

Ingeniería del Software  
Grado en Ingeniería Informática

Procesos del Desarrollo de Software

Practica 8 – Modelado de Componentes  
Grupo 181

Curso 2020-2021

## Arquitectura del software: definiciones

Paul Clements 1996

- La arquitectura del software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema.

Len Bass 1998

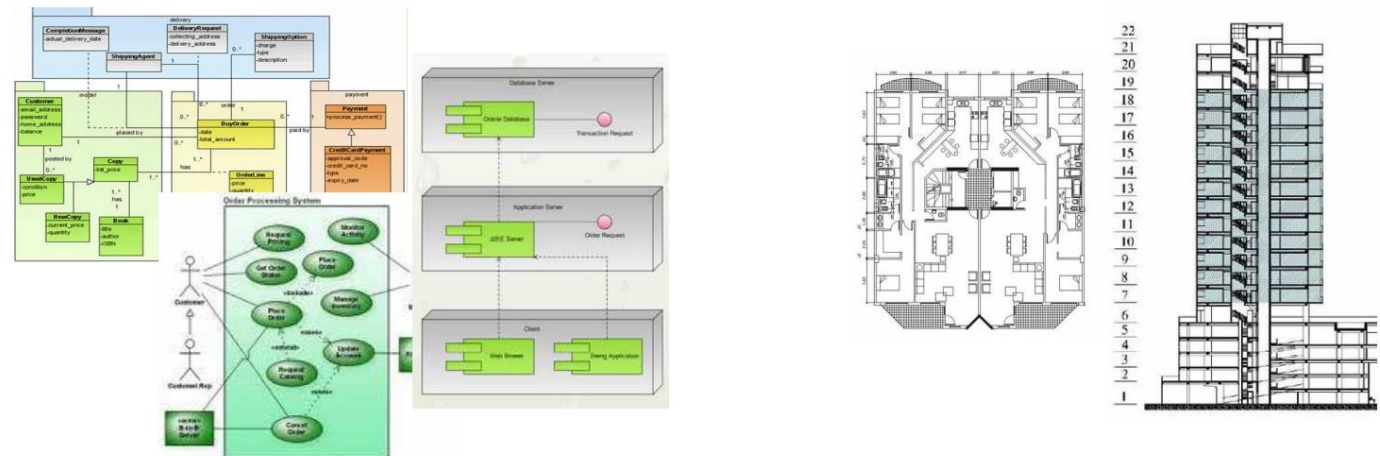
- La arquitectura del software de un programa o sistema de computación es la estructura o las estructuras del sistema, que contienen componentes de software, las propiedades externamente visibles de dichos componentes y las relaciones entre ellos.

