón

- Nombre
  - Define un vocabulario de diseño
  - Facilita abstracción
- Problema
  - Describe cuando aplicar el patrón
  - Conjunto de fuerzas: objetivos y restricciones
  - Prerrequisitos
- Solución
  - Elementos que constituyen el diseño (*template*)
  - Forma canónica para resolver fuerzas
- Consecuencias
  - Resultados, extensiones y *tradeoffs*

# Tipos de patrones

| | Comentario | Problemas | Soluciones | Fase de Desarrollo |
|---|---|---|---|---|
| Patrones de Arquitectura | Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos | Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento | Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad | Diseño inicial |
| Patrones de Diseño | Conceptos de ciencia de computación en general, independiente de aplicación | Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, etc | Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC) | Diseño detallado |
| Patrones de Análisis | Usualmente específicos de aplicación o industria | Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes | Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. *logging* & reinicio) | Análisis |
| Patrones de Proceso o de Organización | Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización | Productividad, comunicación efectiva y eficiente | Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación | Planeamiento |
| Idiomas | Estándares de codificación y proyecto | Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predictibilidad. | Sumamente específicos de un lenguaje, plataforma o ambiente | Implementación, Mantemimiento, Despliegue |

# Patrones de Arquitectura comunes


- Blackboard (Pizarra)
  - Aplicaciones independientes y especializadas colaboran para obtener una solución compartiendo una estructura de datos común.
- Pipes & Filters (Tubos & Filtros)
  - La información se procesa en ráfagas que fluye por los tubos de filtro en filtro.
- MVC (Model-View-Controller)
  - La aplicación se divide en áreas: *Modelo* (reglas de negocio y datos), *Vista* (presentación) y *Controlador* (procesamiento de la entrada).
- Layers (Niveles)
  - Descomposición en distintos niveles de abstracción (niveles de aplicación a tecnológicos).

# Estilos vs. Patrones

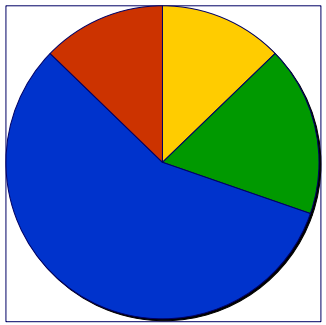
- Hay claras convergencias entre ambos conceptos:
  - Los patrones se refieren a prácticas de re-utilización que se concretizan en un sistema dado
  - Los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas.

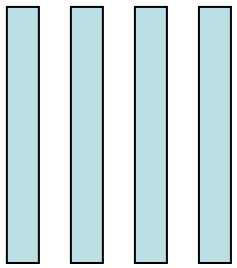
# Estilo de Capas (Layers)

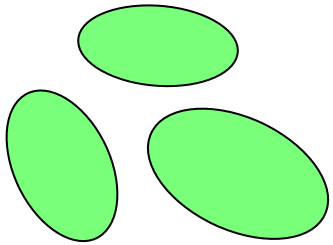
- Cada unidad del estilo es una capa que representa una máquina virtual.
  - Una máquina virtual es un dispositivo abstracto de cómputo; típicamente es un programa que actúa como una interfaz entre otra unidad de software y el hardware (o bien otra máquina virtual).
- El estilo de capas tiene propiedades de modificabilidad muy deseables, lo cual lleva a querer representar sistemas con este estilo aunque no sea lo más apropiado.

# Qué son capas y qué no son?

- Las capas particionan completamente al software.
- Cada capa es una máquina virtual con una funcionalidad cohesionada y una interfaz pública.







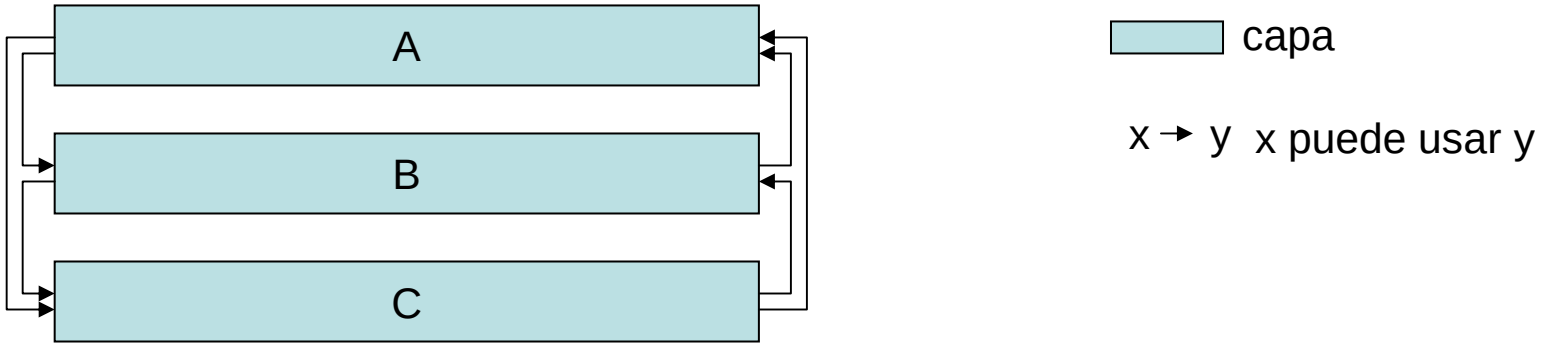
- Si cada una de las divisiones anteriores es una máquina virtual, la descomposición no necesariamente es una arquitectura de capas.

