

Ingeniería del Software I

3 - Arquitectura del Software (Capítulo 4)

Arquitectura del software

“With a small program of a few hundred lines, you can get away without much organization, but as programs scale, they quickly become impossible to manage alone...

Much of this challenge occurs because requirements *change*, and every time they do, code has to change to accommodate. The more code there is and the more entangled it is, the harder it is to change and more likely you are to break things.

This is where architecture comes in.”

— Amy J. Ko

Arquitectura del software

- Todo sistema complejo se compone de **subsistemas** que interactúan.
- Un enfoque para diseñar sistemas es identificar **subsistemas** y la forma que **interactúan** entre ellos.
- La arquitectura de software tiene ese objetivo.
- Es un área que recibe mucha atención.

El rol de la arquitectura de software

Definición: La arquitectura de SW de un sistema es la estructura del sistema que comprende los elementos del SW, las **propiedades externamente visibles** de tales elementos, y la relación entre ellas.

- Por cada elemento sólo interesan las propiedades externas necesarias para especificar las relaciones.
- Para la arquitectura **no** son importantes los detalles de cómo se aseguran dichas propiedades.
- La definición no habla de la bondad de la arquitectura; para ello se necesita analizarla.

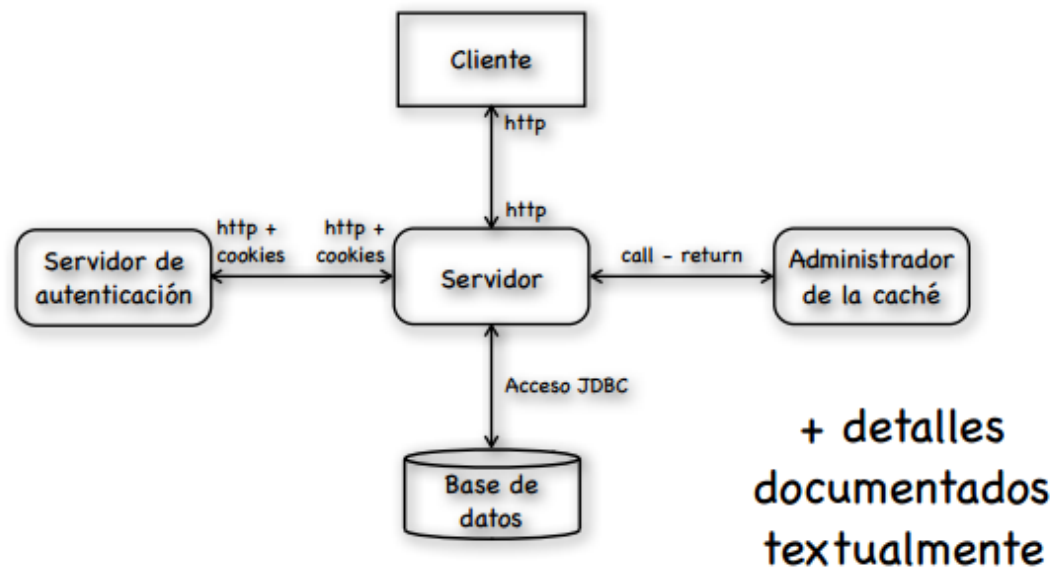
En general, se tienen varias estructuras (de aspectos ortogonales). Una descripción arquitectónica del sistema describe las distintas estructuras del sistema.

Arquitectura del software

- La arquitectura es el diseño del sistema al más alto nivel.
- A este nivel se hacen las elecciones de tecnología, productos a utilizar, servidores, etc.
 - No es posible diseñar los detalles del sistema y luego incorporar estas elecciones.
 - La arquitectura debe ser creada de manera que se adapte a estas elecciones.
- Es la etapa más temprana donde algunas propiedades como confiabilidad y desempeño pueden evaluarse.

Arquitectura del software

- La arquitectura es el diseño del sistema al más alto nivel.
- A este nivel se hacen las elecciones de tecnología, productos a utilizar, servidores, etc.
No es posible diseñar estas elecciones.
La arquitectura debe documentar estas elecciones.
- Es la etapa más temprana en la que se debe considerar la confiabilidad y desempeño.



El rol de la arquitectura de software

- La arquitectura es un diseño del SW que da una visión de muy alto nivel de las partes del sistema y de las relaciones entre ellos que conforman el todo.

Divide al sistema en partes lógicas tal que cada una puede ser **comprendida independientemente**.

Y describe las relaciones entre ellas (que puede ser un proceso complejo).

- Un sistema (SW) complejo puede dividirse de varias formas, cada una proveyendo una visión distinta.
Además hay varias estructuras posibles.

El rol de la arquitectura de software

¿Por qué?

Comprensión y comunicación:

- Al mostrar la estructura de alto nivel del sistema ocultando la complejidad de sus partes, la descripción arquitectónica facilita la comunicación.
- Define un marco de comprensión común entre los distintos interesados (usuarios, cliente, arquitecto, diseñador, etc.).
- Útil para negociación y acuerdos.
- También puede ayudar a comprender el sistema existente.

El rol de la arquitectura de software

¿Por qué?

Reuso:

- El reuso es considerado una de las principales técnicas para incrementar la productividad.
- Una forma de reuso es componer el sistema con partes existentes, reusadas, y otras nuevas.
- Se facilita si a un alto nivel se reusan componentes que proveen un servicio completo.
 - => La arquitectura es muy importante: se elige una arquitectura tal que las **componentes existentes encajen** adecuadamente con otras componentes a desarrollar.
 - => Las decisiones sobre el uso de componentes existentes se toman en el momento de diseñar la arquitectura.

El rol de la arquitectura de software

¿Por qué?

Construcción y evolución:

- La división provista por la arquitectura servirá para **guiar** el desarrollo del sistema:
 - Cuáles partes son necesarias construir primero, cuáles partes ya están construidas.
 - Ayuda a **asignar equipos de trabajo**: las distintas partes pueden construirse por distintos grupos (las partes son relativamente independientes entre sí).
- Durante la evolución del SW, la arquitectura ayuda a decidir **cuáles partes necesitan cambiarse** para incorporar nuevas características o cambios, y a decidir **cuál es el impacto** de tales cambios en las otras componentes.

El rol de la arquitectura de software

¿Por qué?

Análisis:

- Es deseable que propiedades de **confiabilidad y desempeño** puedan determinarse en el diseño; esto permite considerar distintas alternativas de diseño hasta encontrar los niveles de satisfacción deseados.

Ej.: la confiabilidad/desempeño de un sistema pueden predecirse parcialmente a partir de la arquitectura, si se proveen estimadores de carga o se asume un hw específico.

- Requerirá descripción precisa de la arquitectura así como de las propiedades de las componentes.

Vistas de la arquitectura

- **No** hay una única arquitectura de un sistema.
- Hay distintas vistas de un sistema de software (paralelismo con ingeniería civil).
- Una vista consiste de elementos y relaciones entre ellos, y describe una estructura.
- Los elementos de una vista dependen de lo que la vista quiera destacar.
- Distintas vistas exponen distintas propiedades.
- Una vista que se enfoca en algún aspecto reduce la complejidad con la que debe enfrentarse el lector.

Vistas de la arquitectura

- Muchos tipos de vistas fueron propuestos.
- La mayoría pertenece a alguno de estos tres tipos:
 - Módulo
 - Componentes y conectores (C&C)
 - Asignación de recursos
- Las distintas vistas están correlacionadas (todas representan al mismo sistema!)
 - Existen relaciones entre los elementos de una vista y los de otra; tal relación puede ser muy compleja.

Vistas de la arquitectura

Vista de módulos:

- Un sistema es una colección de **unidades de código** (ellas no representan entidades en ejecución).
i.e., los elementos son módulos. Ej.: Clases, paquetes, funciones, procedimientos, métodos, etc.
- La relación entre ellos está basada en el código.
Ej.: "parte de", "usa a" o "depende de", llamadas, generalización o especialización, etc.

Vistas de la arquitectura

Vista de componentes y conectores:

