ACTIVIDAD 02 – PATRONES

PATRONES DE PRESENTACIÓN

DESARROLLO RÁPIDO DE APLICACIONES

Por Juan José Camacho Hidalgo
UNIVERSIDAD DE ALMERÍA

Contenido

1. Introducción a los patrones

Vamos a hablar de los diferentes patrones de presentación:

- **Diseño monolítico:** es un grupo de estructuras fijas, las cuales funcionan entre si. Está agrupado por capas. Se separa la interfaz de usuario, los componentes visuales, la lógica de negocio y el acceso a datos.

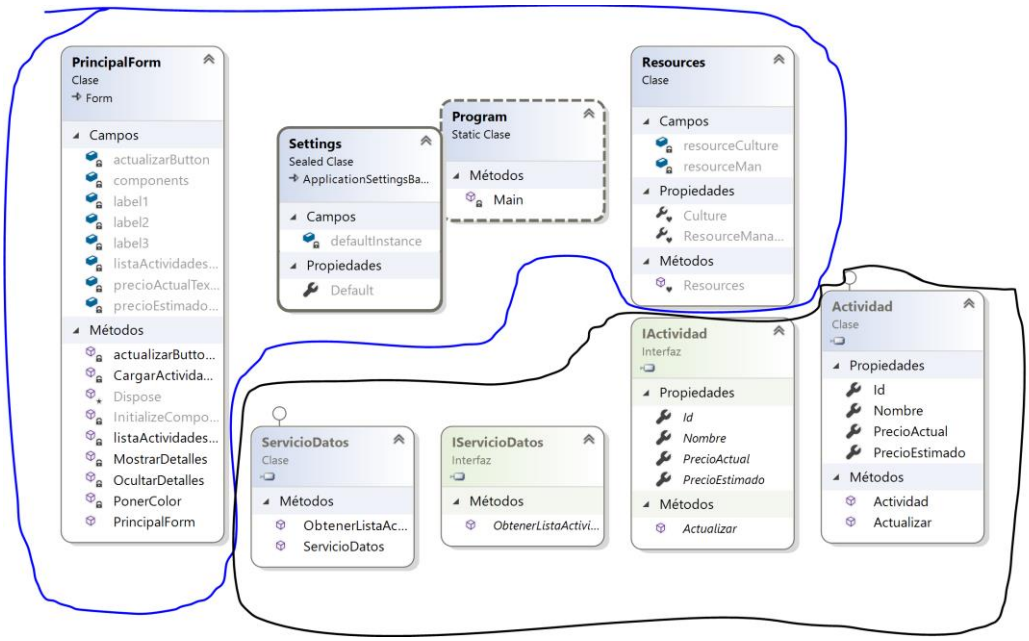
- **Patrón MVC (Modelo-Vista-Controlador):** Se trata de un patrón arquitectónico que separa en tres la lógica de una aplicación: el modelo, la vista y el controlador. Es decir, por una parte define componentes para la representación de la información, y por otro lado para la interacción del usuario. El modelo se encarga de los datos, la vista de la representación visual de los datos, y el controlador se encarga de recibir las ordenes del usuario y solicita los datos al modelo para comunicárselos a la vista.

- **Patrón MVP (Modelo-Vista-Presentador):** este patrón, dispone igual que el MVC de tres partes en la aplicación, pero de forma distinta: el modelo define los datos, la vista los exhibe aparte de las órdenes de usuario de las rutas (eventos) que proporciona al presentador. Por tanto referencia. El presentador actúa sobre ambos recuperando datos del modelo y se lo comunica a la vista formateándolos, por lo que la vista no tiene constancia de los datos originales.

- **Patrón MVVM (Modelo-Vista-Modelo de la Vista):** el patrón MVVM dispone de un modelo y de vista igual que en el patrón MVC, pero con la diferencia de que en vez de un controlador, dispone de uno o varios modelos de la vista. Esto quiere decir, que el comportamiento de la vista ya no estará en la vista, si no que esta queda fuera: la representación de esta queda en el modelo de la vista, que pueden haber varios distintos, sin afectar a esta. Implementa el comportamiento de la vista para responder a las acciones del usuario y exponer los datos del modelo a través de binding declarativo en la transferencia de datos desde y hacia el modelo a la vista.

