

INTRODUCCIÓN AL PATRÓN ARQUITECTÓNICO MVC

MVC, son las siglas de **modelo-vista-controlador** (o en inglés, model-view-controller), que es uno de los tantos **patrones de arquitectura de software**.

Antes de abordar de lleno este patrón, vamos a intentar hacer una introducción a la arquitectura de software, desmembrándola de lo general hacia lo particular, para al fin llegar al detalle, procurando entender exactamente de que se trata, en el contexto adecuado.

Probablemente, **este capítulo sea el más complejo** (y mucho más extenso en relación al Capítulo I). Pero si quieres poder aplicar de forma adecuada el patrón MVC, debes hacer el esfuerzo de seguirlo con especial interés y actitud “entusiasta”.

INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE

¿QUÉ ES LA ARQUITECTURA DE SOFTWARE?

Es necesario aclarar, que no existe una definición única, exacta, abarcativa e inequívoca de “arquitectura de software”. La bibliografía sobre el tema es tan extensa como la cantidad de definiciones que en ella se puede encontrar. Por lo tanto trataré, no de definir la arquitectura de software, sino más bien, de introducir a un concepto simple y sencillo que permita comprender el punto de vista desde el cual, este libro abarca a la arquitectura de software pero, sin ánimo de que ello represente “una definición más”.



A grandes rasgos, puede decirse que “la Arquitectura de Software es la forma en la que se organizan los componentes de un sistema, interactúan y se relacionan entre sí y con el contexto, aplicando normas y principios de diseño y calidad, que fortalezcan y fomenten la usabilidad a la vez que dejan preparado el sistema, para su propia evolución”.

Tal vez estés pensando “...al fin y al cabo, me haz dado una definición más...”. La respuesta es sí. Probablemente no pude contenerme de hacerlo – no lo niego -. Pero, **no debes tomar lo anterior como una definición**, sino como **la forma en la que en este libro, se abarca la arquitectura de software**.

TENDENCIAS DE LA ARQUITECTURA DE SOFTWARE

Si bien no puede decirse o mejor dicho, no es muy académico decirlo, voy a tomarme el atrevimiento de mencionar **solo dos tendencias arquitectónicas**, claramente diferenciadas entre sí:

- La **Arquitectura de Software Orientada a Objetos** (como “ingeniería” de sistemas)
- La **Arquitectura Estructurada** (como “desarrollo” de una aplicación)

Como podemos ver, en este libro, es a la primera a la cual nos enfocamos.

A fin de satisfacer la inquietud de curiosos (y de paso, no ocultar que existen), voy a mencionar la existencia de otras dos corrientes: la **arquitectura basada en patrones** y la **arquitectura basada en procesos y metodologías**.

Personalmente, **no las considero como cuatro corrientes** ¿Por qué? Porque independientemente de cual te agrade más, si la AS orientada a objetos o la AS estructural, implican necesariamente dos corrientes que deben aplicarse de forma optativa (o utilizas una o la otra pero no las dos simultáneamente), mientras que la AS basada en patrones y la AS basada en procesos, pueden utilizarse como complemento de las dos primeras. Pero esto, **es mera filosofía propia**, con la cual, no he encontrado demasiado consenso. Pues no los aburriré con ello e iremos a lo práctico.

CARACTERÍSTICAS DE LA ARQUITECTURA DE SOFTWARE: ATRIBUTOS DE CALIDAD

La **Calidad del Software** puede definirse como **los atributos implícitamente requeridos en un sistema que deben ser satisfechos**. Cuando estos atributos son satisfechos, puede decirse (aunque en forma objetable), que la calidad del software es satisfactoria.

Estos atributos, se gestan desde la arquitectura de software que se emplea, ya sea cumpliendo con aquellos requeridos durante la ejecución del software, como con aquellos que forman parte del proceso de desarrollo de éste.

Atributos de calidad que pueden observarse durante la ejecución del software

1. **Disponibilidad** de uso
2. **Confidencialidad**, puesto que se debe evitar el acceso no autorizado al sistema
3. Cumplimiento de la **Funcionalidad** requerida
4. **Desempeño** del sistema con respecto a factores tales como la capacidad de respuesta
5. **Confiabilidad** dada por la constancia operativa y permanente del sistema
6. **Seguridad externa** evitando la pérdida de información debido a errores del sistema
7. **Seguridad interna** siendo capaz de impedir ataques, usos no autorizados, etc.

