

ARQUITECTURA DE SOFTWARE

Según las características de software y los requisitos de software levantados en secciones anteriores se tomó la decisión de utilizar la herramienta WPF de Microsoft para el desarrollo de aplicaciones de escritorio. Ya con estos elementos procedemos a listar las herramientas que se utilizarán en el desarrollo de software.

1. Herramientas de desarrollo

1.1. Visual Studio

Entorno de desarrollo de Microsoft compatible con múltiples lenguajes de programación como C++, C#, Java, Python y muchos más, al igual que entornos de desarrollo web. Para este proyecto en específico se utilizará Visual Studio 2017 Profesional.

1.2. WPF

Windows Presentation Foundation (WPF) es una tecnología de Microsoft, presentada como parte de Windows Vista. Permite el desarrollo de interfaces de interacción en Windows tomando características de aplicaciones Windows y de aplicaciones web. WPF ofrece una amplia infraestructura y potencia gráfica con la que es posible desarrollar aplicaciones visualmente atractivas, con facilidades de interacción que incluyen animación, vídeo, audio, documentos, navegación o gráficos 3D. Separa, con el lenguaje declarativo XAML y los lenguajes de programación de .NET, la interfaz de interacción de la lógica del negocio, propiciando una arquitectura Modelo Vista Controlador para el desarrollo de las aplicaciones.

1.3. Prism

Prism es un marco para crear aplicaciones de XAML débilmente acopladas, sostenibles y comprobables en WPF, Windows 10 UWP y Xamarin Forms. Las versiones separadas están disponibles para cada plataforma y se desarrollarán en líneas de tiempo independientes. Prism proporciona una implementación de una colección de patrones de diseño que son útiles para escribir aplicaciones XAML bien estructuradas y mantenibles, que incluyen MVVM, inyección de dependencias, comandos, EventAggregator y otros. La funcionalidad principal de Prism es una base de código compartida en una biblioteca de clases portátil dirigida a estas plataformas. Las cosas que deben ser específicas de la plataforma se implementan en las bibliotecas respectivas para la plataforma de destino.

1.3.1. DelegateCommand

Además de proporcionar acceso a los datos que se mostrarán o editarán en la vista, ViewModel probablemente definirá una o más acciones u operaciones que puede realizar el usuario. Las acciones u operaciones que el usuario puede realizar a través de la interfaz de usuario generalmente se definen como comandos. Los comandos proporcionan una forma conveniente de representar acciones u operaciones que pueden vincularse fácilmente a los controles en la

interfaz de usuario. Encapsulan el código real que implementa la acción u operación y ayudan a mantenerlo desacoplado de su representación visual real en la vista. Definición de un Delegatecommand:

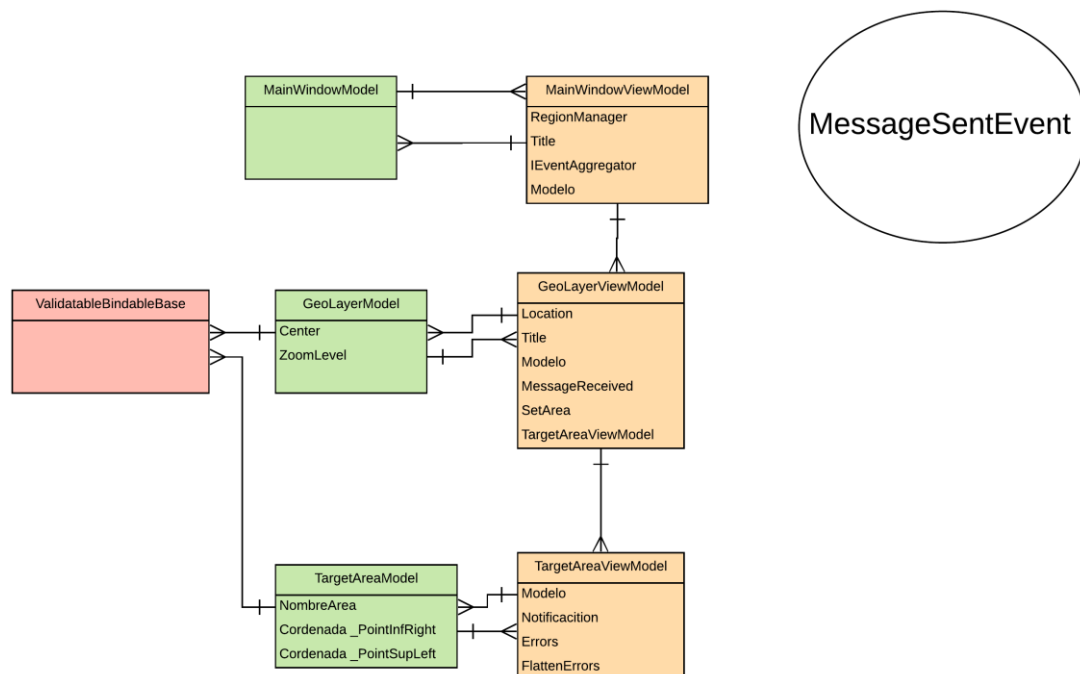
```
public DelegateCommand SubmitCommand { get; private set; }

public ArticleViewModel()
{
    SubmitCommand = new DelegateCommand(Submit, CanSubmit);
}
```