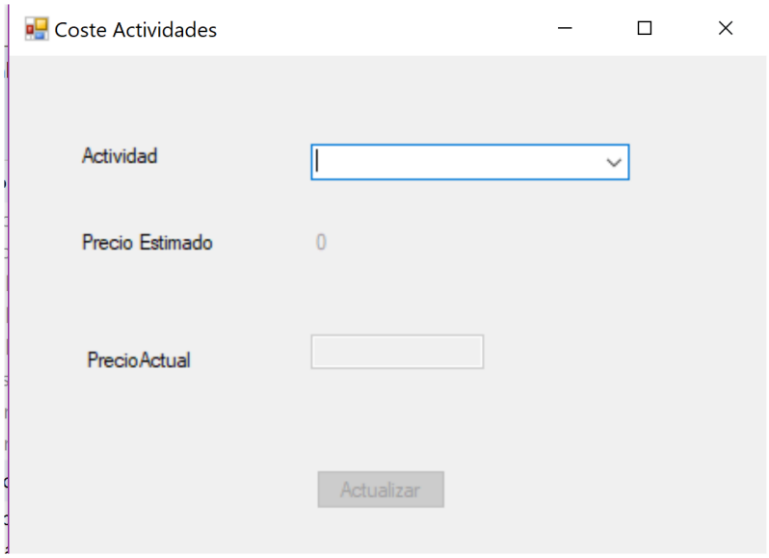
2. Comparativa de patrones

En la solución propuesta obtenemos los siguientes diagramas:

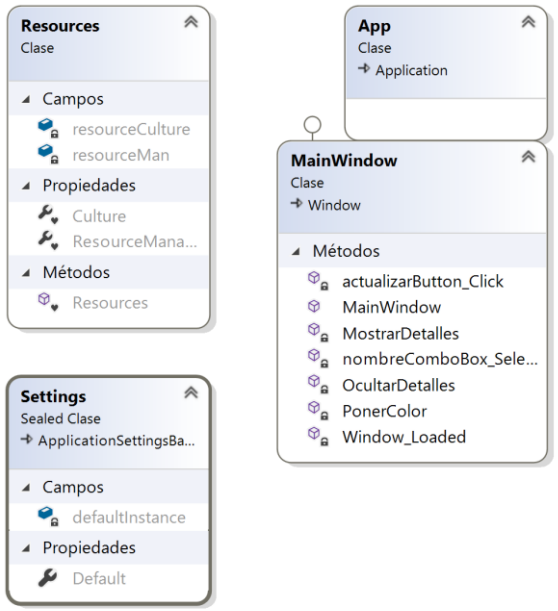


En este diagrama correspondiente a MonoliticoWFA se observa el diseño monolítico. Las líneas negras rodean a la biblioteca de clases de la que hace uso la aplicación (la biblioteca se llama CosteActividades). Esta incluye las clases Actividad y ServicioDatos, que definen el modelo, a través de sus interfaces. La aplicación general , rodeada de líneas negras, referencia a la biblioteca y hace uso de ella, por tanto es un todo que hace uso de una parte.

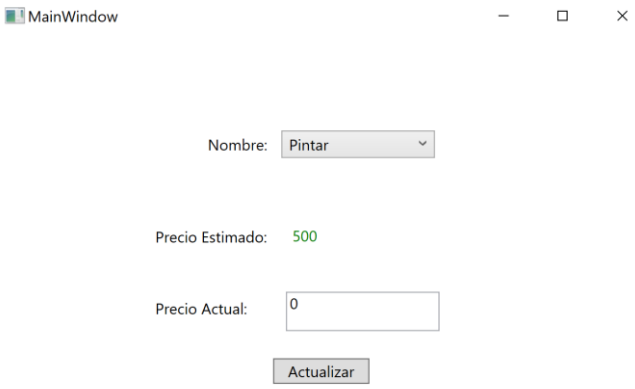
Aquí vemos la ejecución, con todo integrado:



Este segundo diagrama pertenece a MonoliticoWPFApp. Es exactamente el mismo diseño que el anterior pero en WPF. Por lo que en el XAML, tiene las referencias a la biblioteca de clases, que aporta los datos a la representación por así decirlo, integrándolos en un todo. Para ello hace uso de Resources.



Esta es la ejecución de este programa:



Ahora procedemos a comparar los otros tres proyectos, donde se muestran los patrones MVC, MVP y MVVM.

En los tres diagramas señalamos la vista, que es donde nos vamos a centrar en el análisis.