IEEE Std. 1471-2000

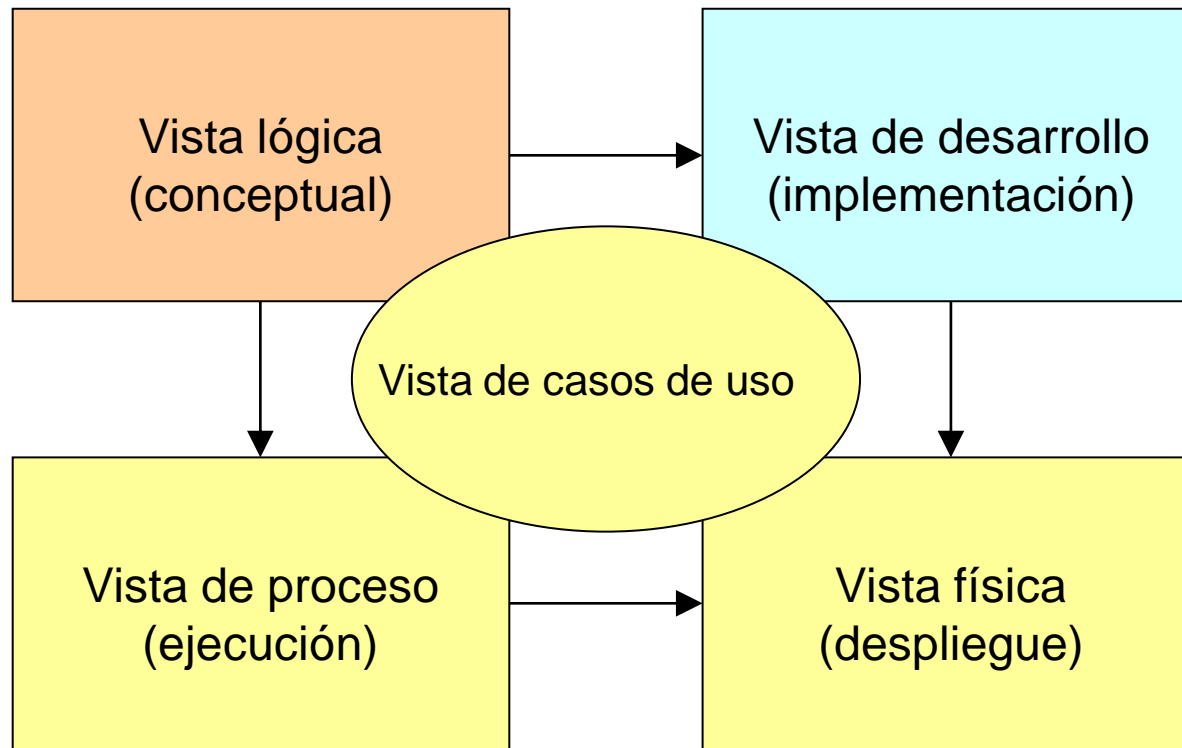
- La arquitectura del software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y con el entorno, y los principios que orientan su diseño y evolución.

# Vistas arquitecturales

- Durante el desarrollo de un sistema software se requiere que éste sea visto desde varias perspectivas.
- Diferentes usuarios miran el sistema de formas diferentes en momentos diferentes.
- La arquitectura del sistema es clave para poder manejar estos puntos de vista diferentes:
  - Se organiza mejor a través de vistas arquitecturales interrelacionadas.
  - Proyecciones del modelo del sistema centradas en un aspecto particular.
- No se requiere una vista que contenga la semántica completa de la aplicación. La semántica reside en el modelo.
- UML cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software



## El modelo de 4+1 vistas arquitectónicas



## Características de cada vista en el modelo 4+1

Vista	Lógica (conceptual)	Proceso (ejecución)	Desarrollo (implementación)	Física (despliegue)
Aspecto	Modelo de información	Concurrencia y sincronización	Organización del software en el entorno de desarrollo	Correspondencia software-hardware
Stakeholders	Usuarios finales	Integradores del sistema	Programadores	Ingenieros de sistemas
Requisitos	Funcionales	Rendimiento Disponibilidad Fiabilidad Concurrencia Distribución Seguridad	Gestión del software Reuso Portabilidad Mantenibilidad Restricciones impuestas por la plataforma o el lenguaje	Rendimiento Disponibilidad Fiabilidad Escalabilidad Topología Comunicaciones
Notación	Clases y asociaciones	Procesos y comunicaciones	Componentes y relaciones de uso	Nodos y rutas de comunicación

