Modelo Vista Controlador

Alumnos:
Omar Alejandro Abasta Lucio
Luis Carlos Estrada Cadena
Luis Clemente Zúñiga Flores
Profesor:
Ángel Santiago Jaime Zavala

7 de diciembre de 2017

Contents

1 Síntesis de la arquitectura

El Modelo Vista Controlador es un patrón de arquitectura de Software que se distingue, entre otras cosas, por la estructura que otorga a las aplicaciones al mantener independientes los datos, la lógica que los manipula, y la interfaz que los muestra al usuario. Este patrón ha ganado fama en los últimos años sobre todo con la aparición de numerosos frameworks de desarrollo web, debido a que esta arquitectura permite un ciclo de vida más adecudado para las aplicaciones, además de facilitar el mantenimiento y la reutilización de código, pero sobre todo, la separación de sus conceptos y componentes.

2 Historia y background del patrón MVC

El Modelo Vista Controlador fue creado por Trygve Reenskaug en los años 1978/1979. Como él mismo menciona en la recopilación de sus reportes originales, "MVC was conceived as a general solution to the problem of users controlling a large and complex data set"[1], dándonos con ello una idea muy clara de la problemática que dio origen a MVC.

Una de sus primeras implementaciones exitosas fue en la biblioteca de clases de Smalltalk-80 , hecho que dio lugar a un artículo en el cual se formalizó el concepto de Modelo Vista Controlador. El nombre original que Reenskaug le dio a este patrón fue "Thing-Model-View-Editor" (Cosa- Modelo-Vista-Editor" con lo cual el científico computacional enmarcaba la intención de separar la información de los métodos utilizados para manipularla. En ese mismo paper el autor nos ofrece las definiciones de dichos componentes, las cuales enumeraremos para realizar un acercamiento histórico al patrón MVC.

- Thing: "Something that is of interest to the user. It could be concrete, like a house or an integrated circuit. It could be abstract, like a new

[1] Trygve Reenskaug, "The Original MVC reports", University of Oslo, 2007.

idea or opinions about a paper. It could be a whole, like a computer, or a part, like a circuit element."

- Model: "A model is an active representation of an abstraction in the form of data in a computing system".

- View: "To any given Model there is attached one or more Views, each View being capable of showing or more pictorial representations of the Model on the screen and on hardcopy. A view is also able to perform such operations upon the Model that is reasonabely associated with that view".

- Editor: "An editor is an interface between a user and one or more views. Ir provides the user with a suitable command system, for example in the form of menus that may change dynamically according to the current context. It providesthe Views with the necessary coordination and command messages".

Como acabamos de observar, las anteriores definiciones ponen de relieve el contexto histórico sobre el que nació el Modelo Vista Controlador. En primer lugar el hecho de que a fines de los años 70 del siglo pasado no existía todavía una idea convencional de Interfaz Gráfica de Usuario (IGU, por sus siglas en inglés), tanto así que Reenskaug distigue "Vista" de "Editor", siendo que hoy día (algo que el mismo Reenskaug después simplificó) notamos ambos aspectos bajo el rubro de Vista. A continuación analizaremos con un poco más de detalle los componentes de esta arquitectura.

2.1 Modelo

Siguiendo con Reenskaug, nos dice que los modelos "[...] are represented in the computer as a collection of data together with the methods necessary to process these data".[2] Un modelo es, en este sentido, un bloque de información autoconsistente, que junto con otros modelos conforma un proyecto cualquiera. En otras palabras, dada cierta problemática de la vida real, ésta es dividida en modelos, mismos que atienden un aspecto específico de esa problemática, y en su conjunto la resuelven. Esto permite la reusabilidad del código, admeás de la portabilidad del mismo hacia otras problemáticas parecidas o incluso distintas, ya que un modelo resuelve un problema tan particular que puede presentarse en proyectos de la más diversa índole.

[2] idem, p. 2.

2.2 Vista

"A view is a (visual) representation of its model."[3] Dentro de este concepto cabe resaltar que Reenskaug mismo encierra entre paréntesis el que sea visual o no dicha representación, esto es, una vista no necesariamente tiene que ser una representación gráfica, sino que "vista" aquí indica que se trata de un "filtro de presentación".

2.3 Controlador

"A controller is the link between a user and the system".[4] El controlador provee al usuario las herramientas necesarias para interactuar con la aplicación, sobre todo los mecanismos para ingresar datos y tomar decisiones de acuerdo a las opciones que les sean mostradas en las vistas. Algunas de sus características clave son:

- Proporcionar al usuario los medios de salida (output) con menús y otros medios para insertar comandos y datos.
- Recibe las salidas del sistema (outputs) las traduce en mensajes apropiados al usuario y pasa los mensajes a una o más vistas, dependiendo del caso.

Restricciones y advertencias a considerar:

- Un controlador nunca debe sustituir a las vistas.
- Una vista nunca debe "conocer" acerca de las entradas que proporciona el usuario, es decir, al ser sólo un filtro de presentación, no los manipula ni decide cuál es el dato que va a mostrar.

3 Objetivos

Como ya mencionamos, el MVC fue concebido para lidiar con el tratamiento de grandes cantidades de información, pero con respecto al usuario y las aplicaciones reales en que este patrón puede aportar su estructura, su objetivo primordial es el de dar soporte a los modelos mentales de los usuaios acerca del espacio de información relevante para ellos, y permitirles inspeccionar y editar dicha información. Bajo este rubro, el modelo incorpora la capa de

[3] idem, p. 12.
[4] idem, p. 13.

dominio y persistencia, es pocas palabras, es la que guarda los datos e indica el cómo se estructuran.

