Patrones arquitectónicos

PROGRAMACIÓN DE DISPOSITIVOS MÓVILES

Carlos Rojas Sánchez

Licenciatura en Informática

Universidad del Mar

Contenido

1. Qué son los patrones arquitectónicos
2. Qué son los patrones de diseño
3. Algunos patrones arquitectónicos

Qué son los patrones arquitectónicos

Patrones arquitectónico

Los patrones arquitectónicos son soluciones generales reutilizables a problemas comunes en el diseño de la arquitectura de software. Representan esquemas ampliamente aceptados que definen cómo organizar los componentes de un sistema para asegurar escalabilidad, mantenibilidad, rendimiento y otros atributos de calidad.

Patrones arquitectónico

Cliente-Servidor

Descripción Divide el sistema en dos partes: cliente (interfaz de usuario) y servidor (procesamiento y datos).

Ejemplo Aplicaciones web, como Gmail o Facebook.

Ventajas Separación clara, mantenimiento sencillo, escalabilidad.

Desventajas Dependencia de la red, cuello de botella en el servidor.

Patrones arquitectónico

Modelo-Vista-Controlador (MVC)

Descripción Separa la lógica del negocio (modelo), la interfaz de usuario (vista) y el control de flujo (controlador).

Ejemplo Frameworks como Django, Ruby on Rails, Laravel.

Ventajas Separación de responsabilidades, facilita pruebas y mantenimiento.

Desventajas Puede ser complejo para proyectos pequeños.

Patrones arquitectónico

Capas (Layered)

Descripción Organiza el sistema en capas, como presentación, lógica de negocio, acceso a datos.

Ejemplo Aplicaciones empresariales típicas.

Ventajas Modularidad, separación de responsabilidades.

Desventajas Puede causar sobrecarga si hay muchas capas.

Patrones arquitectónico

Event-Driven (Orientado a Eventos)

Descripción Los componentes se comunican mediante eventos asincrónicos.

Ejemplo Aplicaciones en tiempo real, como juegos o trading financiero.

Ventajas Alta capacidad de respuesta, desacoplamiento.

Desventajas Difícil de depurar y mantener.

Patrones arquitectónico

Pipe and Filter (Tubería y Filtro)

Descripción Los datos pasan a través de una serie de filtros que los transforman.

Ejemplo Compiladores, procesamiento de datos.

Ventajas Fácil de extender, reutilizable.

Desventajas Procesamiento secuencial, difícil manejo de errores.

Patrones arquitectónico

Broker

Descripción Usa un intermediario (broker) para coordinar la comunicación entre componentes distribuidos.

Ejemplo Middleware, sistemas de mensajería como RabbitMQ o Kafka.

Ventajas Desacoplamiento, escalabilidad.

Desventajas Mayor latencia y complejidad.

Patrones arquitectónico

Arquitectura en Hexágono (Ports and Adapters)

Descripción Aísla el núcleo de la lógica del negocio del mundo exterior mediante puertos y adaptadores.

Ejemplo Aplicaciones con alta portabilidad y facilidad de pruebas.

Ventajas Independencia de tecnologías externas, prueba sencilla.

Desventajas Mayor curva de aprendizaje.

Qué son los patrones de diseño

Patrones de diseño

- Soluciones reutilizables a problemas comunes de diseño de software.
- Proporcionan buenas prácticas y facilitan el mantenimiento del código.
- Clasificados por el ”Gang of Four” (GoF) en:
  - Creacionales
  - Estructurales
  - Comportamiento

Patrones Creacionales

- **Singleton**: Garantiza una única instancia global.
- **Factory Method**: Delegación de la creación de objetos a subclases.
- **Abstract Factory**: Crea familias de objetos relacionados.
- **Builder**: Construcción paso a paso de objetos complejos.
- **Prototype**: Clonación de objetos.

Patrones Estructurales

- **Adapter**: Permite la compatibilidad entre interfaces distintas.
- **Decorator**: Agrega responsabilidades dinámicamente.
- **Composite**: Composición de objetos en estructuras jerárquicas.
- **Facade**: Proporciona una interfaz simplificada.
- **Proxy**: Representante o sustituto de otro objeto.
- **Bridge**: Separa la abstracción de su implementación.
- **Flyweight**: Reduce uso de memoria compartiendo objetos.

Patrones de Comportamiento

- **Observer**: Notificación automática a múltiples objetos.
- **Strategy**: Cambia algoritmos en tiempo de ejecución.
- **Command**: Encapsula una solicitud como objeto.
- **State**: Cambia el comportamiento según su estado interno.
- **Template Method**: Esqueleto de algoritmo con pasos personalizables.
- **Chain of Responsibility**: Cadena de manejadores para solicitudes.
- **Mediator**, **Memento**, **Visitor**, **Interpreter**.