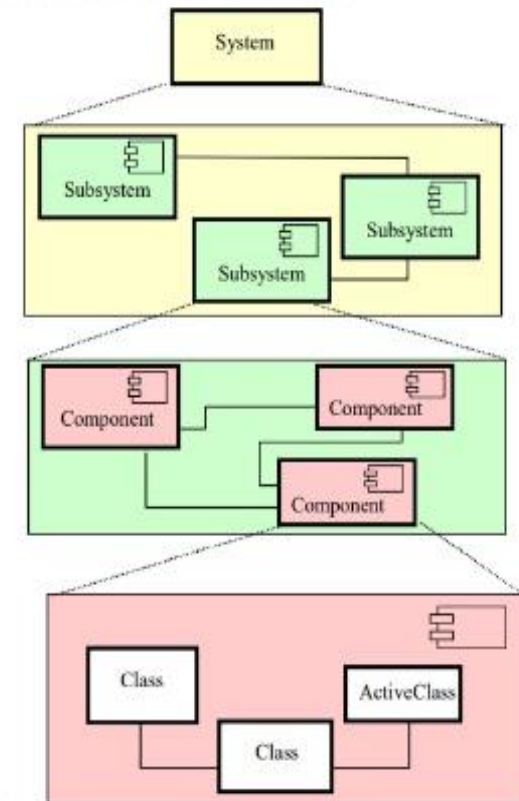
## Vista Lógica: modelo conceptual

- El propósito es especificar la arquitectura de información del sistema que se desea construir, mediante un conjunto de diagramas de clases adecuadamente explicados.
- El modelo de información, o modelo conceptual, debe estar justificado a partir de los requisitos. No tiene sentido que en él aparezcan clases, atributos, operaciones y otros elementos que no hayan aparecido anteriormente en los requisitos. Igualmente, no tiene sentido que en los requisitos se mencionen conceptos importantes que no aparezcan reflejados de ninguna manera en el modelo conceptual.
- El vocabulario del modelo conceptual define todos los términos significativos y específicos del problema que aparecen en los requisitos. Es un puente importante que vincula los requisitos con el modelo conceptual.



## Vista de Desarrollo: modelo de implementación

- Define la **descomposición del sistema en subsistemas y componentes**, y se especifican las dependencias entre los distintos componentes que hayan resultado de la descomposición
- Los **componentes** se refieren a:
  - Elementos estático y estructurales del sistema que **representa una funcionalidad concreta** o un conjunto de ellas utilizados en la implementación: funciones, librerías, etc.
  - Conjunto de datos usados en la implementación: ficheros, bases de datos, etc.
- La arquitectura de desarrollo se representan a través de los **diagramas de componentes**.



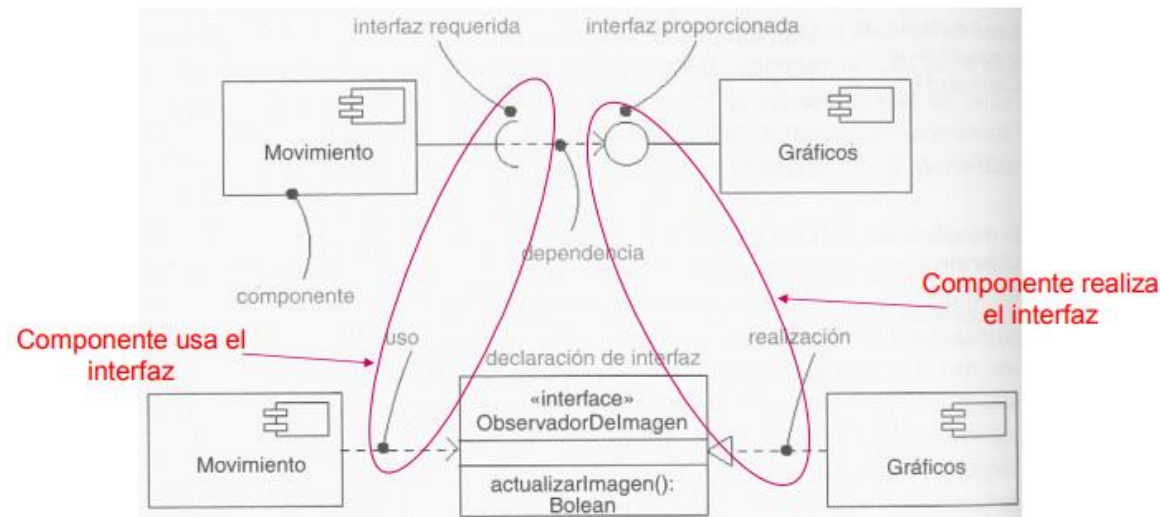
## Vista de desarrollo (Implementación)

### Componentes vs Clases

- Se parecen a las clases en que tienen nombres, realizan interfaces, pueden participar en relaciones,
- Pero se diferencian en que las clases son abstracciones lógicas (se instancian como objetos) y los componentes son fragmentos físicos.

