Patrones de diseño

Sesión 04:

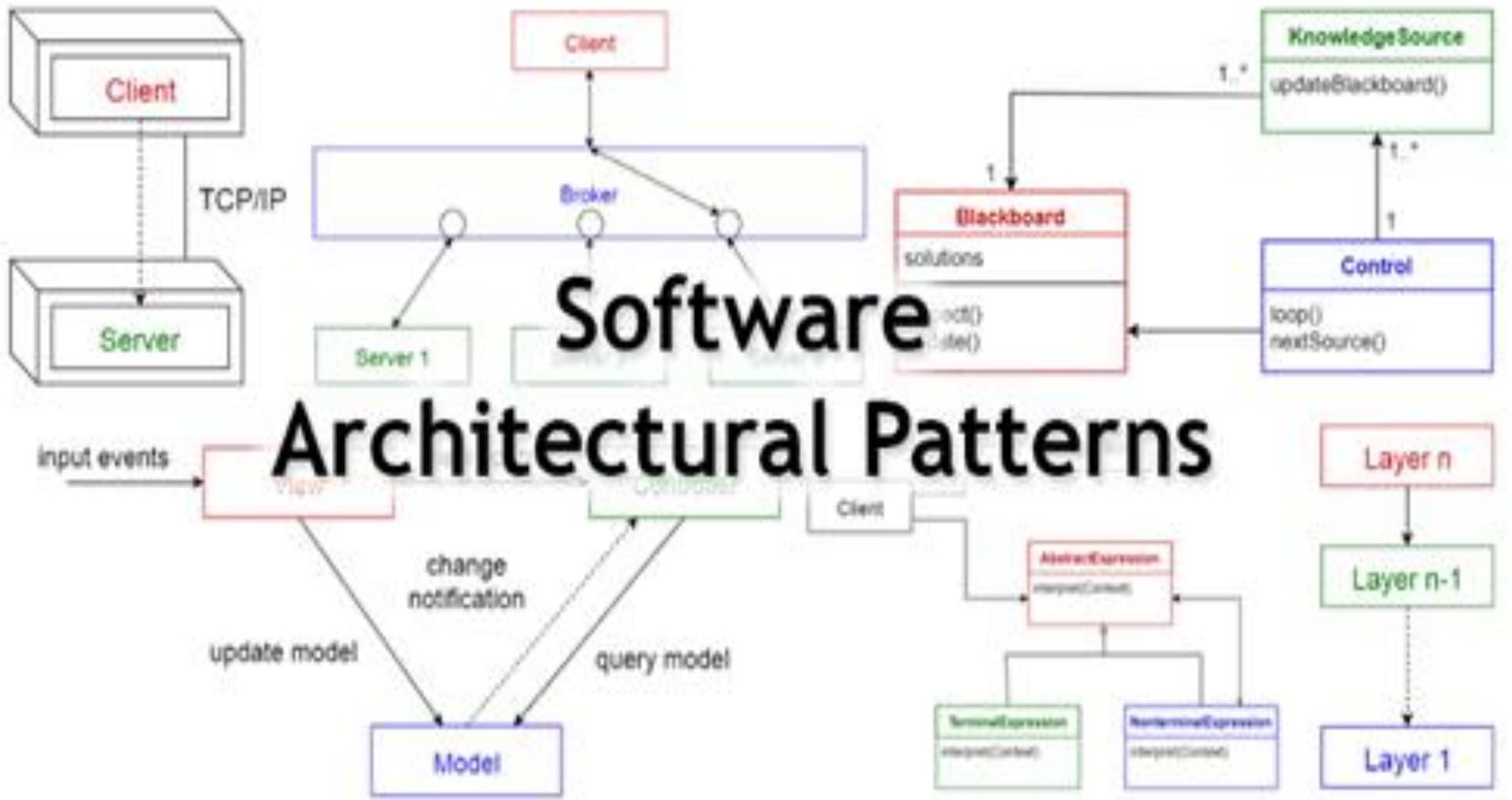
PATRONES ARQUITECTONICOS VS PATRONES DE DISEÑOS: PATRONES DE DISEÑO GOF (GANG OF FOUR)

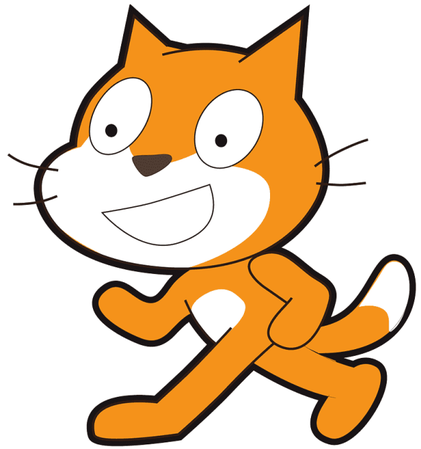
Logro de la sesión

Al finalizar la sesión, el estudiante elabora su programación usando Patrones arquitectónicos vs patrones de diseños: patrones de diseño GOF (GANG OF FOUR) empleando los diversos patrones de diseño aplicados a casos matemáticos, financieros y físicos.

Universidad Tecnológica del Perú

PATRONES ARQUITECTÓNICOS





Universidad Tecnológica del Perú

TIPOS DE PATRONES DE DISEÑO

CREACIONAL

1. Abstract Factory
2. Builder Patterns
3. Factory Method
4. Prototype
5. Singleton

ESTRUCTURAL

6. Bridge
7. Composite
8. Decorator
9. Facade
10. Flyweight
11. Proxy
12. Adapter

COMPORTAMIENTO

13. Chain of responsibility
14. Command
15. Interpreter
16. Iterator
17. Mediator
18. Memento
19. Observer
20. State
21. Strategy
22. Template method
23. Visitor

Universidad Tecnológica del Perú

Patrones en la Arquitectura de Software

Un patrón arquitectónico es un conjunto de decisiones de diseño que se repiten en la práctica, que tienen características bien definidas y que pueden reutilizarse, describiendo así características de una arquitectura bien diferenciada.

Se abordarán los patrones arquitectónicos más comunes aplicados a desarrollos de software, con sus principales características.

Universidad Tecnológica del Perú

Capas (Layer pattern)