1.4. Material Design In XAML

Es una biblioteca de interfaces de diseño de materiales más completas y fáciles de usar en cualquier plataforma. Con Material Design En XAML Toolkit, puede dar vida a hermosas aplicaciones de escritorio con un lenguaje de diseño moderno y popular. De código abierto y una de las bibliotecas GUI más populares para WPF

2. Patrón de diseño MVVM



En la figura se ilustra cómo se implementó el patrón de diseño MVVM en el desarrollo de los casos de uso CU-PMR-01 y CU-PMR-02, de este modo MainWindowViewModel implementará la navegación entre las vistas con la propiedad RegionManager, GeoLayerViewModel será la

vista modelo donde centralizará la información de los modelos TargetAreaViewModel, RadarDevicesModel y demás modelos con la idea de contener la información de los radares, blancos y área en la vista modelo GeoLayerViewModel donde se encuentra el manejo del mapa geográfico.

Todos los modelos que tengan un ingreso de datos por medio de la interfaz de usuario implementaran de ValidatableBindableBase clase que implementa las interfaces para la validación de los datos en los formularios de ingreso de datos. La clase MessageSentEvent se encontrará en la raíz del proyecto y podrá ser accesible por todas las viewmodel y será la encargada de implementar los eventos para comunicar los datos entre viewmodel.

Teniendo en cuenta el patrón de diseño anteriormente mencionado se definirán algunas reglas y excepciones a la hora de programar las funcionalidades del software PRORAM, con el fin de dar orden y entendimiento al código desarrollado y así poder ser más fácil de reutilizar y mantener con el paso del tiempo.

Reglas

- Todas las vistas terminaran con el sufijo View
- Todos las viewmodel terminaran con el sufijo ViewModel
- Todos los modelos terminaran con el sufijo Model
- Los archivos creados durante el proceso de programación deberán llevar documentación de sus clases y sus métodos.
- No se permitirá código comentado
- No se permitirá la asignación de nombre aleatorios y/o sin sentido
- Los commit realizados llevaran un mensaje claro sobre el contenido del mismo
- No se permitirá ejecución de múltiples bloques de ejecución en una sola línea de código (Por legibilidad)
- Las vistas no tendrán ninguna referencia al modelo más allá de la necesaria para la implementación del patrón de diseño
- Se evitará el código en el code behind de las vistas
- Los métodos serán escritos en Pascal case y las instancias y objetos en Camel case
- El nombre de un objeto será idéntico a la clase instanciada, pero en Camel case

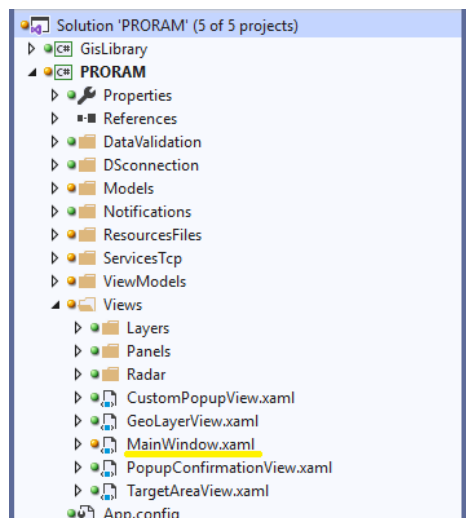
Excepciones

- El patrón de diseño se podrá evitar siempre y cuando la implementación sea ineficiente al desarrollo de las funcionalidades
- El código en el code behind será permitido cuando no se esté utilizando el patrón de diseño

2.1. Comportamiento consola PRORAM

En esta sección se explica el comportamiento de la solución para la consola **PRORAM** como fue desarrollado siguiendo el patrón de diseño MVVM, las vistas, modelos y vista-modelos creados para la solución. La aplicación esta dividida en el patrón de diseño MVVM como ya se mencionó anteriormente, entre las cuales se pueden destacar 3 grandes secciones, la vista principal, la vista geográfica y las vistas modal de añadir y configurar dispositivos radar.

La vista principal *MainWindow* ubicada en la solución de software del proyecto PRORAM en la raíz de la carpeta *Views* podremos encontrar la vista *MainWindow*, La *MainWindow* contiene la primera imagen de la consola **PRORAM** y contiene la mayoría de los paneles y llamado vistas modal.