### Componentes e interfaces


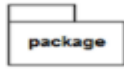






- Definen su comportamiento en base a interfaces requeridas y ofertadas.
- Unos componentes implementan las interfaces y otros acceden a los servicios proporcionados por esas interfaces.
- Estas relaciones se pueden mostrar con dos tipos de notación: icónica o expandida.

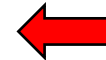




# Vista de desarrollo (Implementación)

## Diagrama de Componentes: notación

Elemento	Símbolo/notación	Explicación
Componente ( <i>component</i> )		Símbolo para representar los módulos de un sistema (la interacción y la comunicación tienen lugar a través de interfaces).
Paquete		Un paquete combina varios elementos del sistema (por ejemplo, clases, componentes o interfaces) en un grupo.
Artefacto		Los artefactos son unidades físicas de información (por ejemplo, código fuente, archivos .exe, scripts o documentos) que se generan en el proceso de desarrollo o el tiempo de ejecución de un sistema o son necesarios para estos.
Interfaz ofrecida		Símbolo para una o más interfaces claramente definidas que proporcionan funciones, servicios o datos al mundo exterior (el semicírculo abierto también se denomina enchufe o <i>socket</i> ).
Interfaz requerida		Símbolo de una interfaz necesaria para recibir funciones, servicios o datos del exterior (la notación del círculo con palo también se denomina <i>lollipop</i> o <i>piruleta</i> ).
Puerto		Este símbolo indica un punto de interacción independiente entre un componente y su entorno.
Relación		Las líneas actúan como conectores e indican las relaciones entre los componentes.
Relación de dependencia		Conector especial para expresar una relación de dependencia entre los componentes del sistema (no siempre se indica).

