UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

INGENIERÍA DE SOFTWARE II

ACTIVIDAD NÚMERO 2

DOCENTE:
ING. Efrain Rodrigo Fonseca Carrera

INTEGRANTES:
CARPIO CELI LUIS ALFREDO
CARRASCO PINTO ALINE MARIBEL
PILA IZA NARCISA NATALY
TOAQUIZA PUCO GEOVANY NELSON

DEPARTAMENTO DE
CIENCIAS DE LA COMPUTACIÓN

CARRERA DE:

TECNOLIGÍAS DE LA INFROMACIÓN Y COMUNICACIÓN

12 DE DICIEMBRE DE 2022

Pichincha, Quito

1. Objetivo

Exponer un ejemplo que haga uso del patrón MVC para el diseño de un aplicativo.

2. Problemática

En una empresa manufacturera de pantalones jeans de hombre en tallas desde el 32 al 40. Quiere llevar únicamente un registro del número de prendas armadas, su talla y color. La empresa consigue piezas para el armado del pantalón para su posterior ensamblaje y etiquetado. Los trabajadores registraran el código de barras de la etiqueta en el sistema cada vez que un pantalón se termina de armar. Cantidad, en base a color y talla serán actualizadas cada vez que se registre el código de barras. Con el tiempo se prevé que la empresa también confeccione pantalones de mujer en 3 tallas y 3 colores diferentes. Ademas de llevar un control del tiempo que toma el armado de un pantalón.

3. Introducción

En la mayoría de los sistemas existe la interacción usuario – maquina la cual busca y recoleta información de diferentes fuentes para luego ser presentada al usuario mediante una interfaz gráfica que presente únicamente lo que el usuario solicita. El usuario no requiere conocer cómo funciona el sistema y lo que hay detrás del flujo de datos. La idea es construir un sistema que independice los aspectos relacionados con el flujo de datos de forma limpia, intérprete de manera correcta lo que el usuario solicita, fácil de mantener y permita cambios sencillos en casos de requerir un escalamiento. Con el fin de exponer las dinámicas mencionadas.

4. Justificación

Al usar patrón MVC para la arquitectura del sistema se proporciona la ilusión de estar viendo y manipulando directamente la información que se genera en el área de manufactura de la empresa. Ademas, la empresa planea en un futuro medir otras variables que ellos consideran importantes, razón por la cual en ese instante se requeriría un cambio en la interfaz de usuario, incremento de funcionalidades que se acoplen a los mecanismo de acceso a datos. Motivos por lo cual el patrón MVC es el ideal ya que separa de mejor manera cada una de las capas que componen el sistema y ademas facilitara su mantenimiento y escalabilidad a futuro.

5. Marco Teórico

5.1. MVC: Modelo-Vista-Controlador

Es una plantilla mental que hace referencia a la forma de estructurar una aplicación que especifica las propiedades estructurales de un aspecto de un problema al cual el sistema debe brindar solución, y su aplicación tiene un impacto a su vez en la arquitectura de sus subsistemas. [1]

El patrón de componer de 3 elementos; el modelo, vista y controlador.

5.2. Modelo

Sus objetos representan la información y el comportamiento de las cuestiones funcionales acerca del dominio del problema. Este elemento esta desconectado de la parte gráfica y del comportamiento de las interfaces. Contienen todos los datos y funcionalidades que serán presentados en las interfaces de usuario. En su forma más pura de Orientación a Objetos, el modelo es un objeto dentro del modelo del dominio.

5.3. Vista

Este elemento muestra información a los usuarios y personifica la exposición del modelo en la interfaz de usuario. La vista tan solo trata cuestiones concernientes a la presentación de información extraída del modelo. Generalmente cada vista se asocia a un controlador.

5.4. Controlador

Este elemento se encarga de capturar las entradas del usuario, manipular el modelo, ademas de permitir que las vistas se actualicen apropiadamente. También se ubica parte de la lógica del negocio.

