

# Principios de Arquitectura Empresarial

Fabio Castro Rozo



# Contenido

1. Estilos de Arquitectura
2. Atributos de Calidad



# Arquitectura de Software

## Estilos Arquitectónicos



# Arquitectura de Software



La arquitectura de un sistema intensivo en software es la estructura o estructuras del sistema, la cual comprende los elementos de software, las propiedades externas visibles de esos elementos y las relaciones entre ellos.

Software Architecture in Practice (2nd edition), Bass, Clements, Kazman; AddisonWesley 2003

## Pero entonces .... ¿Qué trata de resolver?

- ☐ Identificar la mejor manera en que se puede descomponer un sistema
- ☐ Identificar los componentes del sistema
- ☐ Determinar como se comunican dichos componentes
- ☐ Establecer como fluye la información entre ellos
- ☐ Determinar como los elementos de un sistema pueden evolucionar independientemente.
- ☐ Describir TODO ESTO por medio de notaciones formales e informales



# Arquitectura de Software

## ¿Y cómo lo resuelve?



Construcción

Algunas decisiones de diseño regularmente resultan en soluciones con propiedades inesperadas: más efectivas, eficientes, confiables, **evolucionables** y **escalables**.

La experiencia y conocimiento común se generalizó y adoptó diferentes formas dependiendo del contexto de aplicación con el fin de ser reutilizado:

arquitecturas de software específicas de un dominio

estilos arquitectónicos

patrones arquitectónicos.

# Arquitectura de Software

## Estilos Arquitectónicos

Un estilo arquitectónico es una colección de decisiones de diseño arquitectónicas que tiene un nombre específico y que:

- ☐ Es aplicable a un contexto de desarrollo dado
- ☐ Restringe decisiones de diseño arquitectónicas que son específicas a un sistema particular dentro de aquel contexto
- ☐ Garantiza ciertas calidades del sistema resultante




*Taylor, Medvidovic and Dashofy - Software Architecture: Foundations, Theory, and Practice*



# Arquitectura de Software

## Estilos Arquitectónicos

Pero ... ¿Para que sirven?

- 1 Reutilización de diseño**  Las soluciones con propiedades bien entendidas se pueden volver a aplicar a nuevos problemas
- 2 Reutilización de código**  Los aspectos invariables de un estilo se prestan a implementaciones compartidas
- 3 Comprensibilidad de la organización del sistema**  Solo saber que algo es una arquitectura "cliente-servidor" transmite mucha información

# Arquitectura de Software

## Estilos Arquitectónicos

Y ... ¿Para que más sirven?

### 4 Interoperabilidad



Compatible con la estandarización de estilos (por ejemplo, CORBA, SoftBench)

### 5 Análisis específicos de estilo



- Espacio de diseño limitado
- Algunos análisis no son posibles en arquitecturas ad-hoc o arquitecturas en ciertos estilos

### 6 Visualizaciones



Representaciones específicas de estilo que coinciden con los modelos mentales de los ingenieros





# Arquitectura de Software

## Estilos Arquitectónicos

### ¿Cuáles son sus propiedades?

- 1 Proporcionan un vocabulario de elementos de diseño.

Tipos de **componentes** y **conectores** (p. ej., tuberías, filtros, servidores ...)

- 2 Definen un conjunto de reglas de configuración

Restricciones topológicas que determinan la composición permitida de elementos (p. ej. los elementos de una capa se pueden comunicarse sólo con los del capa inferior)

- 3 Definen una interpretación semántica

Composiciones de elementos de diseño tienen significados bien definidos

# Arquitectura de Software

## Componente

- Elementos que encapsulan procesamiento y datos
- Entidad arquitectónica que:

Ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto

1 **Encapsula** un subconjunto de funcionalidad y / o datos del sistema


2 Restringe el acceso a ese subconjunto a una **interfaz** definida explícitamente

Medio común para que los objetos no relacionados se comuniquen entre sí. Estas son definiciones de métodos y valores sobre los cuales los objetos están de acuerdo para cooperar.

3 Tiene definidas dependencias explícitas sobre el contexto de ejecución requerido

Los componentes suelen proporcionar los servicios específicos de la aplicación

# Arquitectura de Software

**Conectores**  En sistemas muy complejos, la interacción puede llegar a ser más difícil e importante que la funcionalidad de los componentes

Es una pieza de la arquitectura que se ocupa de llevar adelante y regular interacciones entre los componentes

- 1 En muchos sistemas, los conectores son por llamadas a procedimientos o accesos a datos compartidos
- 2 Los conectores típicamente proporcionan facilidades de interacción que son independientes de las aplicación
- 3 Se pueden describir independiente de los componentes.

Los componentes suelen proporcionar los servicios específicos de la aplicación

# Arquitectura de Software

## Conectores

Ejemplos:

- Llamadas a procedimientos
- Memoria compartida
- Paso de mensajes
- Streaming
- Conectores de distribución (distribution connectors) –encapsulan “Application programming interfaces (API’s)” que habilitan la interacción de componentes distribuidos



# Arquitectura de Software

## Patrón Arquitectónico

Es una colección de decisiones de diseño arquitectónico aplicables a ***problemas de diseño recurrentes***, y que están parametrizados para tener en cuenta los diferentes contextos de desarrollo de software en el que surge el problema.

*Taylor, Medvidovic and Dashofy - Software Architecture: Foundations, Theory, and Practice*

- 1 Provee un conjunto de decisiones específicas de diseño.
- 2 Estas decisiones de diseño pueden pensarse como parametrizables, ya que necesitan ser instanciadas con los componentes y conectores particulares a una aplicación.



# Arquitectura de Software

## Estilo Vs Patrón (Diferencias)

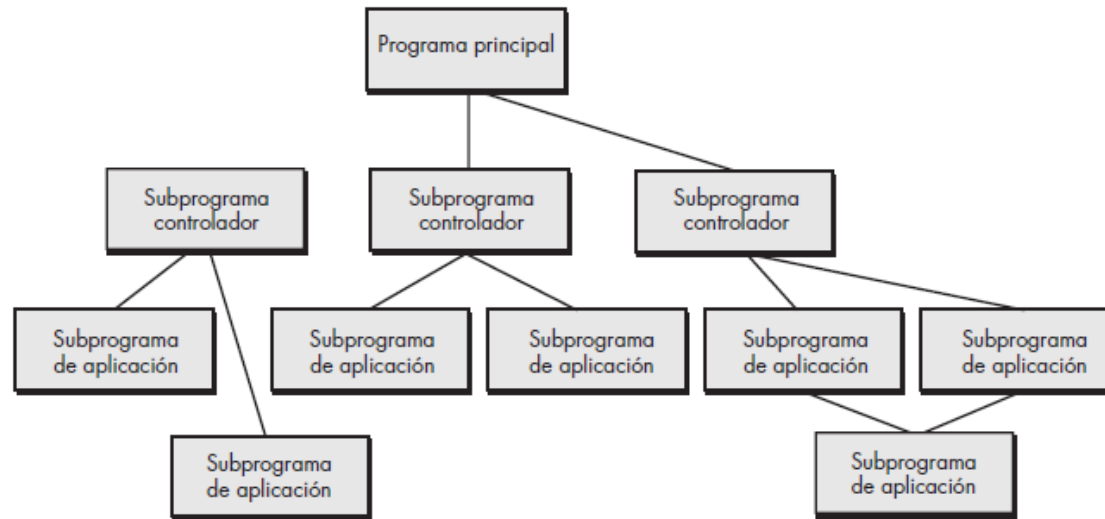
Los dos conceptos son similares y no siempre es posible identificar un límite nítido entre ellos.

	ESTILO	PATRÓN
Alcance	Aplican a un contexto de desarrollo, p.ej.: <ul style="list-style-type: none"><li>• Sistemas altamente distribuidos</li><li>• Sistemas intensivos en GUI</li></ul>	Aplican a problemas de diseño específicos: <ul style="list-style-type: none"><li>• El estado del sistema debe presentarse de múltiples formas</li><li>• La lógica de negocio debe estar separada del acceso a datos</li></ul>
Abstracción	Son muy abstractos para producir un diseño concreto del sistema.	Son fragmentos arquitectónicos parametrizados que pueden ser pensados como una pieza concreta de diseño.
Relación	Un sistema diseñado de acuerdo a las reglas de un único estilo puede involucrar el uso de múltiples patrones	Un único patrón puede ser aplicado a sistemas diseñados de acuerdo a los lineamientos de múltiples estilos

# Arquitectura de Software

## Estilos Arquitectónicos

### 1. Arquitecturas de llamar y regresar



*Arquitecturas de programa principal/subprograma.*

Descompone una función en una jerarquía de control en la que un programa “principal” invoca cierto número de componentes de programa que a su vez invocan a otros

*Arquitecturas de llamada de procedimiento remoto.*

Los componentes de una arquitectura de programa principal/subprograma están distribuidos a través de computadoras múltiples en una red.