Atributos de calidad inherentes al proceso de desarrollo del software

8. Capacidad de **Configurabilidad** que el sistema otorga al usuario a fin de realizar ciertos cambios

9. **Integrabilidad** de los módulos independientes del sistema
10. **Integridad** de la información asociada
11. Capacidad de **Interoperar** con otros sistemas (interoperabilidad)
12. Capacidad de permitir ser **Modificable** a futuro (modificabilidad)
13. Ser fácilmente **Mantenible** (mantenibilidad)
14. Capacidad de **Portabilidad**, es decir que pueda ser ejecutado en diversos ambientes tanto de software como de hardware
15. Tener una estructura que facilite la **Reusabilidad** de la misma en futuros sistemas
16. Mantener un diseño arquitectónico **Escalable** que permita su ampliación (escalabilidad)
17. Facilidad de ser **Sometido a Pruebas** que aseguren que el sistema falla cuando es lo que se espera (testeabilidad)

DE LO GENERAL A LO PARTICULAR: DEL ESTILO ARQUITECTÓNICO AL PATRÓN DE DISEÑO

Existe una diferencia entre **Estilo Arquitectónico**, **Patrón Arquitectónico** y **Patrón de Diseño**, que debe marcarse a fin de evitar las grandes confusiones que inevitablemente, concluyen en el mal entendimiento y en los resultados poco satisfactorios. Éstos, son los que en definitiva, aportarán “calidad” al sistema resultante. En lo sucesivo, trataremos de establecer la diferencia entre estos tres conceptos, viendo como los mismos, se relacionan entre sí, formando parte de un todo: la arquitectura de software.

RELACIÓN Y DIFERENCIA

Estilo Arquitectónico, Patrón Arquitectónico y Patrón de Diseño, representan, de lo general a lo particular, los niveles de abstracción que componen la Arquitectura de Software. En este sentido, puede decirse que:

- **El Estilo Arquitectónico** es el encargado de:
 - Describir la *estructura general* de un sistema, *independientemente de otros estilos*
 - Definir los componentes del sistema, su relación e interactividad
 - Ejemplos: *flujo de datos, llamada y retorno, etc.*
- **El Patrón Arquitectónico** es el nivel en el cual la arquitectura de software:
 - Define la *estructura básica* de un sistema, pudiendo estar *relacionado con otros patrones*
 - Representa una *plantilla de construcción* que provee un conjunto de *subsistemas* aportando las *normas para su organización*
 - Ejemplos: *Capas, MVC, Tuberías y Filtros, Pizarra, etc.*
- **El Patrón de Diseño** es el tercer nivel de abstracción de la arquitectura de software, cuya finalidad es la de *precisar en detalle los subsistemas y componentes* de la aplicación
 - Ejemplos: *Proxy, Command, Factory, etc..*

EL PATRÓN ARQUITECTÓNICO MODELO-VISTA-CONTROLADOR (MVC)

Habiendo dejado en claro de qué hablamos exactamente cuando nos referimos a “patrón arquitectónico”, estamos en condiciones de ahondar en los detalles del patrón MVC.

ACLARACIONES PREVIAS

Antes de caer inmersos en el conocimiento sobre el patrón MVC, quiero dejar en claro que en este libro, no hará referencia a ningún framework.

El objetivo de *POO y MVC en PHP* no es el de formar a programadores en el uso de frameworks, sino que el mismo, parte de una clara pregunta ¿a caso los frameworks no han

sido desarrollados por programadores? Tenemos la alternativa de utilizar frameworks para ahorrar tiempo de programación o, si realmente nos apasiona programar, adquirimos los conocimientos necesarios, tenemos la capacidad y nos animamos, podemos crear nuestros propios frameworks. De hecho, no se trata de reinventar la rueda, sino de crear una, adaptada a nuestras necesidades. Y si lo que

necesitamos es una rueda para acoplar a un mueble que acabamos de crear ¿cuál sería el sentido de intentar modificar una rueda de camión si tenemos todas las herramientas necesarias para ser “creadores” y no “modificadores”?