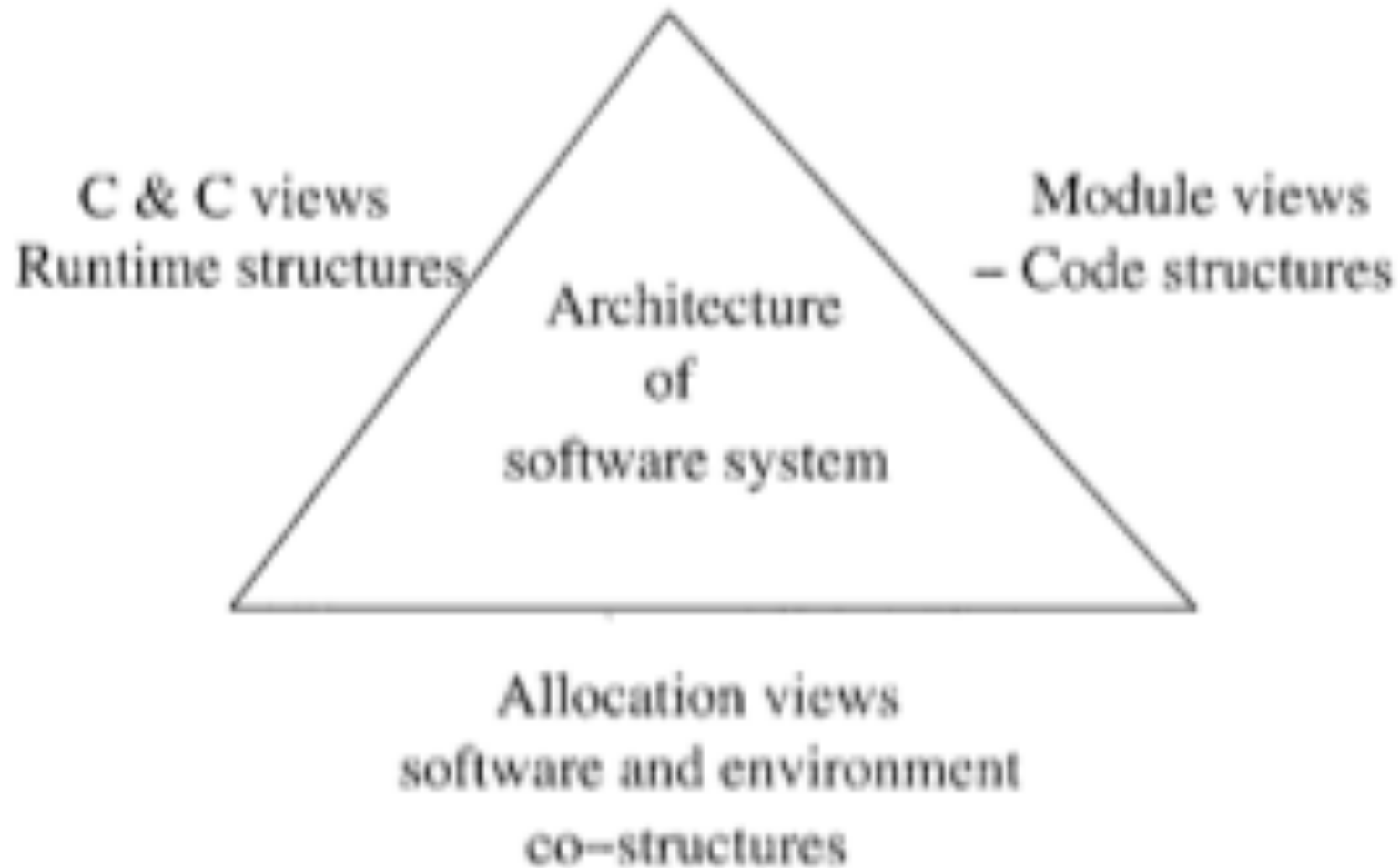
- Los elementos son **entidades de ejecución** denominados componentes, i.e., una componente es una unidad que tiene identidad dentro del sistema en ejecución. Ej.: objetos, procesos, .exe, .dll, etc.
- Los conectores proveen el medio de **interacción** entre las componentes. Ej.: pipes, sockets, memoria compartida, protocolos, etc.

Vistas de la arquitectura

Vista de asignación de recursos:

- Se enfoca en **cómo** las unidades de SW se asignan a recursos como hw, sistemas de archivos, gente, etc.
 - i.e., especifica la relación entre los elementos del SW y las unidades de ejecución en el entorno.
- Exponen propiedades estructurales como qué proceso ejecuta en qué procesador, qué archivo reside dónde, etc.

Vistas de la arquitectura



Vistas de la arquitectura

- Una descripción arquitectónica consiste de vistas de distintos tipos, cada una mostrando una estructura diferente.
Distintos sistemas necesitan distintos tipos de vistas dependiendo de las necesidades.

Ej.: análisis de desempeño => C&C
asignación de recursos => asignación de recursos
planeamiento => módulos
- La vista de C&C es la que casi siempre se hace, y se ha transformado en la vista principal.
Nos enfocaremos en C&C.
La vista de módulos se verá más adelante, en diseño de alto nivel (cuyo enfoque es identificar módulos).

La vista de componentes y conectores

- Dos elementos principales: componentes y conectores.
- **Componentes**: son elementos computacionales o de almacenamiento de datos.
- **Conectores**: son mecanismos de interacción entre las componentes.
- Una vista C&C define las componentes y cómo se conectan entre ellas a través de conectores.
- La vista C&C describe una estructura en ejecución del sistema: qué componentes existen y cómo interactúan entre ellos en **tiempo de ejecución**.
- Es básicamente un grafo donde las componentes son los nodos y los conectores las aristas.

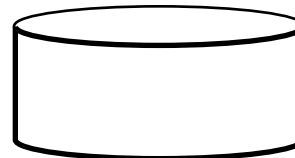
La vista de componentes y conectores

Componentes

- Son unidades de cómputo o de almacenamiento de datos.
- Cada componente tiene un nombre que representa su rol y le provee una identidad.
- Cada componente tiene un tipo.
Los distintos tipos se representan con distintos símbolos.
- Las componentes utilizan interfaces o puertos para comunicarse con otras componentes.



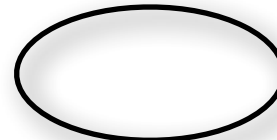
Cliente



Base de datos



Servidor



Aplicaciones

La vista de

Las notaciones varían significativamente según el lenguaje o la herramienta que se utilice.

Ej.: ACME (<http://www-2.cs.cmu.edu/~acme/>);

Wright (<http://www-2.cs.cmu.edu/~able/wright/>);

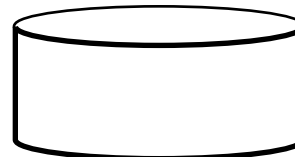
[UML\(www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr008.pdf\)](http://www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr008.pdf)

- S
- Cada co
una i
- Cada componente tiene un tipo.
Los distintos tipos se representan con distintos
- Las componentes utilizan interfaces o puertos para
otras componentes.

Atención: el libro no es muy consistente con esta notación.



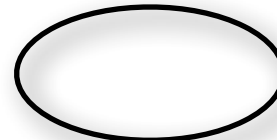
Cliente



Base de datos



Servidor



Aplicaciones

La vista de componentes y conectores

Conectores

- Describen el **medio** en el cual la interacción entre componentes toma lugar.
- Un conector puede proveerse por medio del entorno de ejecución. Ej.: llamada a procedimiento/función.
- Sin embargo, los conectores pueden también ser mecanismos de interacción más complejos, ej.: puertos TCP/IP, RPC, protocolos como HTTP, etc.

Notar que estos mecanismos requieren una infraestructura de ejecución significativa + programación especial dentro de la componente para poder utilizarla.

Importante identificarlos/representarlos explícitamente.

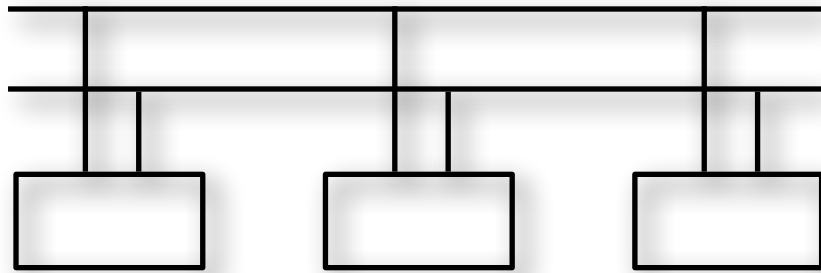
La vista de componentes y conectores

Conectores

- Los conectores no necesariamente son binarios (ej.: bus de broadcasting).
- Los conectores tienen:
 - Nombre, que identifican la naturaleza de la interacción, y
 - Tipo, que identifica el tipo de interacción (binaria o n-aria, unidireccional o bidireccional, etc.)
- Muchas veces los conectores representan protocolos => las componentes necesitan seguir algunas convenciones al usar el conector (ej.: TCP).
- Los distintos tipos de conectores se representan con distintas notaciones.

La vista de componentes y conectores

Conectores



Bus



Acceso a base de datos



Request-reply



Pipe



RPC

La vista de componentes y conectores

Ejemplo

- Se desea diseñar y construir un sistema de encuesta a alumnos de la facultad.

La encuesta es multiple-choice y los alumnos las envían online.

Cada vez que un estudiante envía su encuesta, se le muestran los resultados actuales.

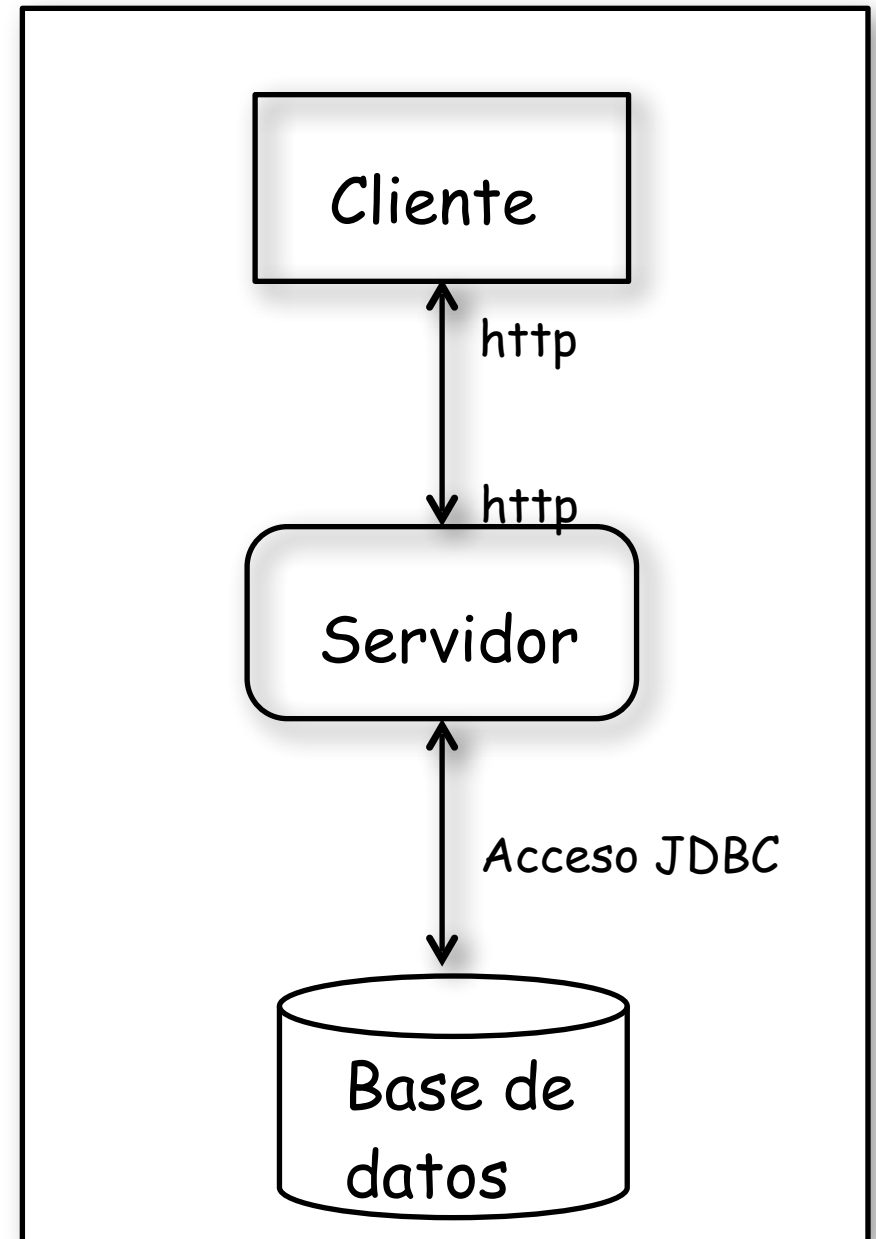
- El sistema se construirá sobre la web para facilitar accesos y desarrollo.

