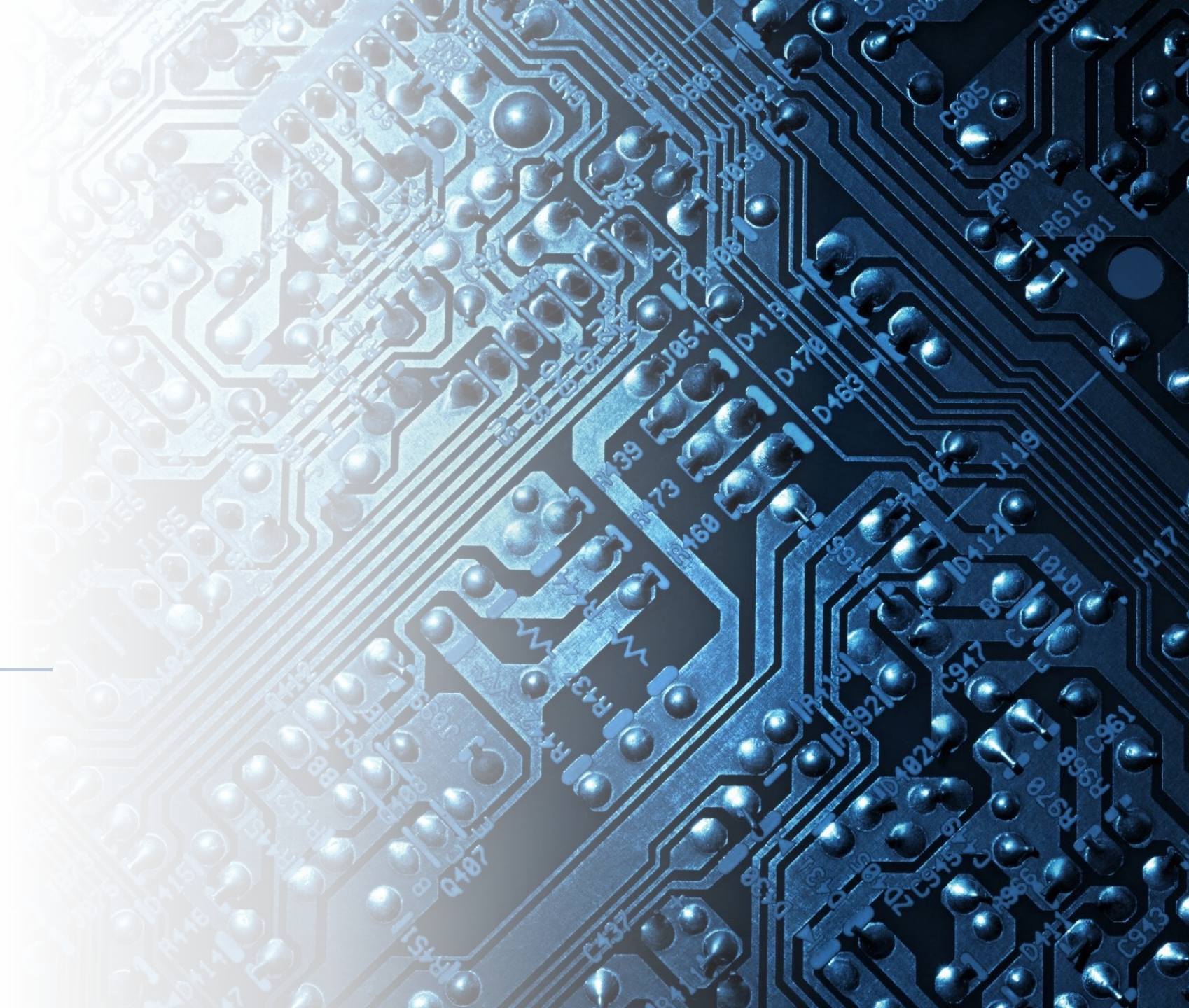
Arquitectura de Software



Arquitectura de Software

La arquitectura es la organización de un sistema descrita a través de:

- Sus componentes.
- Relación entre ellos y con el ambiente.
- La definición de los principios que guían su diseño y evolución.
- Guía la construcción de un software.

Permite a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo.

Permite cubrir todos los objetivos y restricciones de la aplicación.

La arquitectura debe determinar qué computadora tendrá asignada cada tarea.

Tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad), y servir como guía en el desarrollo.

Provee un vehículo para la comunicación entre los distintos stakeholders.

Diseño de la Arquitectura de Software

Se inicia cuando los requisitos funcionales y no funcionales están claramente definidos.

Generalmente se expresa en partes que muestran vistas de un aspecto de la solución: cómo se comparten los datos, cómo son distribuidos, cómo se espera que sean desarrollados los módulos, etc.

Existen muchos estilos arquitectónicos.

Cada estilo es más apropiado para un escenario y para lograr ciertos requisitos de calidad.

Ciclo de desarrollo de la Arquitectura

Requerimientos

Se enfoca en la captura, documentación y priorización de requerimientos que influencian la arquitectura

Diseño

Se definen las estructuras que componen la arquitectura

Documentación

Es necesario poder comunicarlo a otros involucrados dentro del desarrollo lo que va ocurriendo.

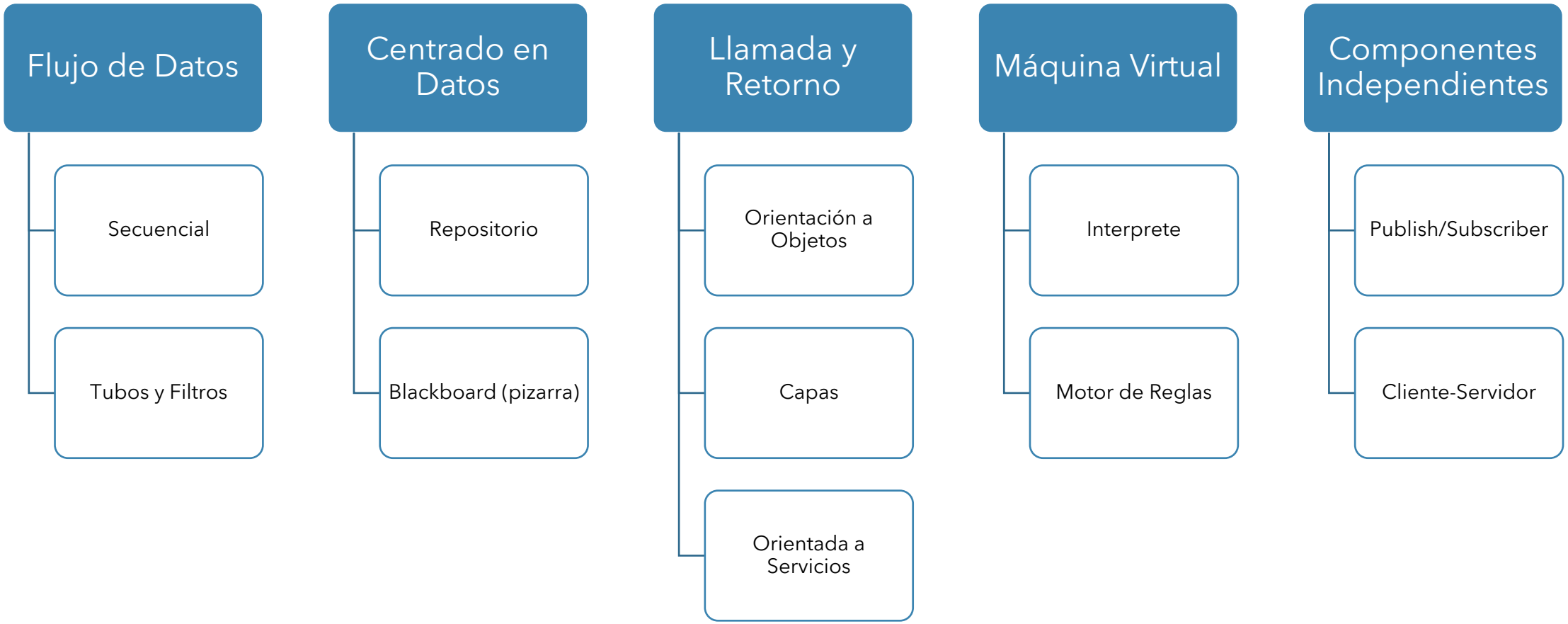
Evaluación

Es conveniente evaluar el diseño una vez que este ha sido documentado con el fin de identificar posibles problemas y riesgos

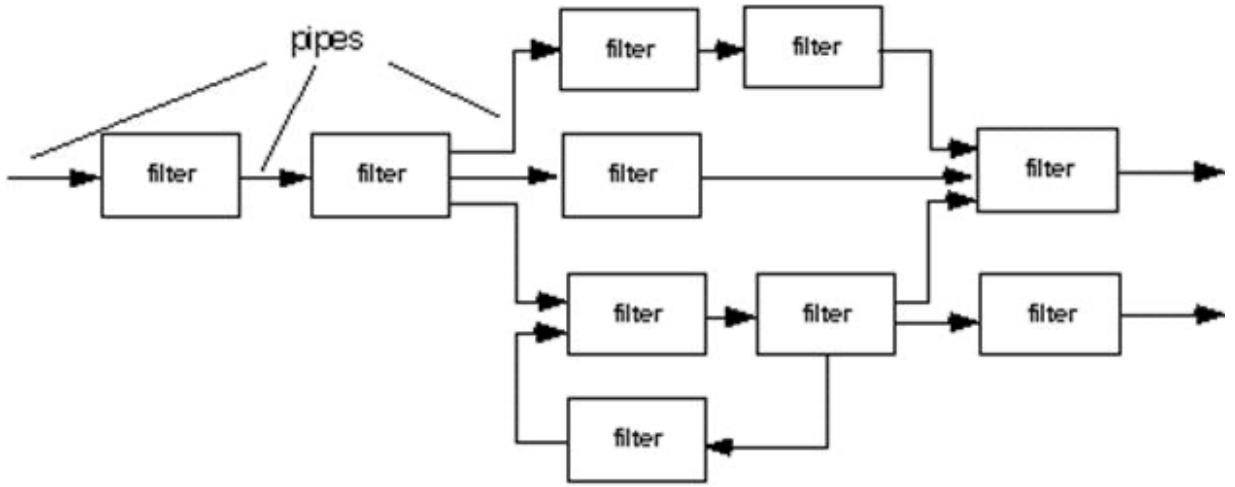
Arquitectura vs Diseño

	Arquitectura	Diseño
Nivel de Abstracción	Alto nivel	Bajo nivel. Enfoque específico en detalles
Entregables	Planear: subsistemas, interfaces con sistemas externos, servicios horizontales, frameworks, componentes reutilizables, prototipo arquitectónico.	Diseño detallado de clases, componentes, etc. Especificaciones de codificación
Áreas de Enfoque	Selección de tecnologías, Requerimientos no funcionales, Manejo de riesgos.	Requerimientos funcionales

