Arquitecturas para aplicaciones de software empresariales

Patrones aplicados en el ejemplo mostrado en clase

PROGRAMA
DE INGENIERÍA ELECTRONICA Y TELECOMUNICACIONES

Ing. Daniel Eduardo Paz Perafán

Definición de arquitectura

La Arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación".

La Arquitectura de Software se refiere a una descripción de cómo está organizado un sistema en componentes y cómo interactúan los componentes.

Cada conjunto de elementos relacionados de un tipo particular corresponde a una estructura distinta, de ahí que la arquitectura esta compuesta por distintas estructuras.

Patrón de diseño

- Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.

- Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

- Debemos tener presente los siguientes elementos de un patrón:
  - Su nombre
  - El problema (cuando aplicar un patrón)
  - La solución (descripción abstracta de la solución)
  - Las consecuencias (costos y beneficios).

¿Cómo organizar la arquitectura de software?

Patrón 1. Patrón n capas

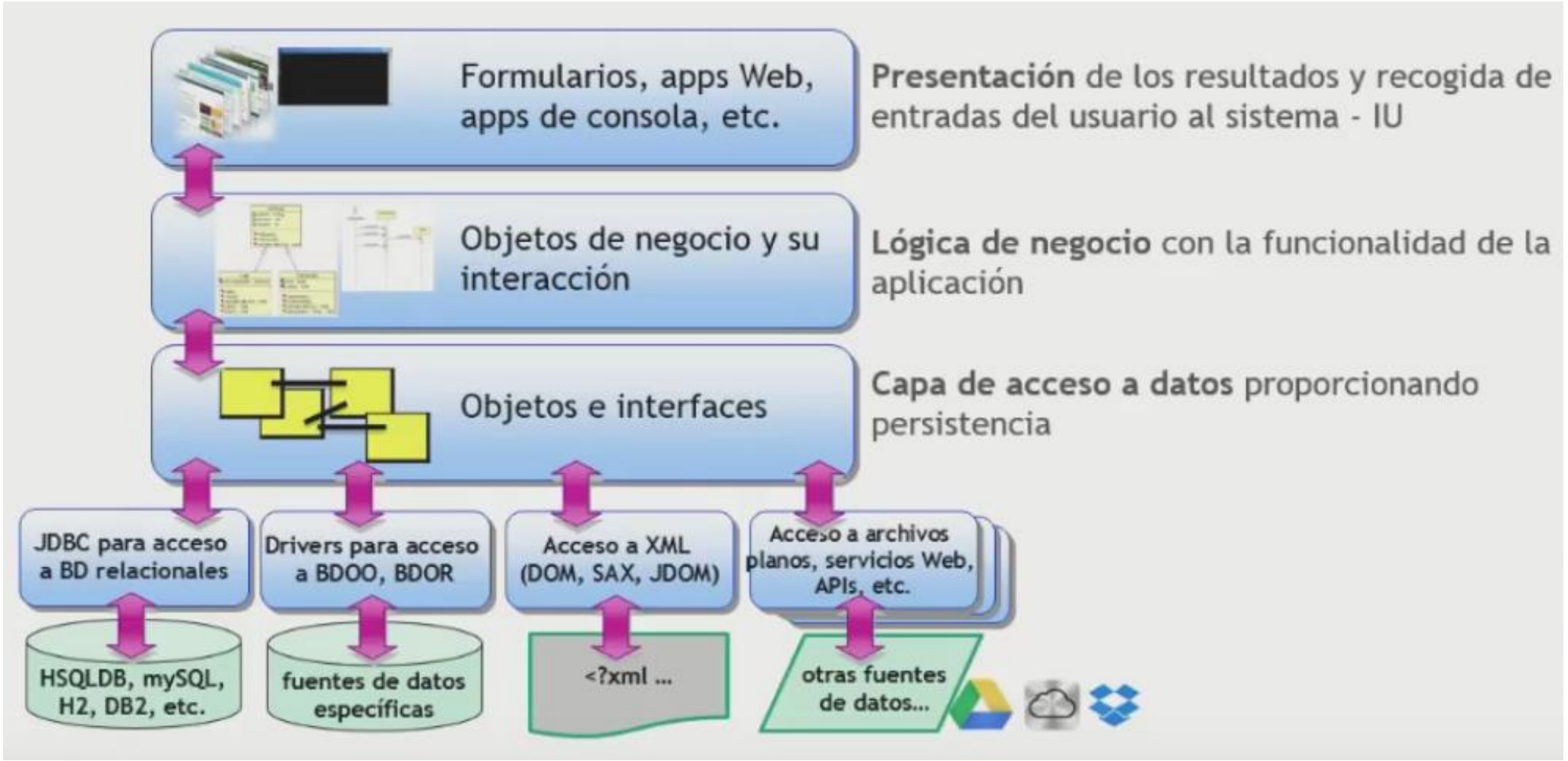
- A medida que los sistemas software se hacen más complejos, se requiere una organización de los mismos.