# Propiedad Esencial de las Capas

- En una arquitectura de capas, las máquinas virtuales deben interactuar de una manera ordenada y preestablecida:
  - Si (A,B) es la relación entre las capas entonces decimos que:
    - la capa B se encuentra por debajo de la capa A, y
    - la implementación de la capa A está autorizada a usar cualquiera de los servicios públicos que ofrece la máquina virtual de la capa B.
- Algunos esquemas de capas permiten usar solamente el nivel inmediatamente inferior, otros todos los de más abajo.

# Uso de Niveles Inferiores

- Ningún esquema de capas permite el uso de servicios de niveles superiores.




- Si bien el diagrama parece una arquitectura de capas, no lo es.
- Las capas inferiores solamente en casos muy excepcionales se les permite usar servicios superiores, y debe ser claramente documentado.

# Cualidades de las Capas


- Una de las principales ventajas de usar máquinas virtuales en arquitecturas de capas es proveer portabilidad.
- Para eso, la interfaz de cada capa no debe exponer funciones que dependen de una plataforma particular.
- Las capas superiores tienen una gran flexibilidad y servicios disponibles para su uso.
- La visión del mundo de las capas inferiores es mucho más restringida y dependiente de la plataforma.

# Capas inferiores y superiores

- Las capas inferiores:
  - incluyen conocimiento de los computadores, canales de comunicación, mecanismos de distribución, despacho de procesos, etc.
  - tienden a ser independientes de la funcionalidad de la aplicación
  - si la aplicación cambia, rara vez se ven afectados
- Las capas superiores:
  - son independientes de la plataforma donde habrán de ejecutar
  - las capas inferiores les resuelven las dependencias del ambiente
  - se ocupan casi exclusivamente de los detalles de la aplicación

# Capas y Código


- Analizando el código fuente, no puede deducirse la estructura de capas:
  - puede verse qué *usa* qué otra parte, pero la relación entre las capas es *está-autorizado-a-usar*.
- Una capa puede tener servicios que ninguna otra capa use
  - generalmente ocurre con módulos comprados, legados y construidos para ser usados en futuras aplicaciones

# Relaciones entre capas


- La relación esencial entre capas es *autorizado-a-usar*.
- Si dos capas tienen esta relación, cualquier módulo de la primera puede usar cualquier módulo en la segunda
  - un módulo A *usa* un módulo B si la corrección de A depende de que B esté presente y correcto.
- La definición y estructuración en capas consiste esencialmente en agrupar los módulos para que se relacionen de esta forma.

# Aplicación de Capas

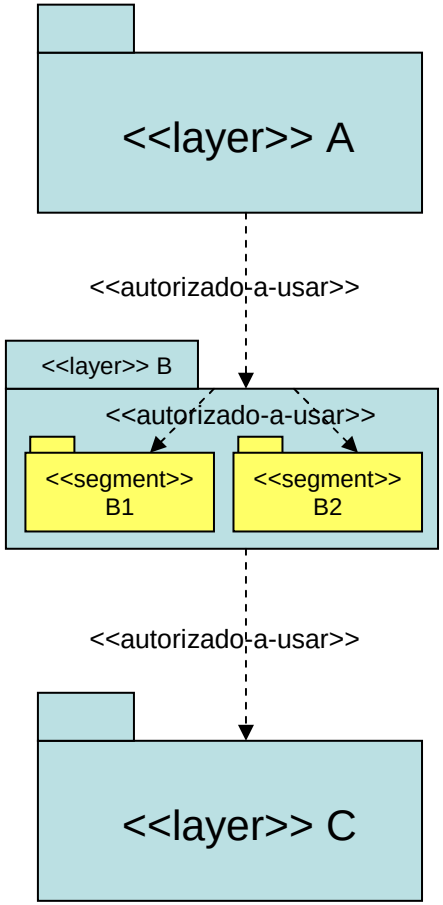
- Las capas ayudan a mejorar la modificabilidad y la portabilidad de los sistemas.
- Los cambios en las capas inferiores pueden ocultarse siempre que se preserve su interfaz. Esta técnica se ha usado mucho y con éxito como apoyo a la portabilidad y mejoramiento del desempeño.
- Ejemplo: Cambiar el procesador por otro más rápido (o más lento) da como resultado que la aplicación resulte más rápida (o más lenta).

# Otras cualidades

- Suele decirse que la arquitectura de capas empeora la performance, pero esto no es necesariamente cierto.
- Las capas pueden representar asignaciones de trabajo, ayudando en la administración del proyecto.
- En grandes sistemas, la cantidad de módulos y su compleja relación de uso puede oscurecer la comprensibilidad
  - organizar los módulos en capas eleva el nivel de abstracción y ayuda a manejar la complejidad.
- Ayudan en el análisis del impacto de los cambios.