La vista de componentes y conectores

Ejemplo

Se propone una arquitectura de 3 niveles (también llamadas "capas"):

- Los conectores entre ellos también tienen distintos tipos.
- Tiene como componentes a un cliente, a un servidor, y a una base de datos (cada una de distinto tipo).



La vista de componentes y conectores

Ejemplo

- A nivel de arquitectura se omiten muchos detalles
(URL del sitio, módulos requeridos para construir cada componente, lenguaje, etc., no son características relevantes a este nivel.)
- Los **conectores** se establecen **explícitamente**
=> la infraestructura debe proveer http, browsers, etc.
- La elección de conectores impone restricciones sobre cómo se diseñarán y construirán finalmente cada componente.

La vista de componentes y conectores

Ejemplo - Extensión I

Esta arquitectura no provee seguridad: la encuesta podría ser realizada por cualquiera.

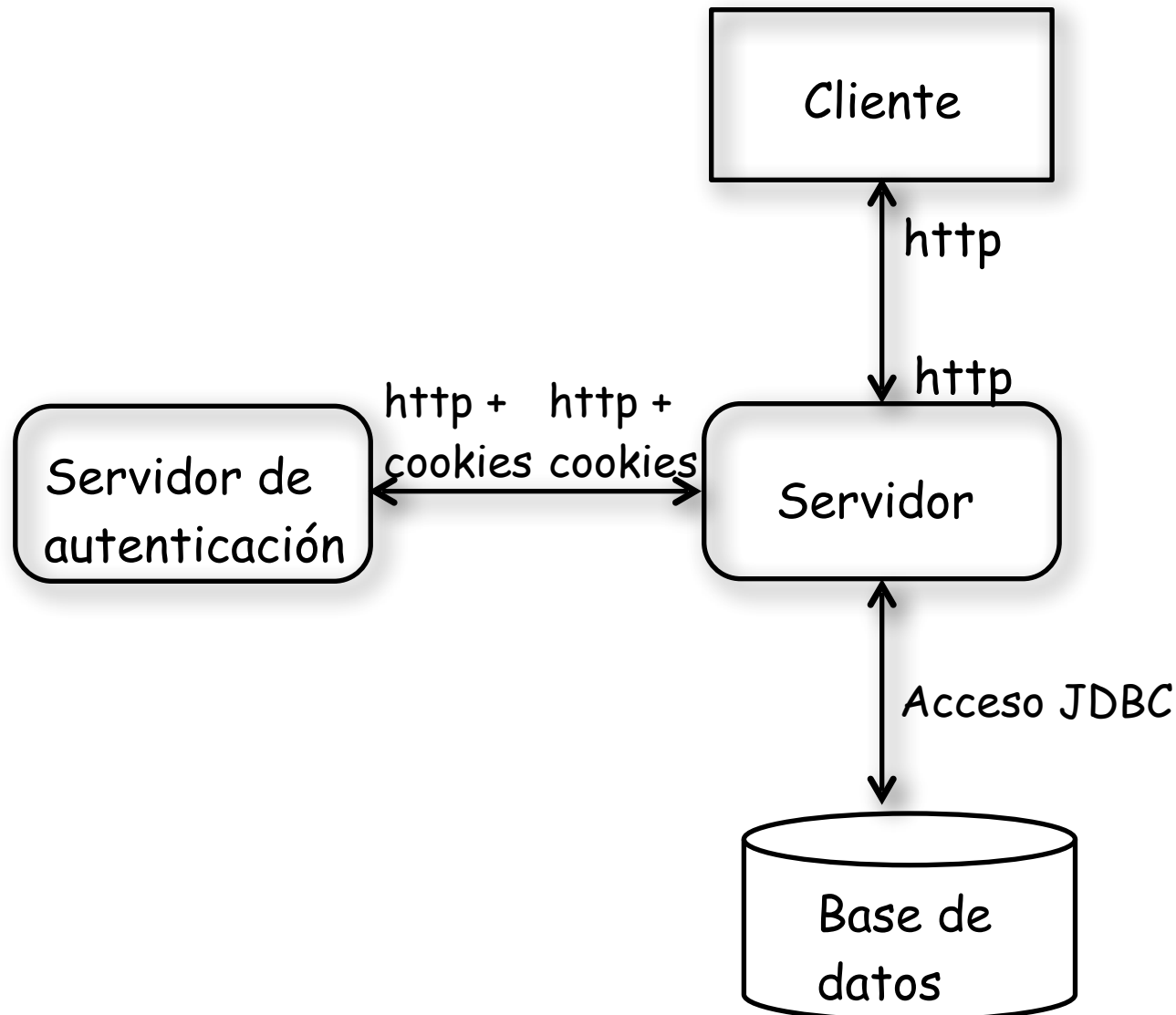
Sin embargo sólo los alumnos deberían responderla y a lo sumo una vez.

Para identificar a los alumnos y verificar que sólo envían una encuesta se requiere un servidor de autenticación.

Se requerirá el uso de cookies, con el servidor construido apropiadamente (usar http con cookies en el conector entre servidor y servidor de autenticación.)

La vista de componentes y conectores

Ejemplo - Extensión I



La vista de componentes y conectores

Ejemplo - Extensión II

Resultó que la base de datos está caída con más frecuencia de la esperada.

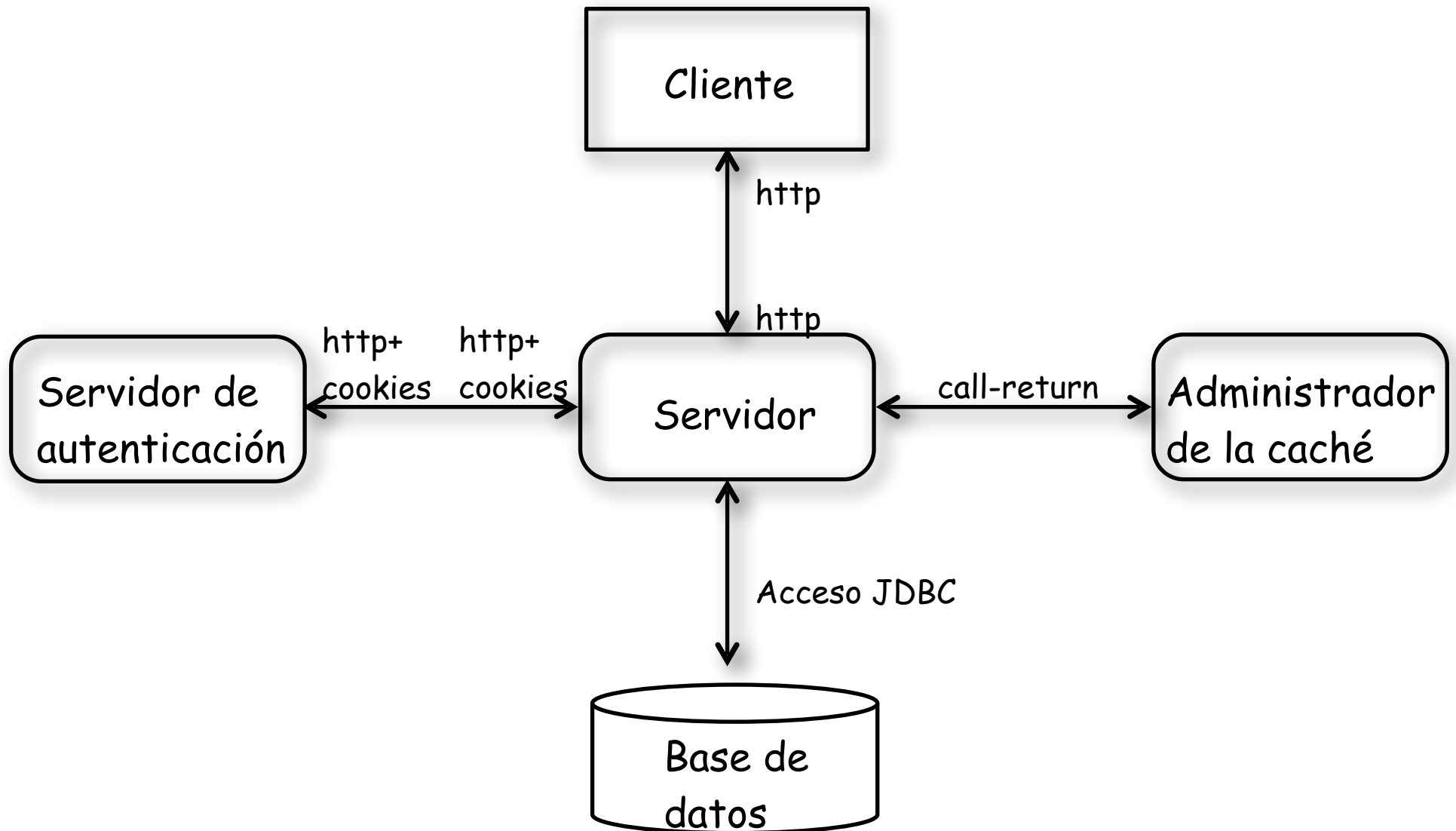
Además, es aceptable que los estudiantes reciban resultados parciales un poco desactualizados (una desactualización de 5 encuestados es tolerable).