Por otra parte, la vista aquí funge como la encargada de presentar la interfaz al usuario; por ejemplo, en un sistema web, esto es lo que se hace con HTML , aunque pueden existir otro tipo de vistas. En este nivel se hacen operaciones simples como if, ciclos, formateos, etc. El controlador escucha los cambios en la vista y los envía al modelo, el cual a su vez devuelve los datos al controlador, y éste nuevamente a la vista.

Un controlador en cierta forma debe tener un registro de la relación entre ordenes que le pueden llegar y la lógica de negocio que le corresponde (Es como una operadora de teléfono que recibe una petición y une dos líneas).

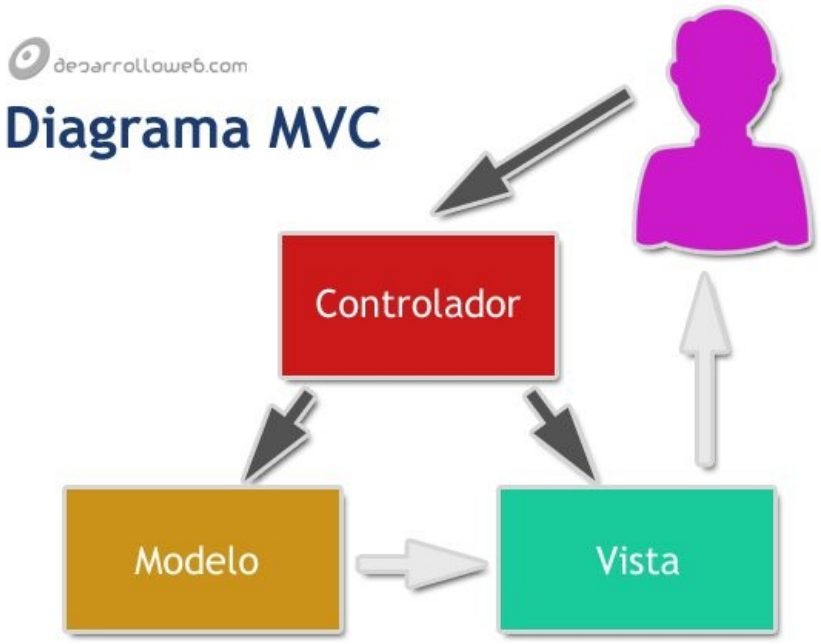

Figure 1: Diagrama del Modelo Vista Controlador

4 Pros y contras

4.1 Ventajas

- 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.

4.2 Desventajas

- 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 sostenible, 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.

5 Similitudes con otros patrones arquitecturales

5.1 MVC y MVVM (Model-View-ViewModel)

Mientras que MVC es un patrón que define en qué bloques (o capas) lógicos vamos a estructurar nuestra aplicación (Modelo, Vista y Controlador), asimismo detallar las responsabilidades exactas de cada capa y la forma que tienen de relacionarse entre sí, MVVM busca separar lo mayor posible la interfaz del usuario de la capa logíca, en este caso, del modelo de datos. Este patrón arquitectural fue creado por Microsoft en 2005 para acompañar su nueva tecnología de interfaces gráficas llamada WPF (Windows Presentation Foundation) la cual mezcla las características de una aplicación de escritorio y las de una aplicación web.

5.2 MVC y SOA (Service Oriented Architecture)

SOA y MVC pueden coexistir pacíficamente. Como se observa, SOA es una abstracción de mayor nivel de cómo el software debería interactuar entre sí, mientras que MVC es una abstracción de nivel ligeramente inferior que hace que sea más fácil escribir aplicaciones reutilizables.

6 Frameworks que proporcionan la implementación de MVC

6.1 Javascript

- Backbone.js
- Angularjs
- Ember
- Knockout

6.2 PHP

- Laravel
- Symfony
- CodeIgniter
- CakePHP

6.3 Java

- Spring MVC
- ActFramework
- Spring-boot
- Play2
- Ninja

6.4 Python

- Django
- TurboGear
- web2py
- Flask
- Pylons

7 Ejemplos de aplicación

7.1 CarTrack

- CarTrackr es una aplicación de ejemplo que utiliza el patrón de repositorio e inyección de dependencias mediante Unity Application Block. Su autor, Maarten Balliauw lo desarrolló como ejemplo para varias demos y presentaciones. En esencia se trata de una aplicación para llevar el control de gastos de cualquier automovil.
- Framework: ASP.NET MVC
- Lenguaje: Lenguajes de la plataforma .NET

7.2 FlickrXplorer

- Se trata de una iniciativa de código abierto consistente en una aplicación que muestra una herramienta de exploración y búsqueda de fotografías entre millones de ellas en Flickr. La aplicación se ha desarrollado utilizando JQuery, ASP.Net MVC y LINQ.

- Framework: ASP .NET MVC
- Lenguaje: Lenguajes de la plataforma .NET

7.3 Instagram

- El desarrollo de Instagram se inició en San Francisco, cuando Kevin Systrom y Mike Krieger optaron por centrar sus múltiples funciones HTML5 check-in Burbn en un proyecto de fotografía móvil. El producto fue lanzado en el Apple App Store el 6 de octubre de 2010 bautizado como Instagram
- Framework: Django.
- Lenguaje: Python.

7.4 Pinterest

- Pinterest permite a los usuarios, guardar y clasificar por categorías, imágenes en diferentes tableros. Pueden también seguir a otros usuarios con los mismos gustos e intereses. Las categorías populares son viajes, coches, películas, humor, diseño para el hogar, deportes, moda y arte.
- Framework: Django.
- Lenguaje: Python.

7.5 CommunityWalk

- permite añadir meta información a mapas de Google
- Framework: Ruby on Rails.
- Lenguaje: Ruby

7.6 BigCartel

- Tienda online con integración para PayPal
- Framework: Ruby on rails.
- Lenguaje: Ruby.

Index