Empezando con el menú donde se puede llamar a las vistas modal: *LayersView*, *TargetAreaView*, *RadarDevicesView*, *PopupConfirmationView*, *LayersView*. Paneles de la *MainWindow*: *RightPanelView*, *SidePanelsView*, *TargetPanelView*, *LogsView* y *GeoLayerView*.

Nota: más adelante se definirán el funcionamiento de estas vistas

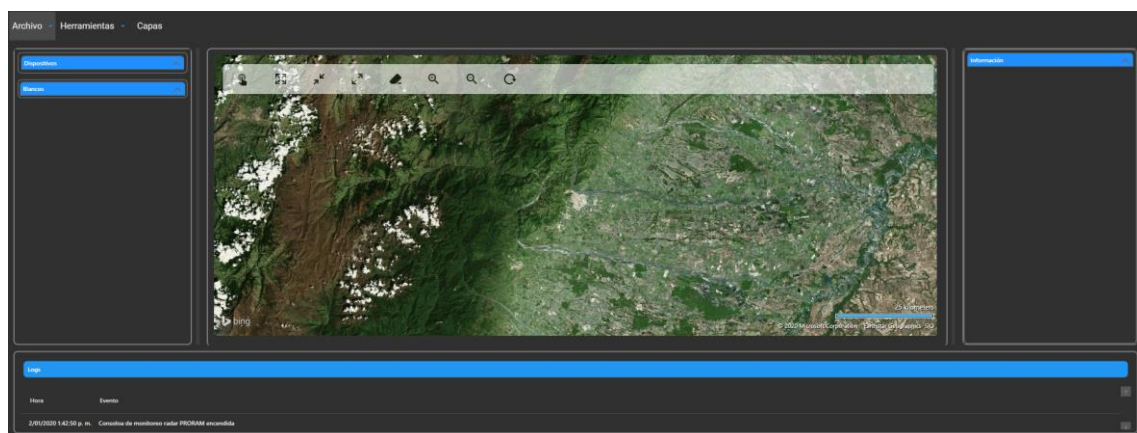


Figura Vista *MainWindow*

Llamado de los paneles en la *MainWindow*:

Paneles: *SidePanelsView* & *TargetPanelView*

```
<RowDefinition Height="auto" />
</Grid.RowDefinitions>
<ContentControl Grid.Row="0" prism:RegionManager.RegionName="SidePanels"/>
<ContentControl Grid.Row="1" prism:RegionManager.RegionName="TargetPanel"/>
</Grid>
```

Panel: *GeoLayerView*

```

<!--Capa geografica-->
<ContentControl Grid.Column="2" prism:RegionManager.RegionName="GeoLayer" prism:RegionManager.RegionContext="{Binding TargetAreaMod}"
Margin="10,10,5,0" MinWidth="300"/>

```

Panel: *RightPanelView*

```

<ContentControl Grid.Column="4">
    <Border CornerRadius="6" BorderBrush="Gray" BorderThickness="2" DockPanel.Dock="Top" Margin="10,10,10,0">
        <ContentControl prism:RegionManager.RegionName="RightPanel"/>
    </Border>
</ContentControl>

```

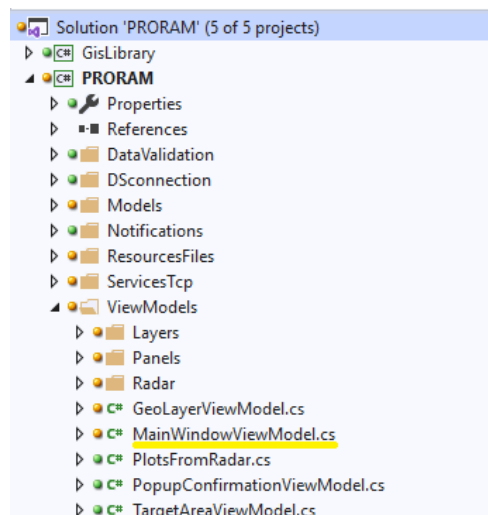
Panel: *LogsView*

```

<Border Grid.Row="2" CornerRadius="6" BorderBrush="Gray" BorderThickness="2" DockPanel.Dock="Top" Margin="10">
    <ContentControl prism:RegionManager.RegionName="LogsView" Margin="5" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
</Border>

```

MainWindowViewModel es la vista-modelo que hace referencia el patrón de diseño MVVM y está ubicada en la solución de software del proyecto PRORAM en la carpeta *ViewModels* en la raíz de esta carpeta.



MainWindowViewModel contiene la interacción con la vista *MainWindow*, los llamados a las vistas modal del menú anteriormente comentados, por medio de los métodos: *RaiseRadarInteraction*, *RaiseLayersNotification*, *RaiseTargetInteraction* y a las vistas de notificación *RaiseCustomPopup*.

Los métodos que hacen llamados a vistas modal implementan una clase que implementa una interface de la vista con una interface a *IConfirmation*.