# Arquitectura de Software

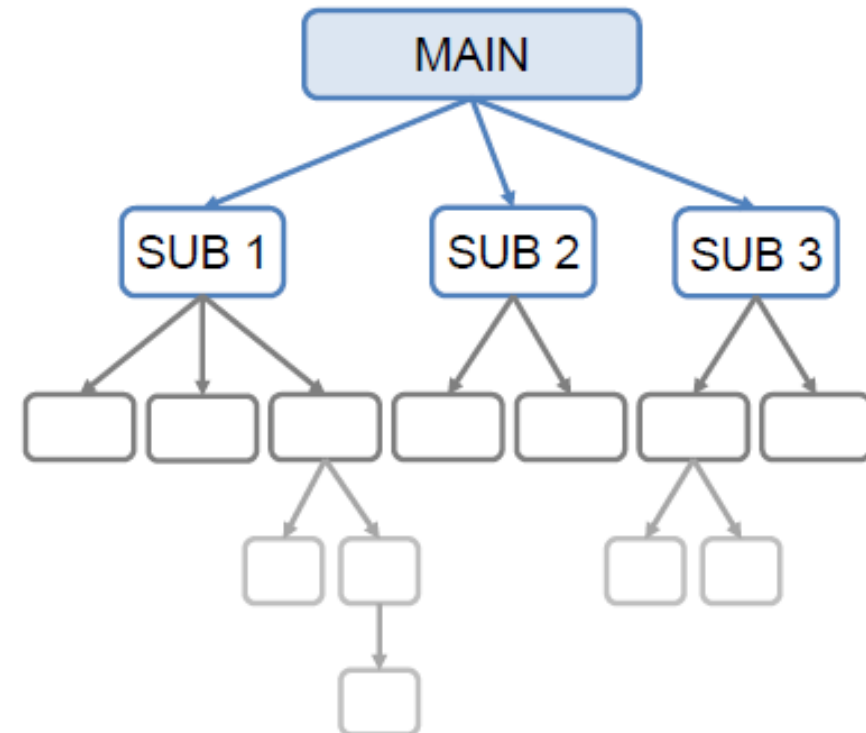
## Estilos Arquitectónicos

### 1. Arquitecturas de llamar y regresar

#### Programa principal y subrutinas

- Influenciado por la Programación Estructurada
- Envío de mensaje / respuesta
- Se cede el control y se espera la respuesta

Es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while)





# Arquitectura de Software

## Estilos Arquitectónicos

### 1. Arquitecturas de llamar y regresar

#### Programa principal y subrutinas

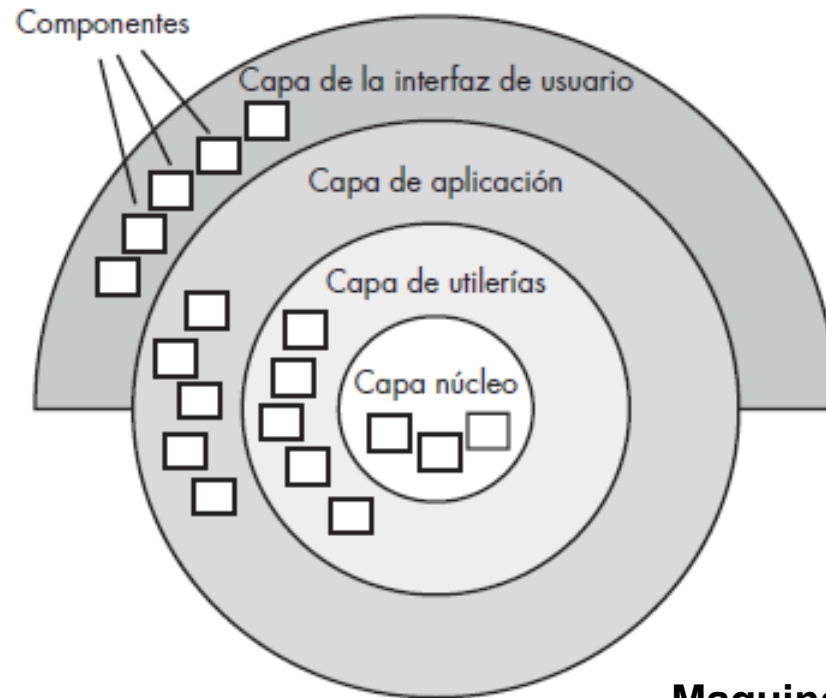
<b>DESCRIPCIÓN</b>	Descomposición basada en la separación de pasos de procesamiento funcionales
<b>COMPONENTES</b>	Programa principal y subrutinas
<b>CONECTORES</b>	Llamadas a procedimientos / funciones
<b>ELEMENTOS DE DATOS</b>	Valores de entrada/salida de las subrutinas
<b>TOPOLOGÍA</b>	La organización estática de los componentes es jerárquica. La estructura es un grafo dirigido
<b>RESTRICCIONES ADICIONALES</b>	No
<b>CUALIDADES</b>	Modularidad: Las subrutinas pueden ser reemplazadas con diferentes implementación mientras su interfaz no sea afectada
<b>USOS TÍPICOS</b>	Pequeños programas. Uso pedagógico.
<b>PRECAUCIONES</b>	<ul style="list-style-type: none"> <li>• Generalmente falla al escalar a grandes aplicaciones</li> <li>• Inadecuada atención a estructuras de datos</li> <li>• Requiere bastante esfuerzo para introducir nuevos requerimientos</li> <li>• Reúso limitado de funciones y procedimientos</li> </ul>

# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

Organiza el sistema en un conjunto de capas, cada una de las cuales provee una serie de servicios a las capas superiores usando los de las capas inferiores.



En la capa externa, los componentes atienden las operaciones de la interfaz de usuario. En la interna, los componentes realizan la interfaz con el sistema operativo. Las capas intermedias proveen servicios de utilerías y funciones de software de aplicación.

Es un estilo simple y de uso clásico para desarrollos en lenguajes de programación como Java o C.

**Maquinas virtuales:** de amplio uso en arquitecturas de computadoras y sistemas operativos.

**Cliente servidor:** presente principalmente en aplicaciones de negocio

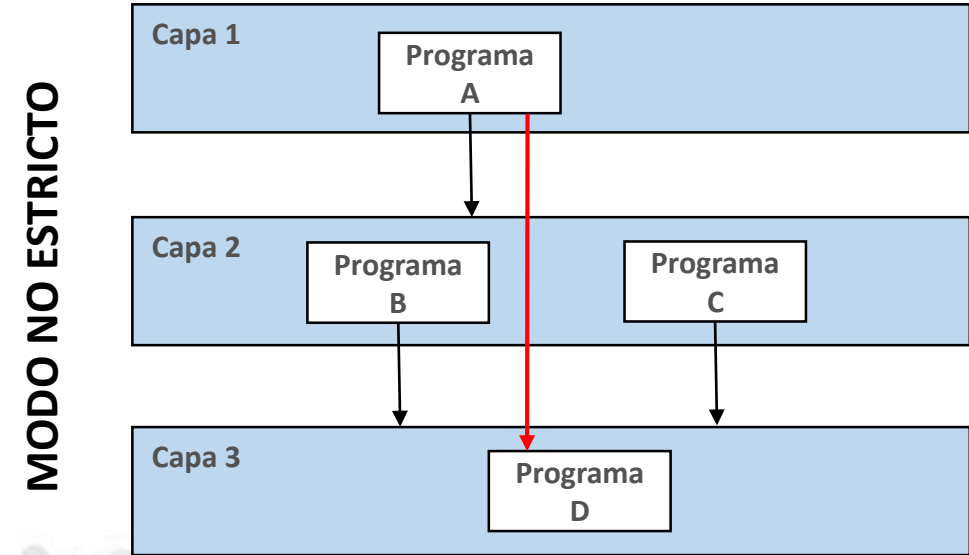
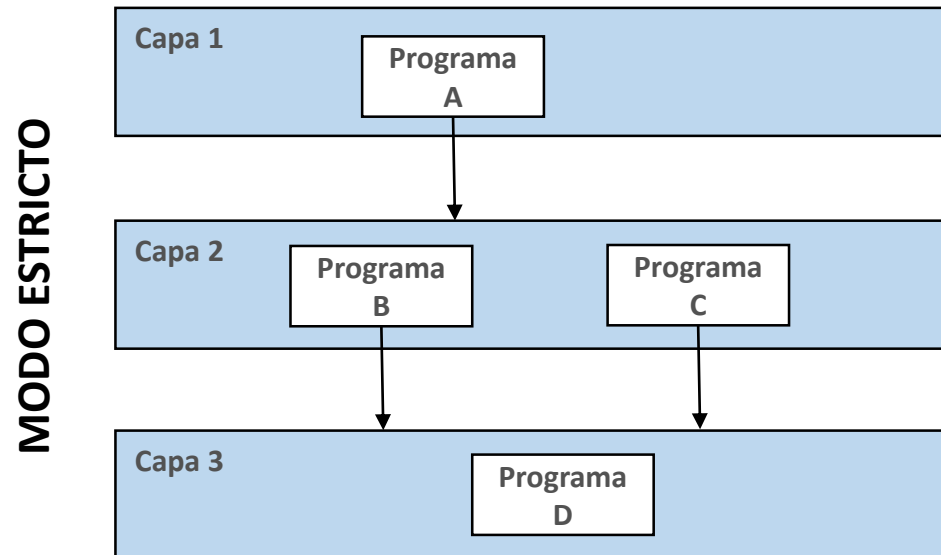
# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

#### Máquinas Virtuales

- Una capa ofrece un conjunto de servicios (interfaz o API).
- Dichos servicios pueden ser accedidos por programas que residen en la capa superior.



# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

#### Máquinas Virtuales

<b>DESCRIPCIÓN</b>	Secuencia ordenada de capas. Cada capa (virtual machine) ofrece servicios a subcomponentes de la capa encima de ella.
<b>COMPONENTES</b>	Capas, generalmente conteniendo subcomponentes
<b>CONECTORES</b>	Llamadas a procedimientos
<b>ELEMENTOS DE DATOS</b>	Parámetros pasados entre las capas
<b>TOPOLOGÍA</b>	Lineal
<b>RESTRICCIONES ADICIONALES</b>	No
<b>CUALIDADES</b>	<ul style="list-style-type: none"> <li>• Dependencias claras y acotadas</li> <li>• Encapsulamiento</li> <li>• Basado en niveles de abstracción</li> <li>• Reusabilidad, Portabilidad</li> </ul>
<b>USOS TÍPICOS</b>	<ul style="list-style-type: none"> <li>• Sistemas Operativos</li> <li>• Stacks de protocolos de red</li> <li>• Aplicaciones empresariales</li> </ul>
<b>PRECAUCIONES</b>	<ul style="list-style-type: none"> <li>• Performance</li> <li>• Una maquina virtual con varios niveles puede resultar ineficiente</li> </ul>

# Arquitectura de Software

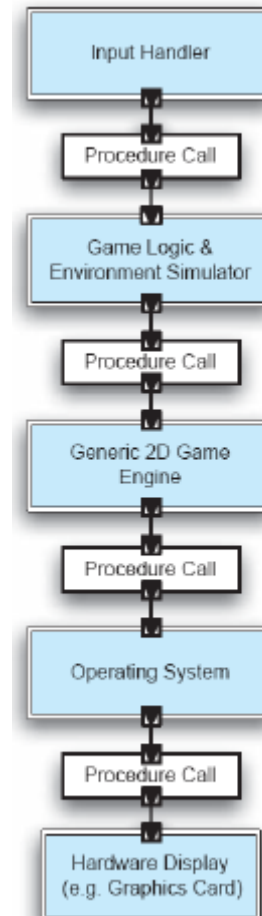
## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

#### Máquinas Virtuales

Ejemplo

CAPA	FUNCIONALIDAD
1	Recibe la entrada del usuario
2	Implementa el simulador del ambiente y la lógica del juego
3	Motor de juegos de gráficos en dos dimensiones genérico.
4	Sistema Operativos proveyendo soporte de GUI
5	Firmware o hardware



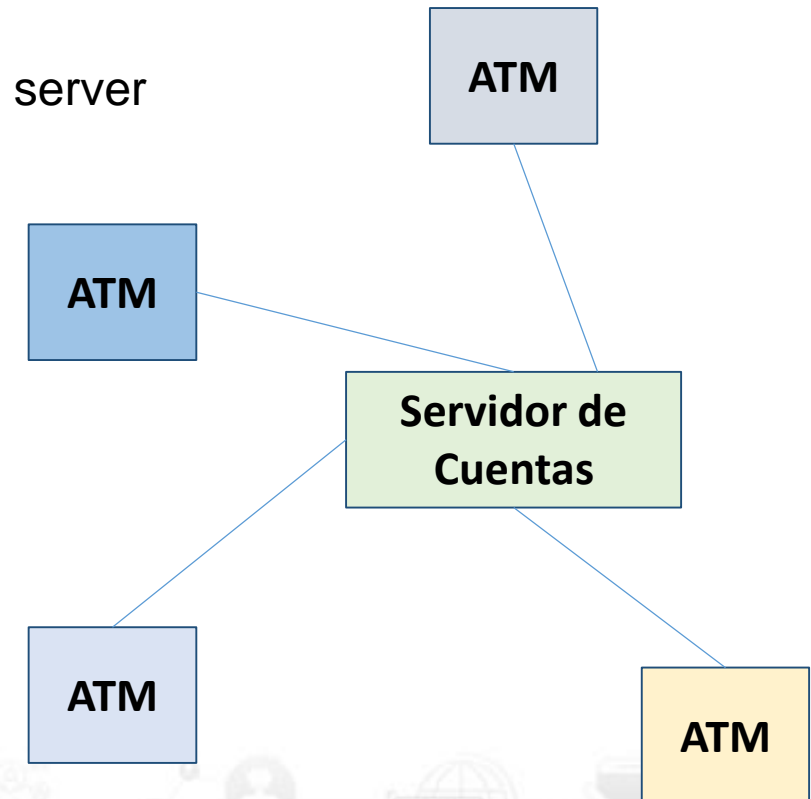
# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

#### Cliente Servidor

- 1 Múltiples clientes independientes accediendo al mismo server
- 2 Especialización de máquina virtual de 2 capas:
  - Capa 1: Cliente
  - Capa 2: Servidor
  - Conexión por red
- 3 Clientes conocen al Servidor
- 4 El Servidor no conoce a los Clientes
- 5 Los Clientes son independiente entre ellos



# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

#### Cliente Servidor

<b>DESCRIPCIÓN</b>	Clientes le envían requerimientos al servidor, el cual los ejecuta y envía la respuesta (de ser necesario). La comunicación es iniciada por el cliente.
<b>COMPONENTES</b>	Clientes y Servidor
<b>CONECTORES</b>	Llamadas a procedimientos remotos (o equivalente)
<b>ELEMENTOS DE DATOS</b>	Parámetros y valores de retorno
<b>TOPOLOGÍA</b>	Dos niveles. Múltiples clientes haciendo requerimientos al server
<b>RESTRICCIONES ADICIONALES</b>	Prohibida la comunicación entre clientes
<b>CUALIDADES</b>	<ul style="list-style-type: none"> <li>• Sencilla y muy utilizada</li> <li>• Centralización de cómputos y datos en el server.</li> <li>• Mantenible</li> <li>• Un único servidor puede atenderá a múltiples clientes</li> </ul>
<b>USOS TÍPICOS</b>	<ul style="list-style-type: none"> <li>• Apps con datos y/ o procesamiento centralizados y clientes GUI</li> <li>• Stacks de protocolos de red</li> <li>• Aplicaciones empresariales</li> </ul>
<b>PRECAUCIONES</b>	Condiciones de la red vs crecimiento de clientes.

# Arquitectura de Software

## Estilos Arquitectónicos

### 2. Arquitecturas en Capas

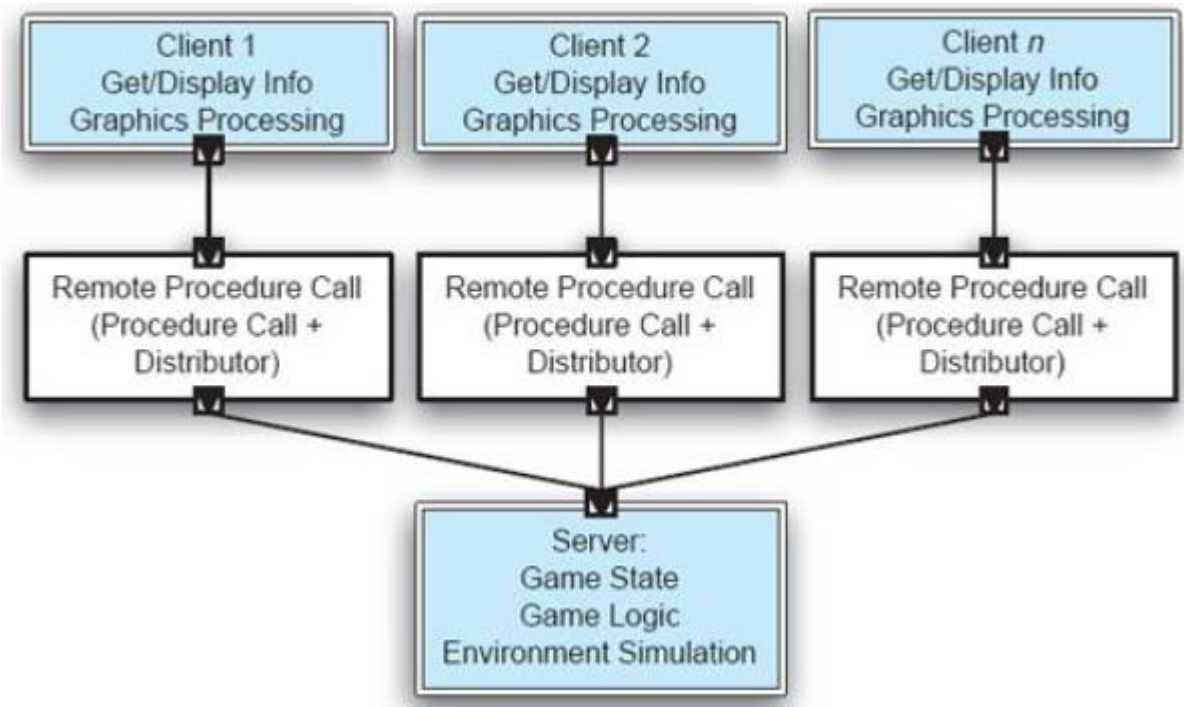
#### Ejemplo

El estado del juego, la lógica y la simulación del entorno se realiza en el servidor

Los clientes realizan las funciones de interface de usuario.