- El código fuente se puede organizar en diferentes capas (abstracción de funcionalidades).

- La capa es una unidad lógica, que establece como organizar el código. Por ejemplo: presentación (vista), controlador, negocio, acceso a datos

- La arquitectura multicapa se basa en organizar la aplicación software en varias capas que ofrecen servicios

Patrón 1. Patrón n capas



Patrón 1. Patrón n capas

Capa de controladores

- Expone una serie de servicios mediante un API.
- Realiza el binding de los datos de entrada.
- Retorna un resultado mediante un formato de datos como JSON.
- Valida automáticamente los datos mediante otro componente.
- Retorna el código asociado al procesamiento realizado.

Patrón 1. Patrón n capas

Capa de negocio

- Expone la lógica necesaria a la capa de presentación para que el usuario, a través de la interfaz, interactúe con las funcionalidades de la aplicación.

- Por negocio podemos entender una industria financiera, un sitio de ecommerce, o un sitio de información deportiva, etc.

- Son componentes que proveen la lógica de negocio de una aplicación. La lógica de negocio radica en operaciones matemáticas, análisis de datos, cálculos estadísticos, filtrado de consultas etc

Patrón 1. Patrón n capas

Capa de acceso a datos

- Gestiona el acceso a los datos de la aplicación.
- A partir de solicitudes de la capa de negocio, emplea gestores de bases de datos que realizan la recuperación y el almacenamiento físico de los datos.
- Se encarga de realizar operaciones como registrar, consultar, eliminar, actualizar.

Patrón 1. Patrón n capas

Recomendación

- La capa superior debe hacer uso de las capas inferiores.
- Se recomienda desarrollar las aplicaciones separando por capas; es decir, diseñar aplicaciones separando entre el modelos de datos, la lógica de aplicación, y la interfaz de usuario.
- Si optamos por esta separación de responsabilidades, podremos compartir totalmente el modelo de datos entre todas las versiones de la aplicación y sólo tendremos que adaptar la interfaz de usuario.

Patrón 1. Patrón n capas

- Aísla la lógica de la aplicación en componentes separados.
- Facilita la distribución de las capas en diferentes tier (máquinas).
- Posibilita el desarrollo de test en paralelo.
- Ayuda a incrementar la reutilización.
- Es un modelo más caro de implementar y de ejecutar.

Patrón 2. Patrón n niveles

Diferencias entre capa y tier (nivel)

El **tier (nivel)** es una unidad física, donde se ejecuta el código / proceso. Un nivel es un contenedor físico de una o más capas, como un servidor a través de una red o varias máquinas virtuales.