Para la creación de una nueva ventana modal se debe crear una *interface* que contenga en sus propiedades el modelo de la vista que quiere invocar o los parámetros que necesarios para su funcionamiento, la *interface* de la vista deberá utilizar la *interface IConfirmation* y crear una clase que implemente la *interface* de la vista modal, con todo esto ya listo podemos crear las propiedades en el *ViewModel* primero una propiedad del tipo *DelegateCommand* que se encarga de capturar las interacciones de la vista y la segunda la propiedad *InteractionRequest* con tipo de la interface que se creó anteriormente. Y por medio de la interacción del usuario en la vista y por medio de los *DelegateCommand* se hace el llamado a la propiedad *InteractionRequest* que llama al método *Raise* y se produce el llamado a la vista, a continuación, se muestra porciones de código donde se realiza el proceso anteriormente mencionado.

```

/// <summary>
/// Interface IRadarDevicesNotification para la implementación de la ventana Dispositivos radar
/// </summary>
6 references
public interface IRadarDevicesNotification: IConfirmation
{
    3 references
    RadarDevicesModel RadarDevicesModel { get; set; }

    4 references
    Location Point1 { get; set; }

    4 references
    Location Point2 { get; set; }
}

```

```

/// <summary>
/// Clase RadarDevicesNotification implementa la interface IRadarDevicesNotification
/// </summary>
2 references
public class RadarDevicesNotification: Confirmation, IRadarDevicesNotification
{
    public RadarDevicesModel RadarDevicesModel_ = new RadarDevicesModel();

    1 reference
    public RadarDevicesNotification()
    {
        RadarDevicesModel_ = null;
        this.Point1 = new Location();
        this.Point2 = new Location();
    }

    4 references
    public Location Point1 { get; set; }
    4 references
    public Location Point2 { get; set; }
    3 references
    RadarDevicesModel IRadarDevicesNotification.RadarDevicesModel { get; set; }
}

```

```

2 references
public InteractionRequest<IRadarDevicesNotification> RadarDevicesNotificationRequest { get; set; }
1 reference
public DelegateCommand RadarDevicesNotificationCommand { get; set; }

```

```

1 reference
private void RaiseRadarInteraction()
{
    _titulo = "Mensaje de notificación ";
    _mensaje = "Necesita definir un área objetivo para poder agregar un dispositivo radar ";

    if (GeoLayerModel_.DefinedMap == true)
    {
        var p1 = new Location() { Latitude = TargetAreaMod.LatitudP1.Value, Longitude = TargetAreaMod.LongitudP1.Value };
        var p2 = new Location() { Latitude = TargetAreaMod.LatitudP2.Value, Longitude = TargetAreaMod.LongitudP2.Value };

        RadarDevicesNotificationRequest.Raise(new RadarDevicesNotification { Title = "Dispositivos radar", Point1 = p1, Point2 = p2 }, r =>
        {
            if (r.Confirmed && r.RadarDevicesModel != null)
            {
                RadarDevicesModel_ = r.RadarDevicesModel;
            }
            else
            {
                Title = "PRORAM Consola de monitoreo";
            }
        });
    }
    else
    {
        RaiseCustomPopup();
    }
    _titulo = string.Empty;
    _mensaje = string.Empty;
}

```

La vista geográfica GeoLayerView está ubicada en la solución de software del proyecto PRORAM en la raíz de la carpeta Views, es el área de mayor confluencia de información donde se centraliza la información de los dispositivos radar, las capas geográficas, la información de Plots y Tracks. La vista GeoLayerView está contenida en la vista principal MainWindow he igual que la mayoría de las vistas las interacciones de la vista son manejadas por el su ViewModel, la vista

GeoLayerView cuenta con 2 principales componentes la mayor de ellas es la capa geográfica donde se visualiza el mapa geográfico de **Bing Maps** que ocupa toda la pantalla de la vista en donde se visualizara el área objetivo de vigilancia, los dispositivos radar, y los blancos detectados por los dispositivos radar, el segundo componente principal es la barra de herramientas que se encuentra en la parte superior y contiene las herramientas típicas de zoom centrar, dibujar y borrar líneas sobre la capa geográfica, estas herramientas son activadas por el usuario y serán controladas en el *ViewModel* por medio de *DelegateCommand*.