# Capas y Descomposición en Módulos

- Suele verse las capas como módulos de una vista de descomposición
  - un módulo se descompone en otros módulos,
  - una capa no se descompone en otras capas, sino en segmentos.


<<layer>> A
<<autorizado-a-usar>>
<<layer>> B
<<autorizado-a-usar>>
<>
B1
<>
B2
<<autorizado-a-usar>>
<<layer>> C

# Capas y Tiers

- Las capas suelen confundirse con los *tiers* en una arquitectura de cliente-servidor de n-tiers.
- Los diagramas de n-tiers suelen tener relaciones bidireccionales entre las tiers.
- La asignación de módulos o componentes se hace teniendo en cuenta distintos criterios:
  - n-tiers
    - eficiencia durante la ejecución, tolerancia a fallas, localidad del procesamiento, no sobrecargar al servidor, uso apropiado del ancho de banda.
  - capas
    - cohesión de la funcionalidad, facilidad para realizar cambios.

# Capas y Uso

- Si en la documentación de un sistema se incluyen tanto la vista de capas como de uso, ninguna relación de *uso* debe violar la relación *autorizado-a-usar*.
- En desarrollos incrementales, en general se comienza con capas con pocos detalles y relaciones *autorizado-a-usar* que se refinan sucesivamente hasta definir todas las relaciones de *uso* efectivo.

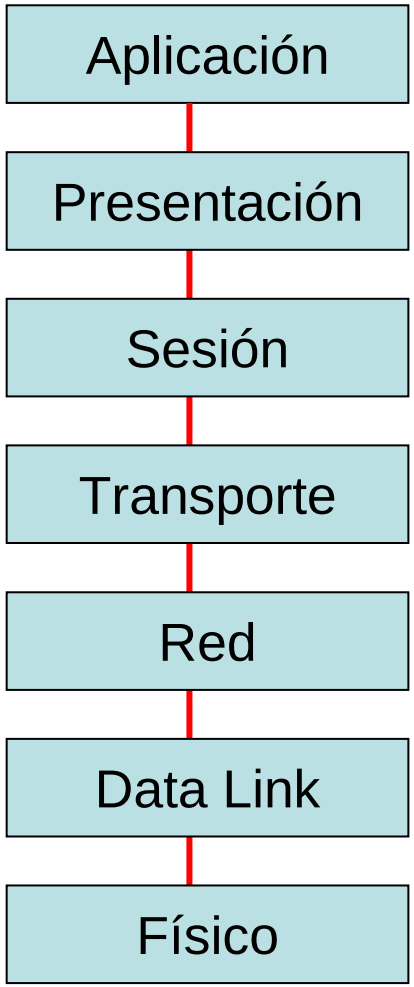
# Capas y Subsistemas

- Los subsistemas son ortogonales a las capas.
- En un sistema de capas segmentado, un subsistema será un conjunto de segmentos de una capa de alto nivel con todos los segmentos de nivel inferior que está autorizado a usar.

# Resumen de Capas

| **Elementos** | **Capas** |
|---|---|
| Relaciones | *Autorizado-a-usar*, lo cual es una especialización de la relación genérica del tipo de vista de módulos *depende-de*. P1 usa P2 si la correctitud de P1 depende de que exista una implementación correcta de P2. |
| Propiedades de elementos | •nombre de la capa<br>•unidades de software contenidas en la capa<br>•el software que la capa está autorizada a usar<br>•cohesión de la capa: máquina virtual que representa la capa |
| Propiedades de relaciones | Igual que para todo tipo de vista de módulos |
| Topología | Si la capa A está por encima de la capa B, entonces la capa B no puede estar encima de la capa A. Cada pieza de software se encuentra en una y sólo una capa. |

# Ejemplo: Modelo OSI de 7 Capas




| | |
|---|---|
| Nivel 7 | Protocolos misceláneos para actividades comunes. |
| Nivel 6 | Estructura la información y le agrega semántica. |
| Nivel 5 | Control de diálogo y facilidades de sincronización. |
| Nivel 4 | Divide los mensajes en paquetes y garantiza el envío. |
| Nivel 3 | Selecciona una ruta de origen a destino. |
| Nivel 2 | Detecta y corrige errores en secuencias de bits. |
| Nivel 1 | Transmite bits: velocidad, códigos de bits, conexión. |

# Estilos de componentes y conectores


- Sirven para razonar acerca de las cualidades que pueden apreciarse durante la ejecución:
  - performace, confiabilidad, y disponibilidad.
- Una vista de C&C bien documentada permite estimar las cualidades del sistema a partir de las propiedades de sus elementos.
- No son buenas para representar elementos que no tienen presencia durante la ejecución.

