Proyecto - MVC

Fidel Lace

Luigui Salazar

Análisis y diseño Orientado a Objetos

Profesor: Eric Gustavo Coronel Castillo

Universidad Científica del Sur

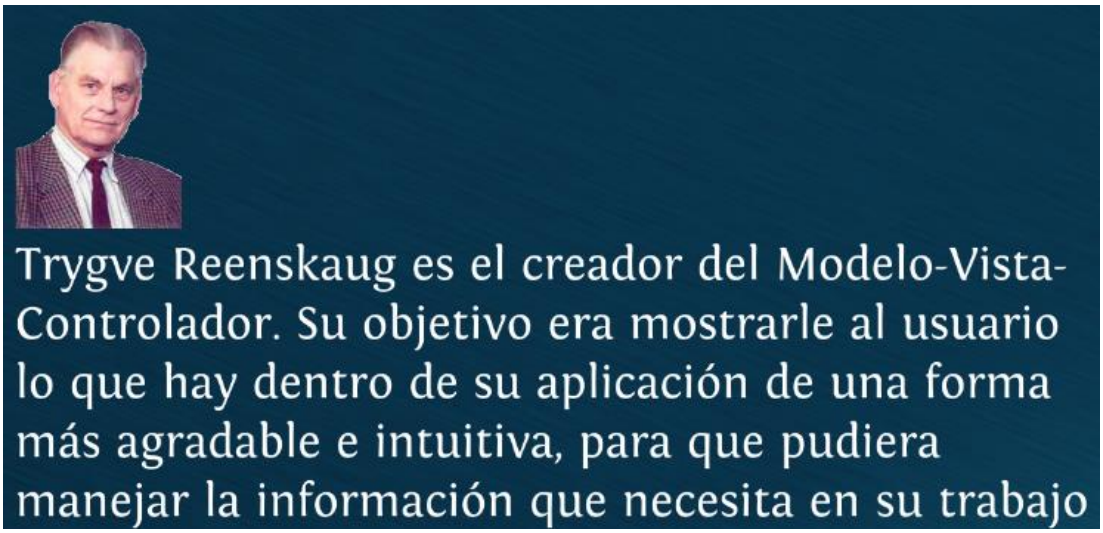
2017

Contenido

1. Origen del MVC

El modelo vista controlador fue un de las primeras ideas en el campo de las interfaces graficas del usuario y uno de los primeros trabajos en describir e implementar aplicaciones software.

El entorno fue introducido por Trygve Reeskaug y es el producto de la problemática de manejar un conjunto de datos complejos de gran volumen.



2. Concepto

Es un patrón de arquitectura de las aplicaciones software que separa la lógica de negocio de la interfaz de usuario:

- Facilita la evolución por separado de ambos aspectos
- Incrementa reutilización y flexibilidad

Está conformado por:

- Un modelo:
    - Contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- Varias vistas:

- Llamada también interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.

- Varios controladores:
  - Actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Entonces podemos decir que:

- Las vistas y los controladores suelen estar muy relacionados.
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista).
- Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador.

3. Responsabilidades

a. El modelo es responsable de:

i. Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

ii. Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor"

iii. Lleva un registro de las vistas y controladores del sistema.

iv. Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un

fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.).

b. El controlador es responsable de:

i. Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).

ii. Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega (nueva_orden_de_venta)".

c. La vista es responsable de:

i. Recibir datos del modelo y las muestras al usuario.

