**Model-View-Controller (MVC)**

Reconocido como estilo arquitectónico por Taylor y Medvidovic [TMA+95], muy rara vez

mencionado en los surveys estilísticos usuales, considerado una micro-arquitectura por Robert

Allen y David Garlan [AG97], el MVC ha sido propio de las aplicaciones en Smalltalk por lo

menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples. En

ocasiones se lo define más bien como un patrón de diseño o como práctica recurrente, y en

estos términos es referido en el marco de la estrategia arquitectónica de Microsoft. En la

documentación correspondiente es tratado a veces en términos de un estilo decididamente

abstracto [MS03a] y otras como patrón de aplicación ligado a una implementación específica en

Visual C++ o en ASP.NET [MS03b]. Buschmann y otros lo consideran un patrón correspondiente

al estilo de los sistemas interactivos [BMR+96].

Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y

mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en

el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la

interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir

ambas piezas para reducir la cantidad de código y optimizar la performance. Sin embargo, esta idea es antagónica al hecho de que la interfaz suele cambiar, o acostumbra depender de

distintas clases de dispositivos (clientes ricos, browsers, PDAs); la programación de interfaces de

HTML, además, requiere habilidades muy distintas de la programación de lógica de negocios.

Otro problema es que las aplicaciones tienden a incorporar lógica de negocios que van más allá

de la transmisión de datos.

 Fig. 3 - Model-View-Controller [según MS03a]

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la

presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes

[Bur92]:

- Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación,

responde a requerimientos de información sobre su estado (usualmente formulados desde la

vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- Vista. Maneja la visualización de la información.

- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista

para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases.

Esta separación permite construir y probar el modelo independientemente de la representación

visual. La separación entre vista y controlador puede ser secundaria en aplicaciones de clientes

ricos y, de hecho, muchos frameworks de interfaz implementan ambos roles en un solo objeto.

En aplicaciones de Web, por otra parte, la separación entre la vista (el browser) y el controlador

(los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho

más taxativamente definida.

Entre las ventajas del estilo señaladas en la documentación de Patterns & Practices de Microsoft

están las siguientes:

Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay

dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar

múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de

una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras

diferentes.

Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con

mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones

de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o

PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de

presentación generalmente no afecta al modelo. Este patrón sentó las bases para

especializaciones ulteriores, tales como Page Controller y Front Controller.

Entre las desventajas, se han señalado:

Complejidad. El patrón introduce nuevos niveles de indirección y por lo tanto aumenta

ligeramente la complejidad de la solución. También se profundiza la orientación a eventos

del código de la interfaz de usuario, que puede llegar a ser difícil de depurar. En rigor, la

configuración basada en eventos de dicha interfaz corresponde a un estilo particular

(arquitectura basada en eventos) que aquí se examina por separado.

Costo de actualizaciones frecuentes. Desacoplar el modelo de la vista no significa que

los desarrolladores del modelo puedan ignorar la naturaleza de las vistas.

Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una

lluvia de requerimientos de actualización. Hace pocos años sucedía que algunas vistas, tales

como las pantallas gráficas, involucraban más tiempo para plasmar el dibujo que el que

demandaban los nuevos requerimientos de actualización.