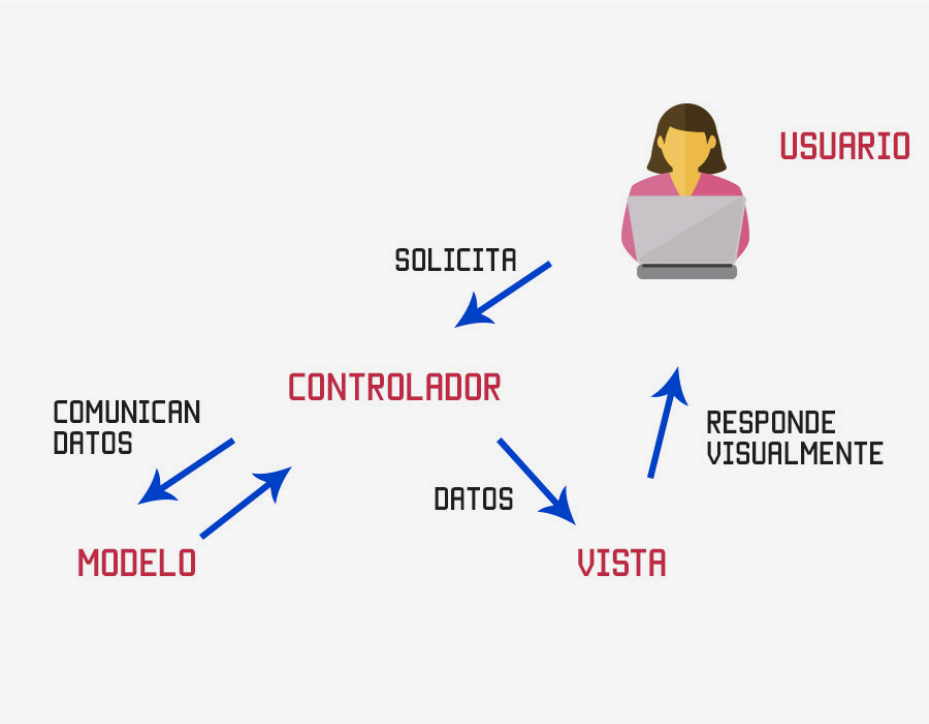
Diferencias con Patrones Arquitectónicos

- **Patrones de Diseño**: nivel medio-bajo, aplicados a clases y objetos.
- **Patrones Arquitectónicos**: nivel alto, definen la estructura global del sistema.
- Ejemplo:
  - Arquitectónico: MVC
  - De diseño: Observer para actualizar vistas.

Algunos patrones arquitectónicos

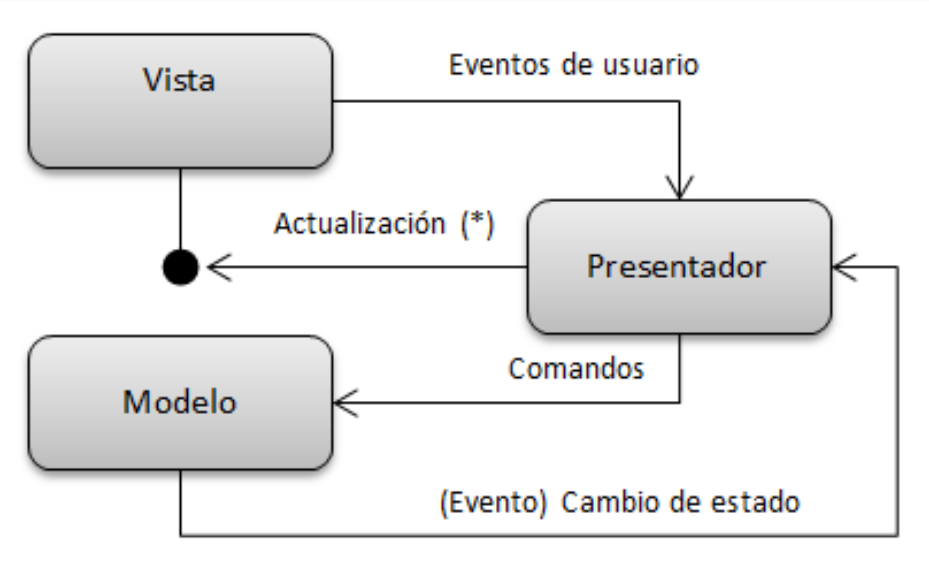
Modelo Vista Controlador (MVC)

- Divide la aplicación en tres componentes principales:
  - **Modelo**: Lógica de negocio y datos.
  - **Vista**: Interfaz de usuario.
  - **Controlador**: Intermediario entre modelo y vista.
- Muy utilizado en aplicaciones web (ej. Ruby on Rails, Django).