5.5. Dinámica de MVC

Esta separación permite múltiples vistas de un mismo modelo i un usuario realiza un cambio en el estado del modelo a través del controlador de una vista, todas las vistas dependientes del dato modificado deben reflejar el cambio. (mecanismo de propagación

---

[1] Camarena Sagredo, J. G., Trueba Espinosa, A., Martínez Reyes, M., & López García, M. D. (2012). Automatización de la codificación del patrón modelo vista controlador (MVC) en proyectos orientados a la Web. CIENCIA ergo-sum, Revista Científica Multidisciplinaria de Prospectiva, 19(3), 239-250.

de cambios En otras palabras, se hace uso del patrón Observer para mantener las vistas actualizadas.

- Construir un sistema flexible que cumpla con estos requerimientos puede llegar a ser muy costoso y propenso a errores si las interfaces se hallan acopladas con la lógica de negocio y el mecanismo de acceso a datos. Puede resultar en mantener algo casi parecido a diferentes sistemas por cada interfaz de usuario existente en la aplicación. [REE]

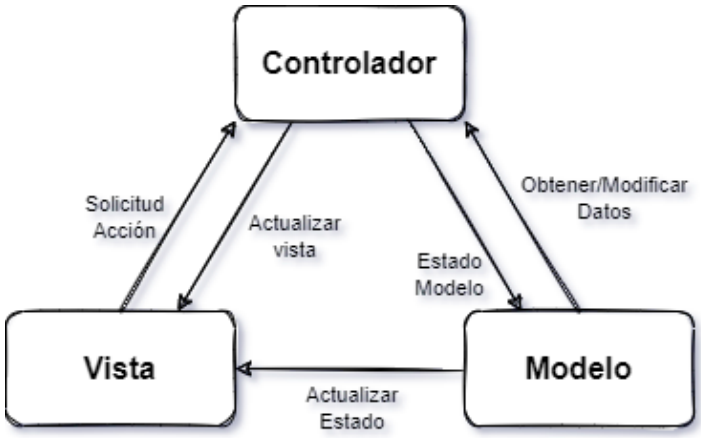

Fig. 1. Esquema de comunicación del patrón MVC.

Estas 3 capas se comunican entre sí bajo el patrón de diseño Observer. No se debe confundir el acceso a datos con una base de datos.

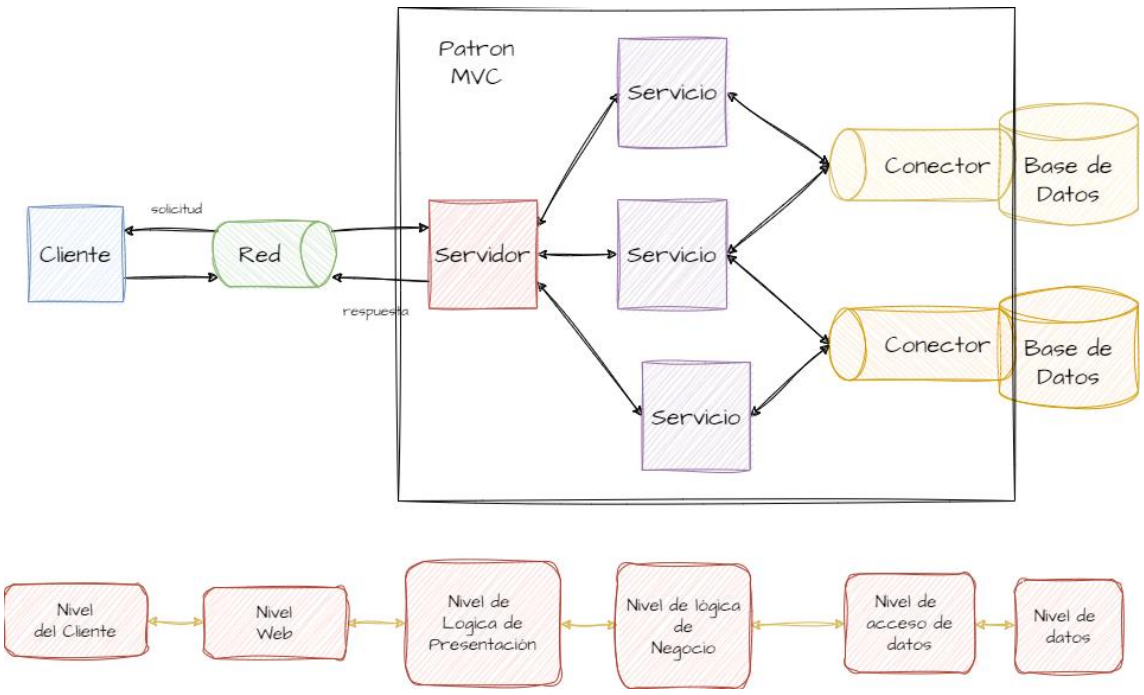

Fig. 2. Funcionamiento del patrón MVC dentro la aplicación en cado uno de los nivel.