# Tipo de Vista de Componentes y Conectores

| | |
|---|---|
| Elementos | Tipos de componentes: unidades principales de procesamiento y almacenamiento de datos<br>Tipos de conectores: mecanismos de interacción |
| Relaciones | Vínculos: los puertos de las componentes se asocian con roles específicos en los conectores; un puerto *p* en una componente se vincula con un rol *r* en un conector si la componente interactúa a través del conector usando la interfaz descrita por *p* y de acuerdo con lo esperado por *r*. |
| Propiedades de los elementos | Componentes.<br>▫ Nombre: debe sugerir su funcionalidad<br>▫ Tipo: define la funcionalidad general, el número y tipo de puertos y las propiedades requeridas<br>▫ Otras propiedades: dependen del tipo de componentes tales como los valores de performance y confiabilidad<br>Conectores.<br>▫ Nombre: debe sugerir la naturaleza de la interacción<br>▫ Tipo: define la naturaleza de la interacción, el tipo y el número de roles y las propiedades requeridas<br>▫ Otras propiedades: dependiendo del tipo de conector puede incluir el protocolo de interacción y valores de performance |
| Topología | El tipo de vista de componentes y conectores no tiene restricciones topológicas inherentes |

# Componentes

- El nombre de una componente debe
  - sugerir su funcionalidad,
  - servir de relación entre la documentación gráfica y la textual.
- Toda componente en una vista de C&C tiene un tipo:
  - tipos genéricos:
    - clientes, servidores, filtros, objetos, bases de datos;
  - tipos más específicos:
    - controlador, sensor.
- Un tipo de componente se define por:
  - la naturaleza de su funcionalidad,
  - su forma.

# Tipos e Instancias de Componentes

- Toda vista de C&C debe definir los tipos de componentes que incluye, pero la vista en sí se compone sólo de instancias.
- La relación entre tipos e instancias de componentes es similar a la que existe en orientación a objetos:
  - una instancia tiene todos los puertos y funcionalidad definidas para el tipo,
  - pero puede definir nuevos puertos o asociar otras implementaciones.

# Interfaces y Puertos

- Las componentes tienen interfaces, que son sus puntos de potencial interacción con el ambiente.
- Las interfaces de las componentes en una vista de C&C se denominan *puertos* para enfatizar su naturaleza dinámica.
- Cada componente puede tener varios puertos, del mismo o de distinto tipo.
- Varias componentes pueden componerse mediante conectores para dar lugar a una componente más compleja.

# Conectores

- Tipos sencillos de conectores pueden ser:
  - llamados a procedimientos entre dos objetos o entre un cliente y un servidor,
  - mensajes asincrónicos,
  - multicast de eventos entre componentes que se comunican mediante publicación-suscripción,
  - tubos que representan datos asincrónicos que preservan el orden.
- Pero existen conectores mucho más complejos (canales de comunicación, etc.) los cuales se pueden expresar desde los mas sencillos.

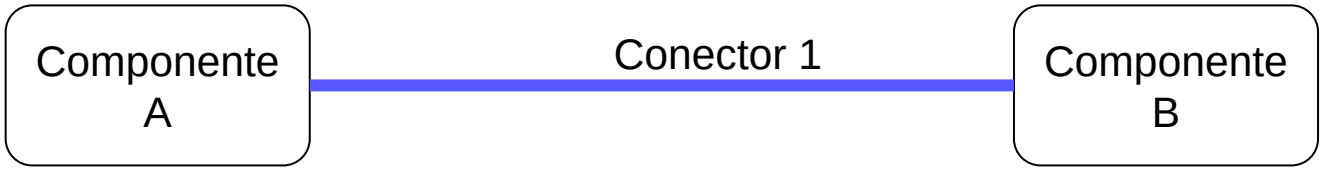
# Tipos de Conectores

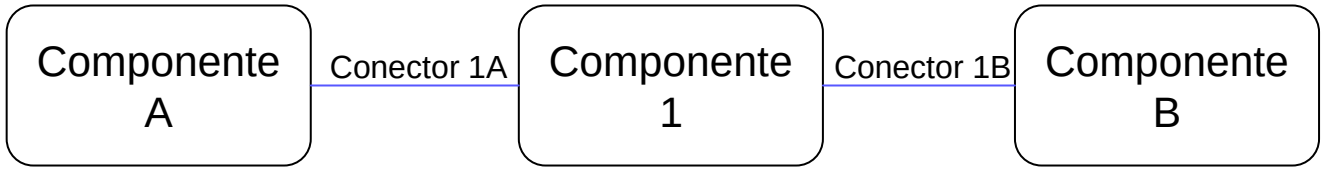
- Todo conector tiene un tipo que define la naturaleza de la interacción que éste soporta.
- También indica la forma que puede tomar el conector:
  - cuántas componentes puede conectar
  - el tipo y número de sus interfaces
  - propiedades requeridas
- Generalmente el tipo de interacción se define mediante un protocolo
  - patrones de eventos o acciones que suceden durante la interacción.

# Interfaces y Roles

- Las interfaces de los conectores se denominan *roles.*
- Los roles definen las formas en que las componentes pueden usar los conectores para interactuar:
  - un conector cliente-servidor tiene los roles *invocar-servicio* y *proporcionar-servicio;*
  - un tubo puede tener los roles *lector* y *escritor;*
  - un conector de publicar y suscribir tiene los roles *publicador* y *suscriptor*.