Estilos Arquitectónicos



Flujo de Datos

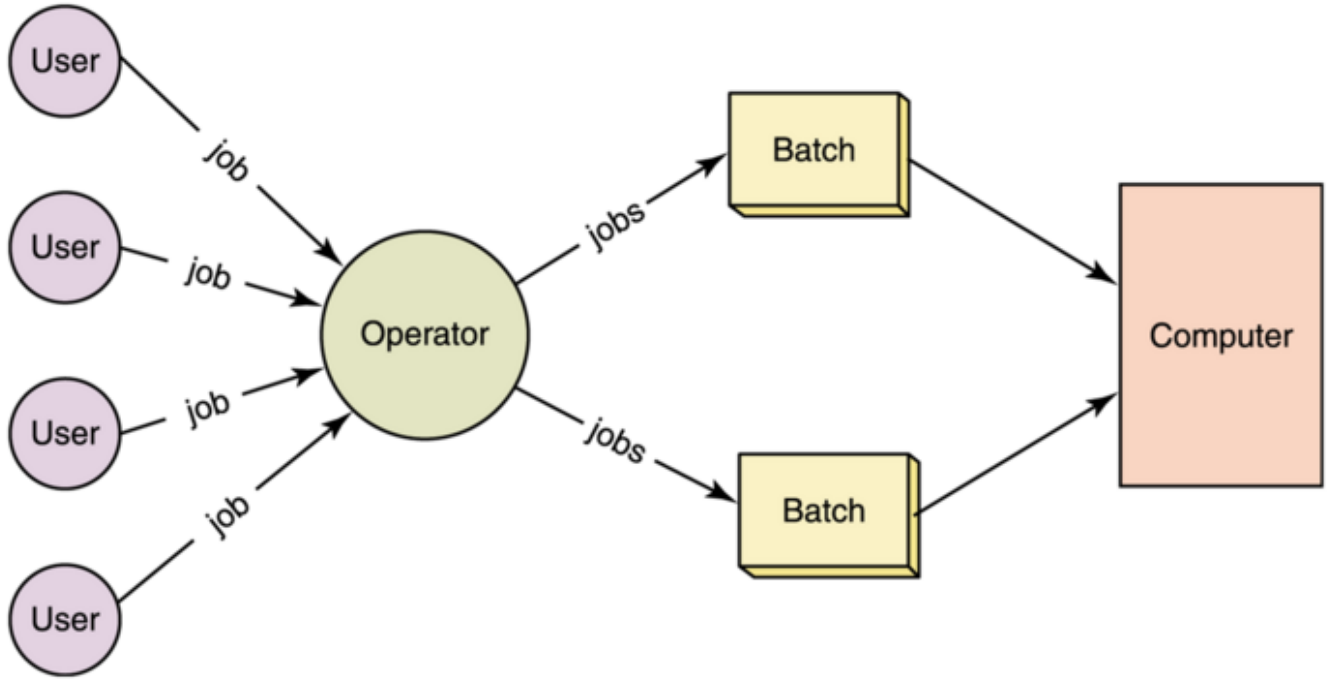


(a) pipes and filters

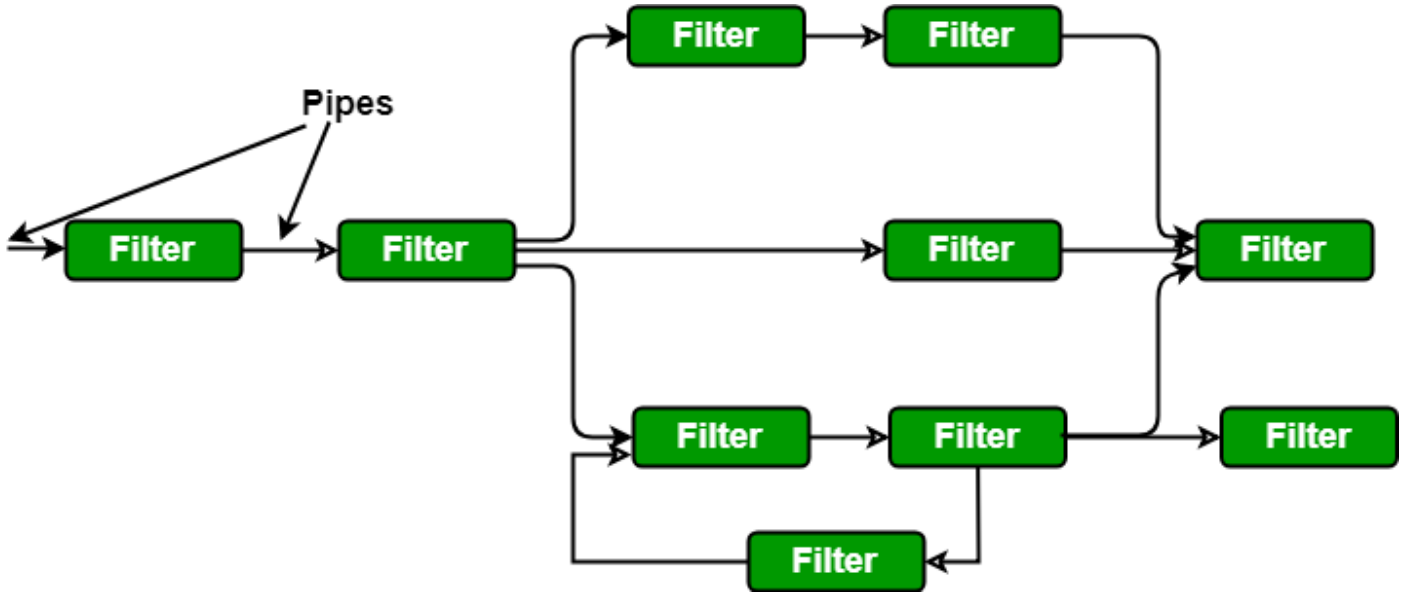
(b) batch sequential

Secuenciales

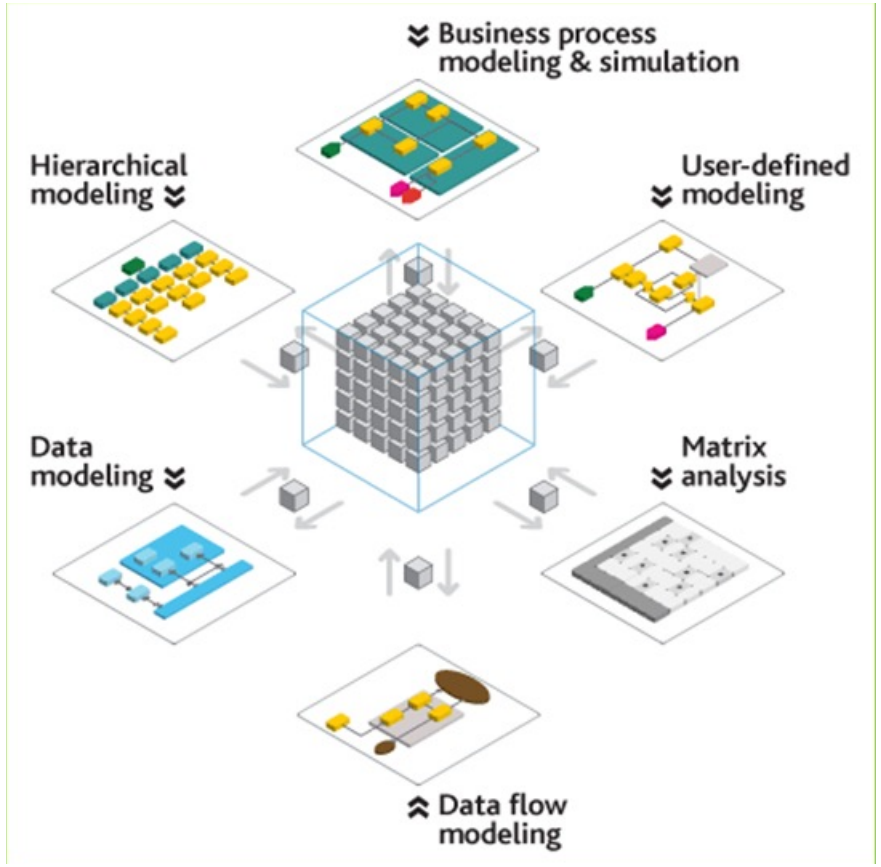
- Los componentes son programas independientes
- El supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.
- Los datos deben pasarse de un programa al siguiente.
- Se puede considerar como el abuelo de los estilos arquitectónicos