Conectores: RPC: llamada a procedimientos y un “Distributor” que identifica los caminos de interacción de la red y las rutas de comunicación

#### Cliente Servidor

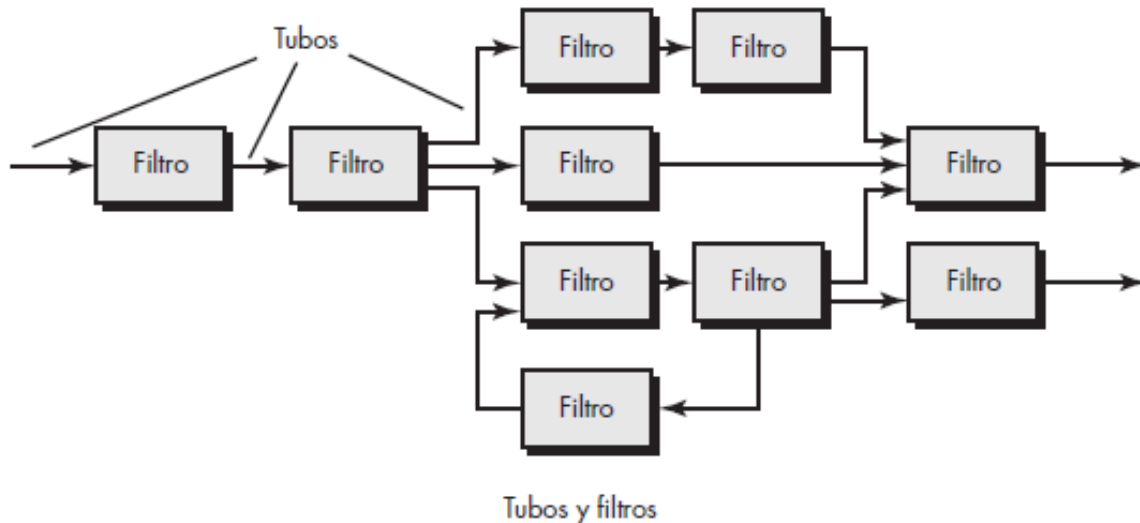




# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos



- considera el flujo de datos entre unidades de procesamiento independientes.
- La estructura del sistema está basada en transformaciones sucesivas de los datos.
- Los datos entran al sistema y fluyen a través de los componentes hasta su destino final.
- Usualmente un programa controla la ejecución de los componentes (lenguaje de control)

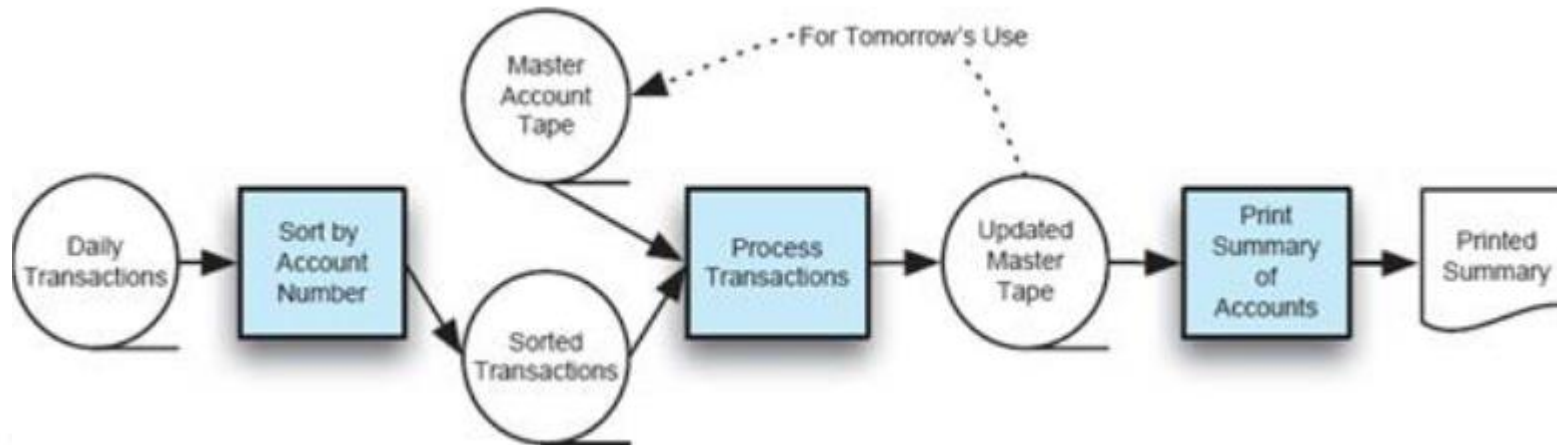
# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

#### Secuencial por lotes

- ★ Es la estructura típica de un sistema de procesamiento de datos tradicional por lotes (Batch)
- ★ Los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.



# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

#### Secuencial por lotes

<b>DESCRIPCIÓN</b>	Programas separados y ejecutados en orden. Los datos son pasados como un lote de un programa al siguiente.
<b>COMPONENTES</b>	Programas independientes
<b>CONECTORES</b>	Distintos tipos de interfaces: desde humana hasta web services
<b>ELEMENTOS DE DATOS</b>	Lotes de datos pasados de un programa al siguiente.
<b>TOPOLOGÍA</b>	Lineal
<b>RESTRICCIONES ADICIONALES</b>	Se ejecuta un programa a la vez, hasta que termina.
<b>CUALIDADES</b>	<ul style="list-style-type: none"> <li>• Sencillez</li> <li>• Ejecuciones independientes</li> </ul>
<b>USOS TÍPICOS</b>	Procesamiento de transacciones en sistemas financieros
<b>PRECAUCIONES</b>	<ul style="list-style-type: none"> <li>• Cuando se requiere interacción entre componentes.</li> <li>• Cuando se requiere concurrencia entre componentes.</li> </ul>

# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

#### Tubos y Filtros

- ★ Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo
- ★ Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas.
- ★ El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes



# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

Tubos y Filtros      ¿Cómo funciona?

- 1 Por los tubos fluyen datos, transmisión de salidas de un filtro a la entrada de otro
- 2 Cada filtro admite una o varias entradas (tubos) y una o varias salidas (tubos)
- 3 Cada filtro es independiente del resto y no conocen la identidad de los filtros antes y después de él
- 4 La transformación del filtro puede comenzar antes de terminar de leer la entrada (distinto al proceso por lotes)

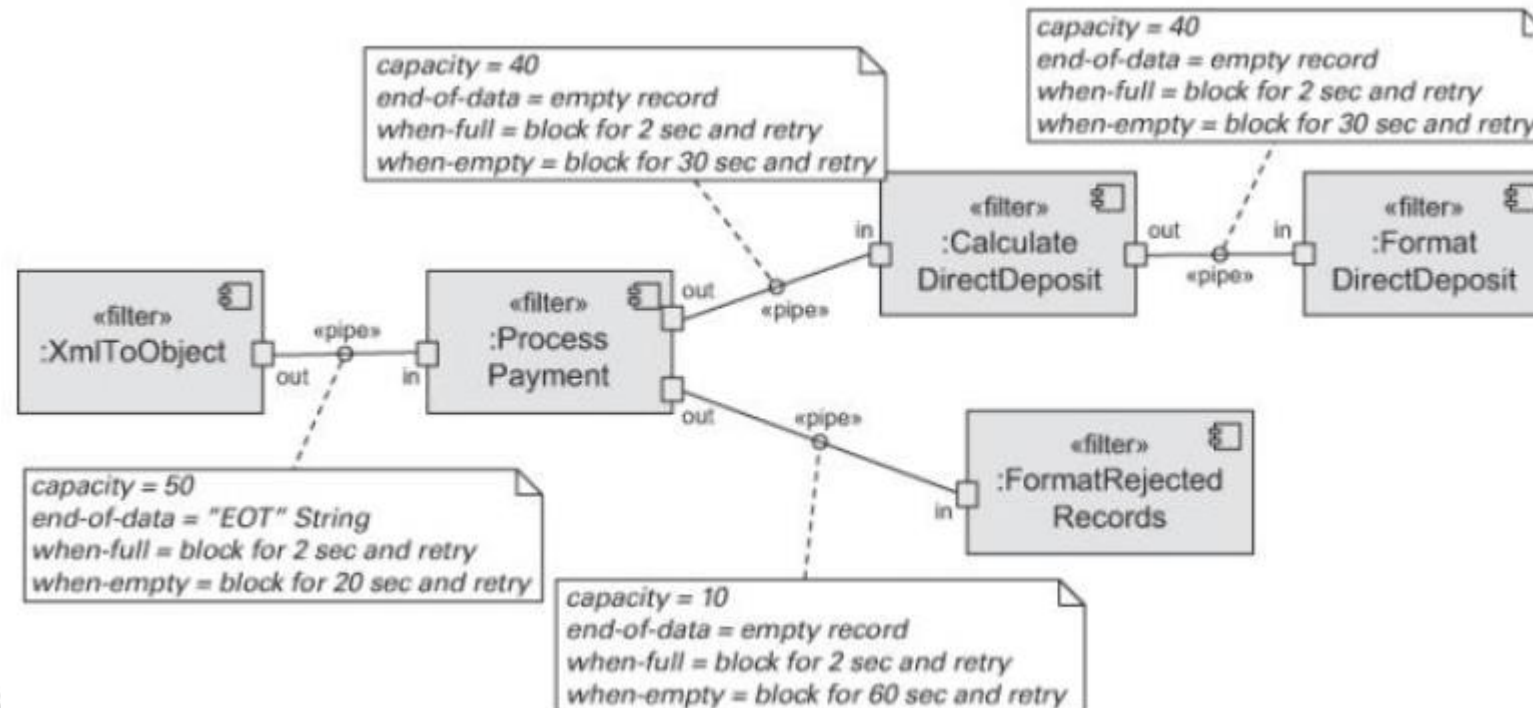
# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