5.6. Patrón Observer

Este patrón, es un patrón de diseño que también es conocido como Publisher-suscriber, o Dependents. El principal propósito del patrón Observer es ayudar a conservar sincronizado el estado de objetos cooperantes en un sistema, que no son más que los nexos entre cada capa. [2]

Para alcanzar su principal propósito se define un mecanismo de propagación de cambios en un solo sentido. Se compone un objeto que contiene el estado de la variable predestinada al cambio, y ante cualquier cambio en el mismo se notifican y actualizan automáticamente todos los objetos que depende de esta variable de actualización.

El patrón Observer describe cómo balancear las fuerzas que configuran los requerimientos y restricciones para obtener una solución reutilizable y flexible. Los objetos claves en este patrón son: el sujeto, que posee los datos, y el observador, que es el objeto dependiente que debe sincronizar su estado según los datos del sujeto. El patrón identifica una serie de actores para su implementación.

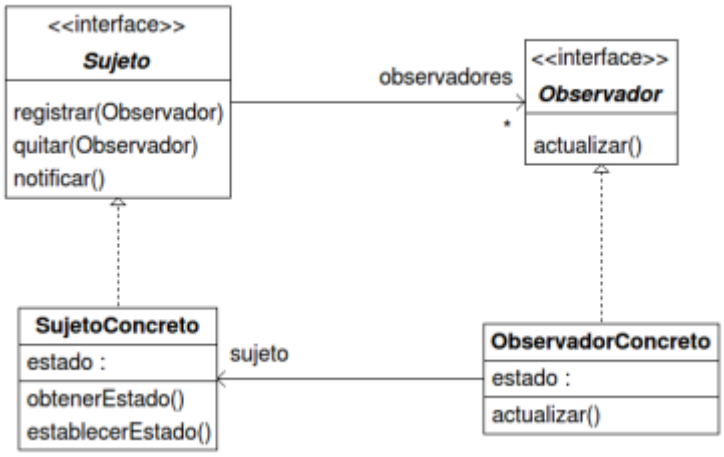

Fig. 3. Estructura del Patrón Observer. **Fuente:** [4]

5.7. Sujeto

Se encarga de mantener una lista de observadores que solo pueden ser reconocidos mediante la interfaz Observador. La interfaz sujeto no tiene conocimiento de las clases

---

[2] Valdecantos, H. A. (2010). Principios y patrones de diseño de software en Torno Al Patrón MVC. Retrieved December 1, 2022, from https://www.researchgate.net/publication/308314622_Principios_y_patrones_de_diseno_de_software_en_torno_al_patron_compuesto_Modelo_Vista_Controlador_para_una_arquitectura_de_aplicaciones_interactivas.

concretas de ningún observador, el sujeto solamente conoce lo que le interesa saber. Ademas de proporcionar una interfaz para agregar y quitar objetos del tipo Observador.

5.8. Observador

Esta interfaz se usa para actualizar los objetos que son notificados de los cambios de estado del sujeto. Establece el estilo de actualización, Pull o Push, si el método actualizar observadores especifica o no parámetros en su firma. El método Pull consiste en una continua actualización del estado cada cierto tiempo. Por otra parte Push, solicita la actualización cada que en verdad ocurrirá una actualización de estado.

5.9. Sujeto Concreto

Mantiene el estado que es de interés para los objetos observadores. Es responsable en notificar a todos sus observadores cuando su estado cambia.

5.10. Observador Concreto

Implementa la interfaz Observador, que es la cara que muestra a los sujetos concretos. Posee una referencia al objeto SujetoConcreto que observa. Mantiene su propio estado sincronizado con el estado del sujeto.