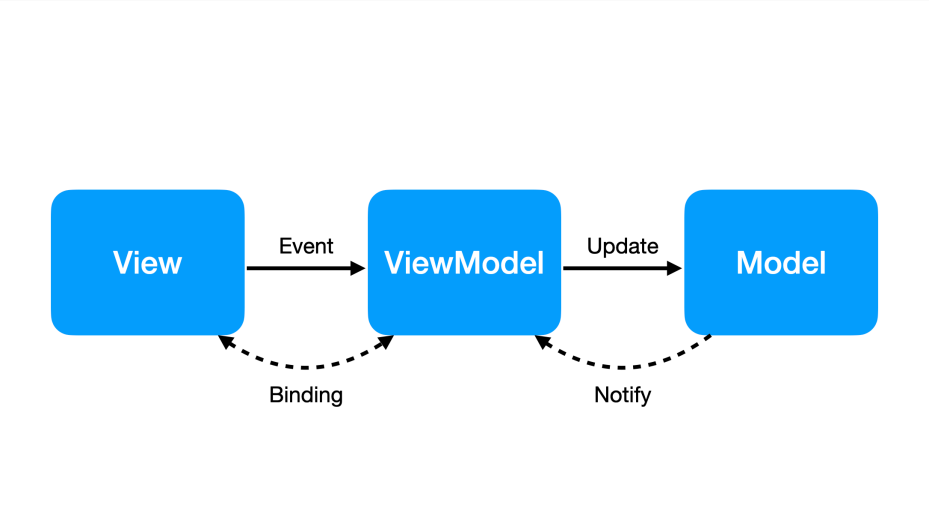
Modelo Vista Presentador (MVP)

- Variante de MVC, con un **Presentador** que:
  - Contiene la lógica de presentación.
  - Se comunica con el modelo y actualiza la vista.
- La vista es más ”pasiva”.
- Popular en aplicaciones de escritorio y móviles.



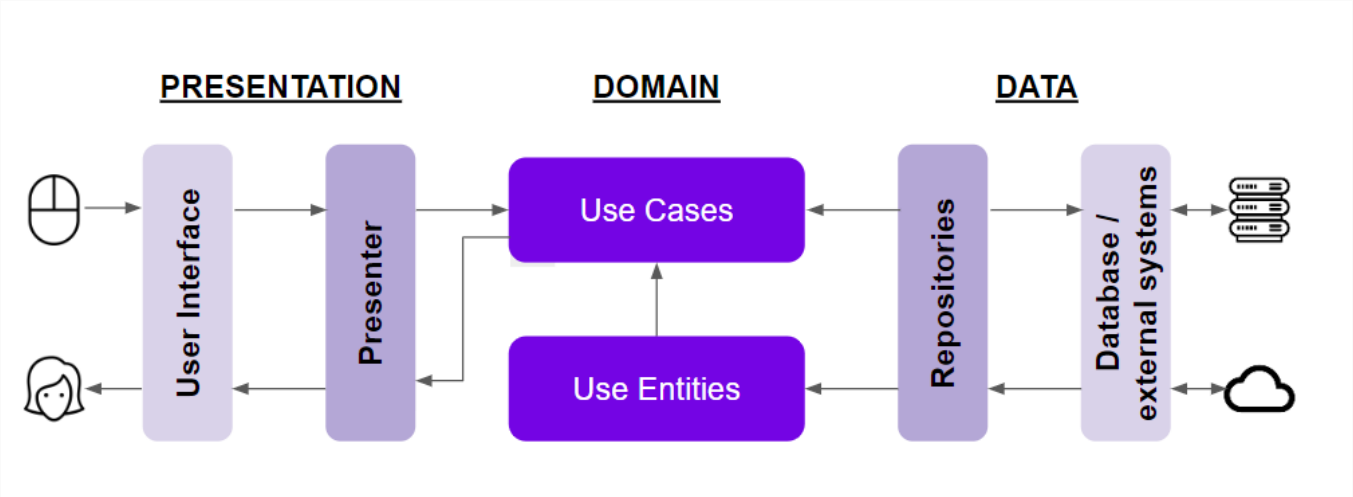
Modelo Vista Modelo de Vista (MVVM)

- Añade el **ViewModel**, que expone datos y comandos a la vista.
- Facilita el **data binding** entre la vista y la lógica.
- Muy usado en frameworks como Angular, React (con hooks), WPF.



Clean Architecture

- Propuesta por Robert C. Martin (”Uncle Bob”).
- Separa la aplicación en capas concéntricas:
  - **Entidad**, **Caso de uso**, **Interfaz**, **Infraestructura**.
- Las dependencias apuntan hacia el centro (regla de dependencia).
- Ideal para grandes proyectos y pruebas automatizadas.



Comparación de Patrones

Aspecto	MVC	MVP	MVVM	Clean Architecture
Lógica en UI	Parcial	No	No	No
Bidireccional	No	Sí	Sí (Binding)	Parcial
Separación de capas	Básica	Mejor	Muy clara	Excelente
Facilidad de testeo	Media	Alta	Alta	Muy alta
Complejidad	Baja	Media	Media	Alta
Usos comunes	Web apps	Desktop/móvil	Apps modernas	Arquitectura empresarial

Table 1: Comparación de arquitecturas

Conclusión

- No hay una única arquitectura correcta: todo depende del contexto.
- MVC, MVP y MVVM son ideales para la interfaz de usuario.
- Clean Architecture es una solución más robusta y flexible.
- Comprender sus diferencias permite elegir la más adecuada para cada proyecto.