# Implementación de MVVM para HID

## La ventana principal

Avisar a Helper que el MainForm es el actual (this)

*[FUTURO]Inicializa las vistas que componen la aplicación, éstas a su vez agregan las sub-vistas si es que corresponde*

Tiene asociada un View Model especifico del tipo MainViewModel. Este VM es mucho mas activo que los demas.

El VM se instancia mediante la factory ViewModelSource.Create()

Inicializa los servicios que utilizara dentro de la vista principal

Si es necesario capturar eventos del VM, hacerlo: por ejemplo eventos de cambios de vista activa

Tenemos que realizar el binding de los botones o item links con los comandos del VM (relativamente facil) 🡪 BindCommands()

Si aplica, agregar el binding de propiedades de controles con propiedades del VM. Conviene usar un BindingDataSource, es casi lo mismo que usar la fuente directa.

### OnLoad

Puntualmente en HID se llama al método TryLogin() del VM.

* Si el login es exitoso, se coloca StartMenu como TIPO de vista actual
* Si no es exitoso, coloca Ninguno como tipo de vista actual (implica NULL)

Otras alternativas:

* Establecer el TIPO de la vista por defecto para la ventana o bien si esta logueado colocar la StartView.
* Se puede llamar a un metodo de VM que sea RestoreDefaultView()

### ViewModelOnViewAdded

Si estoy incorporando una vista concreta (no null) seteo la propiedad Dock de la vista agregada en Fill (para que ocupe todo)

Si la vista que se esta agregando es tipo navegador (o sea una vista que implementa INavigableView y que tiene a lo sumo un ribbon pero ningún contenido propio), le aviso cual es el contenedor donde puede colocar sus work-views (método SetContainer(mainContent)) Con esto le digo que las W-V estarán ubicadas en el mismo lugar que las vistas comunes, pensar que podría ser otro también…

Si es una vista “normal”, seteo la propiedad Parent de la vista agregada con el panel principal de la vista principal

Por ultimo si la vista agregada soporta ribbon, hago el merge de la ribbon principal con la de la vista agregada.

Invoca al método SetMainRibbon() para setear en la vista cual es el ribbon principal (necesario para manejo de eventos de cambio de pagina)

Invoca al método BindEvents() que permite de ser el caso de ser necesario, setear eventos que tienen que ver con la ribbon (p ej SelectedPageChanged) y que no pueden diferirse ya que deben estar listos antes de mostrarla

Invoca al método FocusOnPage() de la vista agregada (que soporta ribbon) para que haga el foco en la pagina que quiera (puede ser la default o la ultima visitada)

### ViewModelOnViewRemoved

Seteo la propiedad Parent de la vista removida en null (hace que la vista sea invisible pero no la destruye de ninguna manera)

Luego si la vista soportaba ribbon, hago el unmerge del ribbon principal para dejar todo como estaba previamente

### Ribbon.OnSelectedPageChanged

Este evento tengo que controlarlo tanto en el formulario principal como en los navegables. Y tiene que estar bien complementado: cuando la pagina corresponde a la navegacion “principal”, tengo que manejarlo en el formulario principal. Si es de la navegacion secundaria, dejo el trabajo para la vista navegable. Lo importante es que NO hay que tratarlo en ambos lados porque en uno de ellos, dará error…

## El MainViewModel

Deriva de ViewModelBase.

Por esa razon, toda propiedad virtual publica declarada en esta clase, cuando se modifica, **llama automágicamente a un método bien definido**. Por ejemplo si la propiedad se llama **Texto**, el método es **OnTextoChanged** y tiene como argumento el valor previo de la propiedad.

Estas propiedades ademas son bindeables desde controles que esten en la vista, los cuales recibiran notificaciones cuando dichas propiedades se modifiquen.

Dentro del VM puedo definir metodos void sin argumentos a los que puedo asociar con comandos ya sea para botones tradicionales o item links de ribbon o status.

Tambien dentro del VM puedo definir métodos que retornen bool, sin argumentos y a los que puedo asociar con el CanExecute de WPF. No es necesario ningun seteo: si el método del comando se llama Procesar() entonces el método que retorna bool se debe llamar CanProcesar() y listo!