6. Desarrollo

En primera instancia se busca identificar el modelo de la aplicación que representara al área de manufactura el cual contara con 1 valor numérico y 2 valores de tipo talla y color, para cada uno de los pantalones, luego se brinda operaciones como obtener y establecer para la modificación de sus contenidos. El estado del modelo estará representado solamente por un campo privado, que serán del tipo “int”, el cual guarda la cantidad de pantalones ya armados.

En el modelo también se considera una lógica funcional de la aplicación que se centra en la solución del problema. A partir de un análisis del dominio del problema de la aplicación, se separa el modelo y el núcleo funcional de la aplicación de todos los otros mecanismos que serán responsables de la interacción hombre-computador que no responden a requerimientos funcionales. El modelo encapsula los datos y funcionalidades necesarias para resolver la lógica fundamental de la aplicación, más que nada relativa al dominio del problema. Ya separado el modelo, se necesita pensar en los requerimientos no funcionales de la aplicación, como el mecanismo de propagación de cambios que

mantendrá sincronizadas las vistas con el estado del modelo. En otras palabras, se necesita aplicar el patrón Observer, para luego asignarle el rol de sujeto o publicador a una clase área de Manufactura. Para esto se crea una clase abstracta Sujeto que se encarga de mantener todos los observadores. En cambio su interfaz ofrece métodos para que estos puedan suscribirse, eliminarse, y ser notificados de los cambios que ocurran en el estado del modelo. Conjuntamente vamos a definir la interfaz Observador a la que deben adherirse los objetos que serán observadores para poder recibir notificaciones del modelo y actualizar su estado. Este es el primer paso para crear nuestro framework MVC de la aplicación interactiva para el departamento de matemáticas.

La clase área de Manufactura debe exponer una interfaz que permita a las vistas obtener el estado y a los controladores manipular los datos del modelo. Se crea una interfaz para separar la implementación del modelo y brindar los métodos necesarios para los observadores.