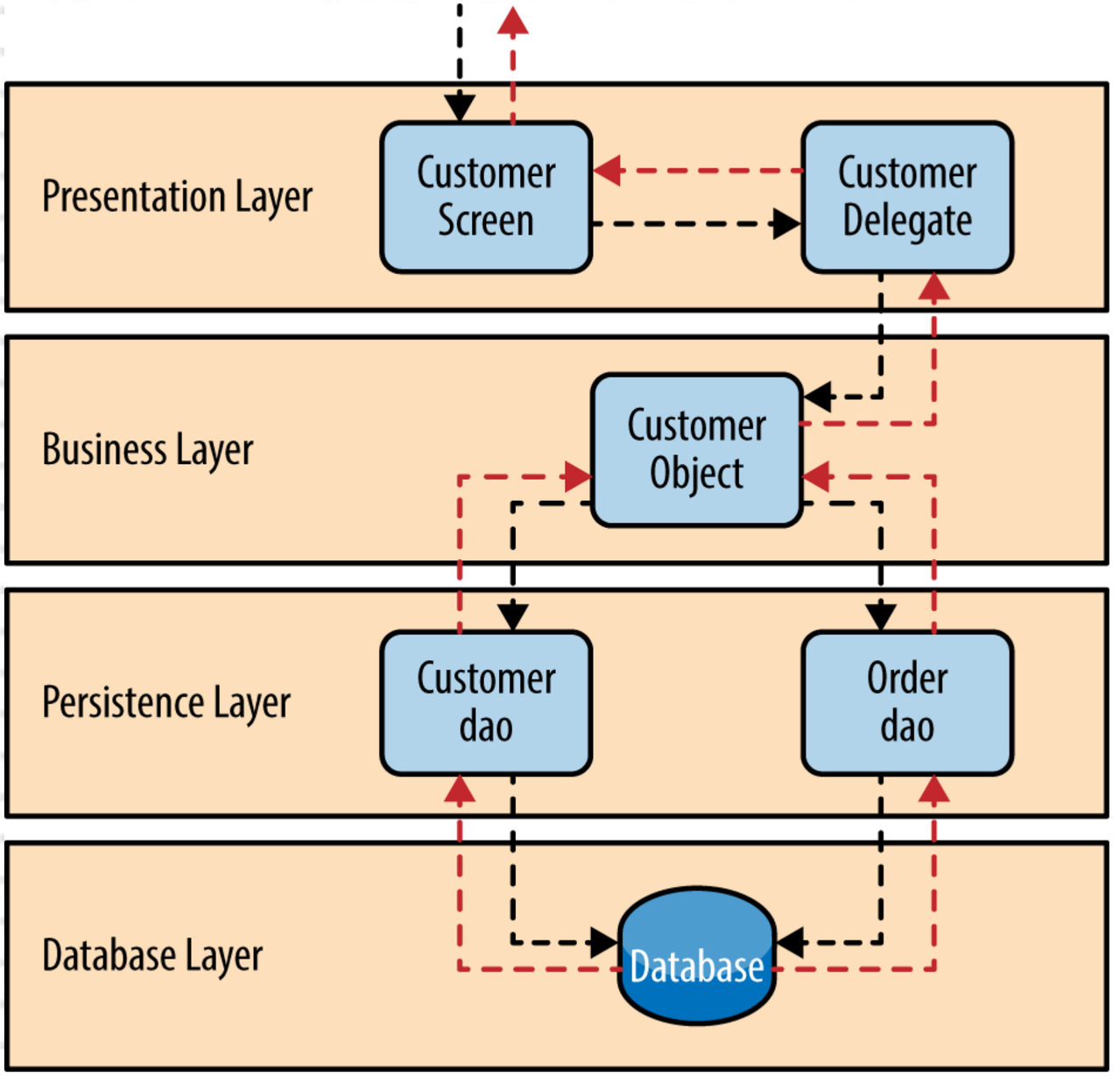
Este patrón suele ser el más utilizado, y se enfoca en estructurar aplicaciones que pueden descomponerse en grupos de subtareas o responsabilidades, las cuales se encuentran en un nivel concreto de abstracción. Cada capa proporciona servicios a la capa inmediatamente superior.

Las cuatro (4) capas más habituales de una aplicación en general, son las siguientes.

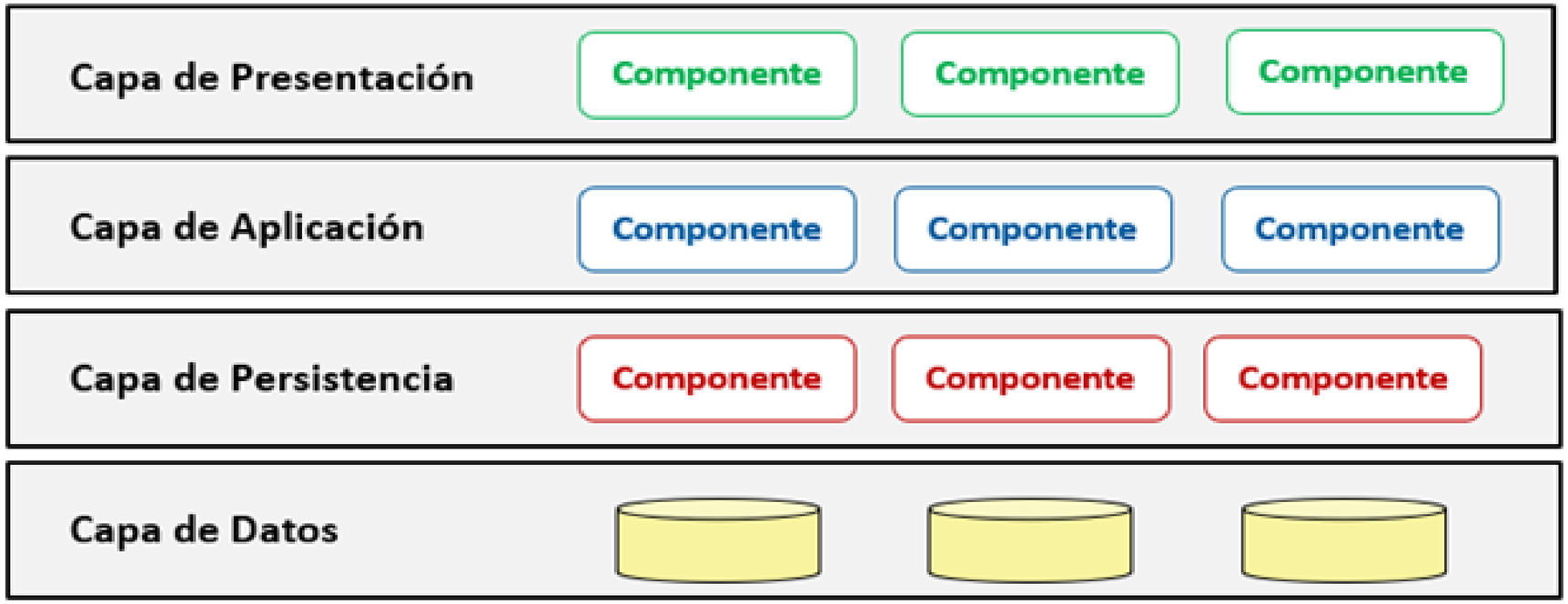
- Capa de Presentación, también conocida como Capa de Interfaz de Usuario (Front-End Layer).
- Capa de Aplicación, también conocida como Capa de Servicio (Services Layer).
- Capa de Lógica de Negocio, también conocida como Capa de Dominio o Persistencia (Back-End Layer).
- Capa de Acceso a Datos, donde residen finalmente los datos almacenados en algún gestor de bases de datos (Data Layer).

Se utilizan en el desarrollo de:

- Aplicaciones generales de escritorio.
- Aplicaciones web en general.



Patrón de Capas

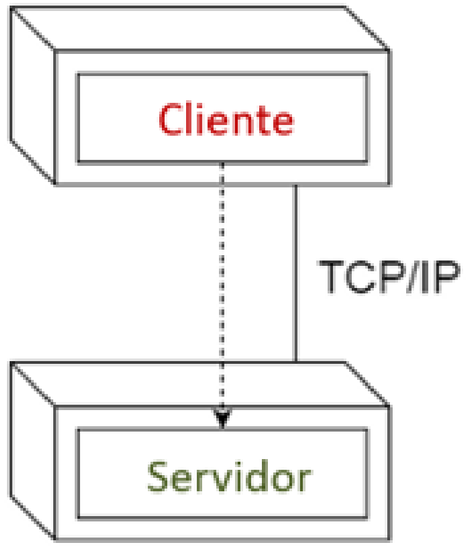


Universidad Tecnológica del Perú

Cliente Servidor (Client-Server pattern)

Este patrón consta de dos (2) partes: un servidor y múltiples clientes. El componente servidor proporciona servicios a múltiples componentes cliente. Los clientes solicitan servicios al servidor, y éste proporciona los servicios pertinentes a dichos clientes. Además, el servidor sigue escuchando las peticiones de los clientes. Algunos ejemplos de su utilización son:

- Aplicaciones en línea como correo electrónico.
- Intercambio de documentos y banca.