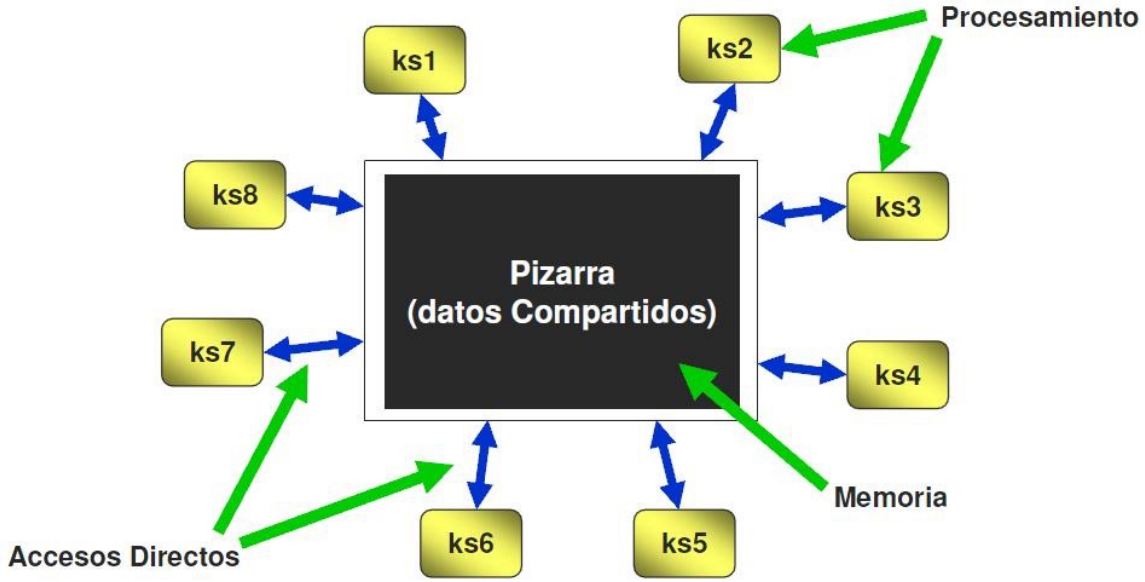


Tuberias y Filtros



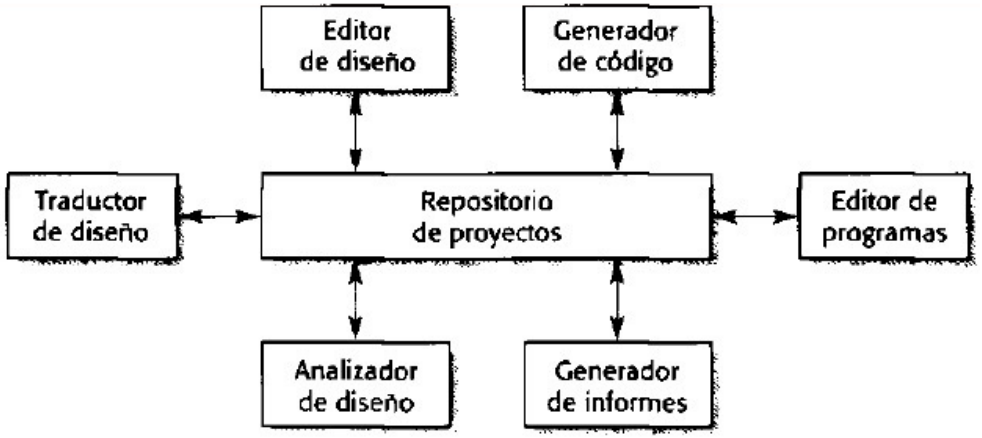
- El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.
- Consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.
- Esta arquitectura es muy común en el desarrollo de programas para el intérprete de comandos, ya que se pueden concatenar comandos fácilmente con tuberías (pipe).





Centrado en Datos

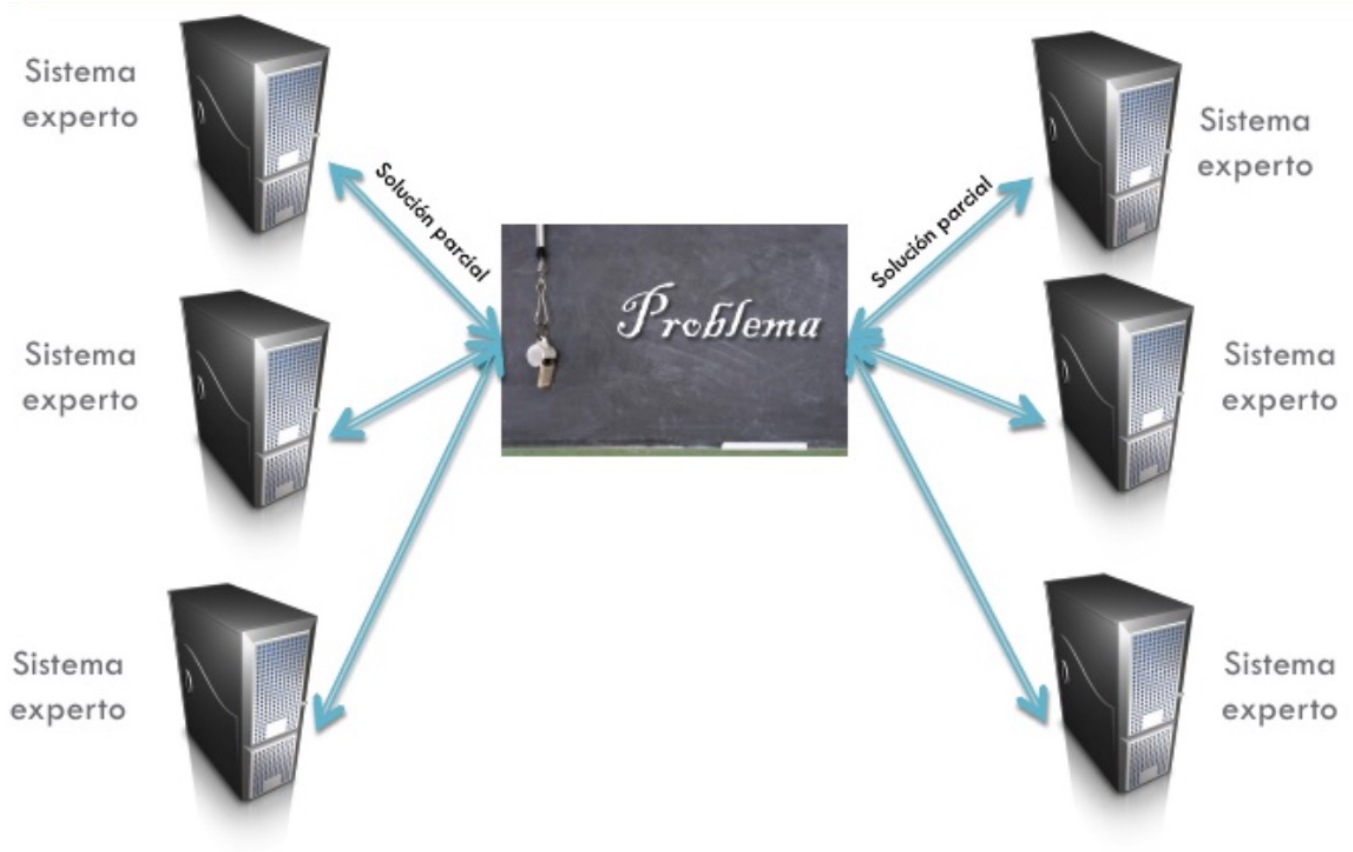
Repositorio



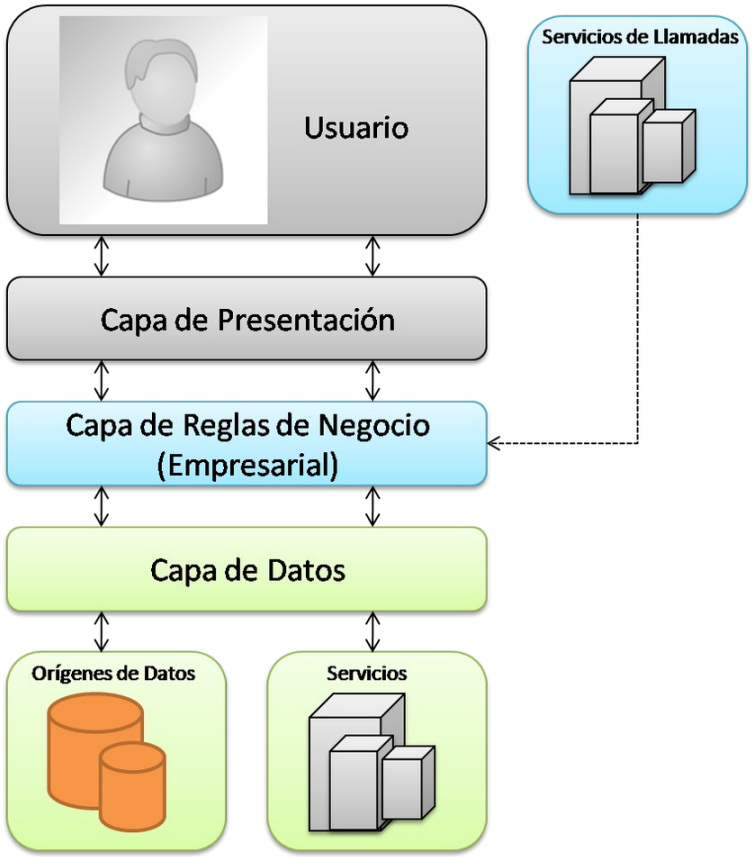
- Se basa en un medio de almacenamiento pasivo o repositorio que es compartido.
- Los subsistemas se comunican con el repositorio para actualizar los datos.
- El repositorio se usa de dos formas:
    1. Los datos compartidos se colocan en un repositorio o base de datos que puede ser accedida por todos los subsistemas. Se llama alternativa centralizada.
    2. Cada subsistema mantiene su propia base de datos y pasa datos explícitos a otros subsistemas. Se llama alternativa distribuída.