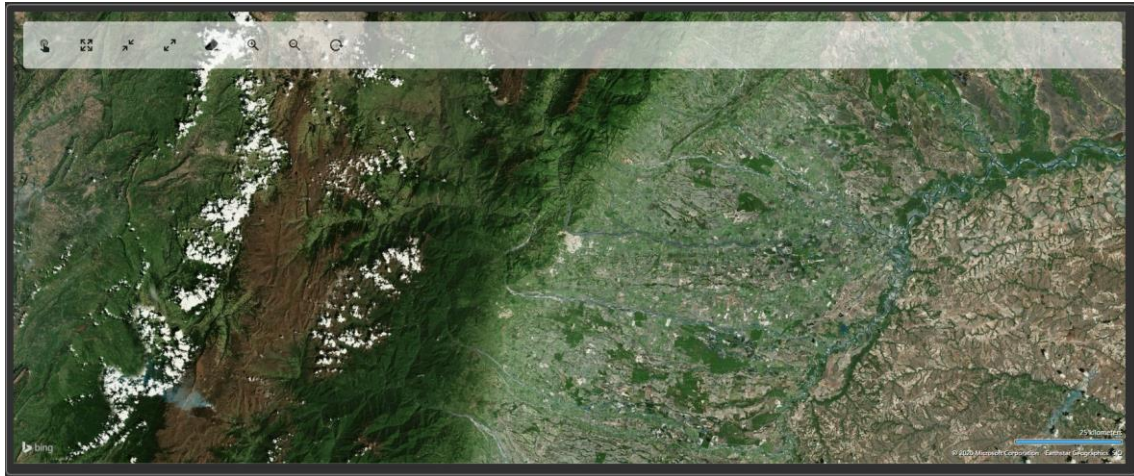


Figura Vista - GeoLayerView

GeoLayerViewModel ubicada en la carpeta *PRORAM > ViewModels* es la vista modelo que controla la interacción con la vista *GeoLayerView* y centraliza la información de la aplicación datos de los dispositivos radar, métodos de visualización blancos Tracks y Plots, manejo de las herramientas en el mapa geográfico y las rutinas de conexión al dispositivo radar de las cuales hablaremos a continuación. *GeoLayerViewModel* contiene una propiedad que es una colección de objetos de conexión *ObservableCollection<ServiceTcp>* *CollectionServicesTcp* esta colección almacenara un objeto de tipo *ServiceTcp*, que contiene las rutinas para la comunicación entre la consola PRORAM y el dispositivo radar, también posee una propiedad de tipo *RadarDevicesModel* que contiene la información del radar al que se va a comunicar.

La vista *RadarDevicesView* y su vista-modelo *RadarDevicesViewModel* son las encargadas de administrar los dispositivos radar agregados. Podemos acceder a la vista *RadarDevicesView* por medio del menú de la vista *MainWindow > Archivo > Dispositivos Radar* y en la solución del proyecto se puede encontrar en *PRORAM > View > Radar*, en esta vista tenemos la opción de agregar radar y la cual nos lleva a la vista *RadarConfigurationView*(Registrar dispositivo radar) en la cual podemos configurar un nuevo dispositivo radar que se desee agregar esta vista esta repleta de campos necesarios para la correcta comunicación y visualización de los blancos.

Dispositivos radar

Agregar radar

Id	Nombre	Azimuth	Coordenadas	Estado conexión	Radiando	Acciones
1	radar 1	2	4,073555 -73,584603 0			Opciones

Cerrar

Figura Vista RadarDevicesView

Registrar dispositivo radar

Nombre del radar (*):

Ingrese el nombre del radar

¿Que modelo radar desea agregar? (*)

Seleccione el tipo de dispositivo

Ingrese una Ip o HostName (*)

Ingrese la ip

Validar

Configuración

Altura del dispositivo (*) (m):

Ingrese la altura (m)

El campo Altitud es obligatorio.

Potencia de transmisión (*) (%):

Ingrese la potencia

El campo Potencia de transmisión es obligatorio.

Angulo de instalación (*) (°):

Ingrese el angulo de instalación (°)

El campo Ángulo de instalación es obligatorio.

Angulo azimut (*) (°):

Ingrese el Azimut (°)

El campo Azimuth es obligatorio.

OffsetXDistance (*) (m):

Ingrese offset distancia en X (m)

OffsetYDistance (*) (m):

Ingrese el offset distancia en Y (m)

Seleccione canal de frecuencia (*):

Seleccione el canal de transmisión

El campo Canal de frecuencia es obligatorio.

Angulo con respecto al norte (*):

Ingrese el angulo con respecto al norte (°) (°)

El campo NorthHeiding es obligatorio.

Coordenadas del dispositivo radar

Latitud (*):

Ingrese la latitud

El campo Latitud es obligatorio.

Longitud (*):

Ingrese la longitud

El campo Longitud es obligatorio.

Cancelar

Aceptar

Figura Vista RadarConfigurationView

La vista modelo *RadarDevicesViewModel* ubicada en *PRORAM > ViewModels > Radar* además de controlar las interacciones con la vista *RadarDeivcesView* genera los eventos de conexión desconexión, encendido y apagado de radiación, estos eventos serán controlados por la *ViewModel GeoLayerViewModel* que controla las comunicaciones con el dispositivo radar.

2.2. Edición de la consola PRORAM

En esta sección se especificará como se deberán hacer las ediciones de la solución de software **PRORAM** tanto el estilo visual como la codificación para que la aplicación mantenga uniformidad y pueda ser fácilmente escalable y reutilizable, manejaremos 3 temas principales de modificación de la solución de software PRORAM estilo y visualización, codificación de nuevas rutinas o métodos y modificación de modelos.