#### Tubos y Filtros

#### Ejemplo



# Arquitectura de Software

## Estilos Arquitectónicos

### 3. Arquitecturas de flujo de datos

#### Tubos y Filtros

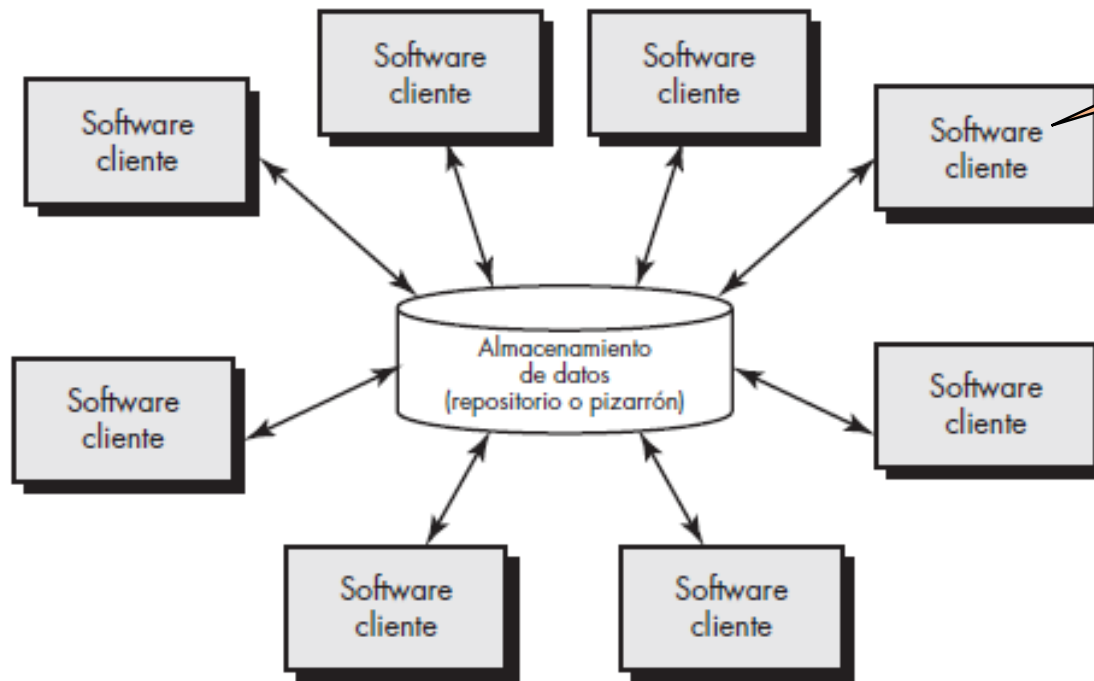
un **stream** es una especie de canal a través del cual fluyen los datos

DESCRIPCIÓN	Programas separados y ejecutados, potencialmente de manera concurrente. Datos son pasados como un stream de un programa al siguiente.
COMPONENTES	Programas independientes (filtros)
CONECTORES	Routers explícitos de streams de datos
ELEMENTOS DE DATOS	Stream de datos
TOPOLOGÍA	Pipeline (conexiones en T son posibles)
RESTRICCIONES ADICIONALES	No
CUALIDADES	<ul style="list-style-type: none"> <li>Filtros mutuamente independientes.</li> <li>Estructura simple de streams de entrada/salida facilitan la combinación de componentes.</li> <li>Flexibilidad: Agregar, eliminar, cambiar y reusar filtros</li> </ul>
USOS TÍPICOS	<ul style="list-style-type: none"> <li>Aplicaciones sobre sistemas operativos</li> <li>Procesamiento de audio, video</li> <li>Web servers (procesamiento de requerimientos HTTP)</li> </ul>
PRECAUCIONES	<ul style="list-style-type: none"> <li>Cuando estructuras de datos complejas deben ser pasadas entre filtros.</li> <li>Cuando se requiere interacción entre filtros.</li> </ul>

# Arquitectura de Software

## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos



Actualizan, agregan, eliminan o modifican los datos de cierto modo dentro del almacenamiento

Promueve la capacidad de integración, es decir, que es posible cambiar componentes existentes y agregar nuevos componentes a la arquitectura sin preocuparse por otros clientes, además es posible pasar datos entre clientes empleando el mecanismo del pizarrón. Los componentes clientes ejecutan los procesos de manera independiente.



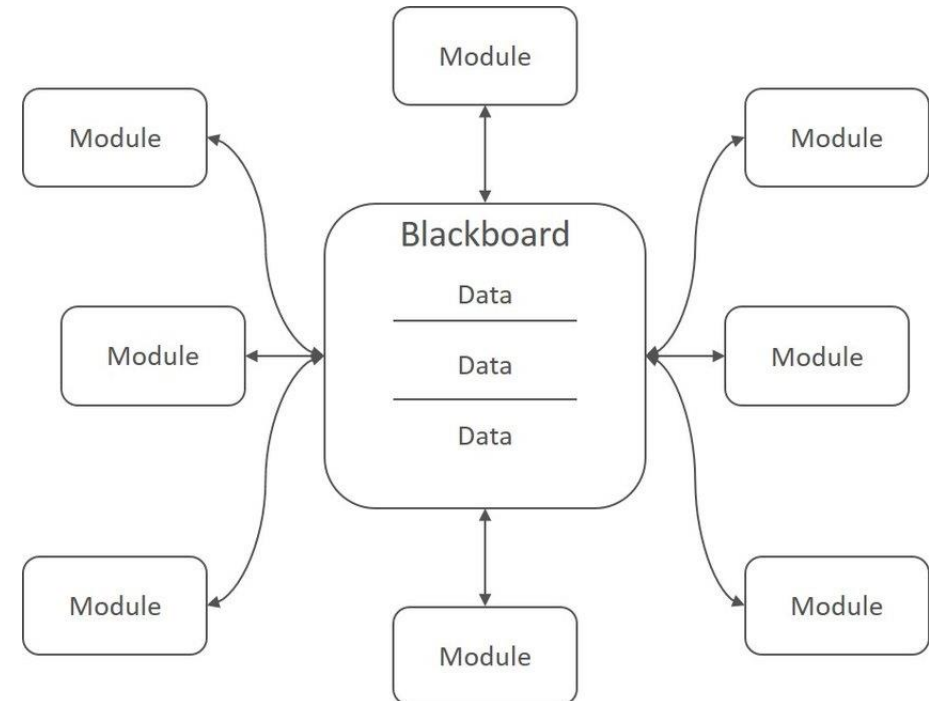
# Arquitectura de Software

## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

#### Blackboard

- ★ Expertos cooperando para resolver un problema planteado en un pizarrón
- ★ Cada uno identifica una parte del problema, la resuelve y pone el resultado de nuevo en el pizarrón
- ★ Con esa solución, otro experto puede identificar otro sub-problema y resolverlo
- ★ Así se continua hasta resolver el problema general
- ★ El estado de la información en el pizarrón determina el orden de ejecución de los distintos programas expertos.