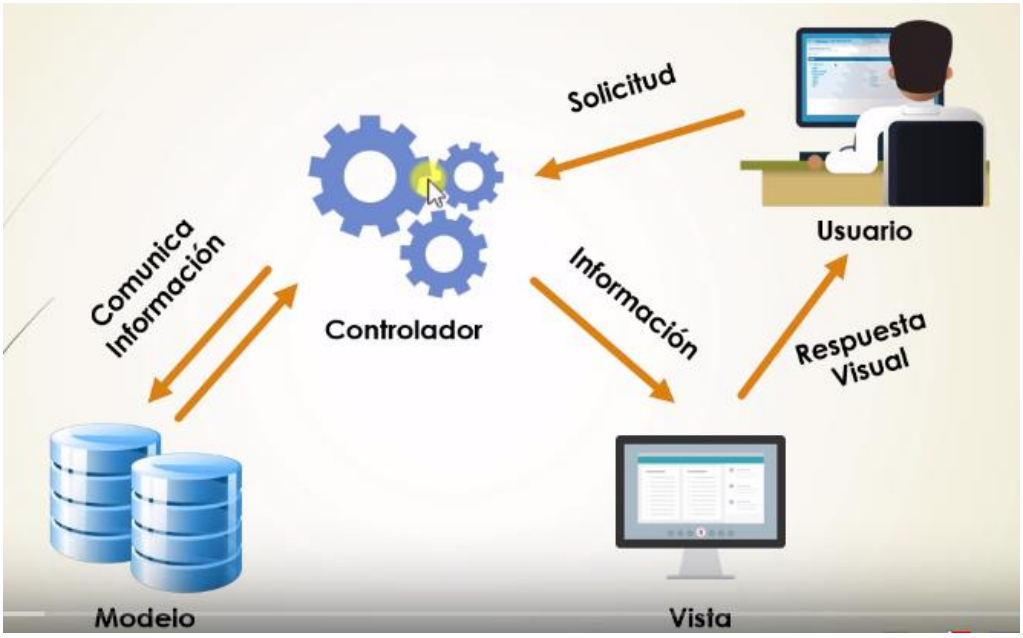
ii. Tienen un registro de su controlador asociado (normalmente porque además lo instancia).

iii. Pueden dar el servicio de "Actualización ()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

}

4. Flujo

El flujo que sigue MVC es el siguiente:



1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

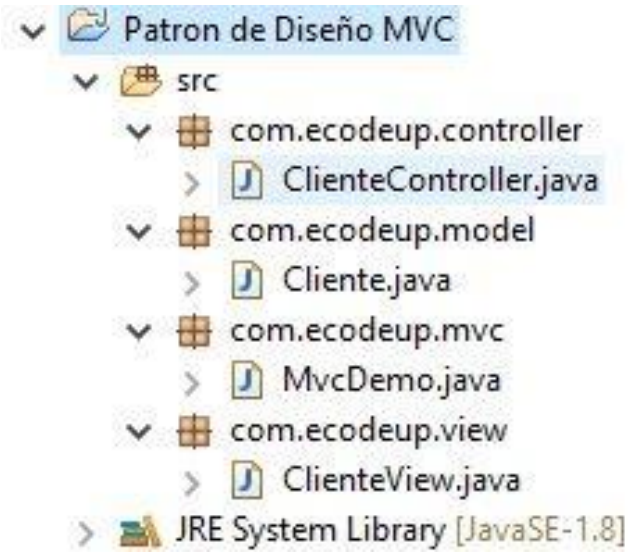
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta dirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

En java:



5. Analogía

Una que me gusta mucho es la de la televisión. En tu televisión puedes ver distintos canales distribuidos por tu proveedor de cable o televisión (que representa al modelo), todos los canales que puedes ver son la vista, y tú cambiando de canal, controlando qué vez representas al controlador.

6. Ventajas y desventajas:

Se tienen muchas ventajas como:

- La implementación se realiza de forma modular.
- Sus vistas muestran información actualizada siempre. El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Cualquier modificación que afecte al dominio, como aumentar métodos o datos contenidos, implica una modificación sólo en el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
- Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información, no su tratamiento.
- MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una

mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

Como desventajas tenemos:

- Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo. Normalmente el patrón exige al programador desarrollar un mayor número de clases que, en otros entornos de desarrollo, no son necesarias. Sin embargo, esta desventaja es muy relativa ya que posteriormente, en la etapa de mantenimiento de la aplicación, una aplicación MVC es mucho más fácil de mantenerlo, extensible y modificable que una aplicación que no lo implementa.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación. Esta arquitectura inicial debe incluir, por lo menos, un mecanismo de eventos para poder proporcionar las notificaciones que genera el modelo de aplicación; una clase Modelo, otra clase Vista y una clase Controlador genéricas que realicen todas las tareas de comunicación, notificación y actualización que serán luego transparentes para el desarrollo de la aplicación.
- MVC es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma

Referencias

- *http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122*
- *https://prezi.com/zcsejbxmrjgm/modelo-vista-controlador/*
- *http://jc-mouse.blogspot.pe/2011/12/patron-mvc-en-java-con-netbeans.html*
- *https://book.cakephp.org/1.3/es/The-Manual/Beginning-With-CakePHP/Understanding-Model-View-Controller.html*