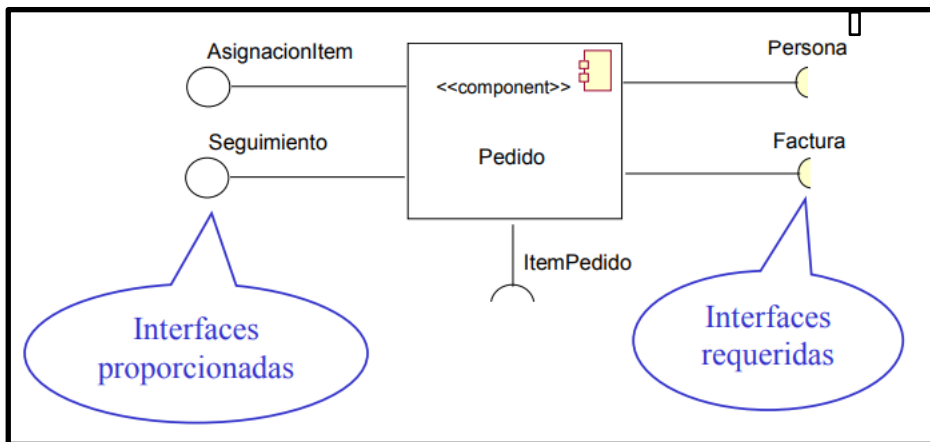


# Vista de desarrollo (Implementación)

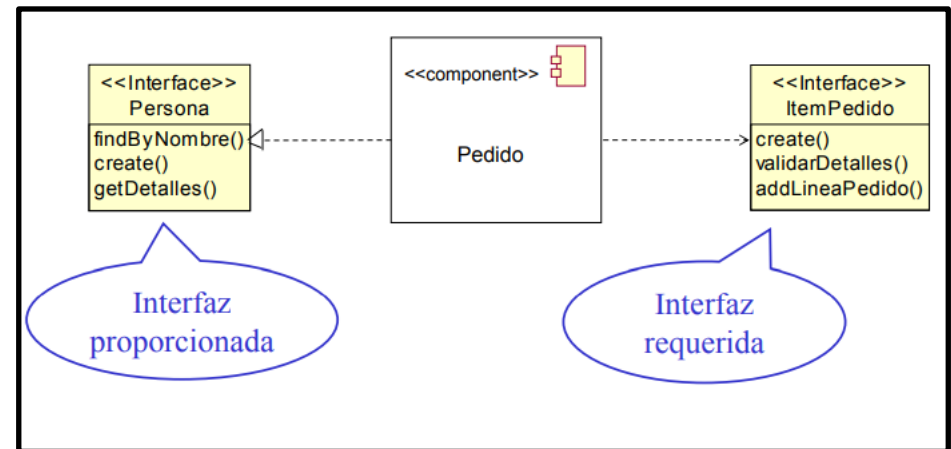
Componentes: notación

Ejemplo de interfaces requeridas y proporcionadas.

Notación icónica

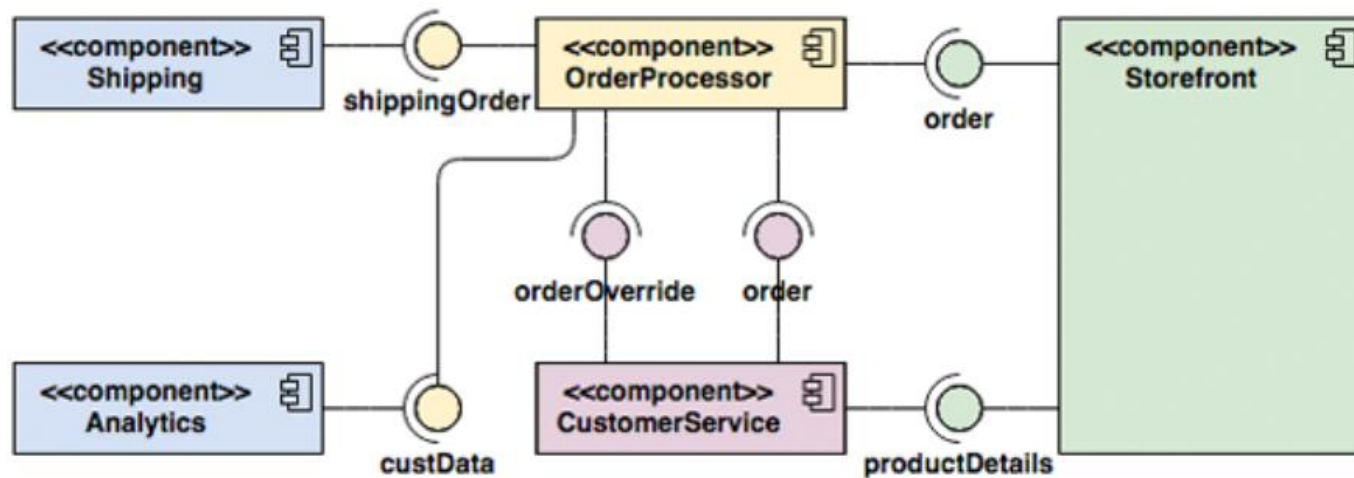


Notación extendida



## Vista de desarrollo (Implementación)

Sistema de pedidos: Ejemplo diagrama de componentes



# Patrones de arquitectura

## Decisiones del diseño

- ¿Existe una arquitectura de aplicación genérica que pueda actuar como una plantilla para el sistema que se está diseñando?
- ¿Qué estilo o estilos arquitectónicos son apropiados para el sistema?
- ¿Cómo debería documentarse la arquitectura del sistema?
- ¿Cómo se descompondrán en módulos las unidades estructurales del sistema?
- ¿Qué sucede si desean realizar modificaciones con otra empresa o desarrollador?
- ¿Qué lenguaje de programación utilizar?

- Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información.
- Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto.
- **Los patrones de diseño** de software ayudan a diseñar una aplicación. **Un patrón es la solución replicable a cierto problema.**
- En una arquitectura de aplicaciones habrá:
  - **servicios de frontend:** se refiere a la experiencia del usuario con la aplicación
  - **servicios de backend:** implica proporcionar acceso a los datos, los servicios de negocio y otros sistemas actuales que permiten el funcionamiento de la aplicación.