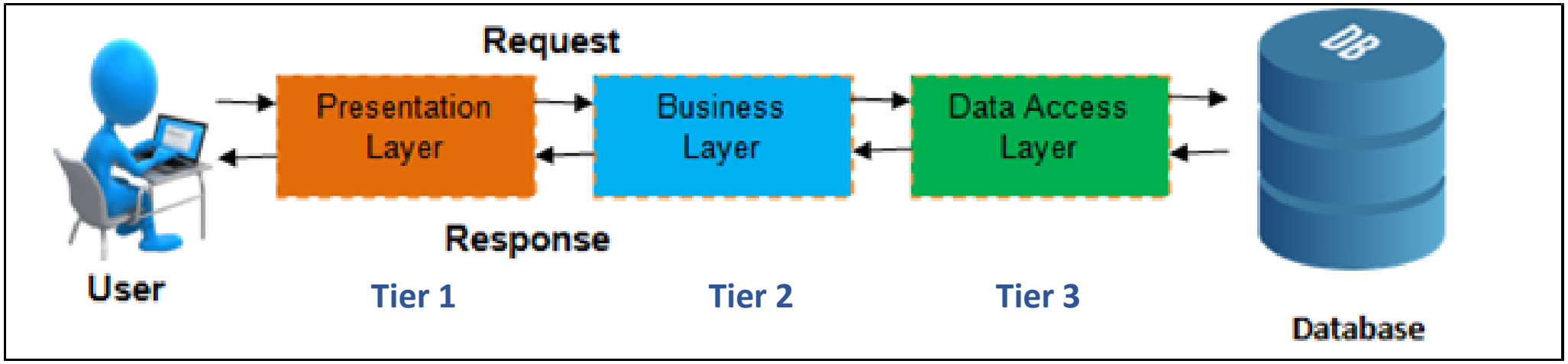
La **capa** es una unidad lógica, que establece como organizar el código. Por ejemplo: presentación (vista), controlador, negocio, acceso a datos

Una capa es un módulo lógico de software con su propia lógica central y límites.