Claro que esto, depende de la pasión, gusto, capacidad y por qué no, de la ambición de cada uno. El tiempo, es discutible. Pues puede demandarte más tiempo modificar algo que ya existe, que crearlo. Si quieres reparar un mueble te llevará más tiempo repararlo que ir a comprar uno nuevo. Y si eres ebanista,

seguramente te llevará menos tiempo hacer un nuevo mueble que ir a comprarlo. Pues sabes exactamente lo que necesitas y como hacerlo y eso, hará que ahorres el tiempo de recorrer todas las mueblerías y termines comprando “el mueble que más se asemeje al que quieres”. El mueble, será similar, pero no exactamente igual al que tu imaginaste. De todas formas, la alternativa de modificar, siempre la tienes... al igual que también tienes la de crear. Todo dependerá del criterio de elección que apliques y sea cual sea, tu elección será la correcta, justamente porque habrás “elegido” y eso, es lo más importante.

¿QUÉ ES EL PATRÓN MVC?

El patrón MVC es un **patrón de arquitectura de software encargado de separar la lógica de negocio de la interfaz del usuario** y es el más utilizado en aplicaciones Web, ya que facilita la funcionalidad, mantenibilidad y escalabilidad del sistema, de forma simple y sencilla, a la vez que **permite** *“no mezclar lenguajes de programación en el mismo código”*.

MVC divide las aplicaciones en tres niveles de abstracción:

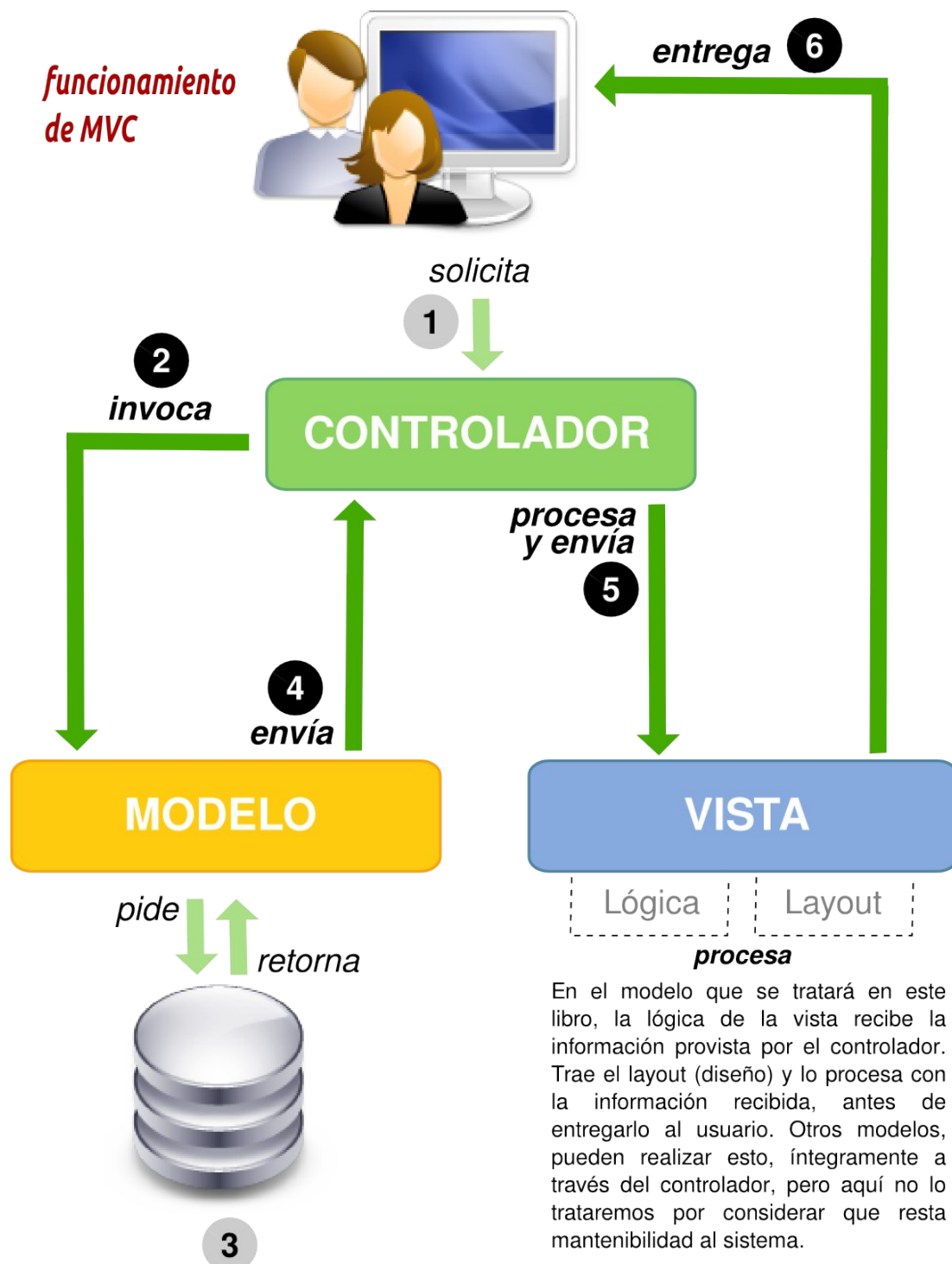
- **Modelo:** representa la lógica de negocios. Es el encargado de acceder de forma directa a los datos actuando como “intermediario” con la base de datos. Lo que en nuestro ejemplo de programación orientada a objetos, serían las clases DBAbstractModel y Usuario.
- **Vista:** es la encargada de mostrar la información al usuario de forma gráfica y “humanamente legible”.
- **Controlador:** es el intermediario entre la vista y el modelo. Es quien controla las interacciones del usuario solicitando los datos al modelo y entregándolos a la vista para que ésta, lo presente al usuario, de forma “humanamente legible”.