## Patrones de arquitectura

- **Monolito**

- Los monolitos son otro tipo de arquitectura asociado con los sistemas heredados.
- Antes las aplicaciones se escribían como una sola unidad de código, en la que todos los elementos compartían los mismos recursos y espacio de memoria.
- Son pilas de aplicaciones únicas que contienen todas las funciones dentro de cada aplicación. Tienen conexión directa, tanto en la interacción entre los servicios como en la manera en que se desarrollan y distribuyen.
- Esto implica que al actualizar o ampliar un solo aspecto de una aplicación monolítica, habrá una repercusión en toda esa aplicación y en la infraestructura subyacente.

- **Cliente-servidor**

Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.

# Patrones de arquitectura

- **Modelo Vista Controlador**

Es el conocido MVC, que divide una aplicación interactiva en tres partes (modelo, vista, controlador) encargadas de contener la funcionalidad, mostrar la información al usuario y manejar su entrada. Este patrón de arquitectura de software separa los datos y la lógica de negocio de una aplicación de su representación.

- **El Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (**lógica de negocio**). En los frameworks actuales normalmente representa una entidad del diagrama entidad-relación.
- **La Vista:** presenta el modelo (información y lógica de negocio) en un formato adecuado para que un usuario pueda interactuar (**usualmente la interfaz de usuario**).
- **El controlador:** es el intermediario entre la vista y el modelo, su función consiste en controlar el flujo de datos, responder a eventos (usualmente provocados por los usuarios) e invocar peticiones al modelo.

- **Maestro esclavo**

Suele utilizarse para replicaciones en la base de datos (la maestra es la fuente autorizada y las esclavas se sincronizan con ella). Estas dos partes distribuyen el trabajo y calculan el resultado final de toda la actividad que realizan dichos esclavos. Este patrón es una arquitectura fundamental que los desarrolladores utilizan cuando tienen dos o más procesos que necesitan ejecutarse de forma simultánea.

## Patrones de arquitectura

- **Igual a igual (pares)**

Todos los elementos individuales se les denomina 'pares', que pueden funcionar tanto como 'cliente', como 'servidor'. Además, pueden ir cambiando su rol con el paso del tiempo.

- **En tuberías**

Arquitecturas de flujo de datos. Esta arquitectura se aplica cuando los datos de entrada son transformados a través de una serie de componentes computacionales o manipulativos en los datos de salida. Típicamente usada en procesamiento de señales y transformación de flujos de datos. Los componentes reciben el nombre de 'filtros' conectados entre sí por 'tuberías' que transmiten los datos.

- **En pizarra**

Es una arquitecturas centradas de datos. Algunas veces llamado Blackboard. En el centro de esta arquitectura se encuentra un almacén de datos (por ejemplo, un documento o una base de datos) al que otros componentes acceden con frecuencia para actualizar, añadir, borrar o bien modificar los datos del almacén. Muy utilizada en sistemas expertos, visión artificial, interpretación sensorial.