# Necesidad de Conectores



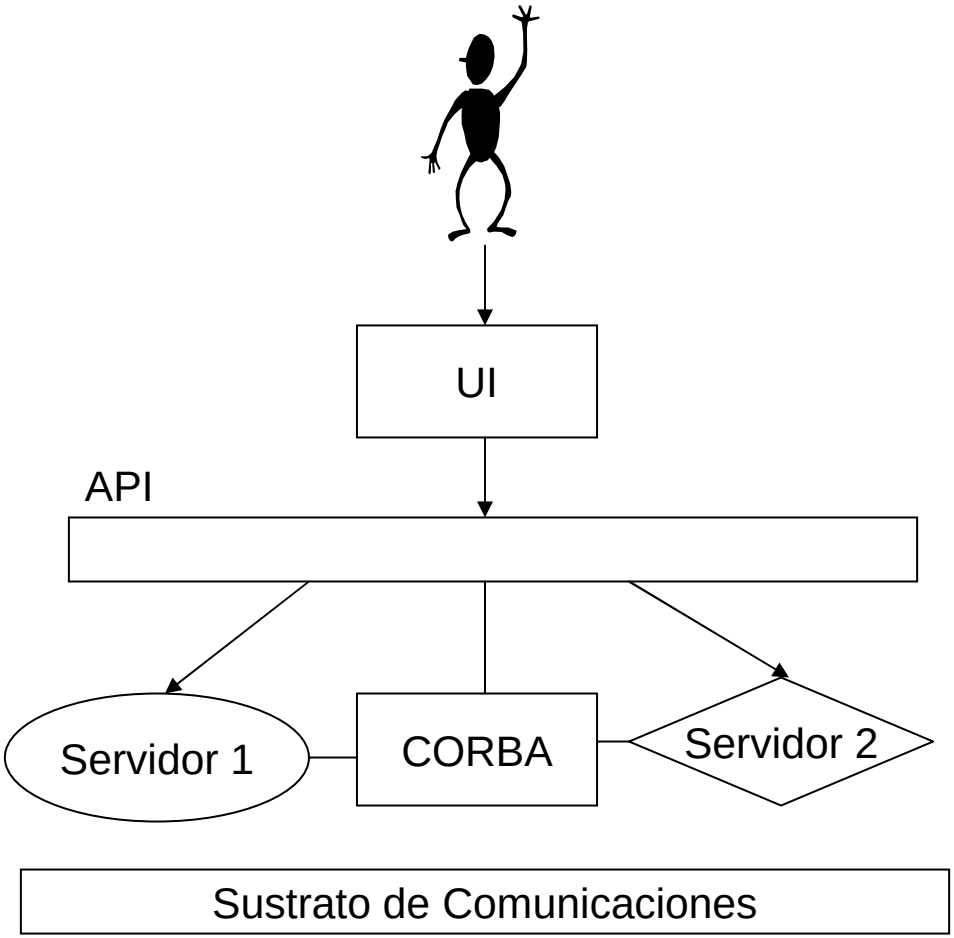

¿Son equivalentes?

# Relaciones

- Las únicas relaciones en los tipos de vistas de C&C son los vínculos (attachments).
- Un vínculo indica qué conectores se vinculan a qué componentes definiendo un grafo.
- Formalmente esto se hace asociando un puerto de una componente a un rol de un conector.

# Malas Prácticas

- No existe explicación de los elementos.
- API es una componente.
- Usa las mismas formas para distintos elementos.
- Usa distintas formas para los mismos elementos.
- Confunde el sistema y el contexto.
- No se explica el uso de flechas.
- Las interfaces no son explícitas.



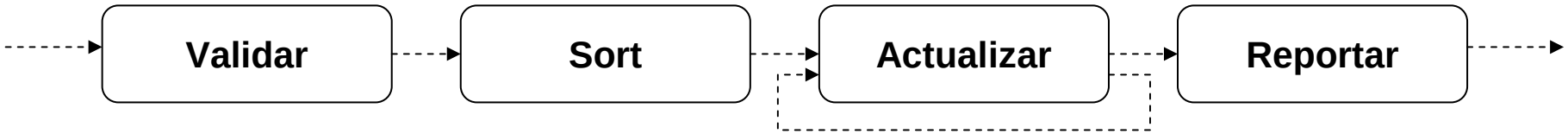
# Distintas Tipos de Estilos de Componentes y Conectores


- Arquitecturas de flujo de datos
- Arquitecturas centradas en datos
- Arquitecturas de llamada-retorno
- Arquitecturas de componentes independientes
- Estilos heterogéneos

# Arquitecturas de Flujo de Datos


- El sistema se percibe como una sucesión de transformaciones que sufre una serie de datos de entrada a través del sistema.
- Los datos ingresan al sistema y fluyen a través de las componentes una a una hasta que se asignan a un destino final: salida o almacenamiento.
- Son típicos casos de arquitecturas que promueven la reusabilidad y la modificabilidad.
- Existen dos subestilos:
  - secuencial por lotes
  - tubos y filtros