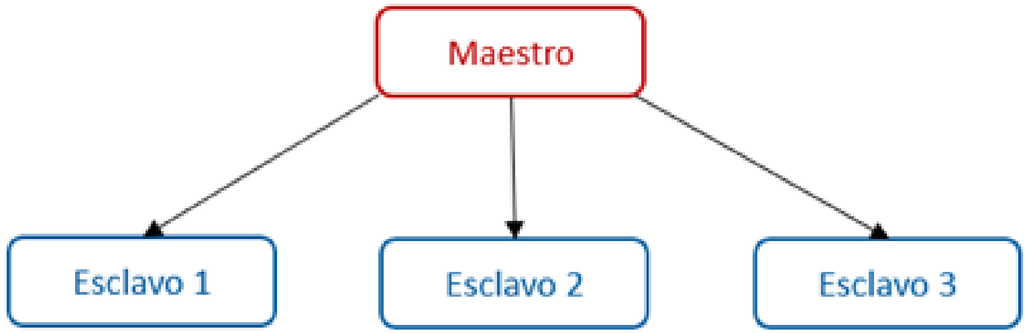
Universidad Tecnológica del Perú

Maestro-Esclavo (Master-Slave pattern)

Este patrón consta de dos (2) partes: maestro y esclavos. El componente maestro distribuye el trabajo entre componentes esclavos idénticos, y determina un resultado final a partir de los resultados que devuelven los esclavos. Entre sus usos están:

•En la replicación de bases de datos, la base de datos maestra se considera la fuente autorizada, y las bases de datos esclavas se sincronizan con ella.

•Periféricos conectados a un sistema informático (unidades maestras y esclavas).

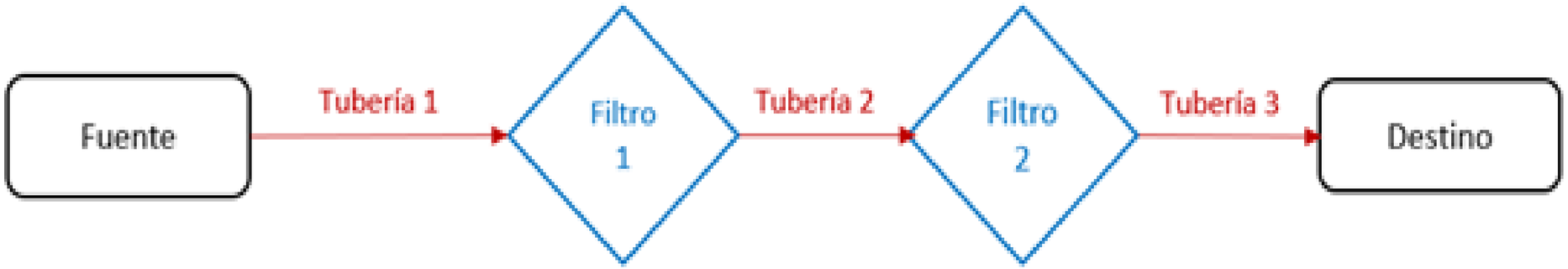


Universidad Tecnológica del Perú

Filtros-Tuberías (Pipe-Filter pattern)

Este patrón puede utilizarse para estructurar sistemas que producen y procesan un flujo de datos. Cada paso de procesamiento está encerrado dentro de un componente de filtro. Los datos que se van a procesar pasan a través de tuberías. Estas tuberías pueden utilizarse para almacenar o sincronizar datos. Se utiliza en desarrollo de aplicaciones que impliquen:

- Compiladores. Los filtros consecutivos realizan el análisis léxico, el análisis sintáctico, el análisis semántico y la generación de código.
- Procesos de Extracción, Transformación y Carga de Datos (ETLs), en soluciones de datos, tales como poblar Data Warehouse o Data Marts, entre otras.



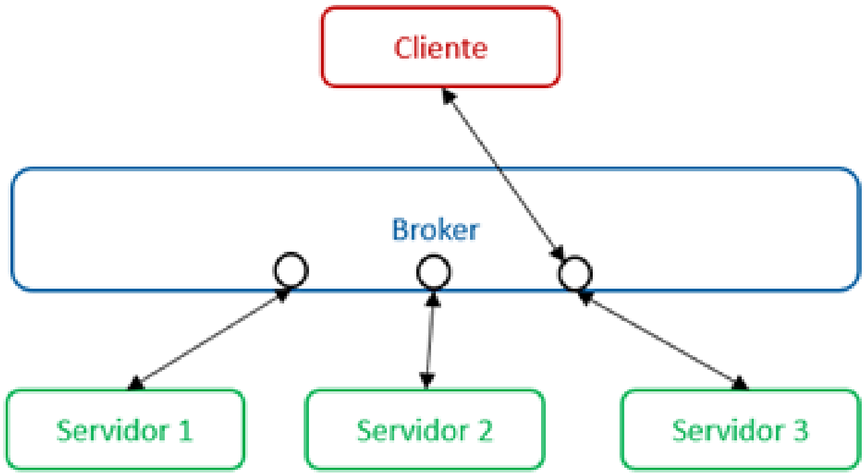
Universidad Tecnológica del Perú

Mediador (Broker pattern)

Este patrón se utiliza para estructurar sistemas distribuidos con componentes desacoplados. Estos componentes pueden interactuar entre sí mediante invocaciones a servicios remotos. Un componente Broker se encarga de coordinar la comunicación entre los componentes.

Los servidores publican sus capacidades (servicios y características) en el Broker. Los clientes solicitan un servicio a éste, y redirige al cliente a un servicio adecuado de su registro. Se suele utilizar en soluciones de software, tales como:

- Software de intermediación de mensajes como Apache ActiveMQ, Apache Kafka, RabbitMQ y JBossMessaging.

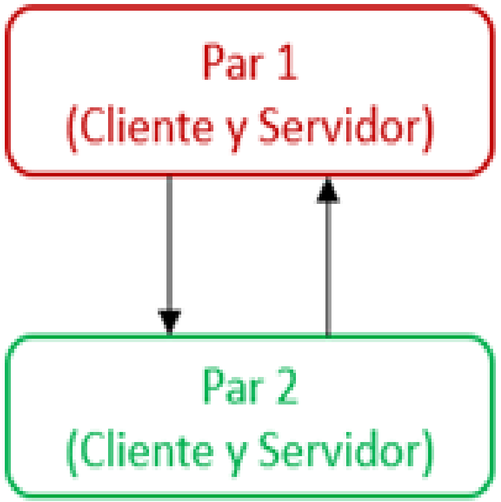


Universidad Tecnológica del Perú

Par-a-Par (Peer-to-Peer pattern)

En este patrón, los componentes individuales se conocen como Par (Peers). Los pares pueden funcionar como clientes, solicitando servicios a otros pares, y como servidores, proporcionando servicios a otros pares. Un par puede actuar como cliente, como servidor o como ambos, y puede cambiar su función dinámicamente con el tiempo. Entra sus aplicaciones prácticas se encuentran:

- Software de redes para intercambio de archivos tales como Gnutella o G2
- Protocolos multimedia tales como P2PTV y PDTP.
- Productos basados en Cadena de Bloques, tales como criptomonedas, Bitcoin y Blockchain.