Para incrementar la disponibilidad del sistema se decidió agregar una caché:

Admite hasta 5 encuestas sin registrar en la BD.
Registra el último resultado parcial posible.

La vista de componentes y conectores

Ejemplo - Extensión II



La vista de componentes y conectores

Ejemplo

Extensión I: agrega seguridad.

Extensión II: incrementa desempeño y confiabilidad.

i.e. las elecciones a nivel de arquitectura tienen un impacto muy grande en las propiedades del sistema.

Elegir la arquitectura adecuada ayuda a construir buenos sistemas.

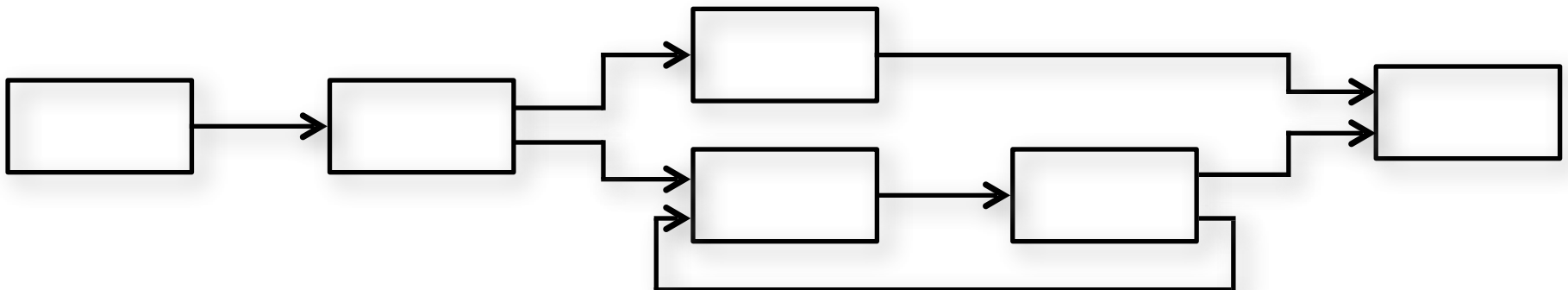
Estilos arquitectónicos para la vista de C&C

- Sistemas distintos tienen estructuras de C&C distintas.
- Algunas estructuras son generales y son útiles para una clase de problemas => estilos arquitectónicos.
- Un estilo arquitectónico define una familia de arquitecturas que satisface las restricciones de ese estilo.
- Los estilos proveen ideas para crear arquitecturas de sistemas.
- Distintos estilos pueden combinarse para definir una nueva arquitectura.

Estilos arquitectónicos para la vista de C&C

Tubos y Filtros (Pipe and Filter)

- Adecuado para sistemas que fundamentalmente realizan **transformaciones de datos**.
- Un sistema que usa este estilo utiliza una red de transformadores para realizar el resultado deseado.
- Tiene un sólo tipo de componente: filtro.
- Tiene un sólo tipo de conector: tubo.
- Un filtro realiza transformaciones y le pasa los datos a otro filtro a través de un tubo.



Estilos arquitectónicos para la vista de C&C

Tubos y Filtros

- Un filtro es una entidad **independiente** y **asíncrona** (se limita a consumir y producir datos).
- Un filtro no necesita saber la identidad de los filtros que envían o reciben los datos.
- Un tubo es un canal unidireccional que transporta un flujo de datos de un filtro a otro.
- Un tubo sólo conecta 2 componentes.
- Los tubos deben hacer "buffering" y sincronización para asegurar el correcto funcionamiento como productor y consumidor.

Estilos arquitectónicos para la vista de C&C

Tubos y Filtros

Restricciones:

- Cada filtro debe trabajar sin conocer la identidad de los filtros productores o consumidores.
- Un tubo debe conectar un puerto de salida de un filtro a un puerto de entrada de otro filtro.
- Un sistema puro de tubos y filtros usualmente requiere que cada filtro tenga su propio hilo de control, lo que implica conocer de qué manera el tubo hace buffering y sincronización.

Estilos arquitectónicos para la vista de C&C

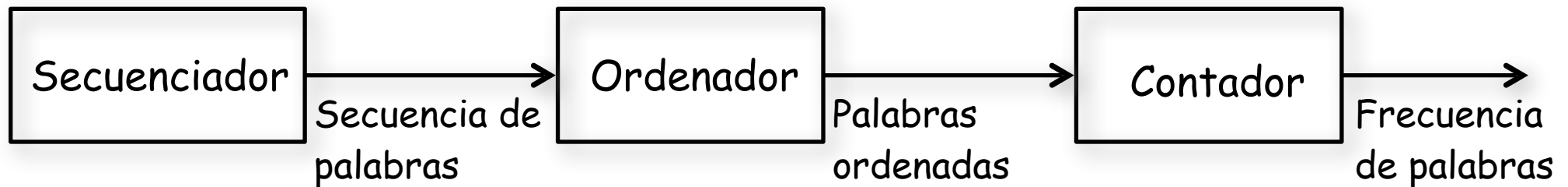
Tubos y Filtros

Ejemplo:

Un sistema requiere contar la frecuencia de distintas palabras en un archivo.

Un enfoque: (1) separar el archivo en palabras; (2) ordenar las palabras; y (3) contar el número de ocurrencias.

La arquitectura del sistema natural responde al estilo de tubos y filtros.



Estilos arquitectónicos para la vista de C&C

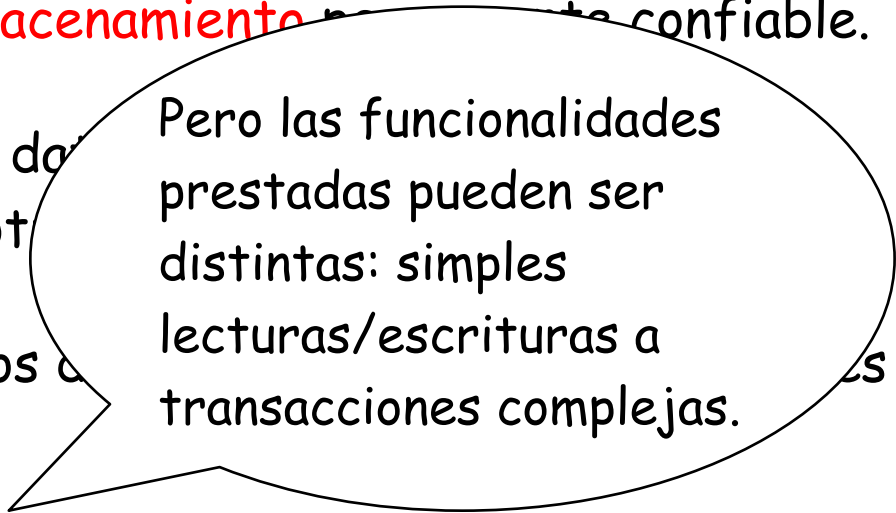
Estilo de datos compartidos

- Dos tipos de componentes: repositorio de datos y usuarios de datos.
- Repositorio de datos: provee **almacenamiento** permanente confiable.
- Usuarios de datos: acceden a los datos en el repositorio, realizan cálculos, y ponen los resultados otra vez en el repositorio.
- La comunicación entre los usuarios de los datos sólo se hace a través del repositorio.
- En este estilo sólo hay un tipo de conector: lectura/escritura.

Estilos arquitectónicos para la vista de C&C

Estilo de datos compartidos

- Dos tipos de componentes: repositorio de datos y usuarios de datos.
- Repositorio de datos: provee **almacenamiento** de datos confiable.
- Usuarios de datos: acceden a los datos, realizan cálculos, y ponen los resultados otros usuarios.
- La comunicación entre los usuarios y el repositorio.
- En este estilo sólo hay un tipo de conector: lectura/escritura.



Pero las funcionalidades prestadas pueden ser distintas: simples lecturas/escrituras a transacciones complejas.

Estilos arquitectónicos para la vista de C&C

Estilo de datos compartidos

Dos variantes principales:

- Estilo pizarra:

Cuando se agregan/modifican datos en el repositorio, se **informa** a todos los usuarios.

i.e.: la fuente de datos compartidos es una entidad activa.

- Estilo repositorio:

El repositorio es pasivo.

Ej.: sistemas orientados a base de datos; sistemas de web; entornos de programación; etc.