2.2.1. Estilo y visualización

Los temas y estilos definidos en la aplicación de la librería de estilos Material Design se pueden encontrar en el archivo *App.xaml* en la raíz del proyecto **PRORAM**, estos estilos están definido como recursos de la aplicación y en la mayoría de los casos no será necesario volver a definirlos en los componentes ya que por defecto los componentes básicos están configurados con estos estilos.

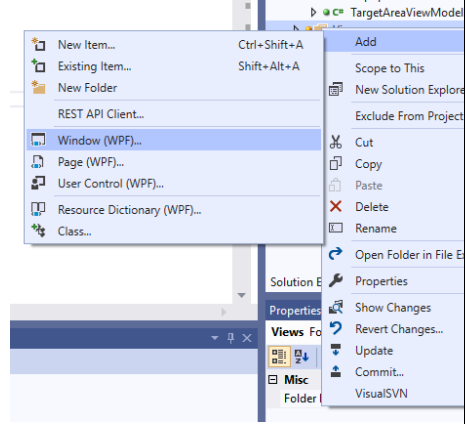
```
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>

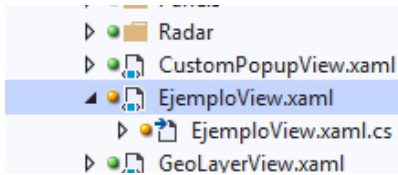
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Button.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.CheckBox.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.ListBox.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.PopupBox.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.RadioButton.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.TextBlock.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.ToggleButton.xaml" />

      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Light.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Primary/MaterialDesignColor.Blue.xaml" />
      <ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Accent/MaterialDesignColor.Lime.xaml" />

    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Application.Resources>
```

Las vistas esta diseñadas en XAML y con la librería de diseños Material Design se le da el estilo actual de la aplicación, a continuación, se muestra el proceso de creación de una vista:

Pa so	Descripción	Imagen
1	<p>Crear una vista clic derecho sobre el área deseada y seleccionar <i>Añadir > Window (WPF)</i>.</p> <p><i>Nota: se recomienda seleccionar <u>Window(WPF)</u> así la vista sea <u>User Control(WPF)</u> más adelante explicaremos el porqué de esta recomendación.</i></p> <p><i>Nota: las vistas tienen que ser creadas dentro de la carpeta Views.</i></p>	