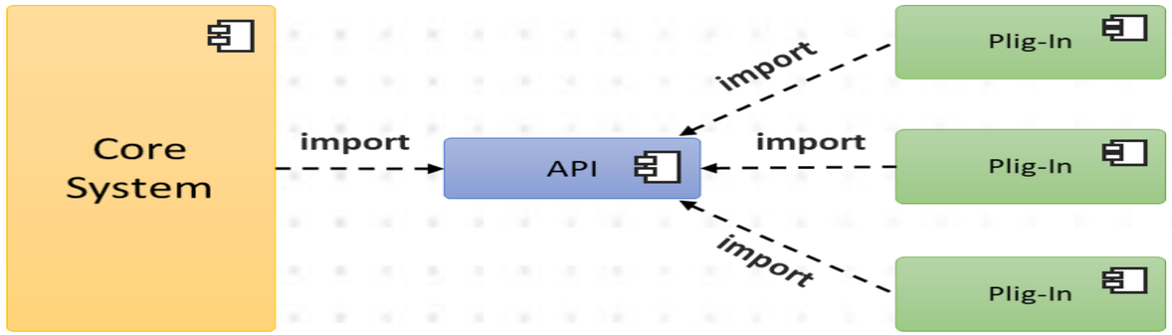
Universidad Tecnológica del Perú

Micronúcleo (Microkernel pattern)

Este patrón se ocupa en situaciones donde el software debe adaptarse a los requisitos cambiantes, pero a la vez aislados. Se parte de un núcleo mínimo de funcionalidades de cara al usuario final, para ir ampliando el software en la medida de que van surgiendo las necesidades. Sirve además como conector para ampliar las capacidades del software sin afectar al núcleo, así como para coordinar la colaboración y el desarrollo de distintos equipos de trabajo.

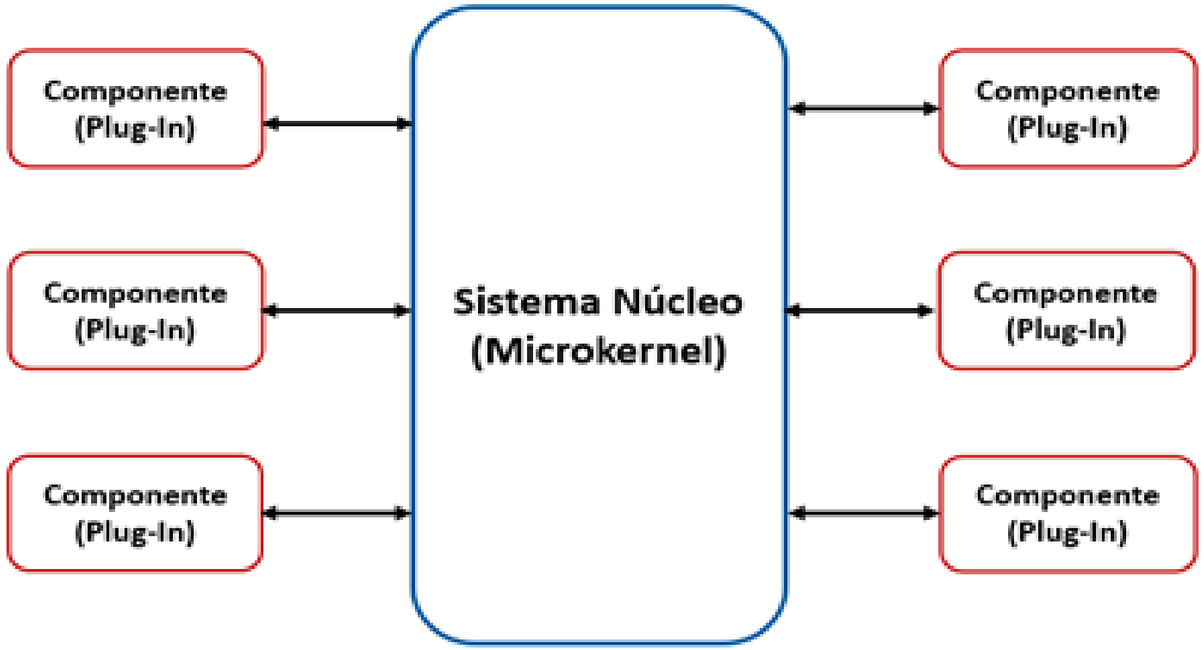
Este patrón consta de dos (2) tipos de componentes de arquitectura: un sistema básico y módulos enchufables. La lógica de la aplicación se divide entre módulos enchufables independientes y el sistema básico del núcleo, lo que proporciona extensibilidad, flexibilidad y aislamiento de las características de la aplicación.

En ese sentido, se usa para implementar aplicaciones basadas en productos. Aquellas que se empaquetan y se ponen a disposición para descargar sus versiones como un producto típico de terceros. No obstante, muchas empresas también desarrollan y publican sus aplicaciones empresariales internas como productos de software, con versiones, notas de publicación y funciones conectables. Algunos ejemplos prácticos de la arquitectura de Micronúcleo son:



Universidad Tecnológica del Perú

- El IDE de Eclipse, al descargar el producto Eclipse se obtiene poco más que un editor. Sin embargo, una vez que se empieza a añadir Plug-Ins, se convierte en un producto altamente personalizable y útil.
- Los antivirus de para computadores de escritorio, donde se van incorporando las semillas de revisión de virus sin afectar la funcionalidad principal.
- Los parches de seguridad o mejoras de los principales productos de software que soportan las operaciones en las empresas, tales como sistemas operativos, manejadores de bases de datos, entre otros.

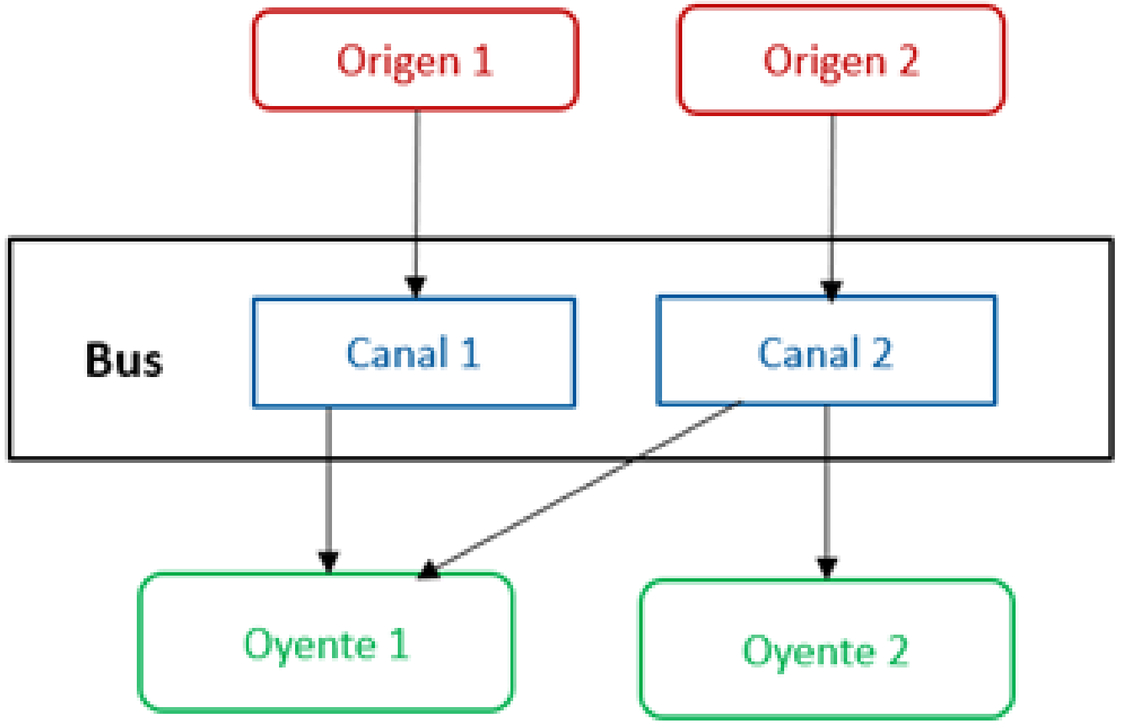


Universidad Tecnológica del Perú

Bus de Eventos (Event-Bus pattern)

Este patrón se ocupa principalmente de los eventos y comprende cuatro (4) componentes principales: origen de eventos, oyente de eventos, canal y bus de eventos. Los orígenes publican mensajes en determinados canales del bus de eventos. Por otra parte, los oyentes se suscriben a canales en el bus. En ese sentido, los oyentes reciben notificaciones de los mensajes que se publican en los canales suscritos, y en consecuencia, envían las respuestas correspondientes. Algunos ejemplos de su utilización son:

- En el desarrollo de Apps para Android.
- Servicios de notificación.