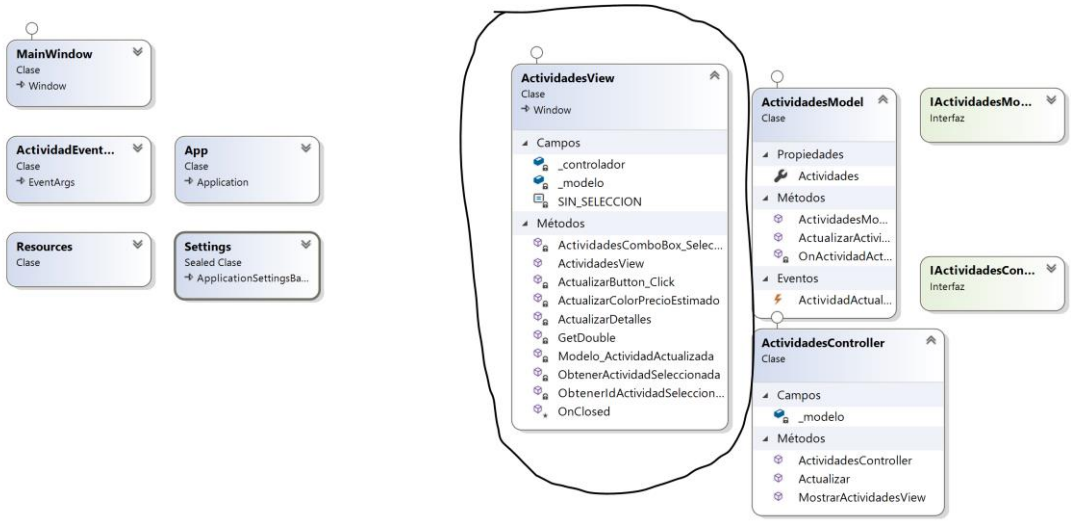


Ilustración 1. MVC_Layered

Como vemos en el patrón MVC, tenemos el modelo de datos definido, con sus propiedades y métodos (clase ActividadesModel), el controlador (ActividadesController), y la vista (ActividadesView). La vista contiene el XAML con toda la interfaz gráfica y además, todos los métodos que necesita para mostrarse al usuario como necesita.

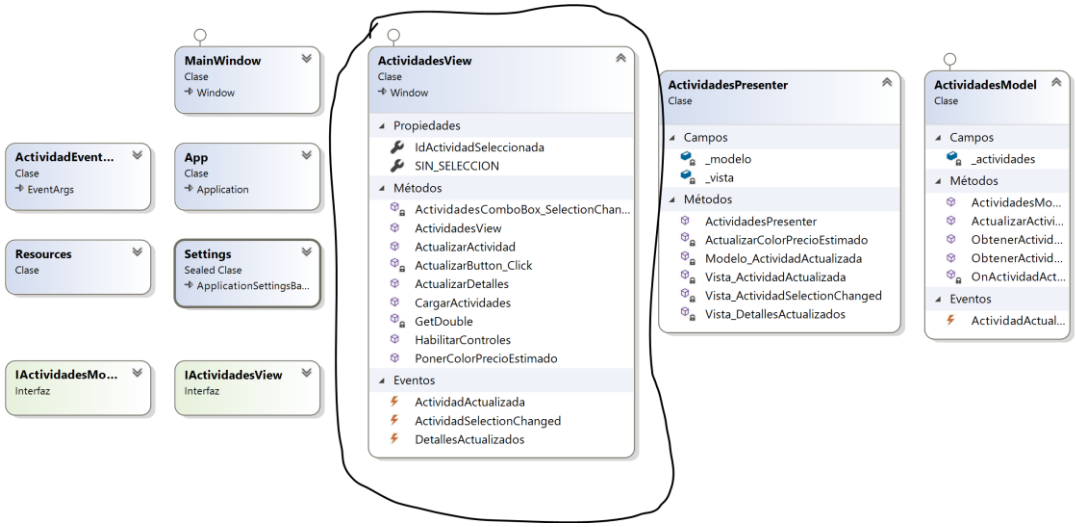


Ilustración 2. MVP_Layered

En cambio, si vamos al MVP, vemos como ActividadesView tiene menos métodos. Algunos métodos que antes tenia ActividadesView, ahora son parte del presentador. Esto quiere decir, que el presentador coge los estados de la vista, los métodos que realizan el comportamiento de la vista, y los hace suyos, para que la vista esté separada del comportamiento. Es decir, la vista ahora no puede coger los datos del modelo mediante el controlador y ofrecerlos así sin más. Ahora el presentador se los da modificados. Un ejemplo claro, que se aprecia visualmente, es el método ActualizarColorPrecioEstimado. Ahora forma parte del presentador, por lo que la vista recibe directamente el color según el precio.