2	Renombrar la nueva vista con la terminación <i>View</i> , debido a la convención del patron de diseño MVVM.	
3	<p>La única vista tipo Window en la aplicación es la MainWindow las otras serán de tipo UserControl (WPF) asi que modificamos el código de la vista por medio de sus etiquetas y cambiamos el tipo a UserControl (WPF).</p> <p><i>Nota: creamos las vistas como de tipo <u>Window</u> para que nos cree los demás objetos asociados a la vista como el <u>Code Behing</u>.</i></p>	<pre> 1 <UserControl x:Class="PRORAM.Views.EjemploView" 2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" 3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" 4 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility" 5 xmlns:local="clr-namespace:PRORAM.Views" 6 mc:Ignorable="d" 7 Height="450" Width="800"> 8 <Grid> 9 10 11 </Grid> 12 </UserControl> </pre>
4	<p>Agregamos las referencias a la librería de estilos <i>Material Design</i>, que tiene que ser igual para todas las vistas.</p> <p><i>Nota: Si la vista es necesita un ViewModel asociado agregar la línea de código <u>"prism:ViewModelLocator.AutoWireViewModel=True"</u>.</i></p>	<pre> 1 <UserControl x:Class="PRORAM.Views.EjemploView" 2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" 3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" 4 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility" 5 xmlns:local="clr-namespace:PRORAM.Views" 6 xmlns:prism="http://prismlibrary.com/" 7 xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes" 8 Background="Transparent" 9 TextElement.Foreground="{DynamicResource MaterialDesignBody}" 10 TextElement.FontWeight="Medium" 11 TextElement.FontSize="14" 12 Width="auto" 13 Height="auto" 14 prism:ViewModelLocator.AutoWireViewModel="True" > 15 <Grid> 16 17 </Grid> 18 </UserControl> </pre>
5	<p>Los campos de entrada de datos deben estar enlazados a un modelo de datos previamente definido y validación para su correcto funcionamiento, importante que los campos de entrada de datos contengan su <i>"Binding"</i> seguido de la propiedad especifica ya que, con el <i>AutoWireViewModel</i> se hace en enlace entre la <i>View</i> y su <i>ViewModel</i>.</p> <p><i>Nota: más adelante se especificará la definición de modelos y validación de datos.</i></p>	<pre> <StackPanel Margin="10"> <Label Name="Label1" Target="{Binding ElementName=NombreArea}" /> <TextBox TabIndex="1" Name="NombreArea" materialDesign:HintAssist.Hint="Ingrese el nombre del área" Text="{Binding TargetAreaMod.NombreArea, Mode=TwoWay, ValidatesOnDataErrors=True, NotifyOnValidationError=True, ValidatesOnExceptions=False, UpdateSourceTrigger=PropertyChanged}" Background="DimGray" BorderBrush="DimGray" BorderThickness="2" Margin="10"> <TextBox.Effect> <DropShadowEffect ShadowDepth="2" Color="white" Opacity=".4" /> </TextBox.Effect> </TextBox> </StackPanel> </pre>
6	<p>Si la vista será llamada dentro de la <i>MainWindow</i> tendrá que definirse una región con una referencia a la vista en el <i>ModulePRORAMModule</i> que se puede encontrar en la raíz del proyecto PRORAM, una vez definido la región se podrá cargar la vista en un <i>ContentControl</i>.</p> <p><i>Nota: Las vistas que serán cargadas como vista modal se definirán en el siguiente paso.</i></p>	<pre> regionManager.RegisterViewWithRegion("GeoLayer", typeof(GeoLayerView)); regionManager.RegisterViewWithRegion("RightPanel", typeof(RightPanelView)); regionManager.RegisterViewWithRegion("SidePanel", typeof(SidePanelView)); regionManager.RegisterViewWithRegion("TargetPanel", typeof(TargetPanelView)); regionManager.RegisterViewWithRegion("LogsView", typeof(LogsView)); <Border CornerRadius="5" BorderBrush="Gray" BorderThickness="2" DockPanel.Dock="Top" Margin="10,10,10,10"> <ContentControl prism:RegionManager.RegionName="RightPanel"/> </Border> </pre>
7	<p>Las vistas que serán cargadas como modal su llamado será diferente que el paso 6, la vista que llamara a la vista modal tendrá que agregar la referencia:</p> <p><i>"xmlns:i=http://schemas.microsoft.com/expression/2010/interactivity"</i></p> <p>y realiza la <i>InteractionRequest</i> que dará lugar a la invocación de la vista en su encabezado como se muestra en el ejemplo.</p>	


```

/// <summary>
/// Propiedad LatitudP1
/// </summary>
[Required]
[Display(Name = "Latitud")]
[Range(-180, 180, ErrorMessage = "Ingrese un valor valido de {0}, entre {1} y {2} ")]
17 references
public double? LatitudP1
{
    get { return _LatitudP1; }
    set { SetProperty(ref _LatitudP1, value); }
}

```

En la figura se pueden apreciar las *DataAnnotations* sobre la propiedad *LatitudP1*, de las cuales podemos identificar el decorador *[Required]* que nos dice que esta propiedad es obligatoria, el decorador *[Display]* en el cual definimos el nombre con cual será visible en la vista si así lo deseamos y por ultimo el decorador *[Ranger]* en el que se le asigna un rango de valores a la propiedad y un mensaje de error por si el valor ingresado no se encuentra entre los designados.

2.3.2. Validación en el ViewModel

Con las *DataAnnotations* tenemos un nivel de seguridad de los datos, pero no nos asegura que los datos sean ingresados de manera correcta, pero si nos ayuda a encontrar estos datos que no fueron ingresados de la manera correcta. Los campos de entrada que no cumplan con lo establecido en el modelo de datos definido para esa vista serán validados en la ViewModel donde los datos se validaran cada vez que se halla realizado un cambio sobre los datos de entrada con el método *FlattenErrors* como se muestra a continuación:

```

public TargetAreaViewModel()
{
    TargetAreaMod.NombreArea = string.Empty;
    TargetAreaMod.ErrorsChanged += (s, e) => Errors = FlattenErrors();
    TargetAreaMod.PropertyChanged += (s, e) => Errors = FlattenErrors();
    SubmitCommand = new DelegateCommand(SubmitTargeArea);
    CancelCommand = new DelegateCommand(CancelInteraction);
    CustomPopupRequest = new InteractionRequest<INotification>();
    CustomPopupCommand = new DelegateCommand(RaiseCustomPopup);
}

private List<string> FlattenErrors()
{
    List<string> errors = new List<string>();
    Dictionary<string, List<string>> allErrors = TargetAreaMod.GetAllErrors();
    foreach (string propertyName in allErrors.Keys)
    {
        foreach (var errorString in allErrors[propertyName])
        {
            errors.Add(propertyName + ": " + errorString);
        }
    }
    return errors;
}

```

Como se muestra en las imágenes al modelo se le subscribe al evento *ErrorsChanged* y *PropertyChanged* el método *FlattenErrors* que se encarga de recorrer las propiedades del modelo y retornar una lista con la cantidad de errores.