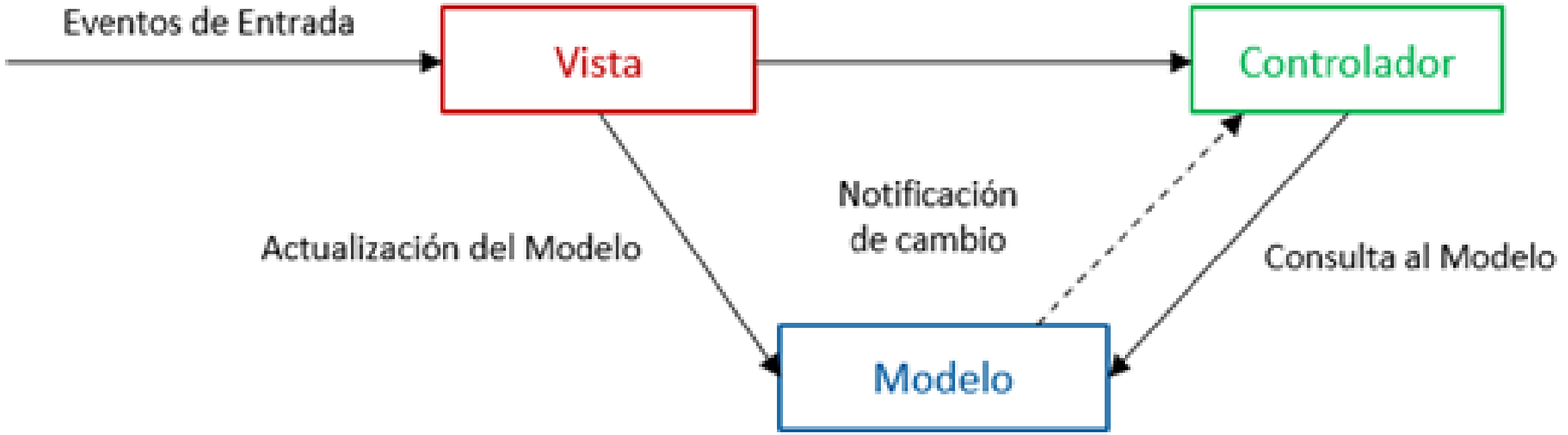
Modelo-Vista-Controlador (Model-View-Controller pattern)

Este patrón, también conocido como patrón MVC, divide una aplicación interactiva en tres (3) partes como:

- Modelo: Contiene la funcionalidad central y los datos.
- Vista: Muestra la información al usuario (se puede definir más de una vista).
- Controlador: Maneja las entradas del usuario.

Esto se hace para separar las representaciones internas de la información de las formas en que la información se presenta al usuario y es aceptada por éste. Desacopla los componentes y permite una reutilización suficiente del código. Ejemplo de su utilización:

- Desarrollo de aplicaciones de la World Wide Web en los principales lenguajes de programación.
- Frameworks web como Django y Rails.



Universidad Tecnológica del Perú

Pizarra (Blackboard pattern)

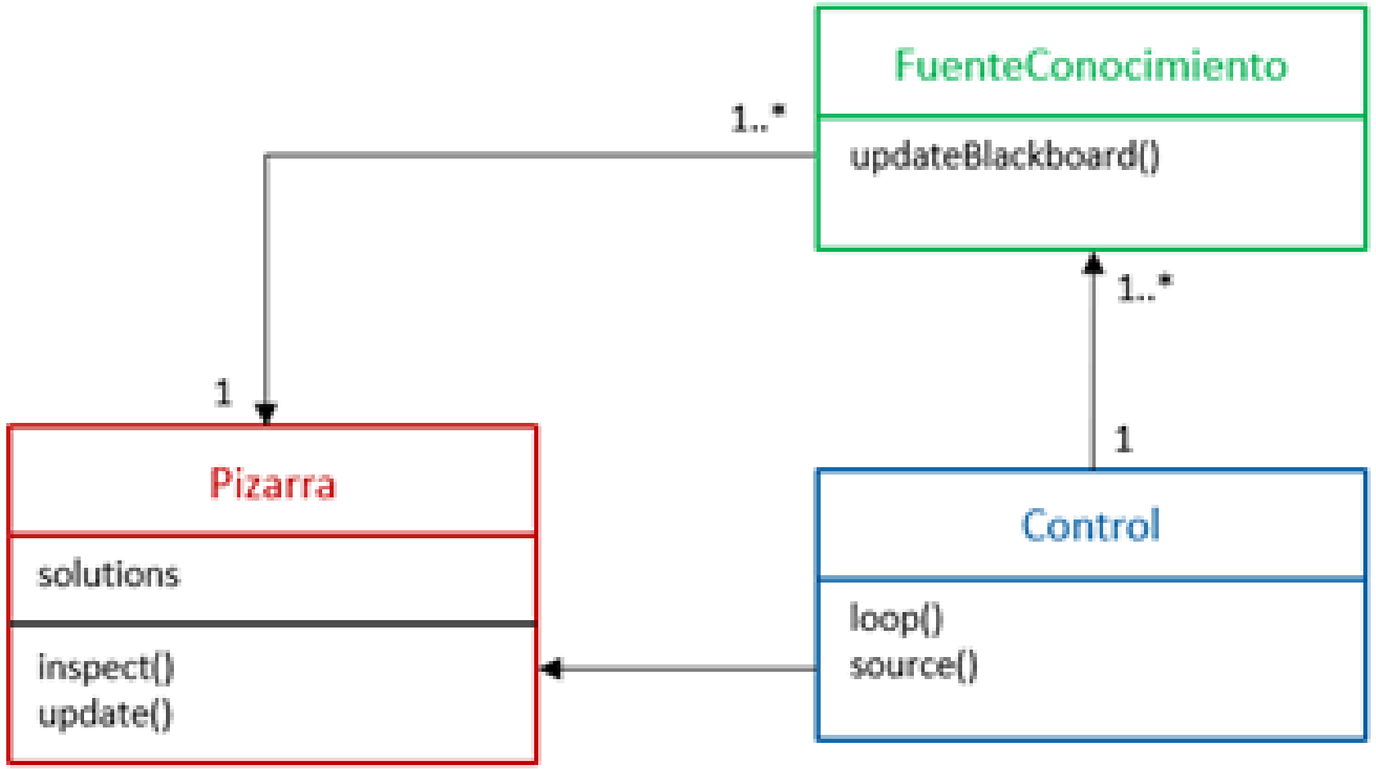
Este patrón es útil para problemas para los que no se conocen estrategias de solución deterministas. El patrón de la pizarra consta de tres (3) componentes principales.

- **Pizarra** - una memoria global estructurada que contiene objetos del espacio de soluciones.
- **Fuente de conocimiento**: módulos especializados con su propia representación.
- **Componente de control**: selecciona, configura y ejecuta los módulos.

Todos los componentes tienen acceso a la pizarra. Los componentes pueden producir nuevos objetos de datos que se añaden a la pizarra. Los componentes buscan determinados tipos de datos en la pizarra, y pueden encontrarlos mediante la concordancia de patrones con la fuente de conocimiento existente. Ejemplos de su utilización:

- Reconocimiento de voz.
- Identificación y seguimiento de vehículos.
- Identificación de estructuras proteínicas.
- Interpretación de señales de sonar.