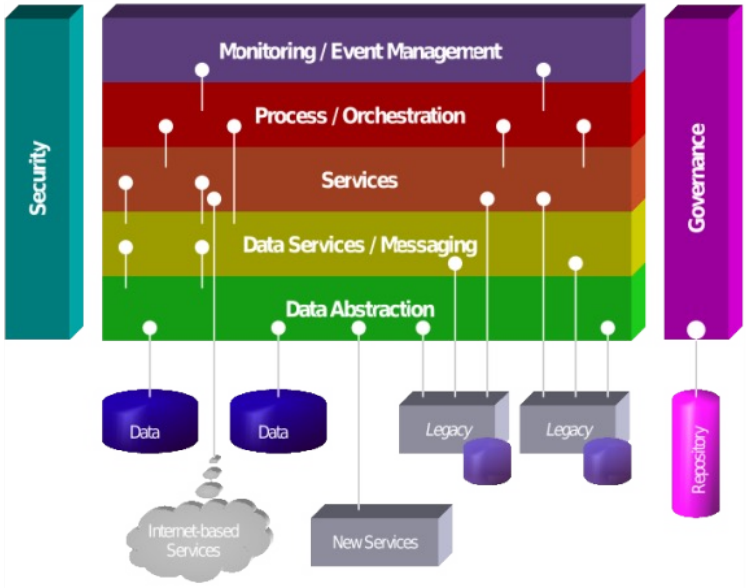
Pizarra

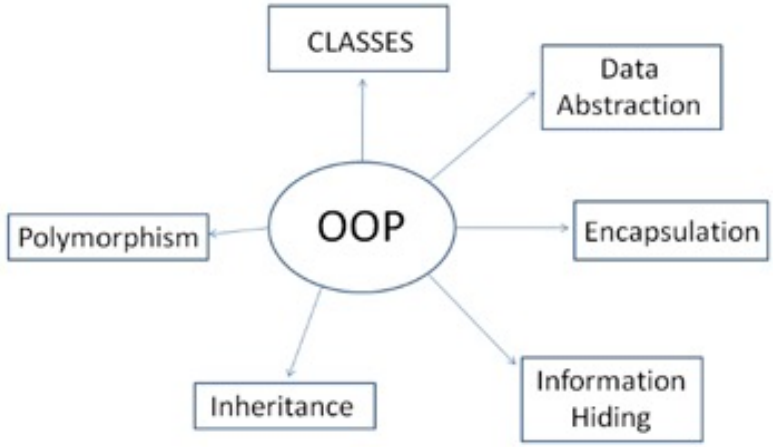
- En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.
- Un sistema de pizarra se implementa para resolver problemas en los cuales las entidades individuales se manifiestan incapaces de aproximarse a una solución, o para los que no existe una solución analítica, o para los que sí existe pero es inviable por la dimensión del espacio de búsqueda.



Llamada y Retorno



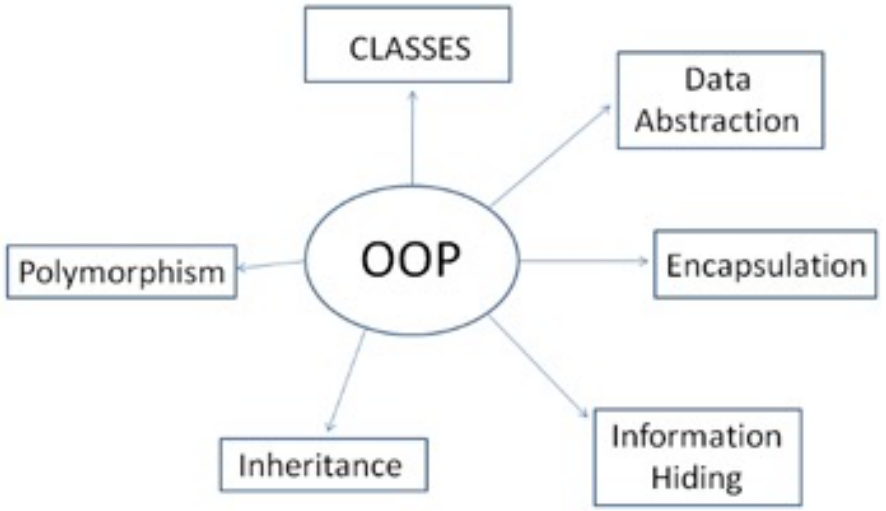


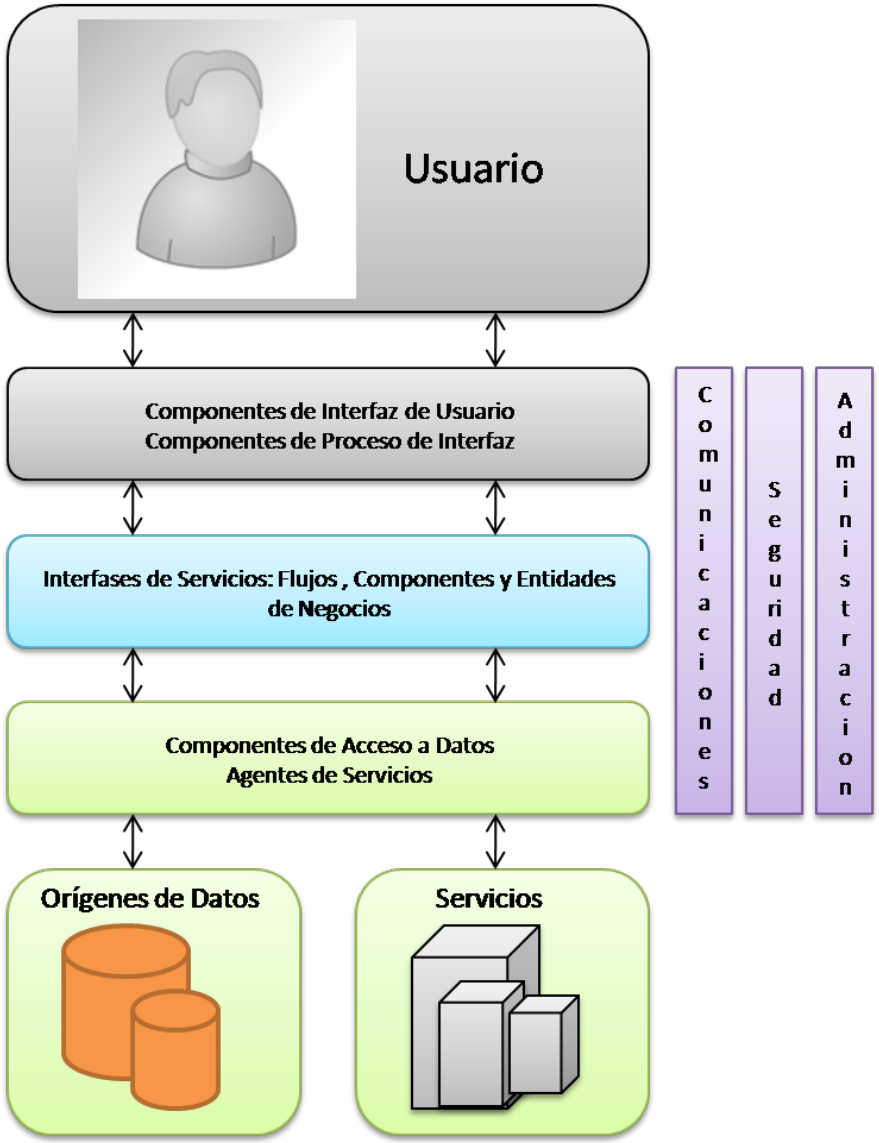


Orientada a Objetos

- Conjunto de objetos que cooperan entre sí en lugar de cómo un conjunto de procedimientos.

- Los objetos son discretos, independientes y poco acoplados, se comunican mediante interfaces y permiten enviar y recibir peticiones.

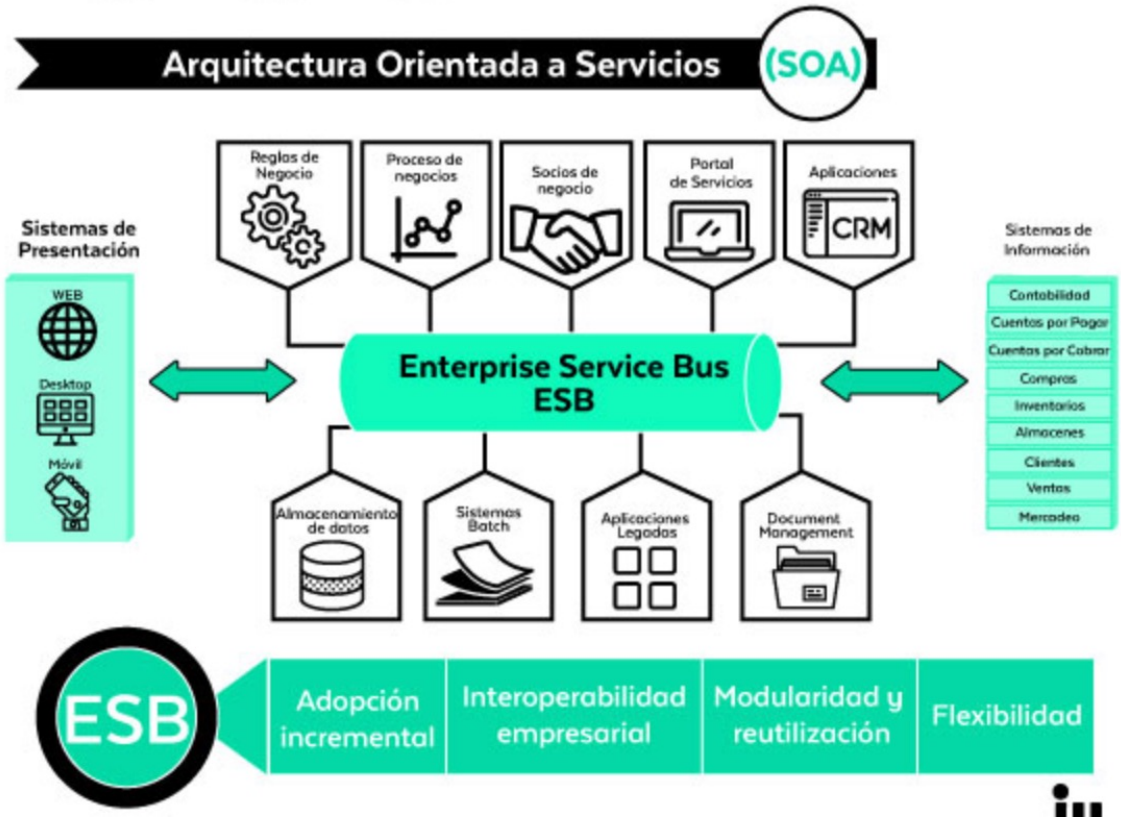




Capas

- Se caracteriza por la distribución jerárquica de los roles y las responsabilidades de las clases, para proporcionar una división efectiva de los problemas a resolver.
  - Los roles indican el tipo y forma de interacción con otras capas y las responsabilidades la funcionalidad que implementan.
  - Cada capa provee servicios a sus capas vecinas.
  - Los conectores son definidos por los protocolos que determinan cómo interactúan las capas.
  - Restricciones topológicas incluyen limitación de interacciones a capas adyacentes.
  - Cada capa sucesiva es construida basada en su antecesor.