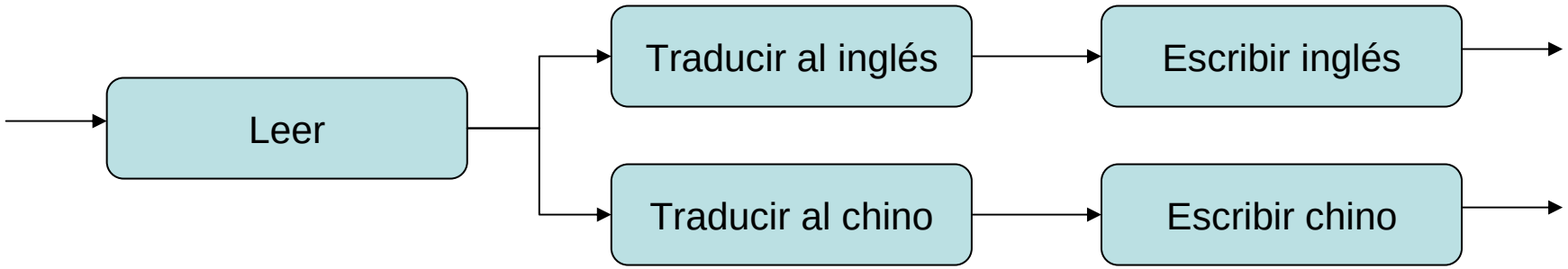
# Arquitectura Secuencial por Lotes




- Es la estructura típica de un sistema de procesamiento de datos tradicional por lotes (batch).
- Cada proceso se ejecuta completamente antes de comenzar la ejecución del siguiente.

# Arquitectura de Tubos y Filtros

- Enfatiza la transformación incremental de los datos a través de las sucesivas componentes.
- Cada componente (filtro) es un traductor que
  - procesa los datos de entrada
  - usa poca información de contexto
  - no retiene información de estado



# Tubos y Filtros: Consecuencias

- Beneficios:
  - los archivos intermedios no son necesarios
  - flexibilidad
  - reutilización de filtros
  - rápida prototipación
  - eficiencia con procesamiento paralelo.
- Desventajas:
  - compartir información de estado es caro y poco flexible
  - ineficiencia por conversión de datos
  - errores pueden implicar reiniciar el sistema.

# Características de Tubos y Filtros

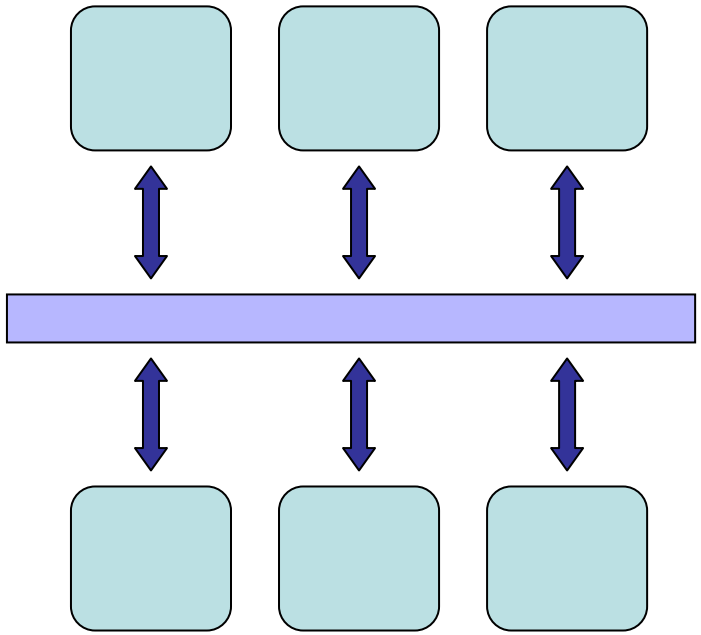
- Los tubos no tienen estado interno y simplemente comunican datos entre los filtros.
- Tubos y filtros ejecutan en forma no determinística sobre los datos hasta que éstos se acaban.
- Restricciones de conexión:
  - existe una fuente de datos conectada al puerto input de un filtro;
  - existe un destino de datos conectado al puerto output de un filtro.

# Más Estilos C&C

- Publicador-suscriptor
- Cliente-servidor
- Peer-to-peer
- Procesos comunicantes

# Publicador-suscriptor

- Los componentes interactúan a través de eventos anunciados.
- Las componentes pueden suscribirse a una serie de eventos.
- La infraestructura de ejecución de un sistema publicador-suscriptor es responsable de distribuir los eventos a todos los suscriptores.
- La forma esencial de conector es un bus de eventos.
- Los componentes publican los eventos en el bus y el conector los dirige a las componentes apropiadas.