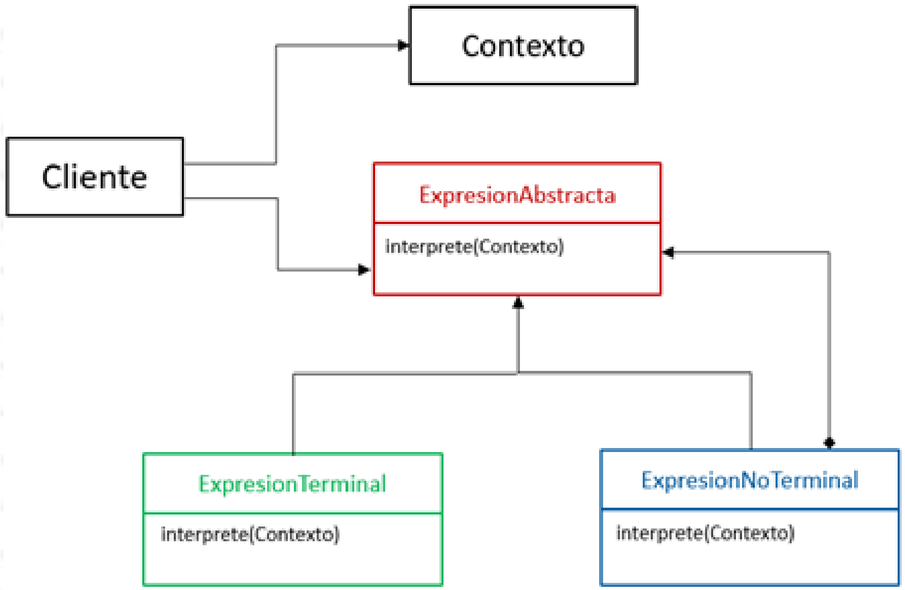
Universidad Tecnológica del Perú



Interprete (Interpreter pattern)

Este patrón se utiliza para diseñar un componente que descifra programas escritos en un lenguaje específico. Especificamente para evaluar líneas de programas, sentencias o expresiones escritas en un lenguaje de programación concreto. La idea básica es tener una clase para cada símbolo del lenguaje. Entre los ejemplos prácticos de su uso están:

- Lenguajes de consulta de bases de datos como SQL.
- Lenguajes utilizados para describir protocolos de comunicación.



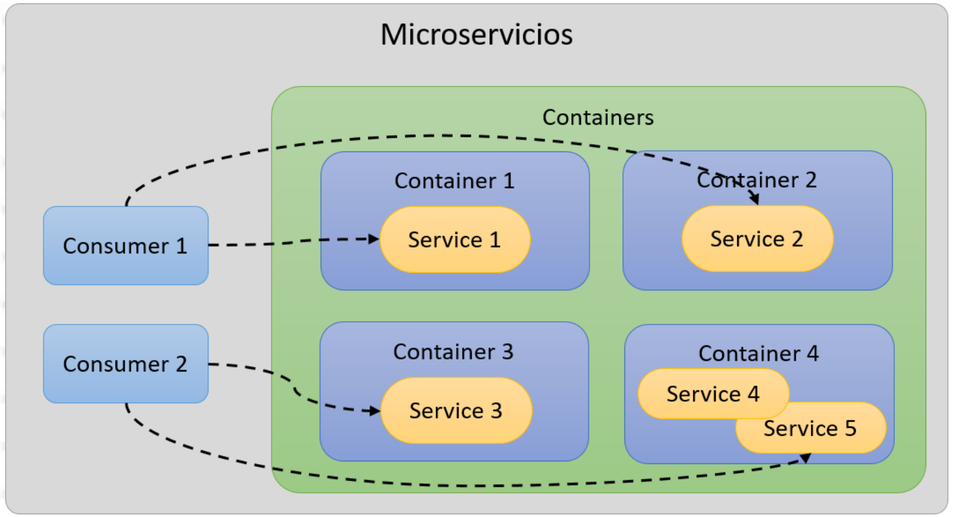
Universidad Tecnológica del Perú

Microservicios (Microservices pattern)

Es uno de los patrones más recientes y se ideó con el objetivo de crear aplicaciones más flexibles y para facilitar la migración de aplicaciones legadas y monolíticas. En la práctica, se trata de varias aplicaciones ligeras desacopladas, que atienden características concretas, qué en su conjunto, conforman un programa principal con múltiples funcionalidades.

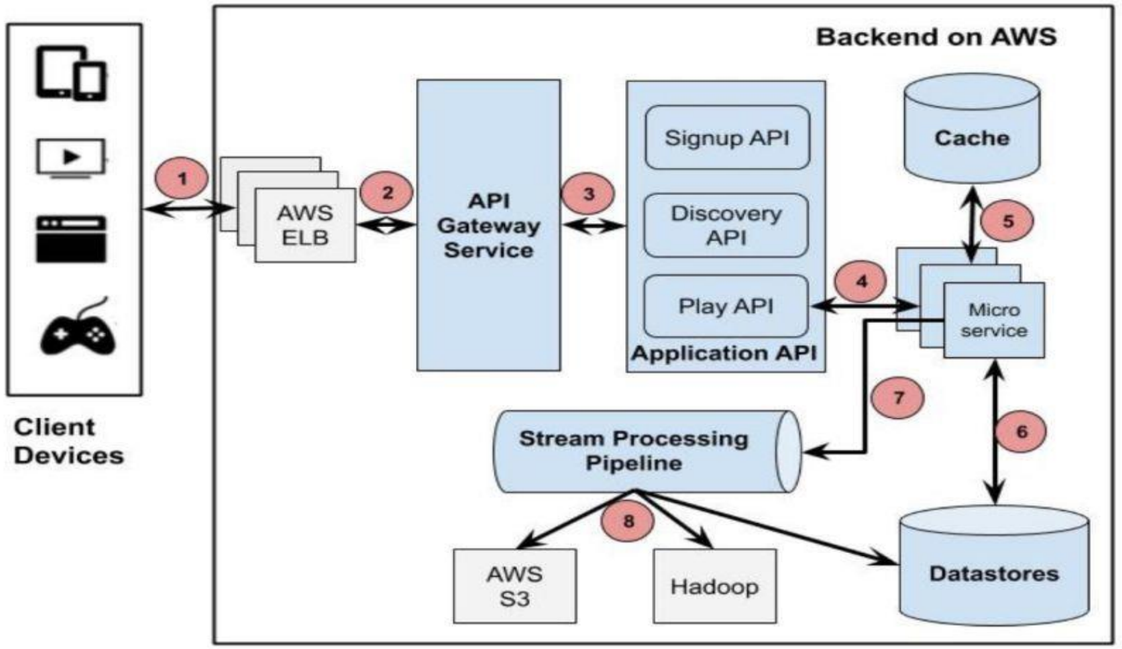
Cada componente del software puede funcionar de manera autónoma e independiente, implementando funcionalidades muy específicas de una aplicación mucho más general.