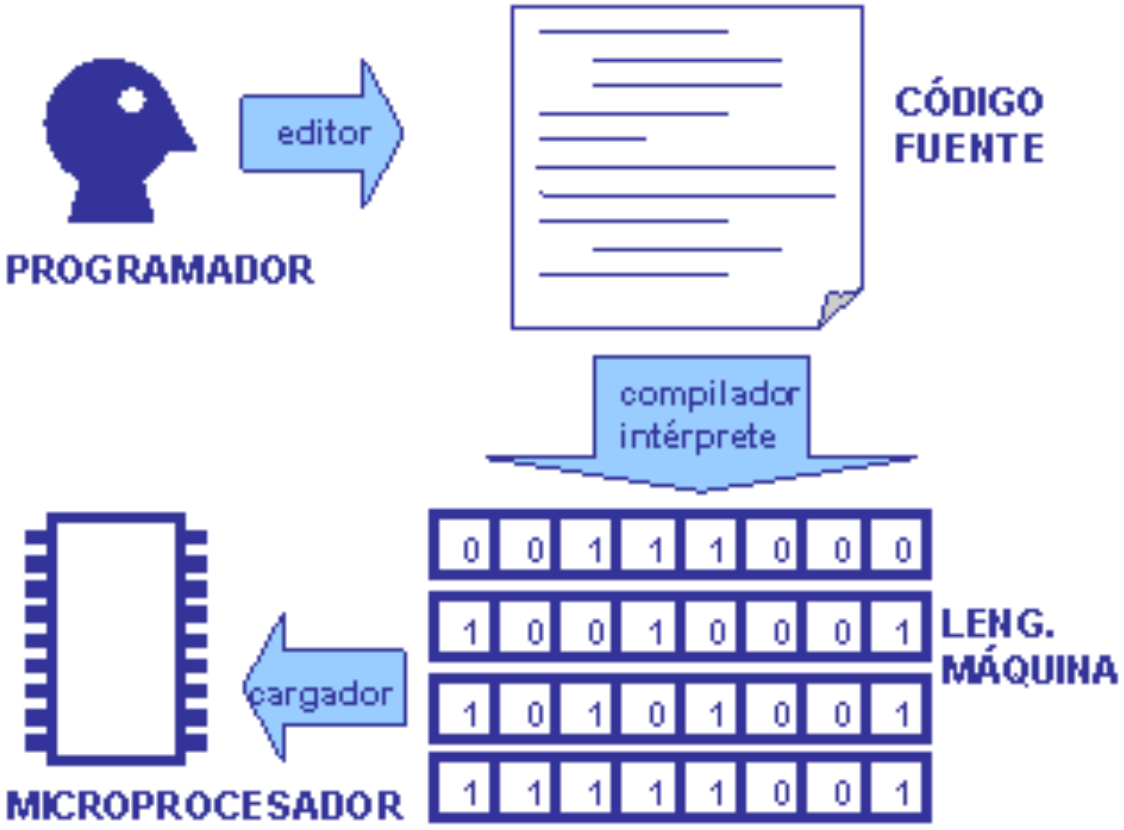
Orientada a Servicios

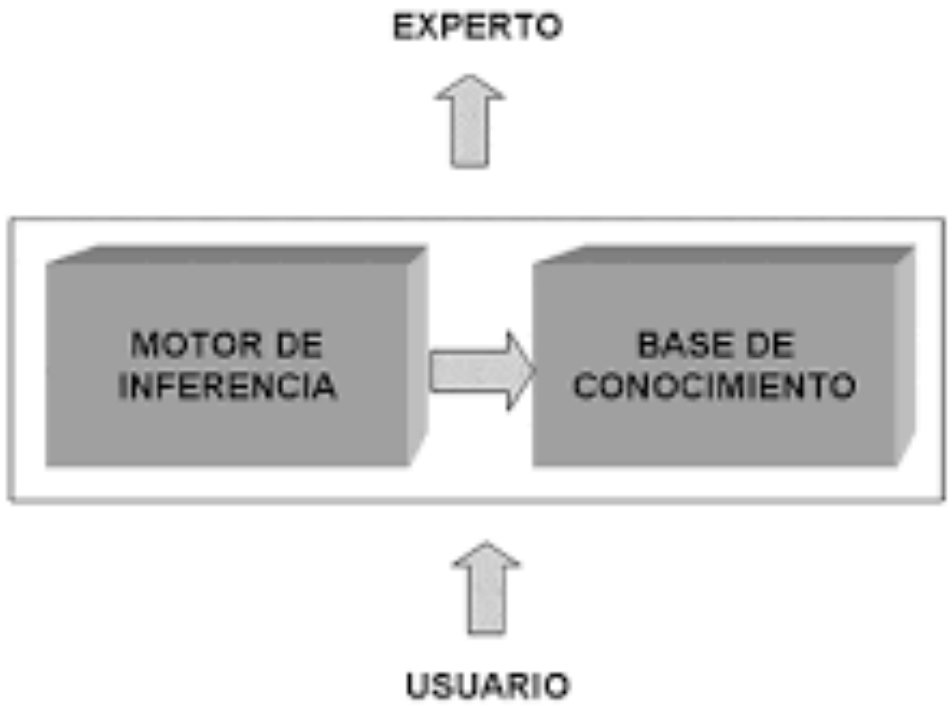


- Es un marco de trabajo que permite diseñar la integración de aplicaciones que incluyen sistemas legados con herramientas modernas en función de los procesos de negocio y los resultados que desean obtener.

- El uso de esta arquitectura reduce costos en la implemententación, mejora los servicios que recibe el cliente y logra que los componentes del procesos se integren y funcionen de una manera más eficiente.

- La gestión de SOA define las políticas, prácticas y soluciones del ciclo de vida de los aplicativos, lo cual permite que las herramientas y procesos funcionen acorde a los estándares de la compañía.

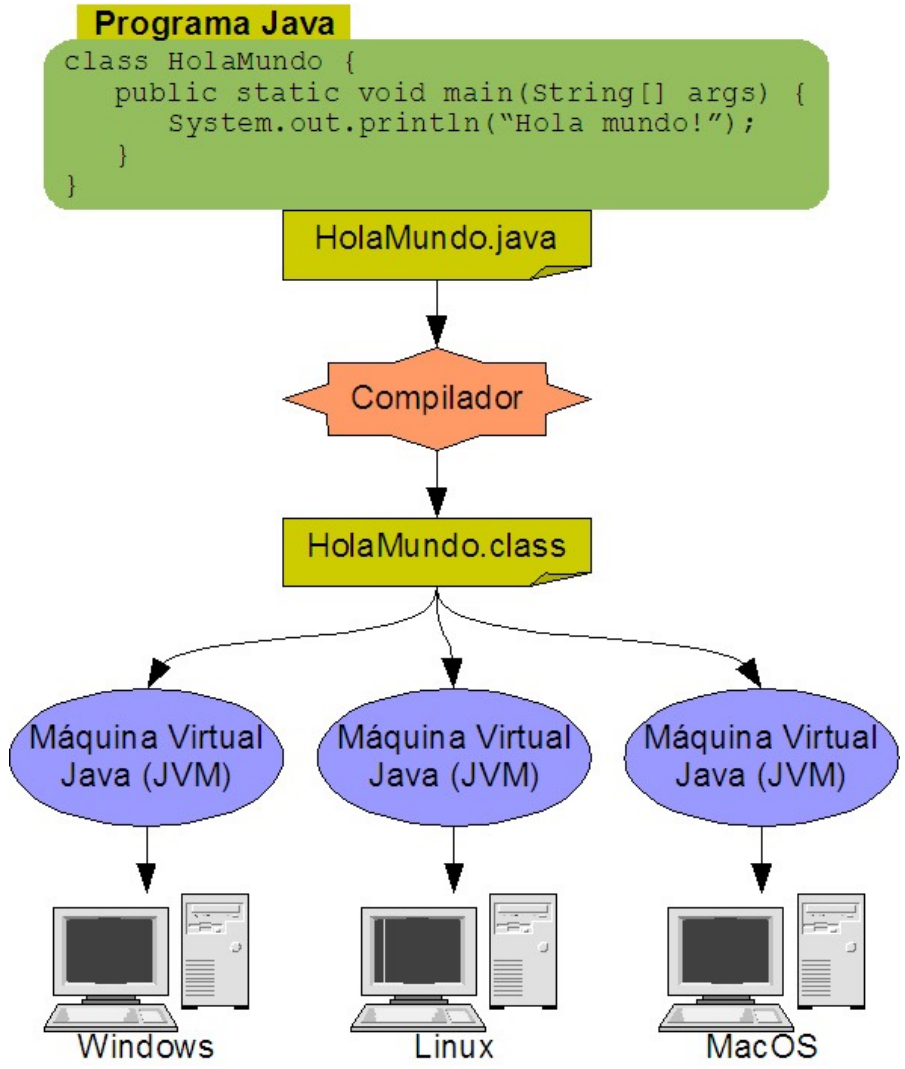




Maquina Virtual

Interprete

- Se puede decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución (o registro de activación). La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución.
- De este modo, un intérprete posee por lo general cuatro componentes:
  - una máquina de interpretación que lleva a cabo la tarea
  - una memoria que contiene el seudo-código a interpretar
  - una representación del estado de control de la máquina de interpretación
  - una representación del estado actual del programa que se simula.