# Arquitectura de Software

## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

La capacidad **heurística** es un rasgo típico de los humanos. Consiste en la capacidad de realizar innovaciones positivas para conseguir los fines que se pretenden. Muchos algoritmos en la **inteligencia artificial** son heurísticos por naturaleza, o usan reglas **heurísticas**

#### Blackboard

DESCRIPCIÓN	Programas independientes que acceden y se comunican a través de un repositorio de datos global (blackboard).
COMPONENTES	<ul style="list-style-type: none"> <li>• Blackboard (estructura central de datos)</li> <li>• Programas independientes operando sobre la pizarra.</li> </ul>
CONECTORES	<ul style="list-style-type: none"> <li>• Acceso al blackboard</li> <li>• Referencia directa a memoria, llamada a procedimiento, consultas a la base de datos, ...</li> </ul>
ELEMENTOS DE DATOS	Datos almacenados en el blackboard
TOPOLOGÍA	Estrella, con el blackboard al medio
RESTRICCIONES ADICIONALES	<ul style="list-style-type: none"> <li>• Detección de cambios en el blackboard</li> <li>• Polling sobre el blackboard</li> <li>• Blackboard Manager se encarga de notificar cambios</li> </ul>
CUALIDADES	<ul style="list-style-type: none"> <li>• La solución completa a un problema no tiene que ser pre-planificada. La evolución del estado determina las estrategias a ser adoptadas.</li> </ul>
USOS TÍPICOS	<ul style="list-style-type: none"> <li>• Resolución de problemas heurísticos en inteligencia artificial</li> <li>• Compiladores</li> </ul>
PRECAUCIONES	<ul style="list-style-type: none"> <li>• Si la interacción entre programas “independientes” necesita de reglas de regulación compleja</li> <li>• Cuando los datos en el blackboard están sujetos a cambios frecuentes y se requiere propagarlos entre todos componentes participantes.</li> <li>• Existe otra estrategia más simple</li> </ul>

# Arquitectura de Software

## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

#### Blackboard

#### Ejemplo

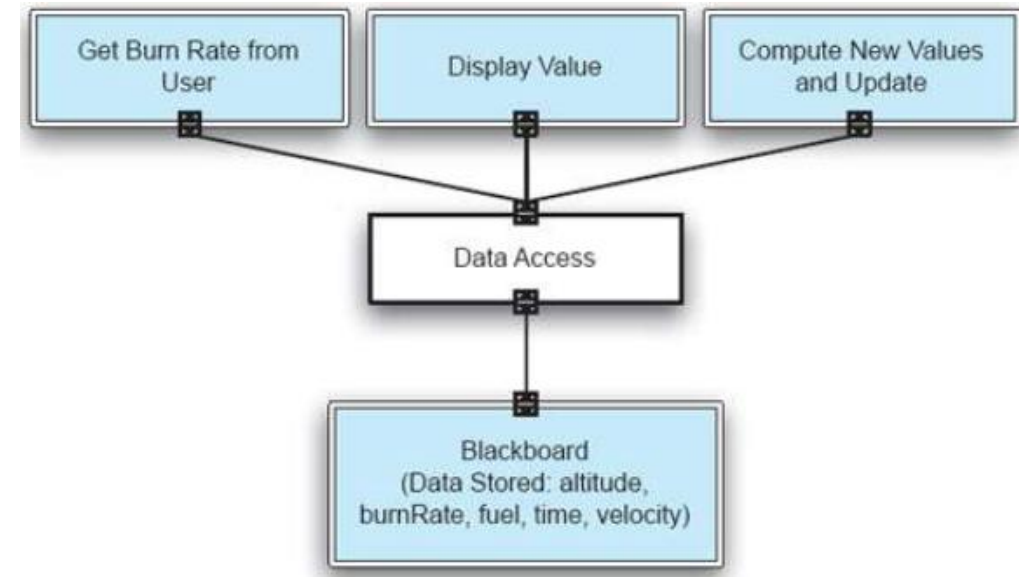
**Blackboard** - mantiene el estado del juego

**Get Burn Rate from User** - actualiza la velocidad de descenso en base el input del usuario

**Display Value** - muestra al usuario el estado de la nave y otros aspectos del juego

**Compute New Values and Update** - actualiza el estado del juego en base al tiempo y el modelo físico

**Data Access** - es el conector



# Arquitectura de Software

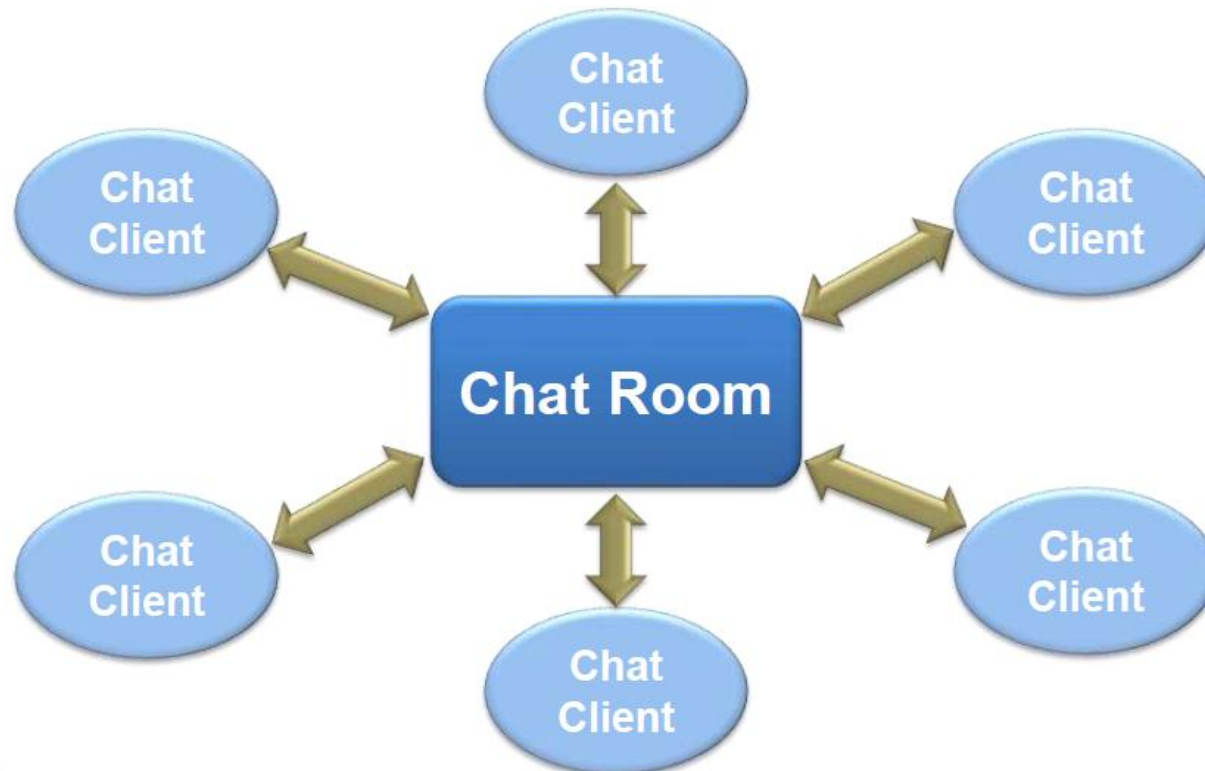
## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

Blackboard

Ejemplo

**Chat Room**



# Arquitectura de Software

## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

#### Repositorio

- ★ Es una especialización de Blackboard
- ★ El repositorio compartido solamente recibe y procesa los requerimientos de sus clientes, no hay notificaciones