Como unico “inconveniente” esta el hecho de que al no existir CommandManager, tengo que avisar manualmente si las condiciones del CanXXX fueron cambiadas. Esto lo hacemos con el método RaiseCanExecuteChanged() al cual le paso como argumento una expresion que invoca al método del comando, ej: RaiseCanExecuteChanged( x => x.Procesar() ). Lo bueno es que todos los controles asociados a Procesar() van a cambiar su estado en virtud de lo que tengo ahora.

### SelectedViewType – OnSelectedViewTypeChanged

Cuando la propiedad virtual SelectedViewType se modifica, automágicamente se llama al método OnSelectedViewTypeChanged (el framework MVVM de DevExpress)

Lo que hago en este metodo es buscar mediante el IViewLocator la instancia que corresponde al tipo de la vista y setear la propiedad SelectedView.

Warning!! En el momento en que seteo la propiedad SelectedView con el resultado del View Locator se llama a OnSelectedViewChanged por lo que hay que tener mucho cuidado de no intentar hacer cosas raras en estos eventos. Creo que al contrario es una ventaja porque luego de setear SelectedView entonces ya tengo la vista actual totalmente estabilizada.

Si la vista soporta vista previa, seteo dicha vista previa con el argumento que me llega en la llamada al método (es el TIPO de la vista previa, no la vista en si…)

### SelectedView – OnSelectedViewChanged

Cuando la propiedad virtual SelectedView se modifica, automágicamente se llama al método OnSelectedViewChanged (DevExpress)

En el método, verifico si hay una vista anterior y si es asi invoco el evento ViewRemoved

Por ultimo invoco al evento ViewAdded con la vista actual

Ambos eventos son manejados supuestamente desde la vista asociada a este VM

## Las Vistas Comunes

Son User Controls (derivan de XtraUserControl) que poseen una ribbon y un contenido

Cuando un tipo de vista comun se selecciona como SelectedView en la ventana principal, si la vista implementa ISupportRibbon la ribbon se mezla con la principal y al area de contenido se le asocia como

## Las Vistas Navegables

Una vista navegable es una vista que no posee ningun contenido, excepto una posible ribbon cuyas paginas estaran asociadas a diferentes work views. Podria ser que una vista navegable NO TENGA ribbon? Habria que verlo… En el caso actual, la vista navegable a su vez implementa ISupportRibbon, con lo cual el cambio de vistas de trabajo se realizara con cambios de página del ribbon.

La vista navegable tiene su propio view locator (que normalmente seria una copia del locator global) ya que las vistas de trabajo son “internas” y la vista principal no tiene por que enterarse de su existencia.

El ctor de la vista navegable, como la vista principal, se encarga de asociar los eventos del view model de vista agregada y vista removida, la diferencia es que el cambio de vistas se hará mediante cambios de página del ribbon.

La diferencia con la vista principal es que el binding de comandos no se hace en el ctor sino en la primera activacion de la vista de trabajo, pasandole la ribbon

Una vista navegable deberia disponer minimamente de los siguientes campos:

|  |  |
| --- | --- |
| \_lastPage : string | guarda el nombre de la ultima pagina seleccionada del ribbon (si no tiene ribbon no seria necesario) |
| \_workViews : Dictionary< ViewType, IWorkView > | diccionario con las vistas de trabajo que manejara esta vista navegable |
| \_locator : IViewLocator | la instancia que se utiliza para buscar y construir las vistas de trabajo |
| \_parent : Control | el control que la vista principal nos cede para ubicar las vistas de trabajo |
| \_ribbon : RibbonControl | en caso de soportar ribbon, este campo deberia mantener una referencia a la ribbon principal (recordar que la que creamos en la vista no se utiliza sino que se crea una copia) |
| \_viewModel : XXXViewModel | referencia al view model que corresponde a esta vista |

### Ribbon en una vista navegable

Como particularidad, cada pagina del ribbon en una vista navegable se corresponde con una vista de trabajo. Para saber como se relacionan, la propiedad Tag de cada item RibbonPage debe contener el texto que representa el valor enum de ViewType que deberia activarse cuando la pagina se activa.

### Constructor

Como template, el ctor de una vista navegable deberia realizar los siguientes pasos:

1. Setear \_locator al argumento
2. Crear el ViewModel pasando \_locator como argumento del ctor
3. Enlazar los eventos WorkViewAdded y WorkViewRemoved del view model a dos metodos privados de la vista
4. Crear las vistas de trabajo
5. Enlazar los botones del ribbon con los comandos de las vistas de trabajo

### CreateWorkViews()

Instanciar el diccionario de vistas de trabajo. Usar locator.GetView() para recuperar una instancia de la vista concreta y agregarla al diccionario.