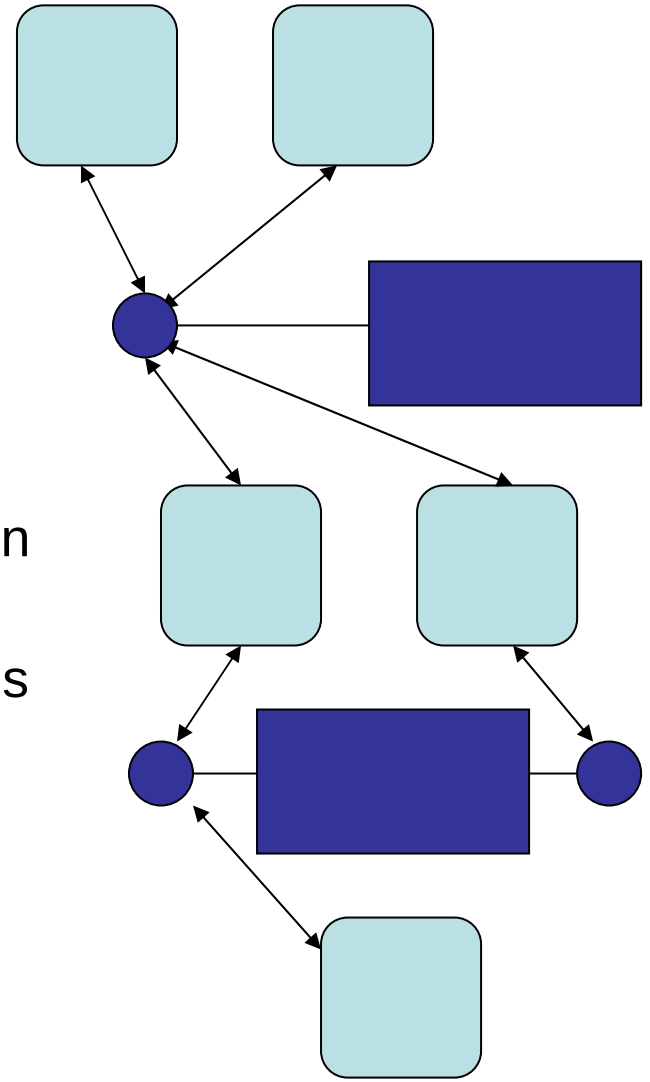


# Resumen de Publicador-Suscriptor

| | |
|---|---|
| Elementos | Tipos de componentes: cualquier tipo de componente con una interfaz que publica y/o suscribe eventos.<br>Tipos de conectores: publicar-suscribir |
| Relaciones | La relación de vínculo asocia componentes con el conector publicar-suscribir |
| Modelo Computacional | Un sistema de componentes independientes que anuncian eventos y reacciones ante otros eventos anunciados |
| Propiedades | Todas las propiedades de C&C y además: qué componentes anuncian qué eventos, qué componentes reaccionan con qué eventos y cuándo se permite a una componente suscribir un evento |
| Topología | Todas las componentes se conectan a un distribuidor de eventos que puede ser considerado como un conector (un bus) o bien una componente |

# Estilo Cliente-Servidor

- En el estilo cliente-servidor, las componentes interactúan solicitando servicios de otras componentes.
- La esencia del estilo radica en que la comunicación se da de a pares y es iniciada por el cliente.
- Una solicitud de un cliente se relaciona con un servicio ofrecido por un servidor.
- Los servidores ofrecen uno o más servicios a través de una o más interfaces.
- Puede haber uno o más servidores dentro del sistema.



# Elementos, Relaciones y Propiedades

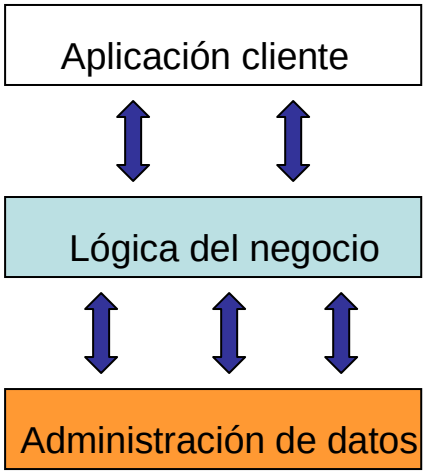
- Los tipos de componentes son *clientes* y *servidores.*
- El tipo de conector principal es *solicitud-respuesta* que se usa para invocar servicios.
- Los servidores tienen interfaces que describen los servicios ofrecidos.
- Los servidores pueden, a su vez, actuar como clientes de otros servidores, formando una jerarquía.

# Dinámica

- El flujo de control en los sistemas cliente-servidor es asimétrico.
- En un sistema cliente-servidor puro:
  - la acción es iniciada por el cliente solicitando un servicio del servidor (el cliente debe conocer la identificación del servidor y el servicio requerido)
  - los servidores desconocen sus clientes potenciales, y responden a quienes los invoquen
- Invocación sincrónica:
  - el solicitante del servicio espera o se bloquea hasta que el servicio sea concluido posiblemente con un resultado devuelto.