Patrón 2. Patrón n niveles

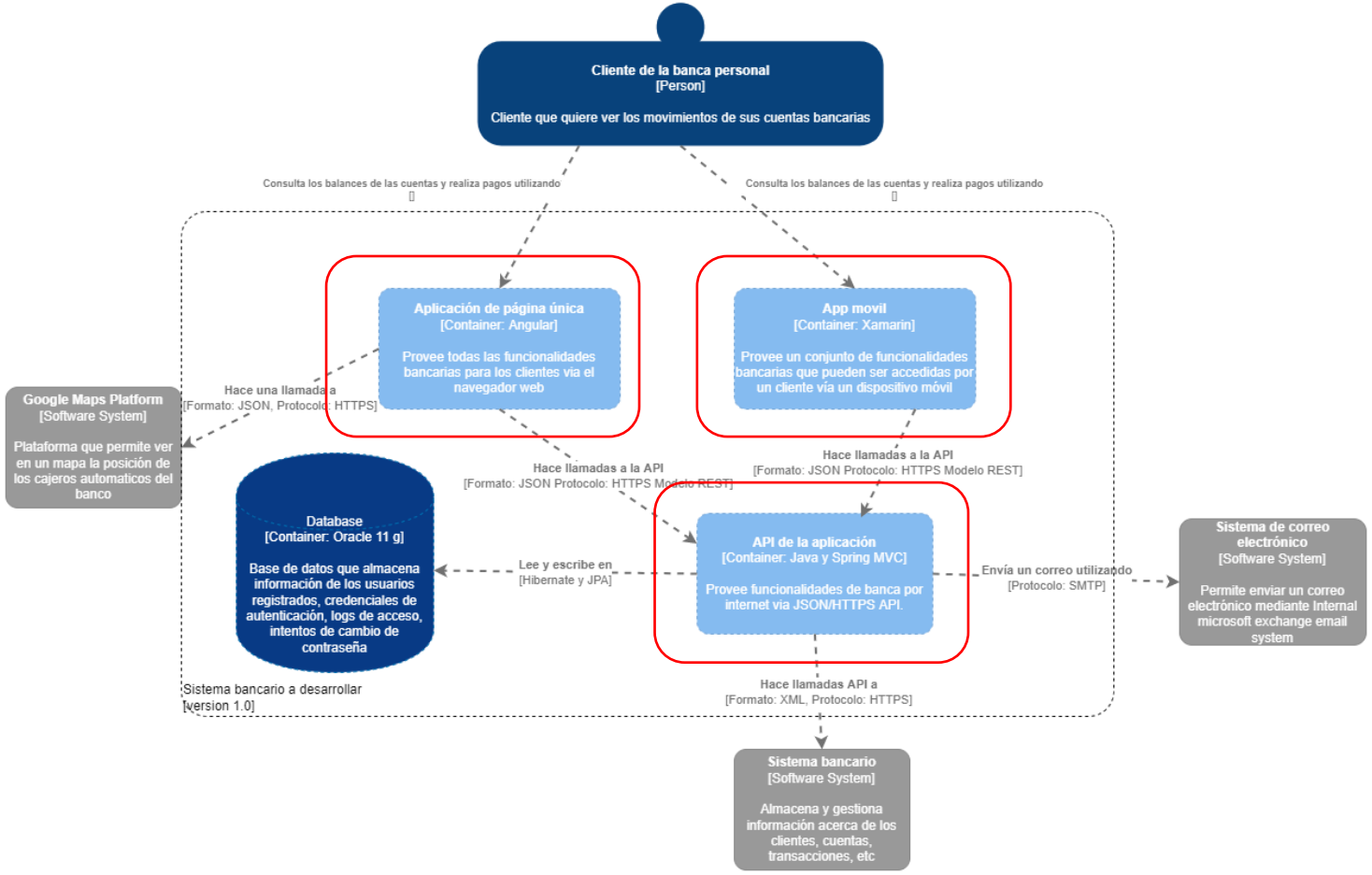
Diferencias entre capa y tier (nivel)

Muchas veces la arquitectura multinivel y la arquitectura multicapa se confunden. La diferencia esta en la separación lógica y física.