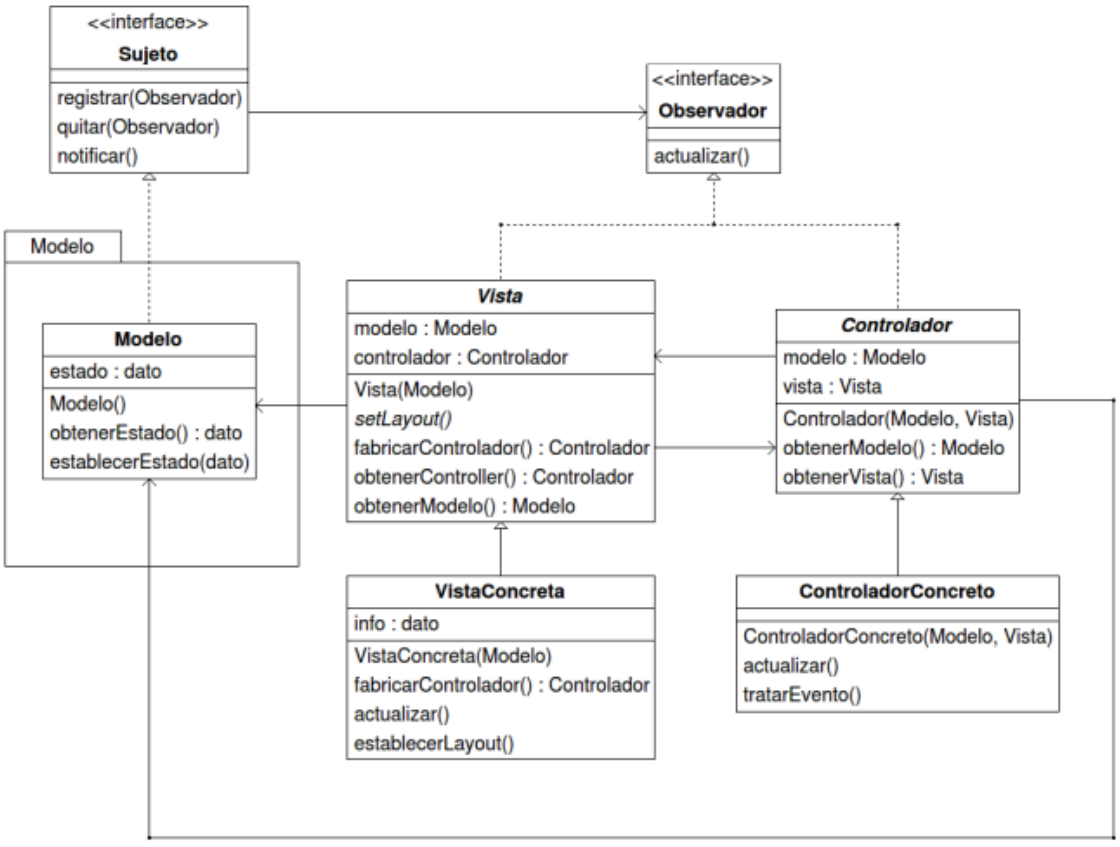

Fig. 4. Estructura de clases del patrón MVC.

El constructor de la clase abstracta Vista emprende el proceso de inicializar una vista, para ello almacena una referencia al modelo, registra la vista como observador, llama al

método setLayout() que debe ser implementado en cada vista concreta, guarda una referencia al controlador adecuado según la vista concreta, y finalmente llama al método actualizar() para mostrar la información actualizada del estado del modelo en la vista. La clase abstracta Vista define el comportamiento base que debe tener una vista concreta, pero no sabe nada sobre cómo se disponen sus elementos visuales. La vista abstracta no sabe sobre cuál es el controlador adecuado para una vista concreta, pero si sabe que en el momento de inicialización debe almacenar una referencia a este controlador.

Para crear un controlador es necesario pasar como parámetros a su constructor el modelo como la vista. Al igual que la clase Vista, la clase abstracta Controlador se registra como observador del modelo y luego llama al método actualizar(). Esta vez se implementa un procedimiento vacío por defecto para el método actualizar(), ya que para esta aplicación los controladores concretos no reaccionan ante un cambio en el modelo, solo reaccionan ante la interacción del usuario con las vistas.

7. Evolución

Habiendo creado las clases abstractas Sujeto, Vista, Controlador, y las interfaces Observador y Modelo, queda implementado un framework MVC para la aplicación, que facilitara la creación de distintas vistas, cada uno con su respectivo controlador, donde sin grandes esfuerzos de programación podrán mantener sus estados sincronizados con el estado del modelo que representa del área de manufactura. De esta forma se puede brindar al jefe del área de manufactura la interacción necesaria para visualizar y establecer la cantidad de pantalones fabricados en el día, ademas que un futuro establecer el numero de pantalones por persona que se obtienen. Otro ejemplo es que si sea una vista diferencia de los datos, se puede modificar únicamente a la vista, añadiendo una nueva función y dando a aviso al modelo que se actualizarán valores.

8. Conclusiones

El patrón MVC permite identificar las estructura de comunicación de cada capa permitiendo que los cambiantes valores no este ligados a la propia capa, si no que sean independientes. Lo cual facilita que el futuro se puedan añadir nuevas funcionalidades sin mayor dificultad.

El patrón arquitectónico MVC no es el único en usarse, también puede ser acompañado por patrones de diseño e implementación que complementan la distribución de los

elementos y el funcionamiento de las capas de modelo, vista o controlador de manera conjunta como independiente.

Se tiene un mejor control de la actualización del parámetro cantidad de jeans manufacturados, lo que permite que se recupere el ultimo valor conocido.

9. Bibliografía

[1] Martin Fowler. Patterns of Enterprise Application Architecture. AddisonWesley Professional, November 2002.

[2] Trygve M. H. Reenskaug. Pages of trygve m. h. reenskaug. Recurso online: http://heim.ifi.uio.no/~trygver/index.html.

[3] Camarena Sagredo, J. G., Trueba Espinosa, A., Martínez Reyes, M., & López García, M. D. (2012). Automatización de la codificación del patrón modelo vista controlador (MVC) en proyectos orientados a la Web. CIENCIA ergo-sum, Revista Científica Multidisciplinaria de Prospectiva, 19(3), 239-250.

[4] Valdecantos, H. A. (2010). Principios y patrones de diseño de software en Torno Al Patrón MVC. Retrieved December 1, 2022, from https://www.researchgate.net/publication/308314622_Principios_y_patrones_de_diseno_de_software_en_torno_al_patron_compuesto_Modelo_Vista_Controlador_para_una_arquitectura_de_aplicaciones_interactivas.