### BindCommands()

Para cada work view en el diccionario…

Recuperar la instancia que corresponde…

Asociar el boton de la ribbon mediante BindCommand(), usando como expresion el resultado del método GetActionFromName() de la propiedad ViewModel (obligatoria) de la work view. El nombre de la accion si bien puede ser cualquiera (y hasta podria tener espacios), normalmente deberia coincidir con el nombre de un Command en el ViewModel de la Work View…

### OnLoad

El método OnLoad de una vista navegable no deberia hacer nada ya que una vista navegable por definicion no tiene contenido visual

Setear una vista en este metodo solamente tendria sentido en la primer carga ya que luego no se invoca mas a OnLoad

De todas maneras, en las vistas navegables que tienen ribbon, se llama a un método de ajuste de pagina de la ribbon y se le da la oportunidad de setear la work view adecuada.

### Ribbon.OnSelectedPageChanged

Este evento tengo que controlarlo tanto en el formulario principal como en los navegables. Y tiene que estar bien complementado: cuando la pagina corresponde a la navegacion “principal”, tengo que manejarlo en el formulario principal. Si es de la navegacion secundaria, dejo el trabajo para la vista navegable. Lo importante es que NO hay que tratarlo en ambos lados porque en uno de ellos, dará error…

## El ViewModel de las Vistas Navegables

El view model de una vista navegable tiene las siguientes caracteristicas:

1. Deriva de ViewModelBase (para favorecer el binding)
2. Su ctor tiene como parametro un IViewLocator
3. Debe disponer de eventos de
   1. WorkViewAdded
   2. WorkViewRemoved
   3. SelectedWorkViewChanged
   4. SelectedWorkViewTypeChanged
4. Propiedades
   1. SelectedWorkViewType : ViewType
   2. SelectedWorkView : object
5. Métodos
   1. SetCurrentWorkViewType(ViewType vt)
   2. SetDefaultView() ???
6. wew

SetCurrentWorkViewType(ViewType) 🡪

## IWorkView -- Las Vistas de Trabajo

Las vistas de trabajo son controles (vistas) que solamente pueden aparecer dentro de una vista navegable. La diferencia es que el cambio de vistas de trabajo se efectua mediante eventos de cambio de página en el ribbon que esta alojado en la vista navegable.

La interface IWorkView obliga a implementar dos miembos:

IWorkViewModel ViewModel {get; set; }

void BindCommands(ISupportRibbon parent);

En el ctor de la work view debemos instanciar un objeto tipo IWorkViewModel. Dicho objeto se retorna en la propiedad ViewModel (de IWorkView). Este VM tiene como uso el enlace de los diferentes comandos de la ribbon

## IWorkViewModel – ViewModel de trabajo

Cada vista de trabajo tendra su View Model que contiene los comandos que se bindearan al ribbon de la vista navegable. Para eso disponemos de una region “Comandos Work Views” donde tenemos que ubicar todos estos comandos.

Tambien el método GetActionFromName() que es obligatorio en el VM

ISupportPreviousView

Esta interface obliga a una vista a tener una propieda PreviousViewType (por que no previous view? creo que mejor view type para que la manera de setear la nueva vista sea la standard, o sea el tipo y luego la vista en si).

En algun momento, como respuesta a algun control visual, se llama al metodo CloseView() de MainViewModel, este metodo toma la SelectedView, se fija si tiene vista previa, y en tal caso la reemplaza

Manejo del SearchControl

El search control esta siempre disponible en la vista principal.

Mi idea es que cuando una vista se activa, si necesita el control, se lo avise para que el mismo actue sobre dicha vista (ojo que despues puede haber distintas workviews…)

Si no lo necesita, pero tampoco tiene que deshabilitarlo, entonces no hace nada

Si tiene que deshabilitarlo entonces tambien tiene que avisarle para que no pueda utilizarse

Otra cosa: cuando use el search, todas las inputs de cualquier vista deberian dirigirse al control (si no hay un control especifico enfocado) Ojo esto no es facil, inicialmente podemos hacer que haya que enfocar el search para que pueda usarse. Tampoco matarse por esta feature