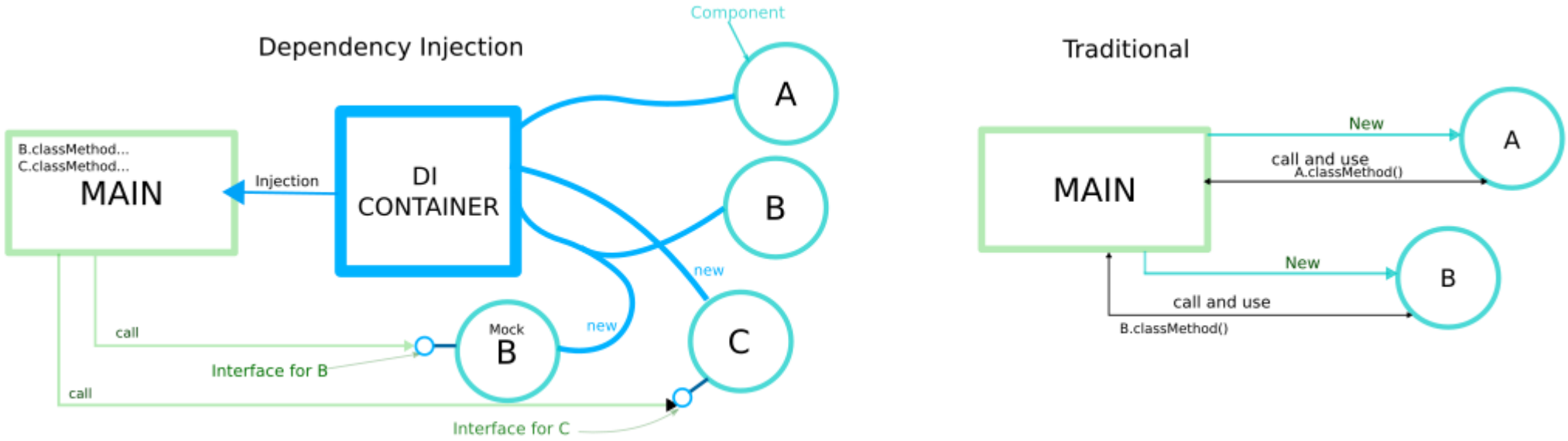
Patrón 2. Patrón n niveles

Contenedores y niveles



Patrón 3. Inyección de dependencias

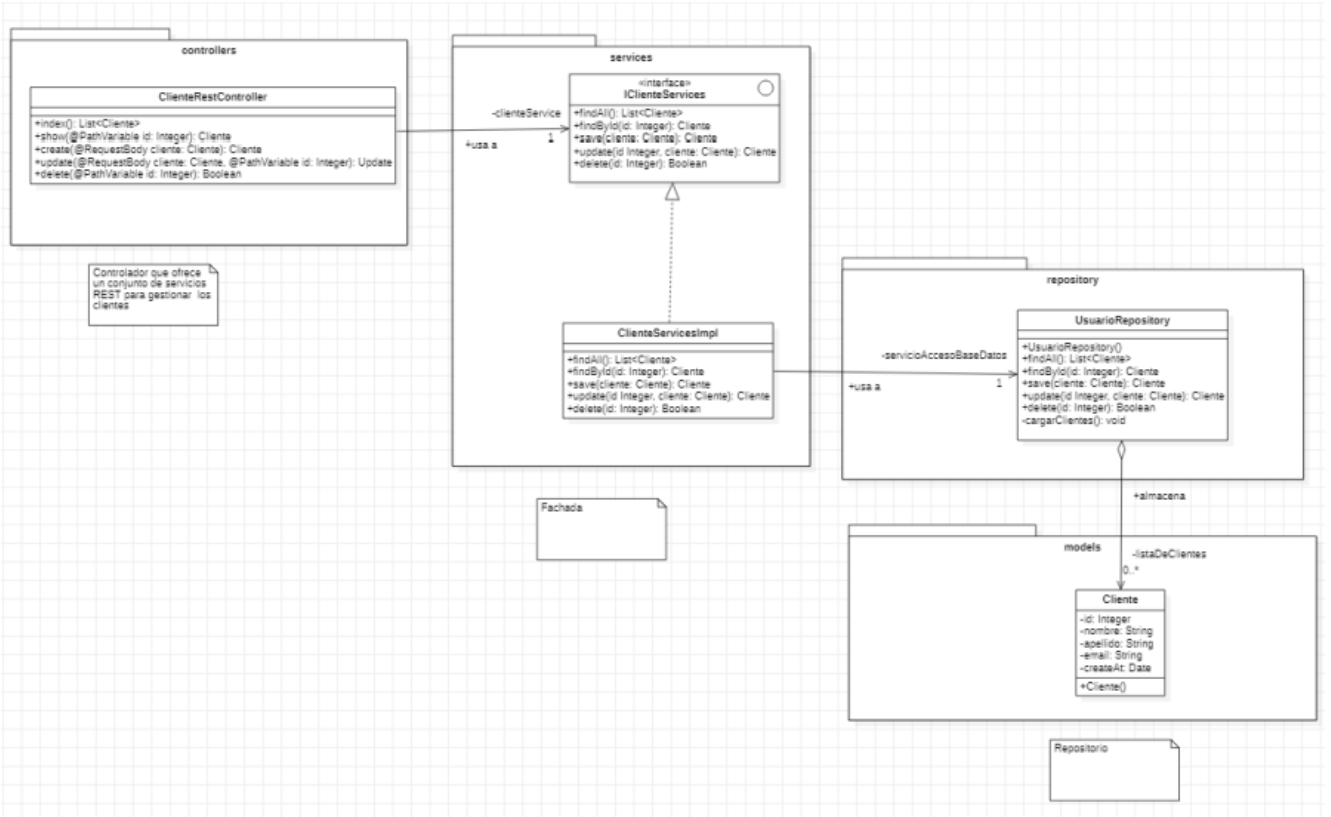
- Normalmente cuando nosotros programamos en el día a día con la programación orientada a objeto nos encontramos construyendo objetos y relacionando objetos utilizando dependencias.