# Forma especializada: n-tiered

- Estructura:
  - los clientes y servidores forman una jerarquía de n niveles
  - los niveles superiores están formados por clientes que invocan servidores en niveles inferiores
- Generalmente se usan en aplicaciones de procesamiento de información, donde n = 3
  - la primera capa es la aplicación cliente
  - la segunda incluye los servicios de la lógica del negocio
  - la tercera capa incluye la administración de los datos



# Resumen de Cliente-Servidor

| | |
|---|---|
| Elementos | Tipos de componentes: clientes (solicita servicios de otra componente) y servidores (proporciona servicios a otras componentes)<br>Tipos de conectores: solicitud/respuesta, invocación asimétrica de un servicio del servidor por parte de un cliente |
| Relaciones | La relación de vínculo asocia los clientes con el rol de solicitud del conector y a los servidores con el rol de respuesta del conector, y determina qué servicios pueden ser invocados por qué cliente |
| Modelo Computacional | Los clientes inician la acción solicitando servicios cuando lo requieran de los servidores y esperando por los resultados |
| Propiedades | Propiedades de C&C agregando el número y tipo de clientes que pueden vincularse y propiedades de performance tales como transacciones por segundo |
| Topología | En general, sin restricciones. Las especializaciones pueden imponer algunas: número de vínculos a un determinado puerto o rol, posibilidad de relacionar servidores, capas. |

# Estilo Peer-to-Peer

- Los componentes interactúan directamente como pares intercambiando servicios.
- La comunicación es una especie de solicitud/respuesta sin la asimetría de cliente-servidor
  - en principio cualquier componente puede interactuar con cualquier otra
- Los conectores pueden involucrar protocolos complejos bidireccionales
- Típicos sistemas peer-to-peer son los basados en infraestructuras de objetos distribuidos tales como CORBA, COM+ y Java RMI.
- Un sistema diseñado con un estilo más restrictivo puede aún ser implementado usando orientación a objetos y una infraestructura de objetos/componentes distribuidos.

# Elementos, Relaciones y Propiedades

- Los típicos elementos son objetos, objetos distribuidos y clientes.
- El conector principar es *invocación-procedimiento*.
- La interacción puede ser inciada por cualquiera de las partes.
- Cada componente tiene interfaces que describen los servicios que solicitan de otras componentes y los servicios que ofrecen.
- Las propiedades esenciales son la modificabilidad y la escalabilidad.

# Dinámica

- El flujo de control es simétrico:
  - un par inicia la acción para realizar una operación mediante la cooperación con otros pares y para ello se solicitan servicios entre ellos.
- La propagación de las invocaciones puede, eventualmente ser restringida.

# Resumen de Peer-to-Peer

| | |
|---|---|
| Elementos | Tipos de componentes: pares<br>Tipos de conectores: invocación de procedimiento |
| Relaciones | La relación de vínculo asocia pares con los conectores de invocación a procedimientos y determina el grafo de interacciones posibles entre componentes |
| Modelo Computacional | Los pares cuentan con interfaces en encapsulan su estado. El cómputo se realiza mediante la cooperación entre pares que solicitan servicios a otros pares. |
| Propiedades | Propiedades de C&C con énfasis en protocolos de interacción y propiedades orientadas a la performance. Los vínculos pueden cambiar durante la ejecución. |
| Topología | Puede haber restricciones acerca del número de vínculos a un cierto puerto o rol. Otras restricciones de visibilidad pueden imponerse para restringir qué componentes saben de la existencia de otras componentes. |

# Estilo de Procesos Comunicantes

- Las componentes son procesos independientes ejecutando en forma concurrente y comunicándose a través de variados mecanismos.
- Los conectores pueden ser: sincronización, pasaje de mensajes, intercambio de datos, inicialización, destrucción, etc.
- Los procesos comunicantes son frecuentes en grandes sistemas, e imprescindibles en sistemas distribuidos.
- El estilo de procesos comunicantes ayuda a comprender el comportamiento asociado a la concurrencia.

# Elementos, Relaciones y Propiedades


- Un sistema se representa como un conjunto de unidades que ejecutan en forma concurrente conjuntamente con su interacción.
- Una unidad concurrente puede ser una tarea, o proceso o un thread.
- Los conectores permiten la comunicación de datos entre las componentes y también su sincronización.

# Resumen de Procesos Comunicantes

| | |
|---|---|
| Elementos | Tipos de componentes: unidades concurrentes tales como tareas, procesos y threads<br>Tipos de conectores: intercambio de datos, pasaje de mensajes, sincronización, control y otros tipos de comunicación |
| Relaciones | La relación de vínculo tal como se define en C&C |
| Modelo Computacional | Componentes que ejecutan concurrentemente interactúan a través de mecanismos de conexión específicos |
| Propiedades de los elementos | Unidad concurrente: *preemptability*, indica que la ejecución de una unidad concurrente puede ser suspendida por otra unidad concurrente o que la unidad concurrente ejecuta hasta que suspende voluntariamente su propia ejecución; *priordad*, determinada por la planificación; parámetros temporales, tales como período o deadline<br>Intercambio de datos: con buffer, indica que los mensajes se almacenan si no pueden procesarse inmediatamente, o protocolo, usado para comunicación |
| Topología | Grafos arbitrarios |

# Créditos

- Cecilia Bastarrica. Universidad de Chile