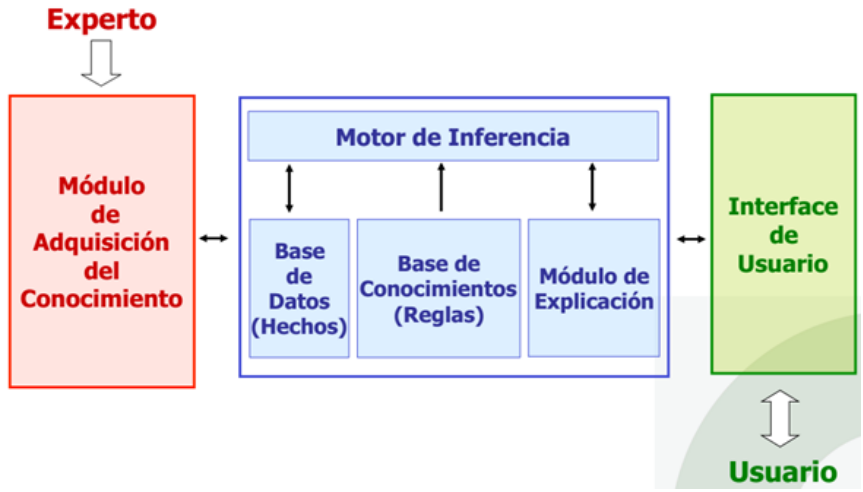


Motor de Regla

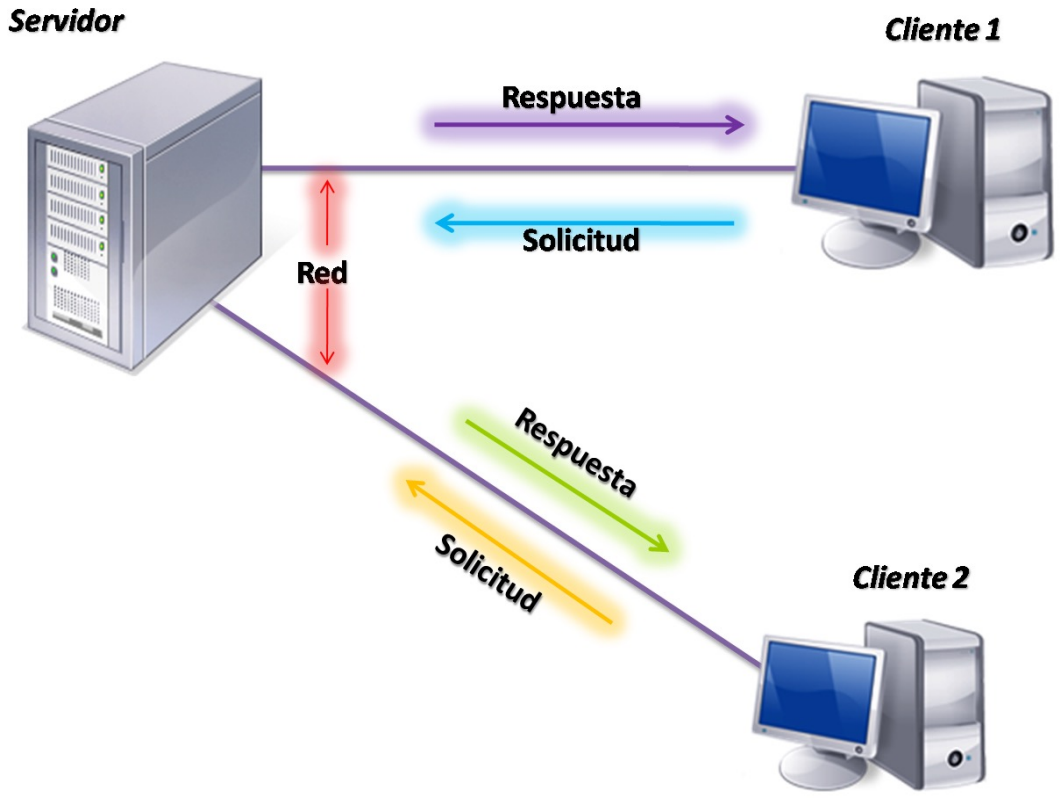
- Los sistemas basados en reglas trabajan mediante la aplicación de reglas, comparación de resultados y aplicación de las nuevas reglas basadas en la situación modificada.
- Una regla es solo una parte del conocimiento con el cual se soluciona el problema. Se almacenan con el siguiente formato:

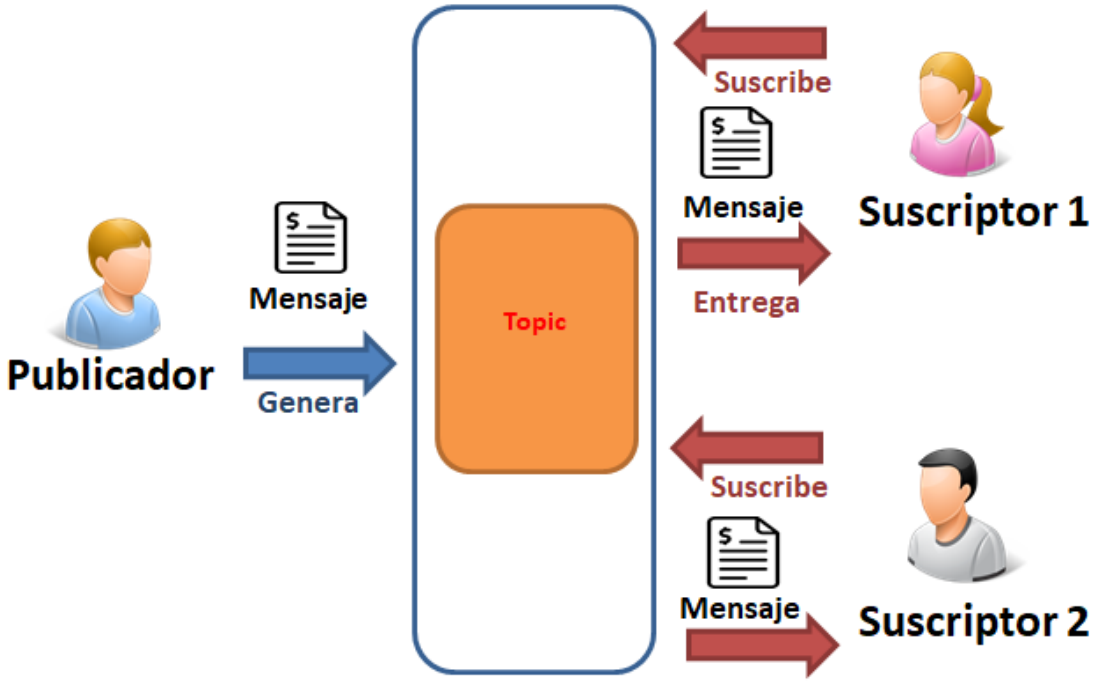
  Hipótesis (Antecedente) -> Conclusión (Consecuente)

  - No son deducciones lógicas, sino más bien el conocimiento adquirido por un experto.
  - Se les puede asignar una prioridad para tener ordenadas las reglas de acuerdo a su importancia de aplicación



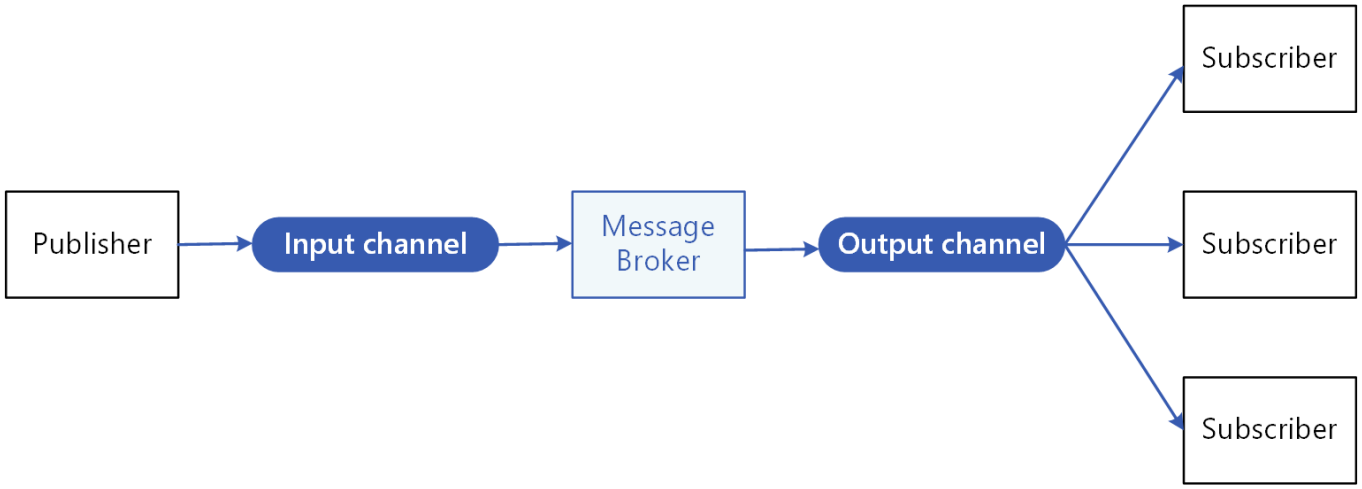
Componentes Independientes



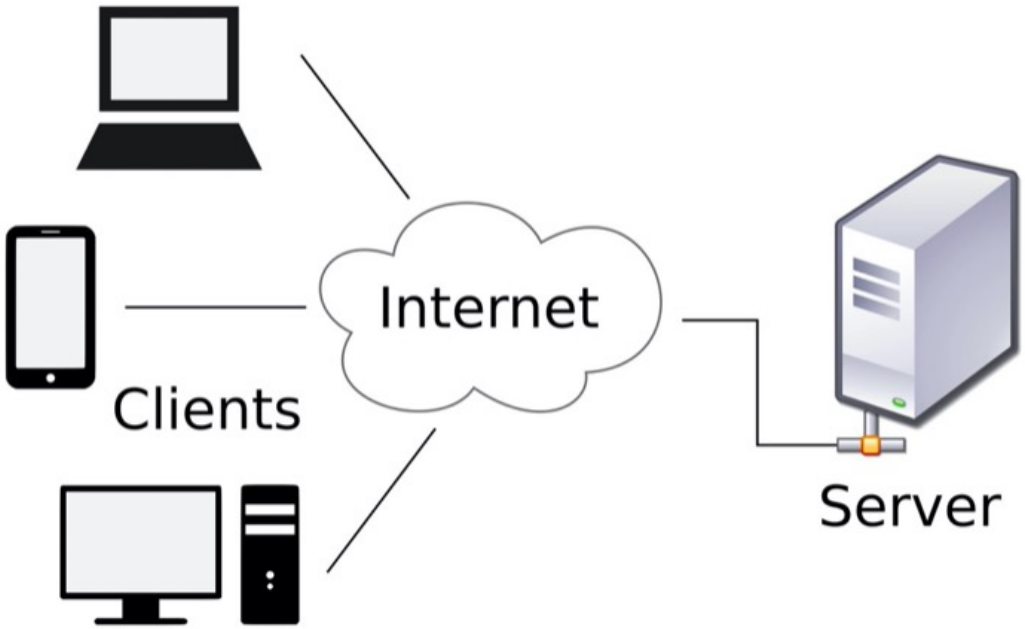


Publish/Subscriber

- Permite que una aplicación anuncie eventos de forma asincrónica a varios consumidores interesados, sin necesidad de emparejar los remitentes con los receptores.
- La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos.
- Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento.