Entre los ejemplos prácticos de aplicaciones basadas en microservicios, se tienen algunas de uso cotidiano, que son incluso lideres en sus correspondientes nichos de mercado, entre estas se tienen:



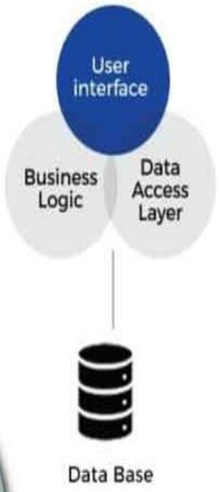
Universidad Tecnológica del Perú

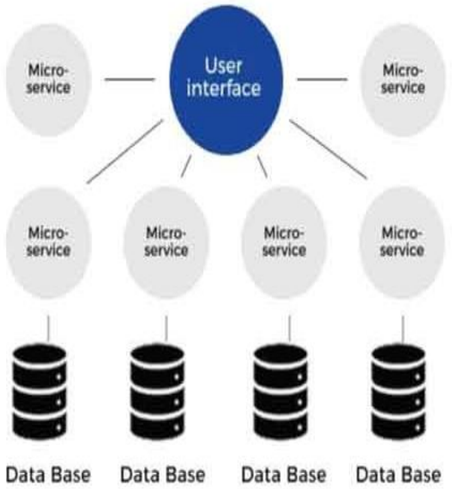
Microservices Architecture at NETFLIX



Credits: Cao Duc Nguyen

Monolithic vs Microservices

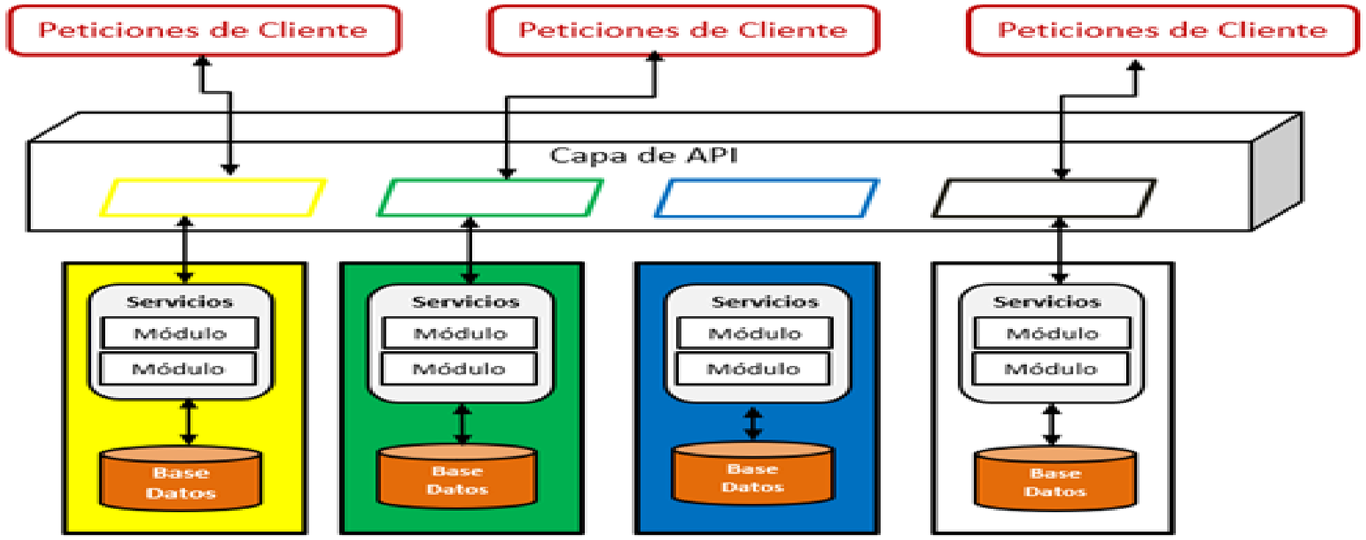




ATLASSIAN NETFLIX

Universidad Tecnológica del Perú

- Netflix: Pionera en la implementación de este patrón arquitectónico, cuenta con una plataforma basada Microservicios desde hace años. Según algunas publicaciones actualmente recibe una media de mil millones de llamadas a sus diferentes servicios y es capaz de adaptarse a más de 800 tipos de dispositivos mediante su API de streaming de vídeo, la cual realiza cinco solicitudes a diferentes servidores para no perder nunca la continuidad de la transmisión.
- Amazon: Migró hace poco más de tres años a la arquitectura de Microservicios siendo una de las primeras grandes compañías que la implementaban en producción. Se desconocen las cifras aproximadas de la cantidad de solicitudes que reciben diariamente, en tanto, no deben ser pocas en virtud al crecimiento sostenido y progresivo que ha tenido Amazon Web Services (AWS).



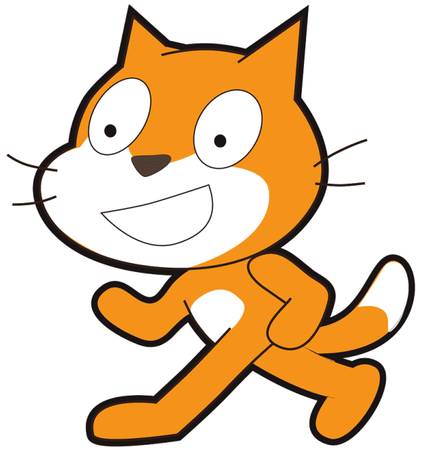
Cuadro resumen comparativo de los Patrones de Arquitectura

No hay un patrón que funcione en todas las situaciones. Al preguntarse cuál es el patrón, o patrones a utilizar en el diseño de una solución de software, aplica la clásica respuesta de siempre: "depende".

Como Arquitecto de Software corresponde sopesar los pros y los contras, para tomar una decisión bien sustentada, seleccionando aquella combinación de los mismos, que impliquen la menor deuda técnica y los menores efectores colaterales para el desarrollo, mantenimiento, despliegue e impacto de la solución en el ambiente productivo organizacional.

Universidad Tecnológica del Perú

Nombre	Ventajas	Desventajas
Capas	• Una capa inferior puede ser utilizada por diferentes capas superiores. • Las capas facilitan la estandarización, ya que podemos definir claramente los niveles. • Los cambios pueden realizarse dentro de la capa sin afectar a otras capas.	• No es aplicable en todas las situaciones. • Es posible que haya que saltarse algunas capas en determinadas situaciones.
Cliente-Servidor	• Buena para modelar un conjunto de servicios donde los clientes puedan solicitarlos y todos queden centralizados.	• Las solicitudes suelen gestionarse en subprocesos independientes en el servidor. • La comunicación entre procesos genera sobrecarga, ya que cada cliente tiene una presentación diferente.
Maestro-Esclavo	• Precisión - La ejecución de un servicio se delega en diferentes esclavos, con diferentes implementaciones.	• Los esclavos están aislados: no hay un estado compartido. • La latencia en el patrón maestro-esclavo puede ser un problema, por ejemplo en sistemas en tiempo real. • Este patrón sólo puede aplicarse a un problema que pueda descomponerse.
Filtros-Tuberías	• Realiza procesamiento concurrente. Cuando la entrada y la salida consisten en flujos, y los filtros empiezan a computar cuando reciben datos. • Fácil de añadir filtros, el sistema puede ampliarse fácilmente. • Los filtros son reutilizables, se pueden construir diferentes filtros recombinando un conjunto dado de filtros.	• La eficacia está limitada por el proceso de filtrado más lento. • Sobrecarga de transformación de datos al pasar de un filtro a otro.
Mediador	• Permite el cambio dinámico, la adición, supresión y reubicación de objetos, y hace que la distribución sea transparente para el desarrollador.	• Exige la normalización de las descripciones de los servicios.
Par-a-Par	• Apoya al procesamiento computacional descentralizado. • Gran robustez ante el fallo de cualquier nodo. • Altamente escalable en términos de recursos y potencia de cálculo.	• No hay garantías sobre la calidad del servicio, ya que los nodos cooperan voluntariamente. • La seguridad es difícil de garantizar. • El rendimiento depende del número de nodos.
Micronúcleo	• Como los componentes son desarrollados de forma por separado, es posible probarlos de forma aislada. • Muchas de las aplicaciones basadas en Micronúcleo trabajan de forma Monolítica por lo que una vez instalado algún Plug-In, todo el procesamiento se haga en una sola unidad de software. • Es posible instalar todas las características adicionales al Núcleos en tiempo en tiempo de ejecución, lo que no impacta a la aplicación.	• Las aplicaciones basadas en Micronúcleo generalmente son desarrolladas para ser ejecutadas en modo independiente, por lo que afecta su escalabilidad. • Son difíciles de desarrollar, se requiere un análisis muy elaborado para identificar hasta qué punto puede ser extendida la aplicación sin afectar la esencia de la aplicación Núcleo.
Bus de Eventos	• Se pueden añadir fácilmente nuevos editores, suscriptores y conexiones. • Eficaz para aplicaciones muy distribuidas.	• La escalabilidad puede ser un problema, ya que todos los mensajes viajan a través del mismo bus de eventos.
Modelo-Vista-Controlador	• Facilita la creación de múltiples vistas del mismo modelo, que pueden conectarse y desconectarse en tiempo de ejecución.	• Aumenta la complejidad. Puede dar lugar a muchas actualizaciones innecesarias para las acciones del usuario.
Pizarra	• Facilidad para añadir nuevas aplicaciones. • Facilita la ampliación de la estructura del espacio de datos.	• La modificación de la estructura del espacio de datos afecta a todas las aplicaciones. • Puede necesitar sincronización y control de acceso
Interprete	• Es posible un comportamiento muy dinámico. • Bueno para la programabilidad del usuario final. • Aumenta la flexibilidad, ya que es fácil sustituir e interpretar el programa.	• Por lo general, un lenguaje interpretado suele ser más lento que uno compilado, por lo que se pueden presentar problemas de rendimiento y tiempo de respuesta.
Microservicios	• Más fáciles de probar y mantener, puesto que son servicios pequeños con un solo propósito. • Fomentan el desacoplamiento de componentes, por lo que facilita el desarrollo y despliegue. • Facilita la escalabilidad, puesto que se ir incorporando capacidades a la aplicación, sin afectar su funcionamiento.	• Implica complejidad adicional de los sistemas por la dispersión que de pueden producir por distribuidos de los microservicios. • Mayor consumo de recursos por sus altos niveles de independencia. Cada microservicio tiene su propio Sistema Operativo y dependencias • Derivado del punto anterior, las aplicaciones basadas en microservicios suelen ser más caras que las monolíticas.