Ejemplo

**Sistema Universitario**



# Arquitectura de Software

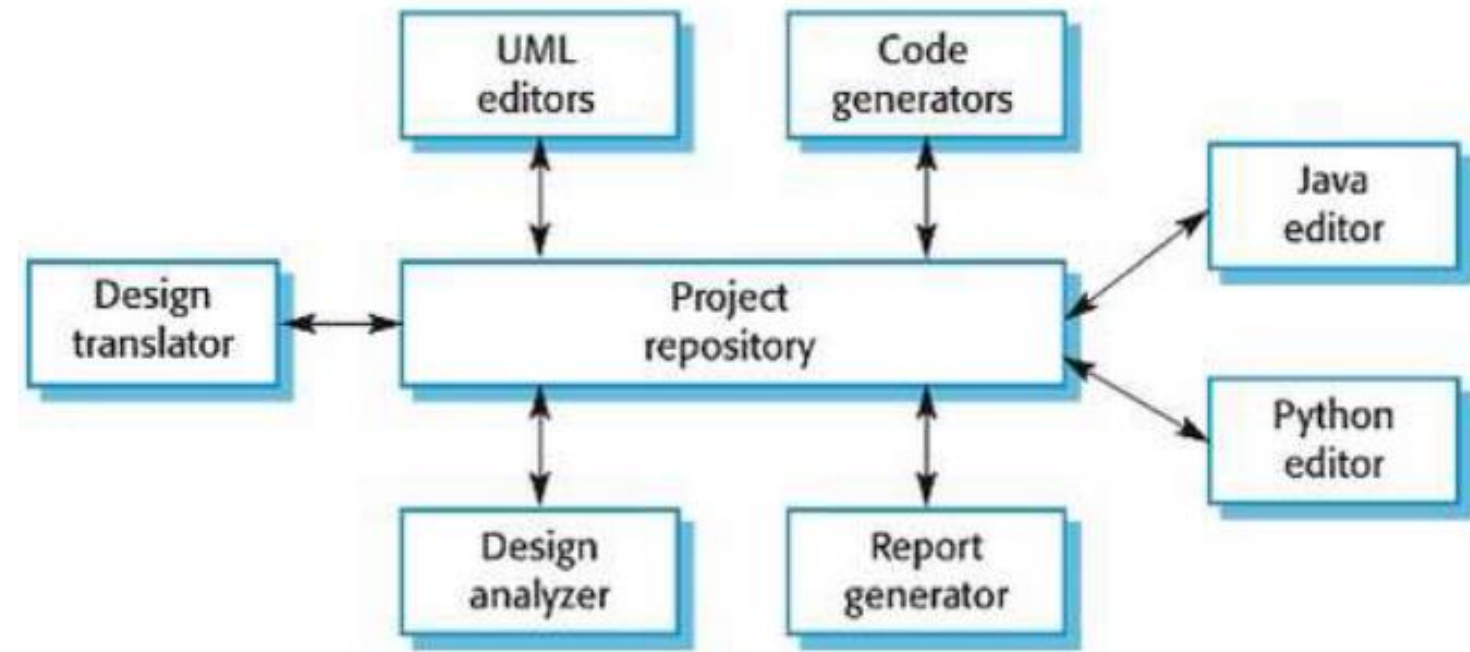
## Estilos Arquitectónicos

### 4. Arquitecturas centradas en los datos

Repositorio

Ejemplo

IDE

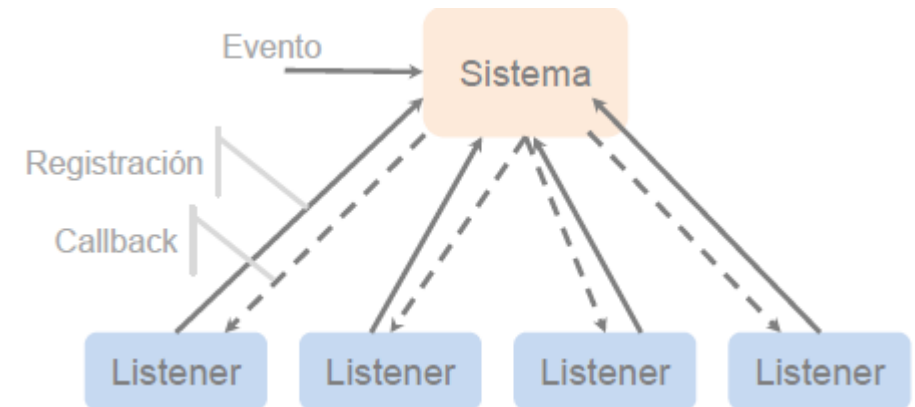


# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

- ★ Se anuncian los eventos en vez de invocarse métodos
- ★ Los “listeners” se registran como interesados y asocian métodos (callbacks) con eventos.
- ★ Al producirse un evento, el sistema invoca a todos los métodos registrados
- ★ Quien anuncia el evento, no sabe a quién afectará
- ★ No hay suposiciones sobre el orden de procesamiento en respuesta a eventos





# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Publisher- Subscriber

- 1 Toma su nombre de la relación análoga entre publicadores de diarios y revistas y sus subscriptores
- 2 Es usado para enviar eventos y mensajes a un conjunto desconocido de receptores.
- 3 El conjunto de receptores es desconocido para el productor del evento, la correctitud del productor no puede depender de los receptores.
- 4 Nuevos receptores puede agregarse sin cambios en el/los productor/es





# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Publisher- Subscriber

- 1 Toma su nombre de la relación análoga entre publicadores de diarios y revistas y sus subscriptores
- 2 Es usado para enviar eventos y mensajes a un conjunto desconocido de receptores.
- 3 El conjunto de receptores es desconocido para el productor del evento, la correctitud del productor no puede depender de los receptores.
- 4 Nuevos receptores puede agregarse sin cambios en el/los productor/es



# Arquitectura de Software

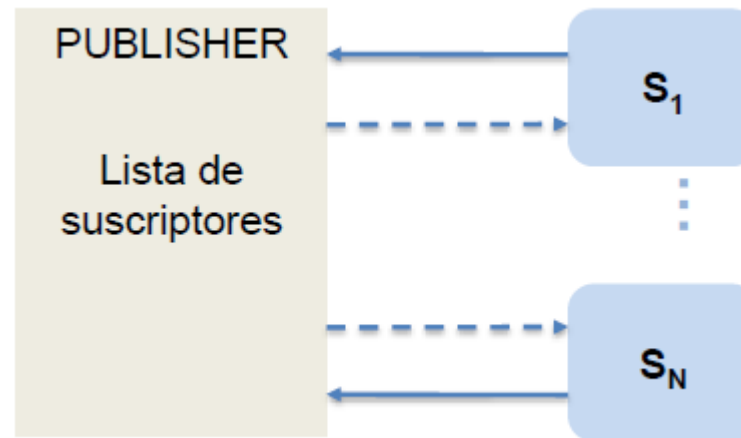
## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

Publisher- Subscriber

Basados en Listas

Cada Publisher mantiene una lista de suscripciones



# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

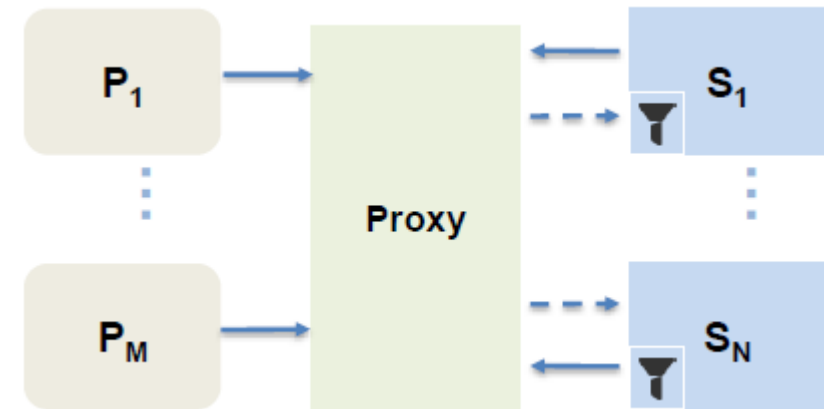
#### Publisher- Subscriber

##### Basados en Broadcast

Los Publishers tienen poco (o ningún) conocimiento de los Subscribers.

Todos los eventos son emitidos a todos los Subscribers

Los Subscribers deben filtrar los eventos que son de su interés



# Arquitectura de Software

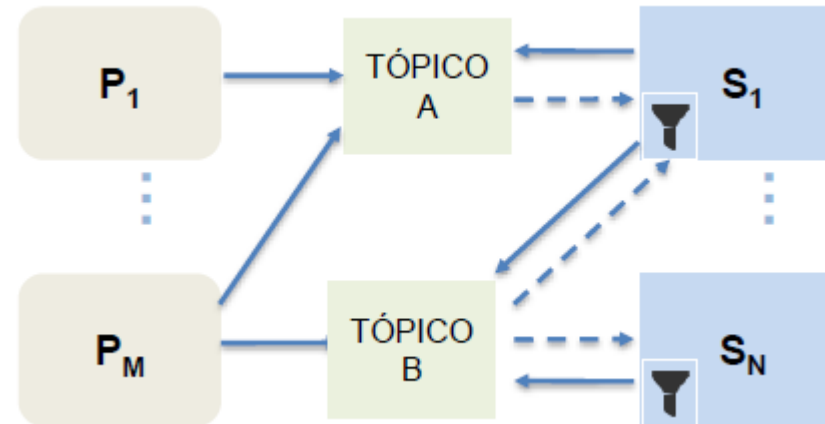
## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

**Publisher- Subscriber**

Basados en Contenidos (Tópicos)

Los tópicos son tipos de eventos o mensajes predefinidos



# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

Publisher- Subscriber

VENTAJAS	DESVENTAJAS
Desacoplamiento	Se agrega una capa de direccionamiento afectando la Latencia
Escalabilidad	No se garantiza la entrega de mensajes, ni el orden en el que llegan
	Disponibilidad



# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

**Publisher- Subscriber**

Ejemplos

---

Interfaces de usuarios gráficas

Listas de correo

---

Aplicaciones basadas en MVC

Redes sociales

---

Ambientes de desarrollo extensibles

---



# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Publisher- Subscriber

<b>DESCRIPCIÓN</b>	Subscribers se registran/desregistran para recibir mensajes o contenidos específicos. Cuando el Publisher publica, el mensaje es enviado a los Subscribers
<b>COMPONENTES</b>	<ul style="list-style-type: none"> <li>• publishers</li> <li>• subscribers</li> <li>• proxies para manejar la distribución</li> </ul>
<b>CONECTORES</b>	<ul style="list-style-type: none"> <li>• Llamadas a procedimientos pueden ser usadas dentro de un programa.</li> <li>• Protocolos de red son más frecuentes.</li> </ul>
<b>ELEMENTOS DE DATOS</b>	<ul style="list-style-type: none"> <li>• suscripciones</li> <li>• notificaciones</li> <li>• información publicada</li> </ul>
<b>TOPOLOGÍA</b>	Subscribers se conectan a Publishers en forma directa o pueden recibir notificaciones de intermediarios
<b>RESTRICCIONES ADICIONALES</b>	No
<b>CUALIDADES</b>	Altamente eficiente para distribuir información en un solo sentido con muy bajo acoplamiento de componentes.
<b>USOS TÍPICOS</b>	<ul style="list-style-type: none"> <li>• o Distribución de noticias</li> <li>• o GUIs</li> <li>• o Juegos en red multi-player</li> </ul>
<b>PRECAUCIONES</b>	Cuando la cantidad de Subscribers para un tópico es muy grande, un protocolo especial puede ser necesario