¿CÓMO FUNCIONA EL PATRÓN MVC?

El **funcionamiento básico del patrón MVC**, puede resumirse en:

- El **usuario realiza una petición**
- El **controlador captura el evento** (puede hacerlo mediante un manejador de eventos – *handler* -, por ejemplo)
- Hace la **llamada al modelo**/modelos correspondientes (por ejemplo, mediante una llamada de retorno – *callback* -) efectuando las modificaciones pertinentes sobre el modelo
- El **modelo será el encargado de interactuar con la base de datos**, ya sea en forma directa, con una capa de abstracción para ello, un Web Service, etc. y retornará esta información al controlador

- El controlador recibe la información y la envía a la vista
- La vista, procesa esta información pudiendo hacerlo desde el enfoque que veremos en este libro, creando una capa de abstracción para la lógica (quien se encargará de procesar los datos) y otra para el diseño de la interfaz gráfica o GUI. **La lógica de la vista, una vez procesados los datos, los “acomodará” en base al diseño de la GUI - layout – y los entregará al usuario** de forma “humanamente legible”.



Funcionamiento del patrón modelo-vista-controlador

Hasta aquí, hemos hablado de estilos arquitectónicos, nos hemos introducido en uno de los patrones arquitectónicos (MVC) pero, en todo esto **¿en qué momento intervienen los patrones de diseño?**

Es aquí donde debemos notar que éstos, intervienen en la forma en la que cada capa (modelo, vista y controlador), “diseña” su estructura. El controlador decidirá (aunque en realidad, nosotros lo haremos) si utilizará un *handler* para manejar los eventos del usuario. En ese caso, estaría optando por un **patrón de diseño**. Si para llamar al modelo, utiliza un *callback*, estaría utilizando otro, y así en lo sucesivo.

Personalmente, sugiero que, a no ser que se trate de sistemas realmente robustos y complejos, no se compliquen demasiado – **por ahora** - en “aplicar” todos y cada uno de los patrones de diseño que encuentren en el camino. La forma de hacerlo **bien**, es:

- Sencillez y simplicidad
- Fácil mantenibilidad
- Practicidad para evolucionar

Si se tienen en cuenta estas tres premisas, el resultado de un código limpio, legible y fácil de interpretar para cualquier programador, estará 90% asegurado.