Universidad Tecnológica del Perú

Patrones de Diseño GoF (Gang of Four)

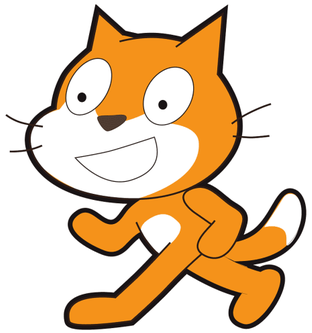
Dentro de los patrones clásicos tenemos los **GoF (Gang of Four),**estudiados por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides en su mítico libro Design Patterns se contemplan 3 tipos de patrones:

Patrones de creación: tratan de la inicialización y configuración de clases y objetos

Patrones estructurales: Tratan de desacoplar interfaz e implementación de clases y objetos

Patrones de comportamiento tratan de las interacciones dinámicas entre sociedades de clases y objetos

Y dentro de cada grupo tenemos:



Universidad Tecnológica del Perú

Patrones de creación

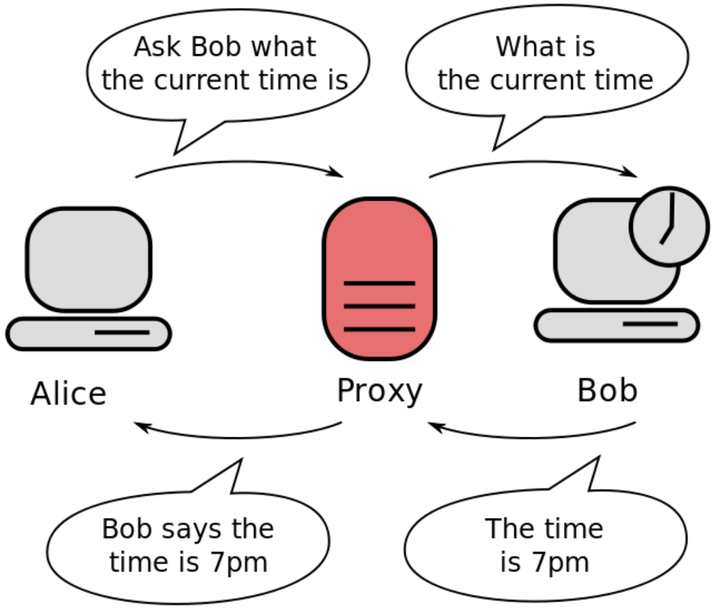
Abstract Factory. Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

Builder. Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

Factory Method. Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

Prototype. Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.

- **Singleton**. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.



Universidad Tecnológica del Perú

Patrones estructurales

Adapter. Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

Bridge. Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

Composite. Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

Decorator. Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

Facade. Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.

Flyweight. Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

- **Proxy**. Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

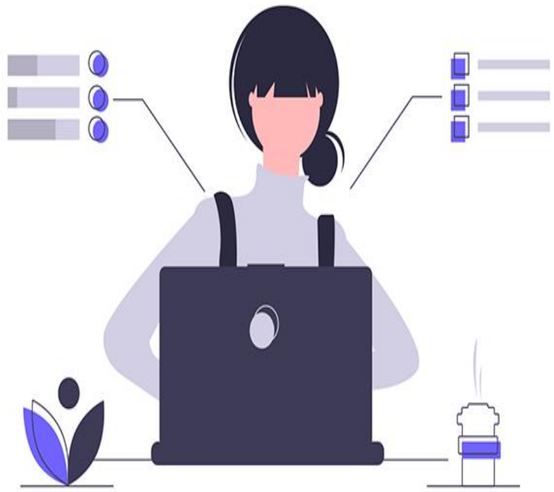
Patrones de comportamiento

Chain of Responsibility. Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

Command. Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

Interpreter. Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

Iterator. Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.



Universidad Tecnológica del Perú

Mediator. Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

Memento. Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.

Observer. Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.

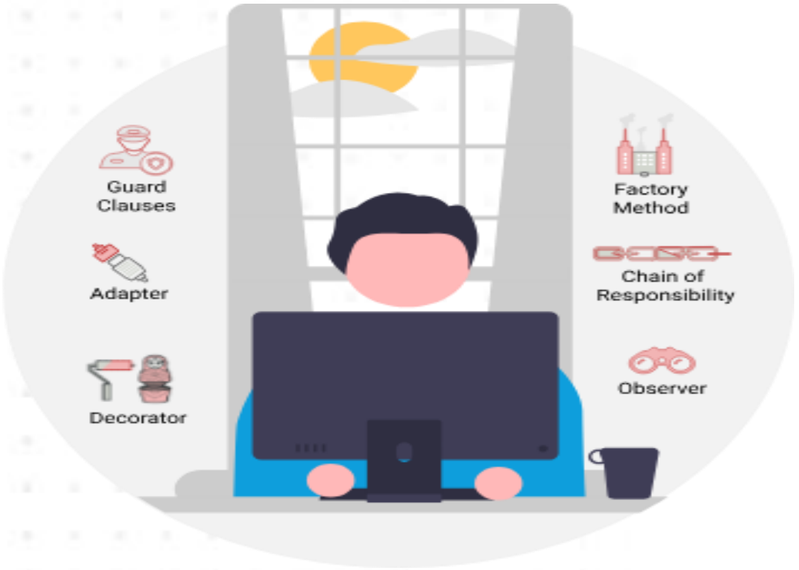
- **State**. Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.

Universidad Tecnológica del Perú

Strategy. Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

Template Method. Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

Visitor. Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.



Universidad Tecnológica del Perú

¿Consultas o dudas?

Universidad Tecnológica del Perú

Actividad

Resolver la actividad planteada en la plataforma.

Universidad Tecnológica del Perú

Cierre

¿Qué hemos aprendido hoy?

Elaboramos nuestras conclusiones sobre el tema tratado

Universidad Tecnológica del Perú

Universidad
Tecnológica
del Perú