# Patrones de arquitectura

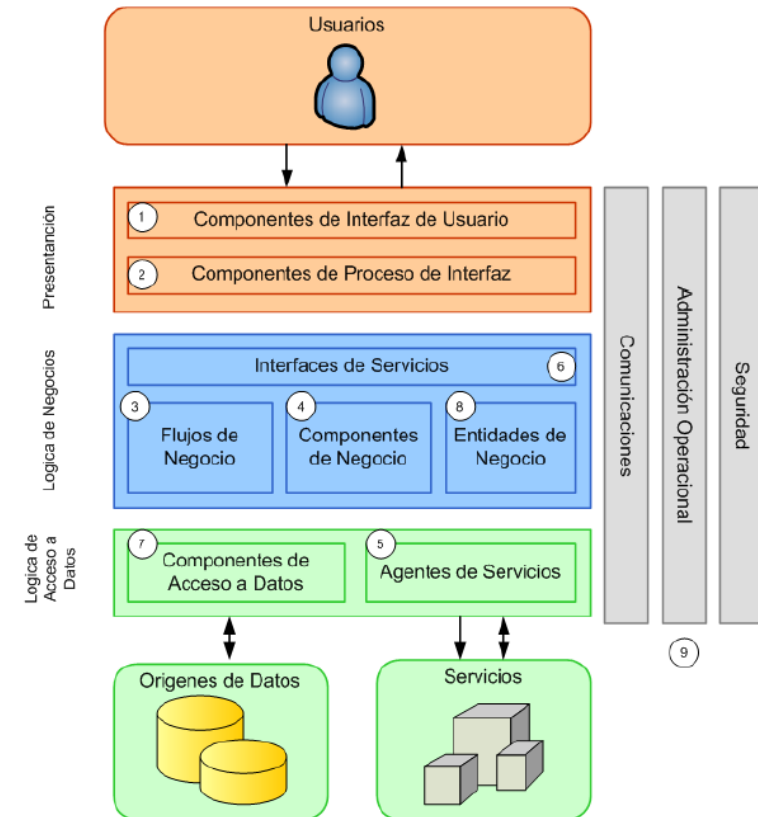
- Arquitectura en capas

Especialización de la arquitectura cliente-servidor donde la carga se divide en capas con un reparto claro de funciones:

- una **capa para la presentación (interfaz de usuario)**,
- **otra para el cálculo (donde se encuentra modelado el negocio)**
- y otra para el **almacenamiento (persistencia)**.

Una capa solamente tiene relación con la siguiente.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

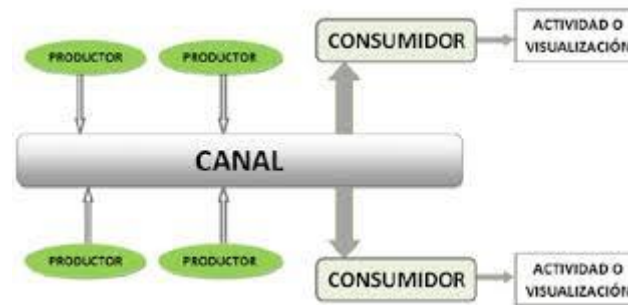




## Patrones de arquitectura

### • Dirigido por eventos

- Este patrón arquitectónico puede ser aplicado por el diseño e implementación de aplicaciones y sistemas que transmitan eventos entre componentes software.
- Los consumidores tienen la responsabilidad de llevar a cabo una reacción tan pronto como el evento esté presente se usan para entornos no predecibles y asíncronos.
- Difiere del modelo tradicional basado en solicitudes.
- Los eventos son aquellos sucesos o cambios significativos en el estado del hardware o el software de un sistema. Los eventos suceden con estímulos internos o externos.
- Está compuesta por consumidores y productores de eventos. El productor detecta los eventos y los representa como mensajes. No conoce al consumidor del evento ni el resultado que generará este último.
- Después de la detección de un evento, este se transmite del productor a los consumidores a través de los canales de eventos, donde se procesan de manera asíncrona con una plataforma de procesamiento de eventos.
- Una arquitectura de este tipo puede basarse en un modelo de publicación/suscripción o en un modelo de flujo de eventos.
- Extendido por ejemplo para la gestión del Internet de las cosas (IoT) entre otros.