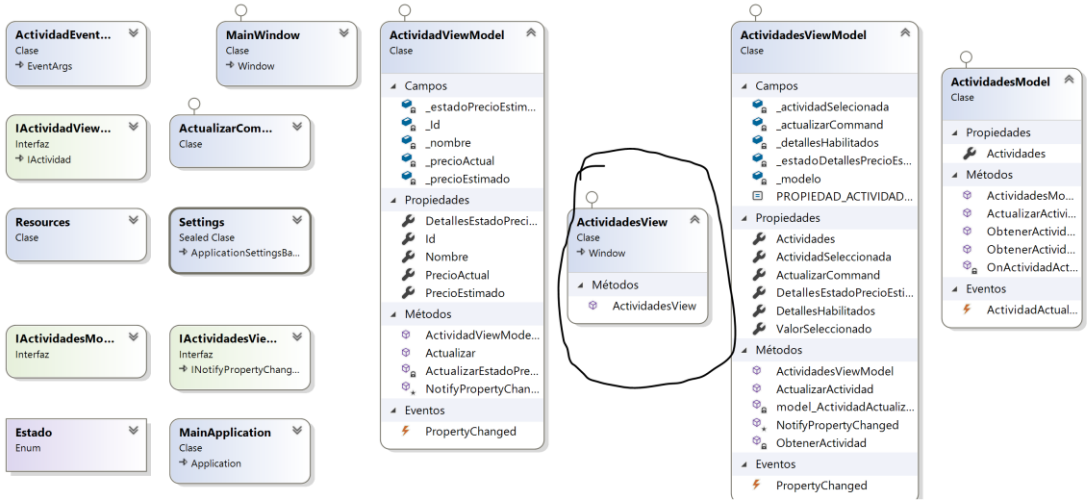


Ilustración 3. MVVM_Layered

Y por último, vemos como ActividadesView, en el patrón MVVM, no tiene métodos. Solo su constructor. Esto quiere decir que solo tiene el XAML que define la interfaz de usuario, pero no tiene nada referente a su comportamiento. Eso está en manos de los modelos de la vista: ActividadViewModel y ActividadesViewModel. Ambos conforman los comportamientos distintos que ofrece la vista al usuario, y disponen de los métodos necesarios para esto, como podemos apreciar. Muy similares, ya que actúan de forma parecida.

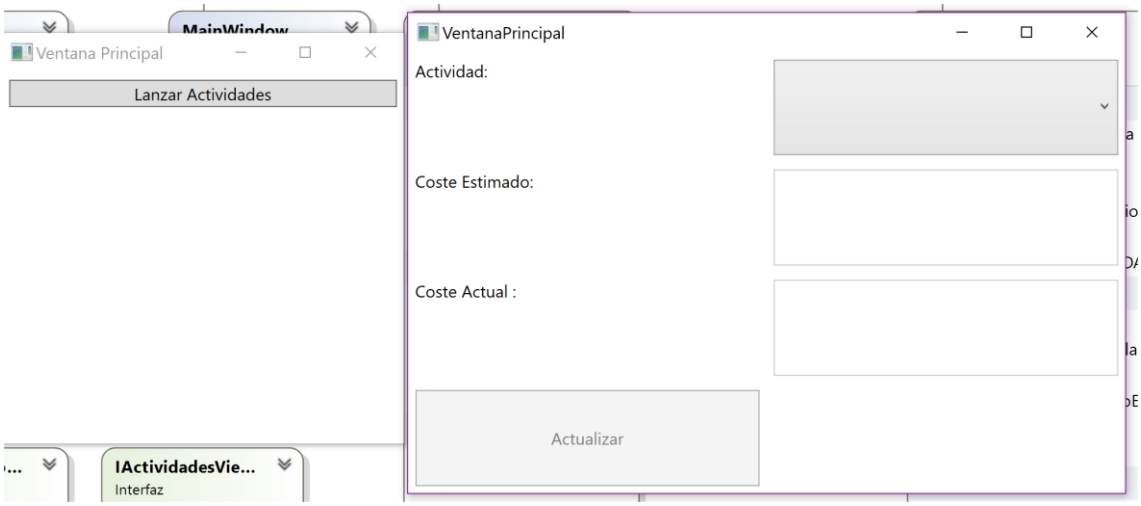


Ilustración 4. Ejecución de la app (común para todos los patrones)