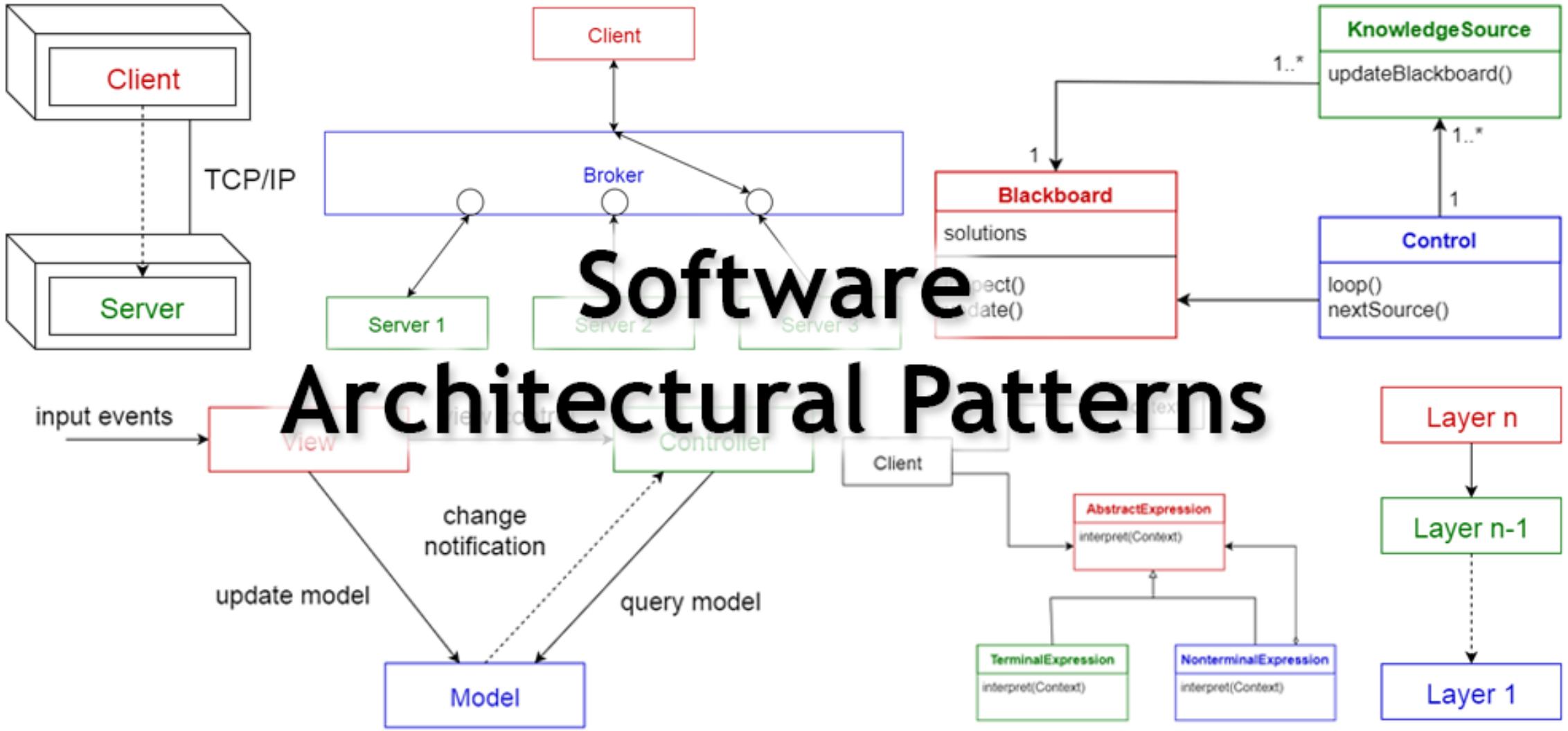


Cliente-Servidor



- Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.
- Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.
- Algunos ejemplos de aplicaciones computacionales que usen el modelo cliente-servidor son el Correo electrónico, un Servidor de impresión.

Software Architectural Patterns



Patrones Arquitectónicos

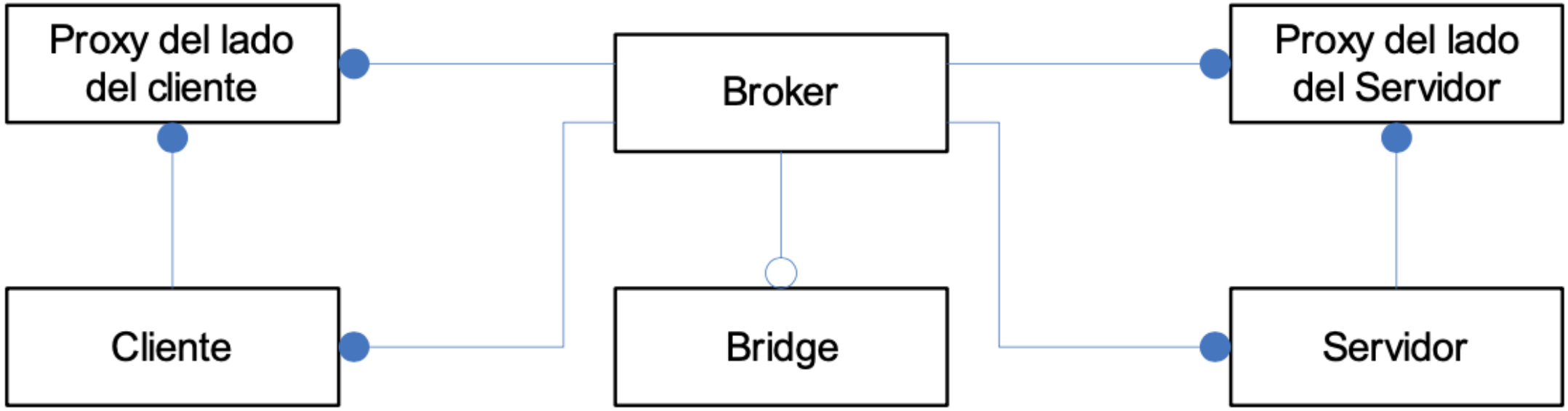
- Los patrones arquitectónicos son plantillas que describen los principios estructurales globales que construyen las distintas Arquitecturas de Software viables.
- Plantean una organización estructural fundamental para un sistema de software, expresando un conjunto de subsistemas predefinidos, especificando responsabilidades y organizando las relaciones entre ellos.
- La selección de un patrón arquitectónico es además una decisión fundamental de diseño cuando se desarrolla un sistema de software.

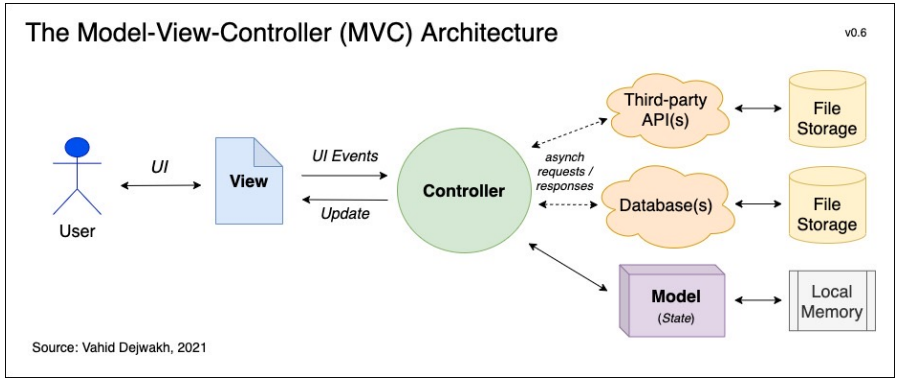
Tipos de Patrones Arquitenctónicos

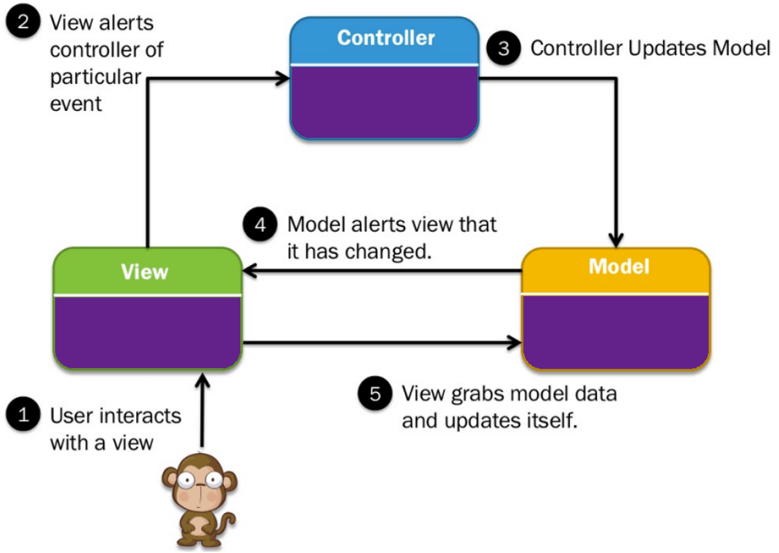
Sistemas Distribuidos	Broker
Sistemas Interactivos	Modelo-Vista-Controlador (MVC) Presentation-Abstraction-Control (PAC)
Sistemas Adaptables	Microkemel Reflection

Broker

- Puede usarse para estructurar sistemas de software distribuidos con componentes desacoplados que interactúan por invocaciones de servicios remotos.
- Un componente Broker es responsable de coordinar la comunicación, remitir las demandas, así como transmitir resultados y excepciones.