Patrón 4. Fachada

- Fachada es un patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

- Una fachada es una clase que proporciona una interfaz simple a un subsistema complejo que contiene muchas partes móviles.



Patrón 5. Patrón Repositorio

- Encapsula el acceso a la base de datos. Por lo que cuando la capa de lógica de negocio necesite interactuar con la base de datos, va a hacerlo a través de la API que le ofrece DAO.

- Generalmente esta API consiste en métodos CRUD (Create, Read, Update y Delete). Entonces por ejemplo cuando la capa de lógica de negocio necesite guardar un dato en la base de datos, va a llamar a un método create().

- Lo que haga este método, es problema de DAO y depende de como DAO implemente el método create(), puede que lo implemente de manera que los datos se almacenen en una base de datos relacional como puede que lo implemente de manera que los datos se almacenen en ficheros de texto.

- La capa de lógica de negocio no tiene porque saberlo, lo único que sabe es que el método create() va a guardar los datos, así como el método delete() va a eliminarlos, el método update() actualizarlos, etc. Pero no conoce como interactúa DAO con la base de datos.

Patrón 6. Patrón DTO

- Una de las problemáticas más comunes cuando desarrollamos aplicaciones, es diseñar la forma en que la información debe viajar desde la capa de servicios a las aplicaciones o capa de presentación

- El patrón DTO tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes

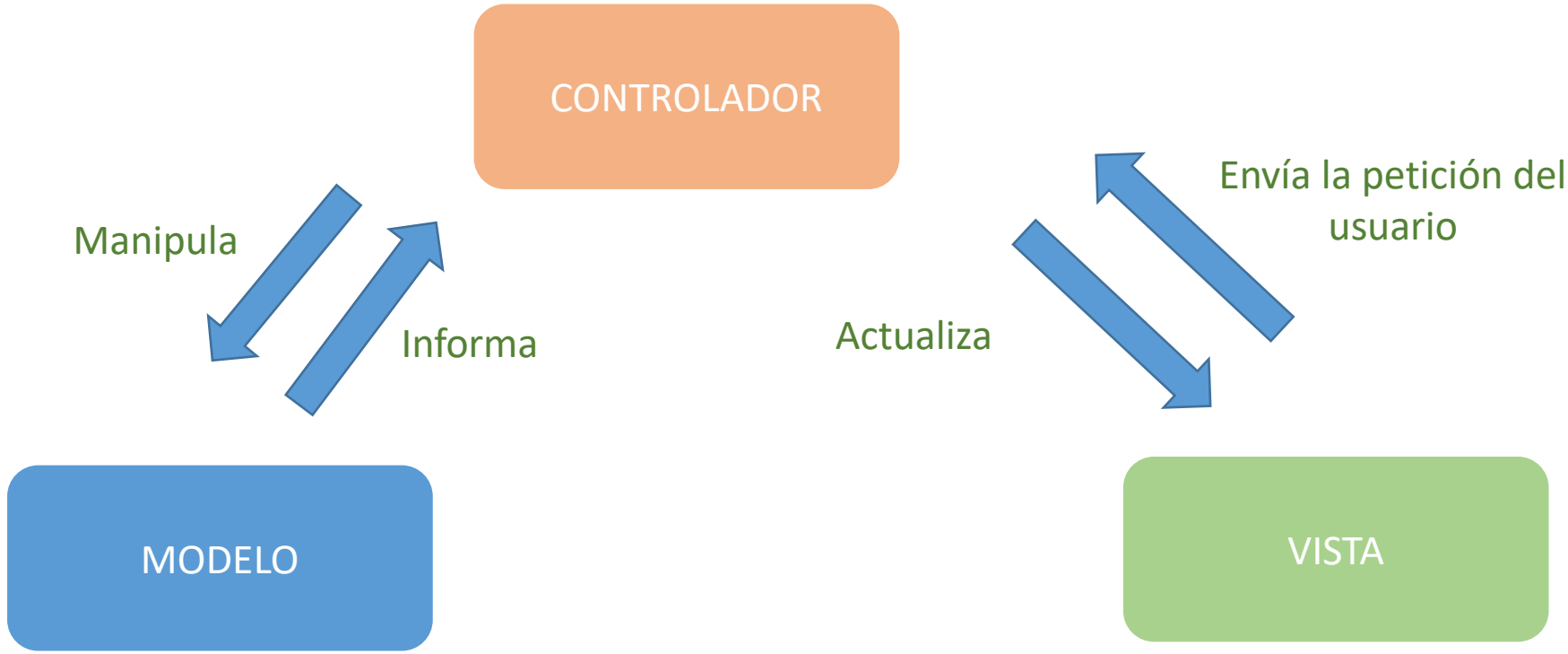
- Dado que el objetivo de un DTO es utilizarlo como un objeto de transferencia entre el cliente y el servidor, es importante evitar tener operaciones de negocio o métodos que realicen cálculos sobre los datos, es por ello que solo deberemos de tener los métodos GET y SET de los respectivos atributos del DTO.