Estilos arquitectónicos para la vista de C&C

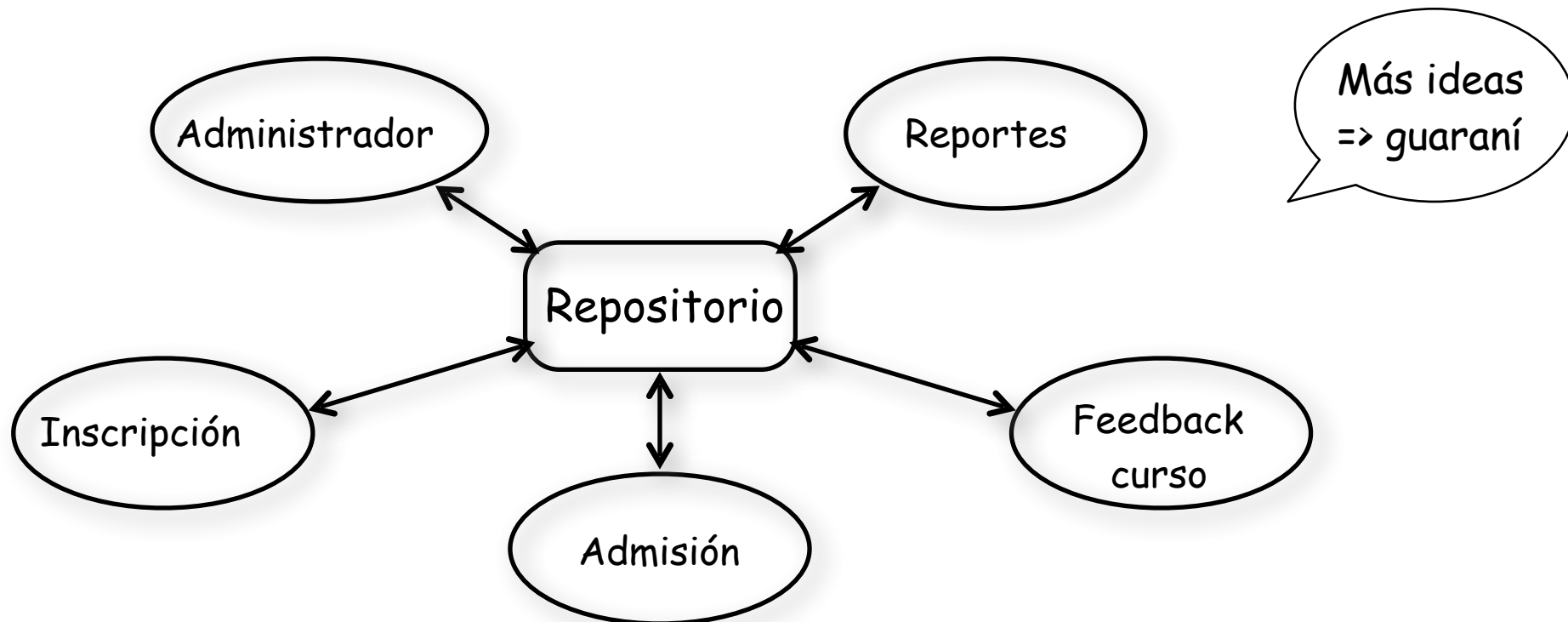
Estilo de datos compartidos

Ejemplo:

Sistema de inscripción de alumnos a los cursos.

El repositorio contiene toda la información sobre los alumnos, cursos, horarios, correlatividades, etc.

Componentes usuarias: administrador, inscripción, admisión, reportes, etc.; que realizan operaciones con los datos.



Estilos arquitectónicos para la vista de C&C

Estilo de datos compartidos

Ejemplo:

Las componentes no actúan directamente entre ellas (ni necesitan hacerlo).

Es fácil de entender: si se necesitara agregar un administrador de aulas y horarios, podría agregarse como una nueva componente usuaria.

=> Ninguna componente existente necesitaría cambiarse.

Estilos arquitectónicos para la vista de C&C

Estilo cliente-servidor

- Dos tipos de componentes: clientes y servidores.
- Los clientes sólo se comunican con el servidor, pero **no** con otros clientes.
- La comunicación siempre es iniciada por el cliente quien le envía una solicitud al servidor y espera una respuesta de éste
=> la comunicación es usualmente sincrónica.
- Solo un tipo de conector: solicitud/respuesta (request/reply) - es asimétrico.
- Usualmente el cliente y el servidor residen en distintas máquinas.

Estilos arquitectónicos para la vista de C&C

Estilo cliente-servidor

Este estilo tiene en general la forma de una estructura multi-nivel.

El servidor también actúa como cliente.

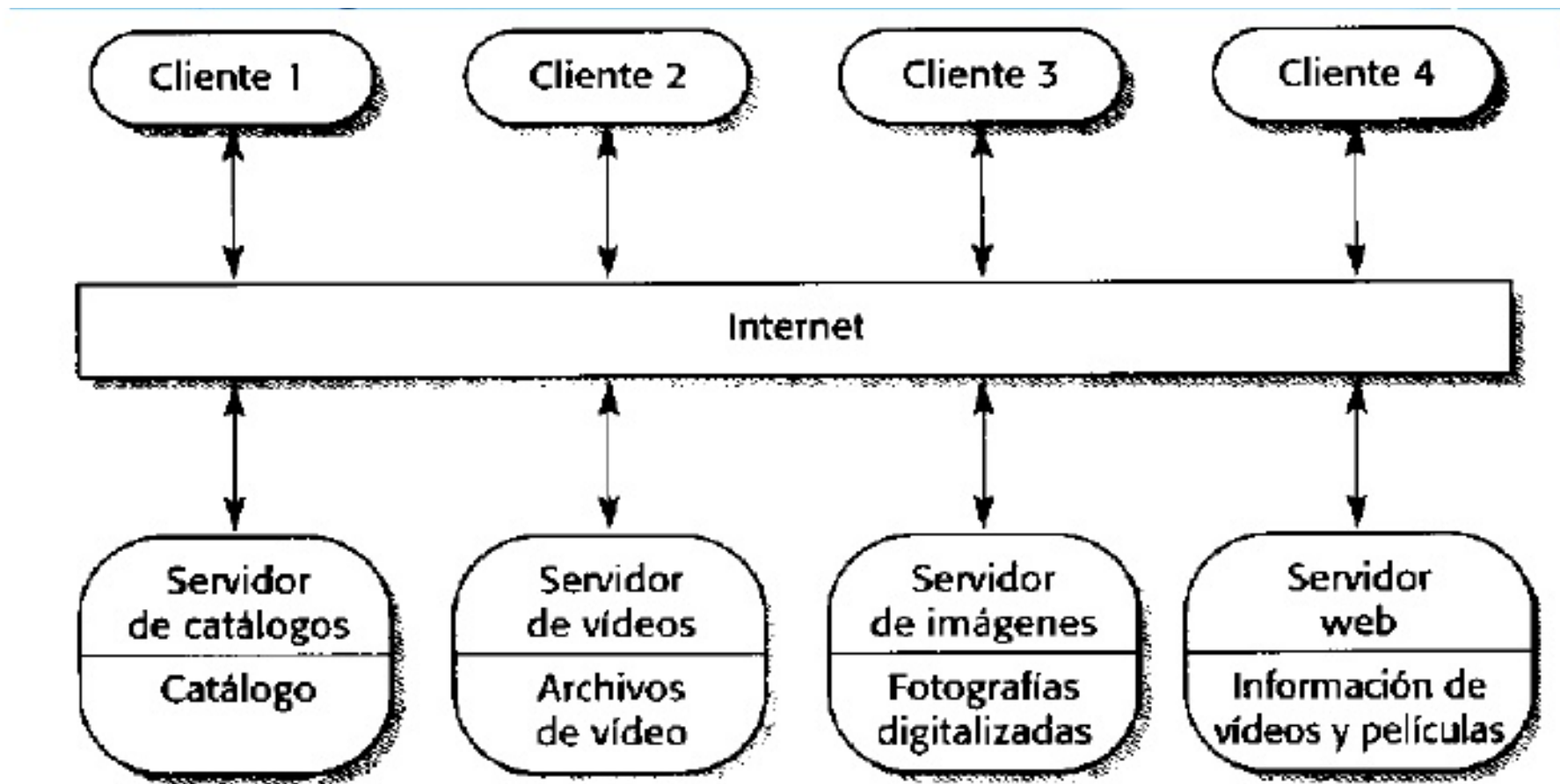
Ej. clásico => 3 niveles:

- Nivel de cliente: contiene a los clientes.
- Nivel intermedio: contiene las reglas del servicio.
- Nivel de base de datos: reside la información.

Estilos arquitectónicos para la vista de C&C

Estilo cliente-servidor

Ejemplo: Sistema de usuarios en web, de una biblioteca de imágenes



Estilos arquitectónicos para la vista de C&C

Otros estilos

Estilo publicar-suscribir (publish-subscribe):

- Dos tipos de componentes: las que publican eventos y las que se suscriben a eventos.
- Cada vez que un evento es publicado se invoca a las componentes suscriptas a dicho evento.

Estilo peer-to-peer:

- Un único tipo de componente.
- Cada componente le puede pedir servicios a otra
≈ modelo de computación orientado a objetos.

Estilo de procesos que se comunican:

- Procesos que se comunican entre sí a través de pasaje de mensajes.

Documentación del diseño arquitectónico

- Los diagramas son un medio adecuado para discutir y crear diseños.
- Sin embargo, no son suficientes para documentar el diseño arquitectónico.
- El documento de diseño arquitectónico debe especificar precisamente las **vistas** y las **relaciones** entre éstas.

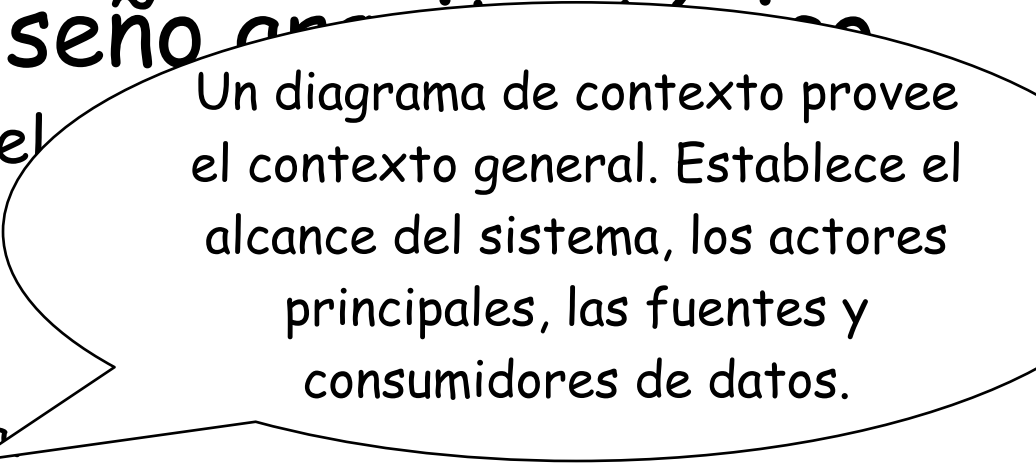
Documentación del diseño arquitectónico

Organización del documento

1. Contexto del sistema y la arquitectura.
2. Descripción de las vistas de la arquitectura.
 - a. Presentación principal de la vista.
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Documentación del diseño arquitectónico

Organización del



Un diagrama de contexto provee el contexto general. Establece el alcance del sistema, los actores principales, las fuentes y consumidores de datos.