# Patrones de arquitectura

- **Orientada a servicios (SOA)**

- Estructura las aplicaciones en **servicios independientes y reutilizables que se comunican a través de un bus de servicios empresariales (ESB)**.
- Un **servicio es una representación lógica de una actividad de negocio** que tiene un resultado de negocio específico (ejemplo: comprobar el crédito de un cliente, obtener datos de clima, consolidar reportes de perforación)
- En esta arquitectura, cada uno de los servicios individuales se organiza en torno a un proceso empresarial específico, y todos cumplen con un protocolo de comunicación y se exhiben a través de la plataforma del ESB.
- **ESB es un middleware. Es el elemento de software que media entre las aplicaciones empresariales y permite la comunicación entre ellas.**
- La arquitectura SOA permite la implantación de procesos de negocio que utilizan los servicios proporcionados por los sistemas actuales. **La arquitectura garantiza la interoperabilidad de los procesos del sistema aunque estos hayan sido construidos en distintos momentos.**
- **Permite la reutilización de activos existentes** para nuevos servicios que se pueden crear a partir de una infraestructura de TIC diseñada con anterioridad. Posibilita:
  - La optimización de la inversión por medio de la reutilización del software.
  - la interoperabilidad entre las aplicaciones y tecnologías heterogéneas.
  - La reducción de costes de implementación, innovación de servicios a clientes, adaptación ágil ante cambios y una rápida reacción ante la competitividad, ya que, combinan fácilmente las nuevas tecnologías con aplicaciones independientes, permitiendo que los componentes del proceso se integren y coordinen de manera efectiva y rápida.

# Patrones de arquitectura

- **Arquitecturas de microservicios**

- Algunos consideran que es una especialización de una forma de implementar SOA. Basa la construcción de las aplicaciones en un conjunto de pequeños servicios
- En el mundo real, no todas las implementaciones de este estilo de arquitecturas siguen las mismas características, pero la mayor parte de las arquitecturas de microservicios tienen la mayor parte de las siguientes características:
  - **Los componentes son servicios.** que se ejecutan en su propio proceso y se comunican con mecanismos ligeros. Por ejemplo: una API con recursos HTTP. Cada uno de estos servicios independientes se encargará de implementar una funcionalidad.
  - **Organizada en torno a las funcionalidades del negocio.**
  - **Productos, no proyectos.** En esta arquitectura normalmente se sigue la idea de que un equipo debe estar a cargo de un componente (servicio) durante todo el ciclo de vida del mismo, desde la etapa de diseño y construcción, la fase de producción y hasta la de mantenimiento.
  - **Tener gobierno descentralizado** permite usar tecnologías que se adapten mejor a cada funcionalidad. Con el sistema con múltiples servicios colaborativos, podemos decidir utilizar diferentes lenguajes de programación y tecnologías dentro de cada servicio.
  - **Gestión de datos descentralizada.** Los microservicios prefieren dejar a cada servicio que gestione su propia base de datos, sean estos diferentes instancias de la misma tecnología de base de datos o sistemas de base de datos completamente diferentes
  - **Diseño tolerante a fallos.**
  - **Automatización a través la entrega e integración continua** (DevOps).
  - **Diseño evolutivo.** Cuando se divide el sistema en servicios hay que tener en cuenta que cada uno tiene que poder ser reemplazado o actualizado de forma independiente. Es decir, tiene que permitir una fácil evolución. El diseño del servicio tiene que ser de tal forma que evite en lo posible que la evolución de los servicios afecte a sus consumidores.

## Diseñar una arquitectura: pasos

- Buscar los patrones más apropiados que den el soporte requerido para alcanzar los atributos de calidad deseados
  - Basarse en Arquitecturas de Referencia reconocidas por tanto por la academia como por la industria
    - Implementaciones conocidas, de amplia difusión y uso
    - Buena documentación
  - Reconocer el tamaño de la aplicación objetivo
    - Aplicaciones pequeñas: Pocos patrones requeridos
    - Aplicaciones grandes: Mezcla de varios patrones
  - Debe justificarse teniendo en cuenta:
    - Los criterios: simplicidad, extensibilidad, modificabilidad, eficiencia.
    - Los requisitos no funcionales.
    - Otros que se consideren relevantes: habilidades técnicas, costes.
- Identificación de módulos: subsistemas y componentes
  - La clasificación de requisitos por áreas temáticas puede también proporcionar pistas importantes para esta descomposición.
  - Asignación de componentes
    - Identificar los componentes principales
    - Identificar como los componentes se ajustan a los patrones
    - Identificar dependencias entre ellos
    - Identificar las interfaces y los servicios que cada componente soporta