Patrón 6. Patrón DTO

Ejemplo

```
package servidor.modelo.dominio;

public class LibroDTO {
    private String codigo;
    private String titulo;
    private String tipo;
    private int cantidad;

    public LibroDTO(String codigo, String titulo, String tipo, int cantidad) {...6 lines }

    public String getCodigo() {...3 lines }
    public void setCodigo(String codigo) {...3 lines }
    public String getTitulo() {...3 lines }
    public void setTitulo(String titulo) {...3 lines }
    public String getTipo() {...3 lines }
    public void setTipo(String tipo) {...3 lines }
    public int getCantidad() {...3 lines }
    public void setCantidad(int cantidad) {...3 lines }

}
```

7. Patrón MVC



Algunas variantes den MVC harán que la vista observe los cambios en los modelos, mientras que otros permitirán que el controlador maneje la actualización de la vista.

7. Patrón MVC

El **controlador** contiene lógica que actualiza el modelo y / o la vista en respuesta a una petición de los usuarios de la aplicación.

El **Modelo** gestiona el acceso a los datos de la aplicación. El modelo no debe conocer sobre html, formularios, apps. Proporciona funcionalidades para consultar y cambiar el estado de los datos sobre algún medio de persistencia. Cuando un modelo cambia notifica a sus observadores que un cambio a ocurrido.

La **vista** es la capa de presentación que contiene las páginas html, formularios, apps . Sirve para que el usuario pueda interactuar a través de los eventos con el controlador y también para que pueda ver los resultados. Estos resultados serán aportados por el Controlador a través del Modelo.

El controlador actualiza la vista cuando el modelo cambia. También agrega detectores de eventos a la vista y actualiza el modelo cuando el usuario manipula la vista.

Bibliografía

Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. (2001). Web engineering: A new discipline for development of web-based systems. In Web Engineering (pp. 3-13). Springer, Berlin, Heidelberg.

Aliaga, C. J., & Quintero, A. I. L. (2018). Arquitecturas de microservicios para aplicaciones desplegadas en contenedores.

Sommerville, I. (2011). Software engineering 9th Edition. ISBN-10, 137035152, 18.

Muchas gracias

Preguntas