1. Contexto del sistema y la arquitectura
2. Descripción de las vistas de la arquitectura.
 - a. Presentación principal de la vista.
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Documentación del diseño arquitectónico

Organización del

Un diagrama de contexto provee el contexto general. Establece el alcance del sistema, los actores

1. Contexto del sistema y la arquitectura
2. Descripción de las vistas de la arquitectura

Uno por cada una de los distintos tipos de vistas que se eligieron representar.

- a. Presentación principal de la vista.
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Documentación del diseño arquitectónico

Organización del

Un diagrama de contexto provee el contexto general. Establece el alcance del sistema, los actores

1. Contexto del sistema y la arquitectura
2. Descripción de las vistas de la arquitectura
 - a. Presentación principal de la vista.
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Casi siempre contiene la descripción gráfica tal como se presentó anteriormente.

Documentación del diseño

Organización del

Un diagrama de contexto provee el contexto general. Establece el alcance del sistema, los actores

1. Contexto del sistema y la arquitectura.
2. Descripción de las vistas de la interfaz.
 - a. Presentación principal de la interfaz.
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Provee más información sobre los elementos que se muestran en la presentación principal. Por c/ elem. describe su propósito y sus interfaces (tanto sintaxis como semántica).

Documentación del diseño arquitectónico

Organización del

Un diagrama de contexto provee el contexto general. Establece el alcance del sistema, los actores

1. Contexto del sistema y la arquitectura
2. Descripción de las vistas de la arquitectura
 - a. Presentación principal de la vista
 - b. Catálogo de elementos.
 - c. Fundamento de la arquitectura.
 - d. Comportamiento.
 - e. Otra información.
3. Documentación transversal a las vistas.

Provee más información sobre los elementos presentados

Justificaciones de las decisiones/elecciones realizadas. Podría proveer también una discusión sobre las alternativas consideradas y descartadas (importante si se realizan cambios de req.)

Documentación del diseño arquitectónico

Organización del documento

1. Contexto del sistema y la arquitectura.

2. Descripción de las vistas de la arquitectura.

a. Presentación principal.

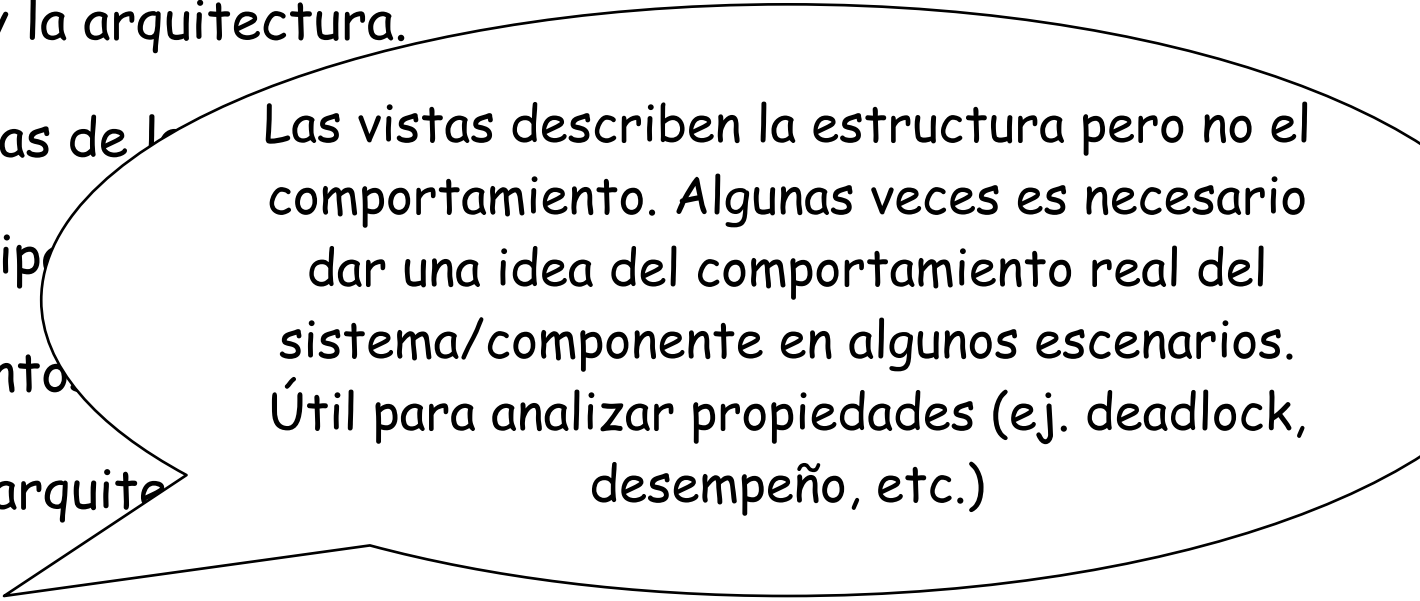
b. Catálogo de elementos.

c. Fundamento de la arquitectura.

d. Comportamiento.

e. Otra información.

3. Documentación transversal a las vistas.



Las vistas describen la estructura pero no el comportamiento. Algunas veces es necesario dar una idea del comportamiento real del sistema/componente en algunos escenarios. Útil para analizar propiedades (ej. deadlock, desempeño, etc.)

Documentación del diseño arquitectónico

Organización del documento

1. Contexto del sistema y la arquitectura.

2. Descripción de las vistas de la arquitectura.

a. Presentación principal.

b. Catálogo de elementos.

c. Fundamento de la arquitectura.

d. Comportamiento.

e. Otra información.

Las vistas describen la estructura pero no el comportamiento. Algunas veces es necesario dar una idea del comportamiento real del sistema/componente en algunos escenarios.

Limitaciones (ej. deadlock, etc.)

Ej.: decisiones dejadas intencionalmente para el futuro;

3. Documentación transversal a las vistas.

Documentación del diseño arquitectónico

Organización del documento

1. Contexto del sistema y la arquitectura.

2. Descripción de las vistas de la arquitectura.

a. Presentación principal.

b. Catálogo de elementos.

c. Fundamento de la arquitectura.

d. Comportamiento.

e. Otra información.

3. Documentación transversal a las vistas.

Las vistas describen la estructura pero no el comportamiento.

Describe cómo los elementos de las distintas vistas (y las vistas mismas) se relacionan entre sí (ej.: como los módulos se relacionan con las componentes). También: justificación de las vistas elegidas + otro tipo de info transversal.

Arquitectura y diseño

Tanto la arquitectura como el diseño dividen al sistemas en partes y dicen cómo éstas se organizan.

¿cuál es la relación/diferencia entre diseño y arquitectura?

- La arquitectura **es** un diseño: se encuentra en el dominio de la solución y no en el dominio del problema.
- La arquitectura es un diseño de muy alto nivel que se enfoca en las componentes principales.
- Lo que usualmente llamamos diseño se enfoca en los módulos que finalmente se transformarán en el código de tales componentes.

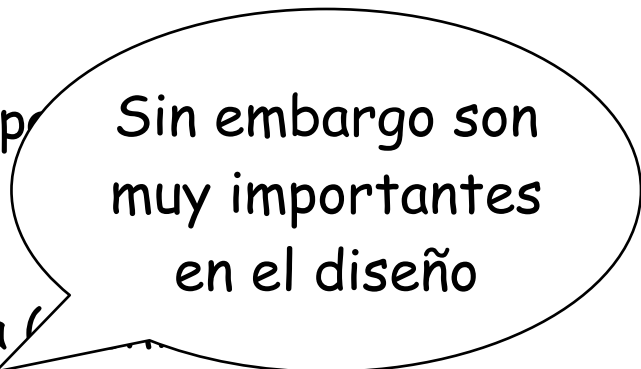
Podemos considerar que el diseño provee la vista de módulos del sistema.

Arquitectura y diseño

- Básicamente, el arquitecto y el diseñador definen donde acaba una tarea y empieza la otra.
- En la arquitectura sólo se necesita identificar las partes necesarias para evaluar las propiedades deseadas.
- La arquitectura no considera la estructura interna (archivos, estructura de datos, etc.)
- La arquitectura impone restricciones sobre elecciones que pueden realizarse durante el diseño.

Arquitectura y diseño

- Básicamente, el arquitecto y el diseñador definen donde acaba una tarea y empieza la otra.
- En la arquitectura sólo se necesita identificar las propiedades para evaluar las propiedades deseadas.
- La arquitectura no considera la estructura interna (estructura de datos, etc.)
- La arquitectura impone restricciones sobre elecciones que pueden realizarse durante el diseño.