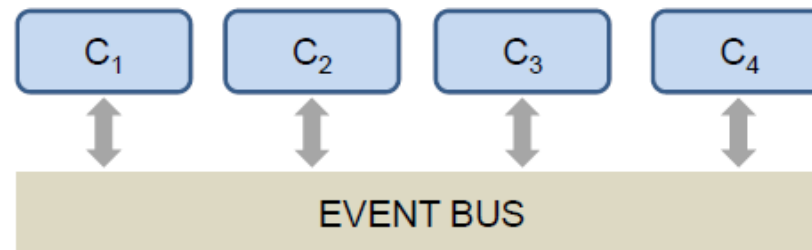
# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Basados en Eventos

- 1 Componentes independientes comunicándose sólo enviando eventos a través de conectores a un event-bus
- 2 Los componentes emiten eventos al event-bus en forma asincrónica, el cual luego los transmite a los otros componentes. Cada componente puede reaccionar ante la recepción de un evento, o ignorarlo.
- 3 Los conectores se encargan de:
  - Optimizar la distribución de eventos
  - La replicación de eventos (transparente al emisor y receptor)





# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Basados en Eventos

##### Tipos de Distribución

**Pull (polling)** - los componentes consultan al conector por eventos

**Push** - los eventos arriban a los componentes ni bien se producen

##### Tipos de Implementaciones

Middlewares o librerías que resuelven el Event-Bus y los conectores (llamados Enterprise Service Bus – ESB)

*Mule, JBoss ESB, Oracle Service Bus, Microsoft BizTalk Services, Windows Azure Service Bus, Spring Integration, IBM WebSphere BUS, IBM Integration BUS*



# Arquitectura de Software

## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

#### Basados en Eventos

<b>DESCRIPCIÓN</b>	Componentes independientes que asincrónicamente emiten y reciben eventos comunicados a través de event-buses.
<b>COMPONENTES</b>	Generadores y/o consumidores de eventos independientes y concurrentes.
<b>CONECTORES</b>	Event-bus. Podría existir más de uno.
<b>ELEMENTOS DE DATOS</b>	Eventos
<b>TOPOLOGÍA</b>	Los componentes se comunican con el event-bus, no directamente entre ellos.
<b>RESTRICCIONES ADICIONALES</b>	No
<b>CUALIDADES</b>	<ul style="list-style-type: none"> <li>• Altamente escalable</li> <li>• Fácil de evolucionar</li> <li>• Efectivo para aplicaciones heterogéneas altamente distribuidas.</li> </ul>
<b>USOS TÍPICOS</b>	<ul style="list-style-type: none"> <li>• Software de UI</li> <li>• Aplicaciones de área amplia que involucran partes independientes (mercados financieros, logística)</li> </ul>
<b>PRECAUCIONES</b>	No existen garantías que un evento sea procesado, ni cuando lo será.

# Arquitectura de Software

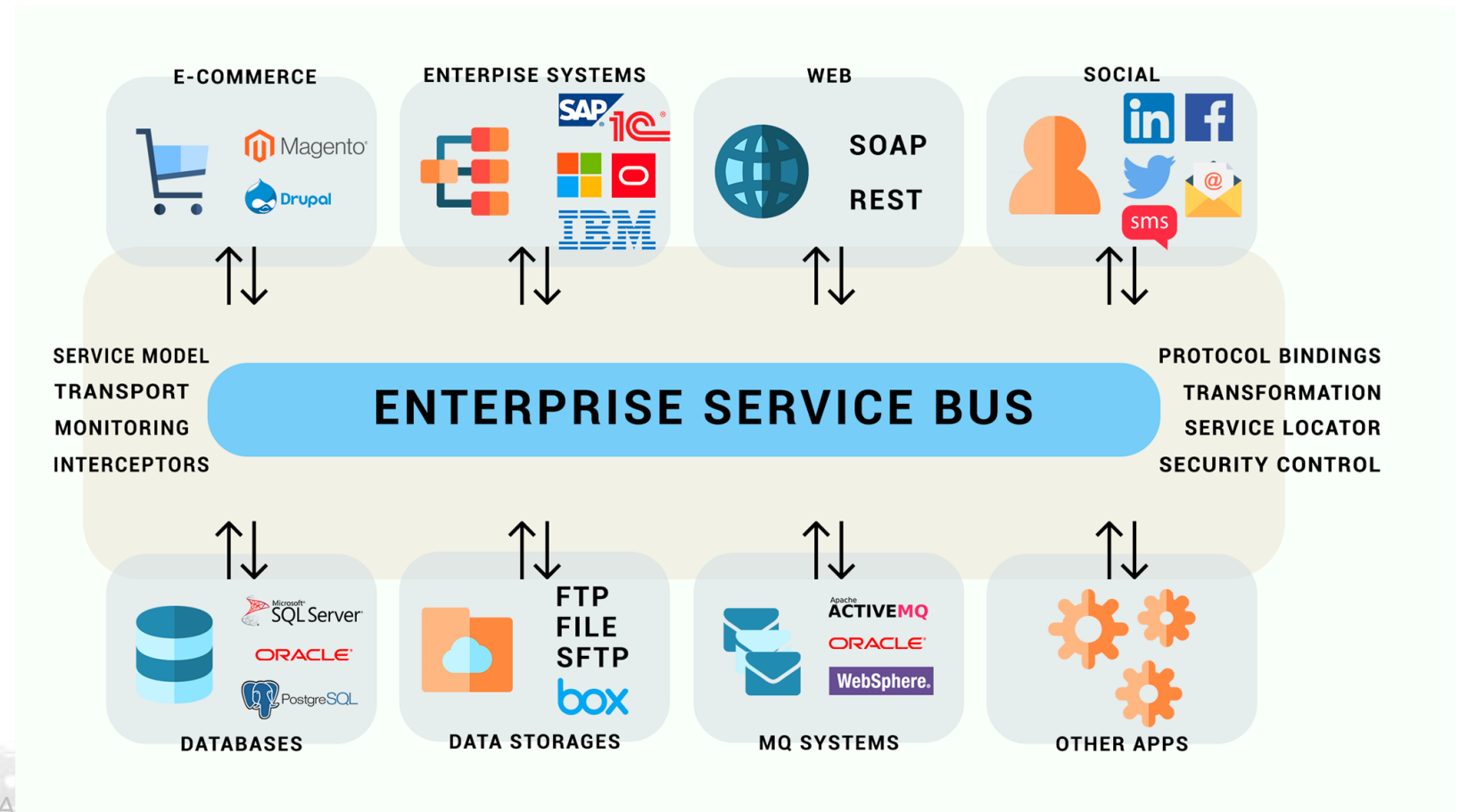
## Estilos Arquitectónicos

### 5. Arquitecturas de Invocación Implícita

Basados en Eventos

Ejemplo

Enterprise Service Bus



# Arquitectura de Software

## Atributos de Calidad



# Arquitectura de Software

## Atributos de Calidad



Hola Felipe, Maria de nuevo. Tengo una pregunta acerca del nuevo sistema de empleados que programaste. Como sabes, este sistema corre en nuestro mainframe y cada departamento tiene que pagar por su almacenamiento en disco y por el consumo de CPU cada mes. Al parecer el nuevo sistema de archivos está utilizando por lo menos el doble de disco que el anterior. Peor aún, los costos por CPU se han triplicado. ¿Puedes decirme que ha pasado?

Claro que si María, *dice Felipe*. Recuerda que requerías que el sistema almacenara mucho más información acerca de cada empleado respecto al sistema anterior, por lo tanto la base de datos es más grande. Y por lo tanto es natural que pague más por mes. También especificaste que requerías que el sistema fuera más sencillo de usar, por lo tanto le adicionamos una interfaz de usuario gráfica amigable. Sin embargo, la interfaz gráfica consume muchos más recursos de CPU que el anterior sistema, que era solo de caracteres. Por eso es que tu costo por sesión de usuario es más alto, pero el nuevo sistema es más fácil de usar que el anterior, ¿cierto?



# Arquitectura de Software

## Atributos de Calidad



Si, lo es, *replicó María*, pero la verdad no pensaba que fuera tan costoso de usar. Estoy en problemas por eso. Mi jefe se está poniendo nervioso, ya que a este paso todo nuestro presupuesto de cómputo se acabará en Abril. ¿Puedes **arreglar** el sistema para que cueste mucho menos?

*Felipe estaba frustrado:* No hay nada que arreglar, realmente. El nuevo sistema de empleados es exactamente lo que tu pediste que fuera. **Asumí** que si tu tenías claro que al almacenar más datos o trabajar más con el computador tus costos iban a subir. **Posiblemente debimos haber hablado de esto antes, porque en este momento no hay mucho que yo pueda hacer.** Lo siento.



# Arquitectura de Software

## Atributos de Calidad



Se enfocan en el comportamiento y en la funcionalidad (las cosas que el software debe hacer)



El éxito del software va mucho más allá de que ofrezca la funcionalidad correcta

# Arquitectura de Software

## Atributos de Calidad



Facilidad de uso

Qué tan rápido corre

Qué tanto falla

Cómo maneja condiciones inesperadas

Estas (y otras) características se conocen colectivamente como “atributos de calidad” (o factores de calidad) y forman parte de los requerimientos no funcionales (o no comportamentales)



# Arquitectura de Software

## Atributos de Calidad