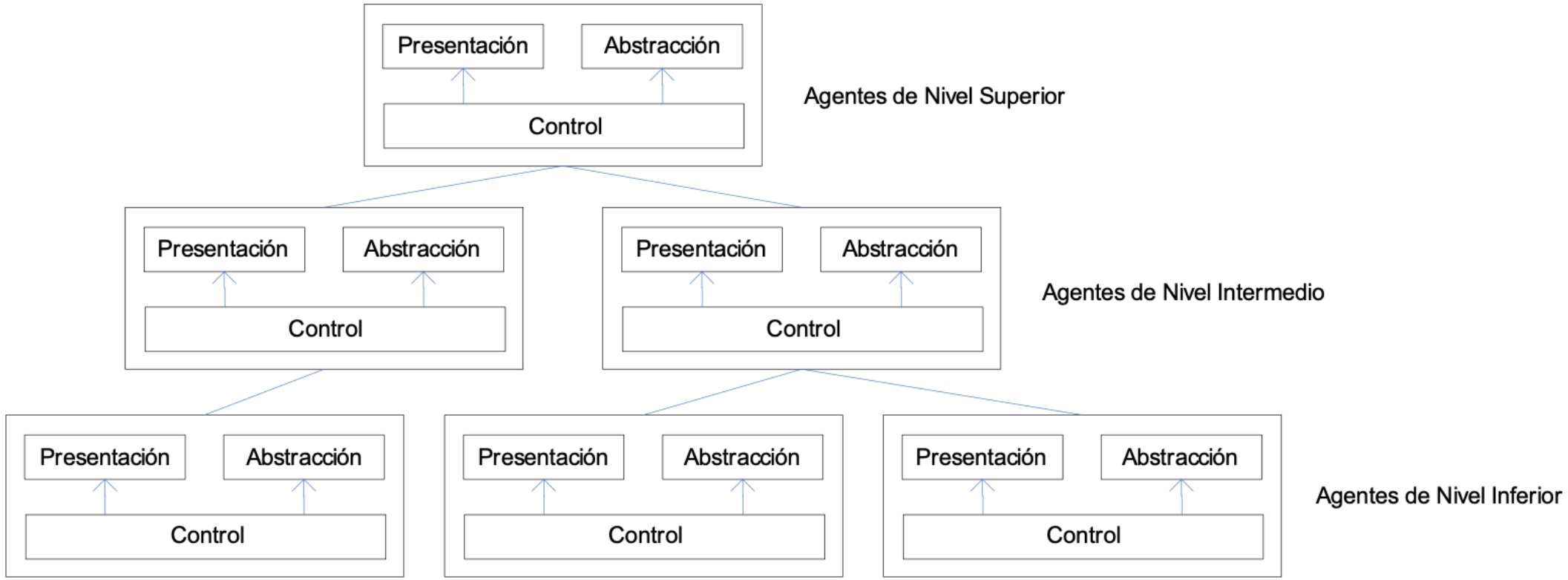
Modelo-Vista-Controlador

- Divide una aplicación interactiva en tres componentes:
  - El Modelo contiene la funcionalidad central y los datos
  - Las Vistas despliegan la información al usuario
  - Los Controladores manejan la entrada del usuario.
- Un mecanismo de propagación de cambio asegura la consistencia entre la interfaz del usuario y el modelo.
- El Model-View-Controller (MVC) divide una aplicación interactiva en tres áreas: procesamiento, entrada y salida.

Presentación-Abstracción-Control

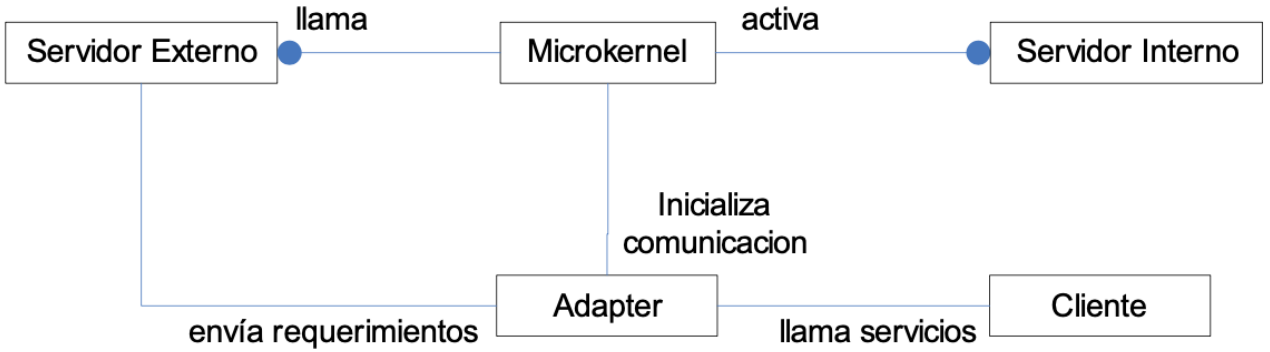
- Define una estructura para los sistemas de software interactivos en forma de jerarquía de agentes cooperantes.
- Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y tiene tres componentes: presentación, abstracción, y control.
- Esta subdivisión separa los aspectos de interacción usuario-computadora del agente central funcional y su comunicación con otros agentes.
- Cada componente de un agente PAC tiene una tarea específica.
  - El componente presentación proporciona la conducta visible del agente.
  - El componente abstracción mantiene el modelo de datos que fundamenta al agente y proporciona la funcionalidad para operar sobre esos datos.
  - El componente control conecta los componentes presentación y abstracción, y proporciona la funcionalidad de comunicación con otros agentes.

Presentación-Abstracción-Control



Microkemel

- Se aplica a los sistemas de software que deben adaptarse a requisitos cambiantes del sistema.
- Separa la funcionalidad central mínima de la funcionalidad extendida y las partes específicas de los clientes.
- El microkernel también sirve como socket para conectar extensiones y coordinar sus colaboraciones.



Reflexión

Provee un mecanismo para estructura cambiante y conducta dinámica de sistemas de software.

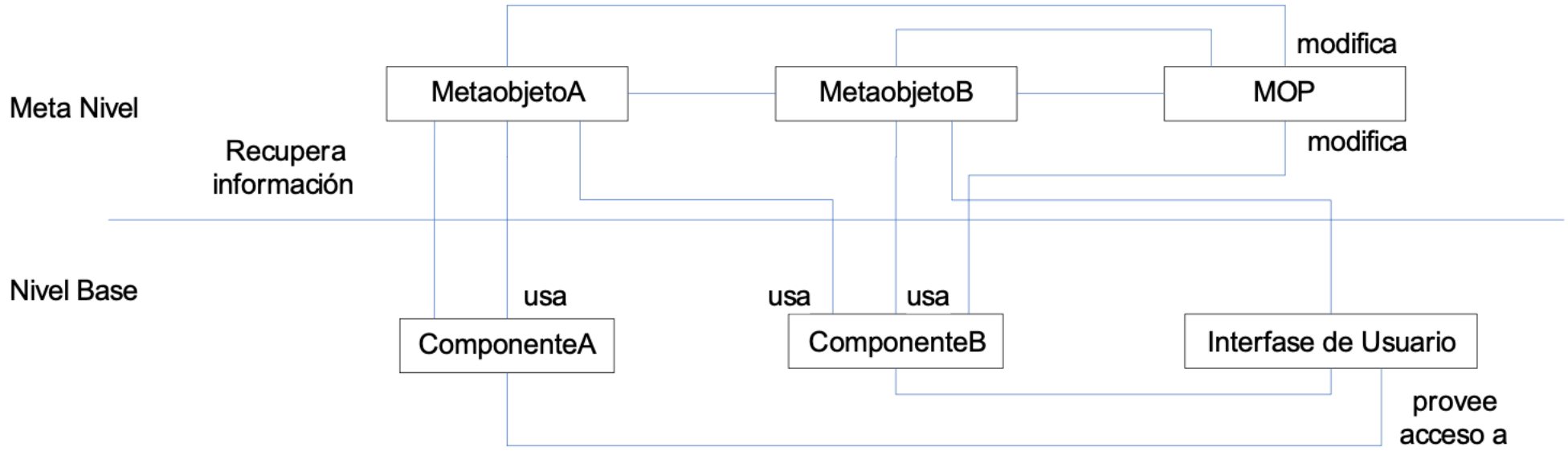
Soporta la modificación de aspectos fundamentales como los tipos de estructuras y mecanismos de llamada a función.

En este patrón, una aplicación es dividida en dos partes

- un meta nivel que provee información sobre las propiedades del sistema seleccionadas
- un nivel base que incluye la lógica de la aplicación

La implementación se construye en el meta nivel. Los cambios en la información contenida en el meta nivel afectan la conducta del nivel base subsiguiente.

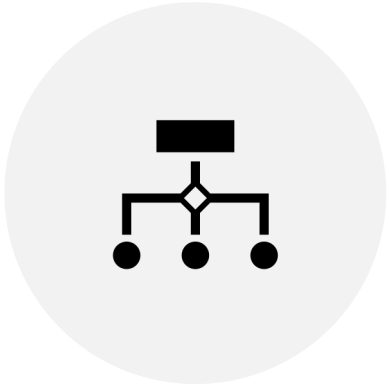
Reflexión


Meta Nivel
Recupera información
Nivel Base
MetaobjetoA
MetaobjetoB
MOP
modifica
modifica
usa
usa
usa
ComponenteA
ComponenteB
Interfase de Usuario
provee acceso a

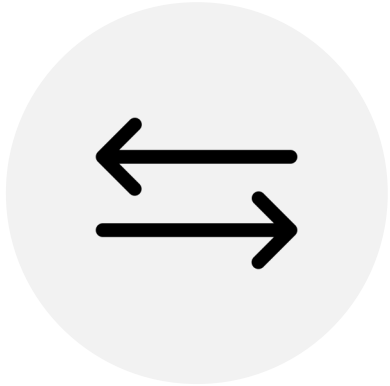
Patrones de Diseño

- Es posible que a lo largo de varios diseños de aplicaciones hayan **problemas que se repitan** o que responden a un cierto **patrón**.
- Es por esto, que los patrones de diseño dan solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).
- Trabajar con patrones es como jugar con un Lego. Cogemos piezas que funcionan y sabemos lo que hacen, las adaptamos a nuestras necesidades y las juntamos.

Clasificación



PATRONES DE CREACIÓN



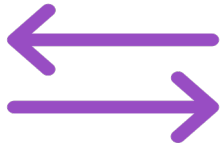
PATRONES ESTRUCTURALES

PATRONES DE COMPORTAMIENTO

Clasificación

Patrones de Creación

Abstract Factory
Factory Method
Builder
Singleton
Prototype



Patrones Estructurales

Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Proxy

Patrones de Comportamiento

Command
Chain of responsibility
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template Method
Visitor