Sin embargo son
muy importantes
en el diseño

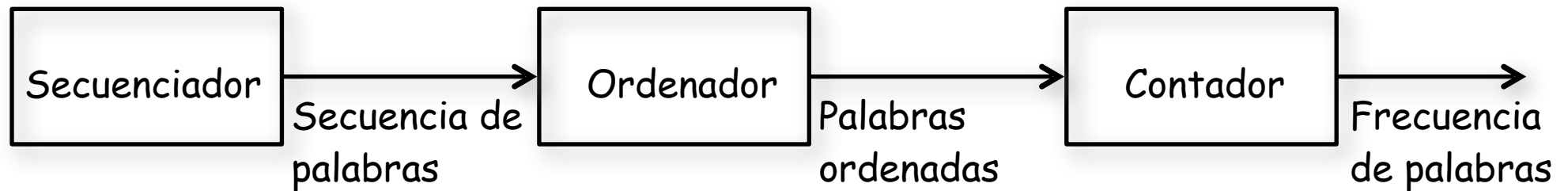
Preservación de la integridad de la arquitectura

¿Cuál es el rol de la arquitectura durante el resto del proyecto?

- Muchos diseñadores/desarrolladores sólo la usan para comprender el sistema y para nada mas.
- La arquitectura impone restricciones que **deben** preservarse en la implementación, inclusive.
- Lamentablemente, es muy fácil ignorar el diseño arquitectónico y continuar con el desarrollo.
- Para que la arquitectura tenga sentido, ésta debe acompañar el diseño y el desarrollo del sistema.

Preservación de la integridad de la arquitectura

Ejemplo



Implementación 1:

Responde exactamente a la arquitectura:

Cada componente es un proceso escrito en C.

Los procesos se comunican entre sí a través del comando pipe.

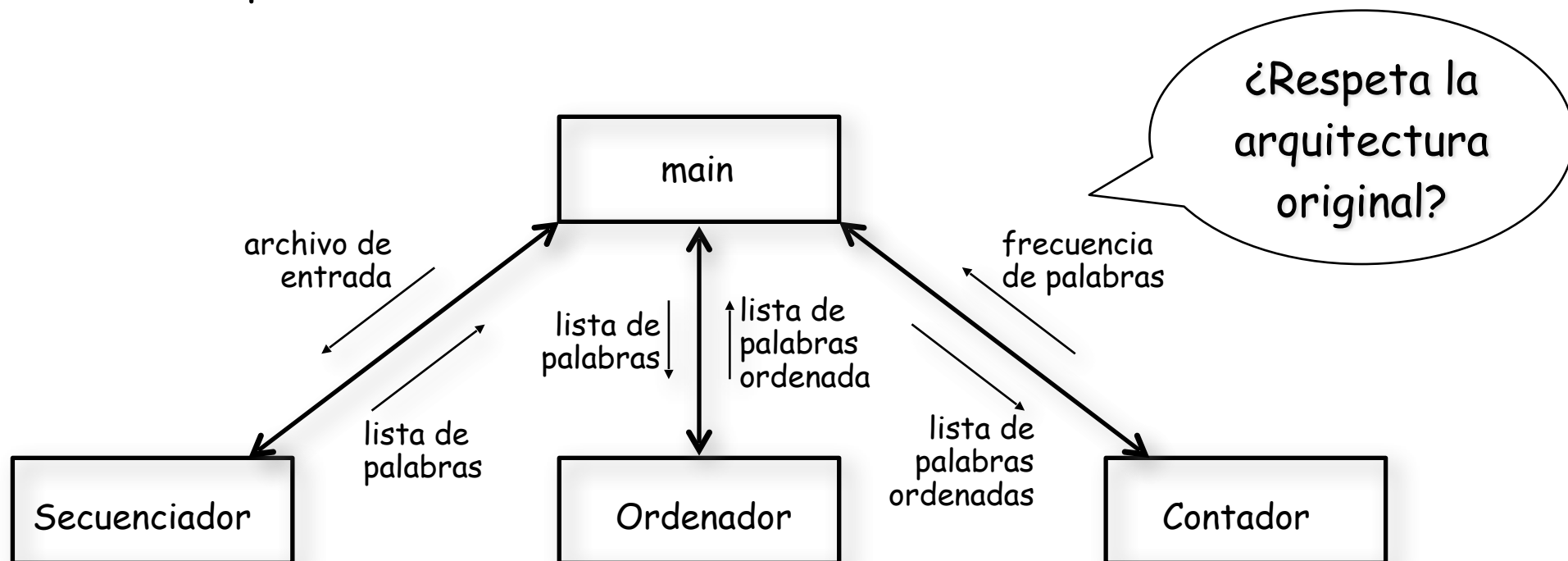
Preservación de la integridad de la arquitectura

Ejemplo

Implementación 2:

Para evitar la sobrecarga en el desempeño inducida por el comando pipe, se decide:

1. implementar cada proceso como una función C , y
2. los conectores se implementan a través de una función main que llama oportunamente a las funciones.

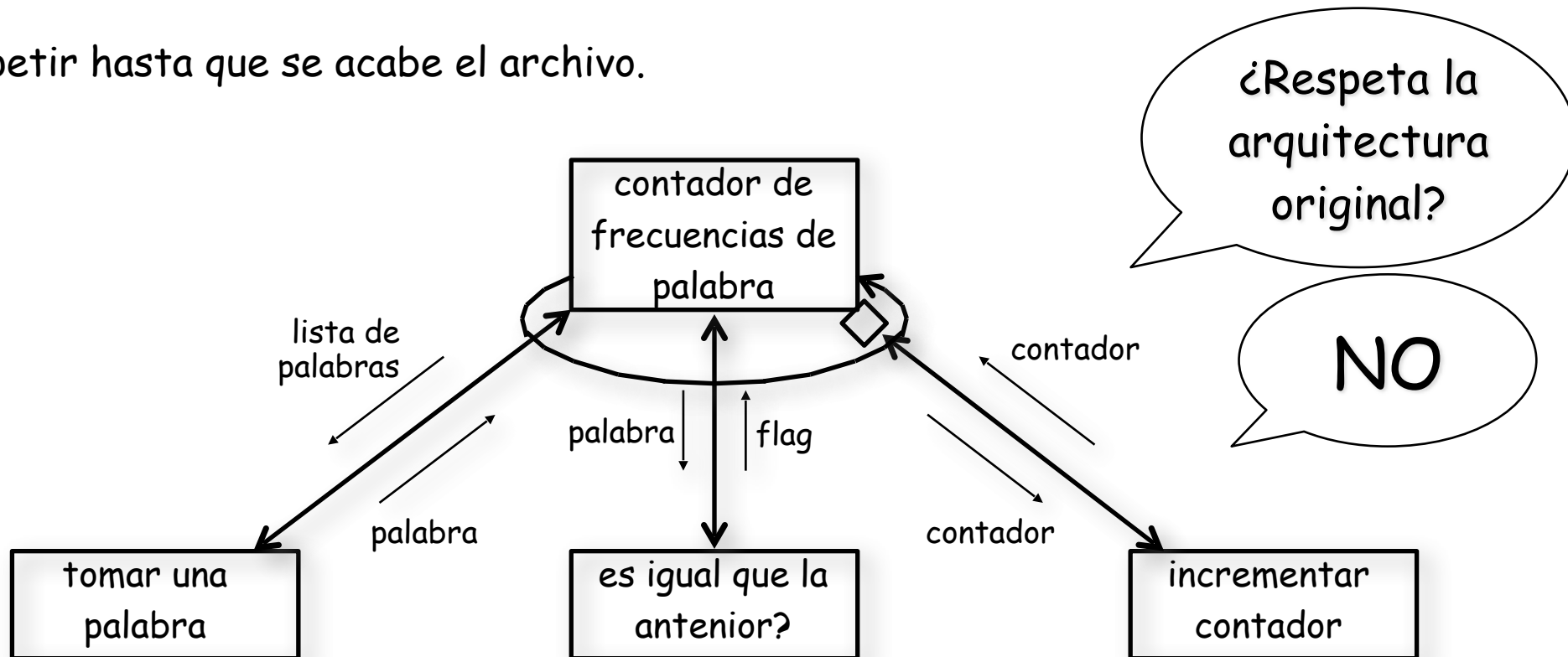


Preservación de la integridad de la arquitectura

Ejemplo

Implementación 3:

- Tomar palabra por palabra.
- Si la palabra ya fue contada, incrementar el contador de esta palabra.
- Si no, incorporarla al listado.
- Repetir hasta que se acabe el archivo.

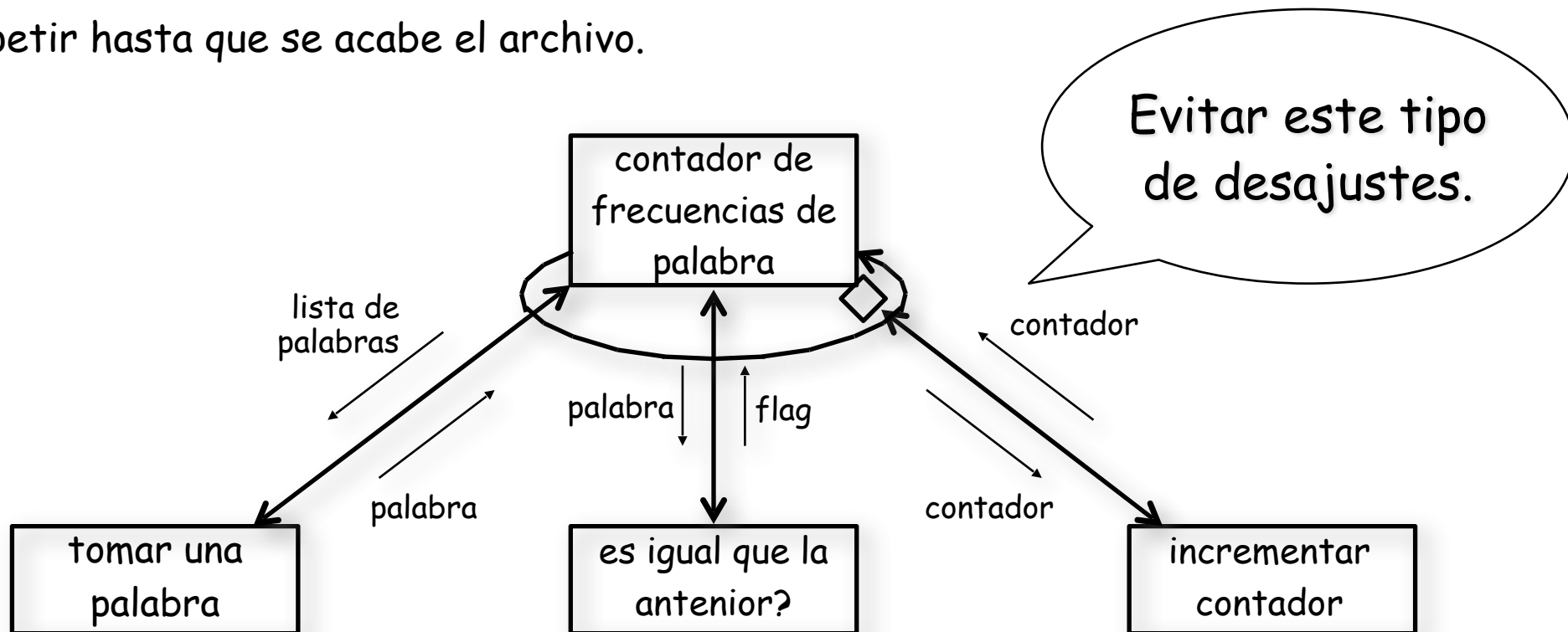


Preservación de la integridad de la arquitectura

Ejemplo

Implementación 3:

- Tomar palabra por palabra.
- Si la palabra ya fue contada, incrementar el contador de esta palabra.
- Si no, incorporarla al listado.
- Repetir hasta que se acabe el archivo.



Evaluación de las arquitecturas

La arquitectura tiene **impacto** sobre los **atributos no funcionales** tales como modificabilidad, desempeño, confiabilidad, portabilidad, etc.

Tanto como las elecciones de diseño y codificación

=> se deben evaluar estas propiedades en la arquitectura propuesta.

¿Cómo hacerlo?

- Una posibilidad => técnicas formales (redes de colas, model checkers, lenguajes de especificación, etc.)
- Otra posibilidad => metodologías rigurosas.

Evaluación de las arquitecturas

El método de análisis ATAM



Architecture
Tradeoff Analysis
Method

Analiza las propiedades y las concesiones entre ellas.

Pasos principales:

1. Recolectar escenarios:

- Los escenarios describen las interacciones del sistema.
- Elegir los escenarios de interés para el análisis (escenarios críticos).
- Incluir escenarios excepcionales sólo si son importantes.

2. Recolectar requerimientos y/o restricciones:

- Definir lo que se espera del sistema en tales escenarios.
- Deben especificar los niveles deseados para los **atributos de interés** (preferiblemente cuantificados).

Evaluación de las arquitecturas

El método de análisis ATAM

3. Describir las vistas arquitectónicas:

- Las vistas del sistema que serán evaluadas son recolectadas.
- Distintas vistas pueden ser necesarias para distintos análisis.

4. Análisis específicos a cada atributo:

- Se analizan las vistas bajo distintos escenarios separadamente para cada atributo de interés distinto.
- Esto determina los niveles que la arquitectura puede proveer en cada atributo.
- Se comparan con los requeridos.
- Esto forma la base para la elección entre una arquitectura u otra o la modificación de la arquitectura propuesta.
- Puede utilizarse cualquier técnica o modelado.

Evaluación de las arquitecturas

El método de análisis ATAM

5. Identificar puntos sensitivos y de compromisos:

- Análisis de sensibilidad: cuál es el impacto que tiene un elemento sobre un atributo de calidad.

Los elementos de mayor impacto son los puntos de sensibilidad.

- Análisis de compromiso:

Los puntos de compromiso son los elementos que son puntos de sensibilidad para varios atributos.

Evaluación de las arquitecturas

El método de análisis ATAM vs CBAM



Cost-benefit Analysis
Method

ATAM

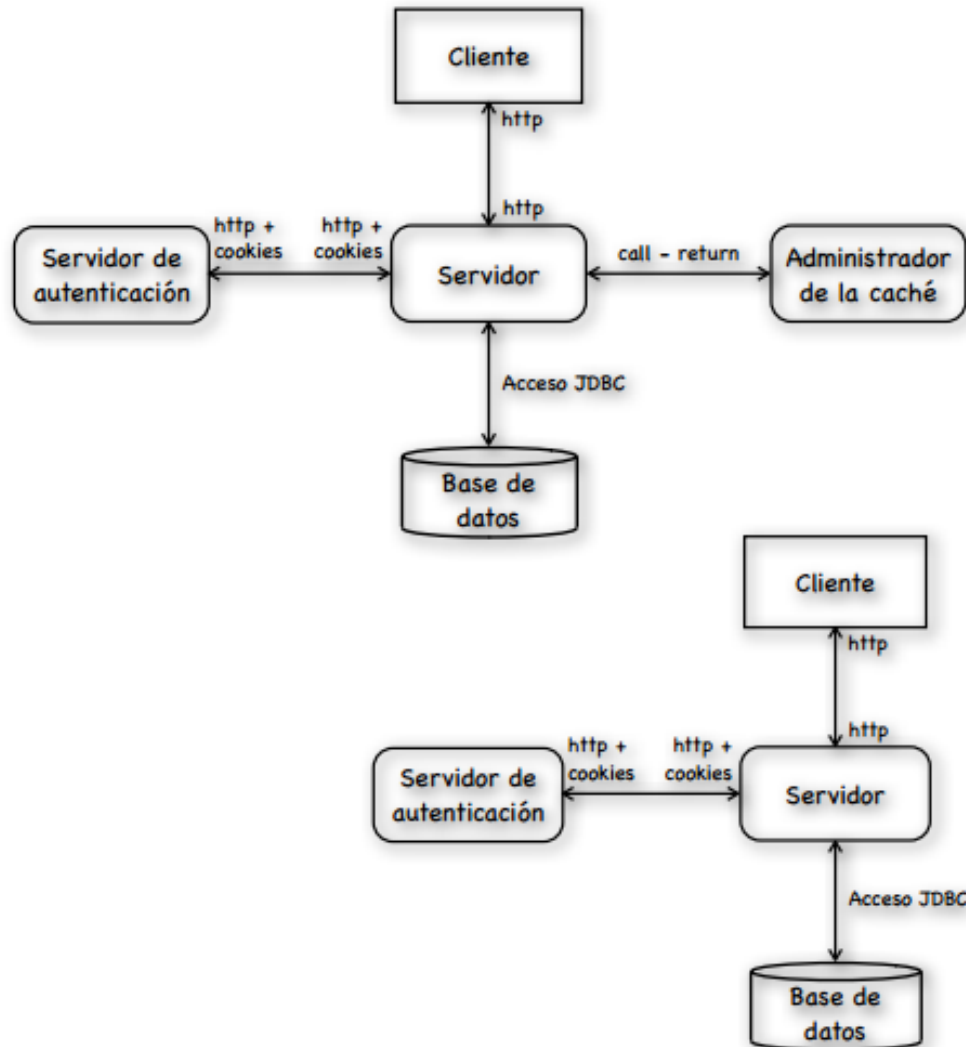
Performance
Scalability
Availability

CBAM

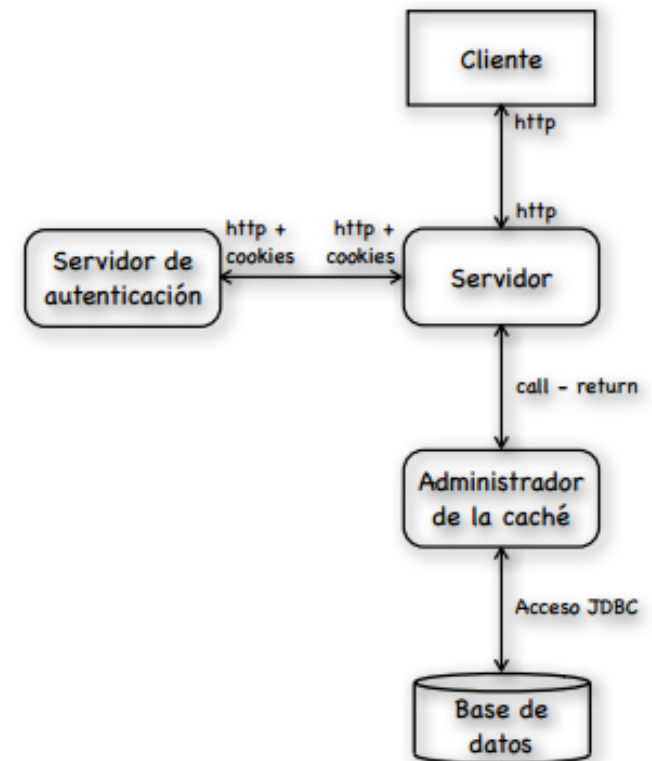
Cost of Performance
Cost of Scalability
Cost Availability

Evaluación de las arquitecturas

El método de análisis ATAM



Leer en el libro la comparación entre estas tres arquitecturas correspondientes al ejemplo de la encuesta de los alumnos.



Arquitectura del software

“The idea of software architecture has at its foundation a principle of **information hiding**: the less a part of a program knows about other parts of a program, the easier it is to change. The most popular information hiding strategy is **encapsulation**: this is the idea of designing self-contained abstractions with well-defined interfaces that separate different concerns in a program. Programming languages offer encapsulation support through things like **functions** and **classes**, which encapsulate data and functionality together. Another programming language encapsulation method is **scoping**, which hides variables and other names from other parts of program outside a scope.”

— Amy J. Ko

Arquitectura del software

Lectura complementaria:

- Capítulo 4 Jalote
- Architecture by Amy J. Ko
- <http://faculty.washington.edu/ajko/books/cooperative-software-development/architecture.html>
- (ATAM)
- https://www.youtube.com/watch?v=52haYbu80e8&ab_channel=MarkRichards