2.3.3. MyConverter

MyConverter es la tercera y ultimo mecanismo de validación de datos. MyConverter es una clase que se encuentra en la raíz de proyecto PRORAM he implementa la interface *IValueConverter* y controla la conversión del componente visual al modelo, este mecanismo se debe implementar en la vista como un recurso para la vista que acepte campos de entrada y sobre cada campo de entrada como se muestra a continuación:

```

<UserControl.Resources>
  <ResourceDictionary>
    <local:MyConverter x:Key="MyConverter" />
    <Style x:Key="ShadowDepth" TargetType="{x:Type TextBox}">
      <Setter Property="BorderBrush" Value="White" />
      <Setter Property="Foreground" Value="White" />
      <Setter Property="Button.Effect">
        <Setter.Value>
          <DropShadowEffect Color="White" Direction="320" ShadowDepth="3" BlurRadius="5" Opacity="0.5" />
        </Setter.Value>
      </Setter>
    </Style>
    <!--<Style TargetType="{x:Type TextBox}" BasedOn="{StaticResource MaterialDesignTextBox}">
      <Setter Property="Margin" Value="0 8 0 8" />
    </Style-->
  </ResourceDictionary>
</UserControl.Resources>

<TextBox Name="TXPower" TabIndex="2"
  materialDesign:HintAssist.Hint="Ingrese la potencia"
  Background="DimGray"
  BorderBrush="DimGray"
  BorderThickness="2"
  Margin="3" >
  <TextBox.Effect>
    <DropShadowEffect ShadowDepth="2" Color="White" Opacity=".4"
      RenderingBias="Performance"></DropShadowEffect>
  </TextBox.Effect>
  <Binding Path="RadarConfigurationModel.TXPower" UpdateSourceTrigger="LostFocus"
    NotifyOnValidationError="True" Mode="TwoWay"
    ValidatesOnExceptions="True" Converter="{StaticResource MyConverter}">
  </Binding>
</TextBox>

```

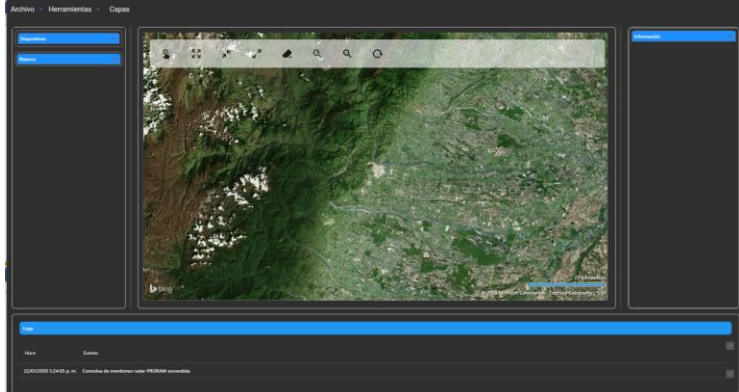
En la primera vista podemos ver como cargamos el recurso *MyConverter* a la vista y la segundo un campo de entrada tipo *TextBox* donde se implementa el *MyConverter* sobre este campo de entrada.

3. Vistas y otros componentes

En esta sección dará una breve explicación de las vistas de la consola de monitoreo PRORAM y su funcionamiento, otros componentes como proyectos creados con el fin de dar manejo a datos de manera externa a la consola.

3.1. Vistas de la consola **PRORAM**

Ya se ha explicado algunas vistas anteriormente en este documento, en esta sección continuaremos detallando el resto de las vistas de la consola de monitoreo **PRORAM** sus funciones como se ven la siguiente tabla:

#	Vista	Función
1		Vista <i>MainWindow</i> , en la vista principal de la aplicación y sirve como marco para contener otras vistas y llamadas a modales.

2		<p>Vista <i>Configuración área objetivo</i>, esta vista modal se encarga de presentar el formulario de ingreso de datos para definir un área objetivo de vigilancia.</p>
3		<p>Vista <i>Configuración de capas</i>, vista modal que permite ocultar o mostrar las capas disponibles sobre el mapa geográfico.</p>
4		<p>Vista <i>Dispositivo radar</i>, vista modal en la cual se generan la mayoría de acciones concernientes a los dispositivos radar</p>

3.2. Proyectos adicionales

En el desarrollo de la solución de software **PRORAM** se generaron proyectos adicionales GisLibrary, ReadingPcap y PRORAM Installer para apoyar el desarrollo de la aplicación, el principal de los proyectos adicionales es GisLibrary que contiene las rutinas de transformación de coordenadas geográficas a coordenadas cartesianas y viceversa, ReadingPcap es un simulador de blancos que fue creado para reproducir blancos previamente guardados y así poder hacer pruebas al proyecto principal, por último el proyecto PRORAM Installer es el encargado de generar el instalador del software de la consola de monitoreo **PRORAM**. A continuación describiremos más en detalle el proyecto adicional GisLibrary

3.2.1. GisLibrary

Este es el principal de los proyectos adicionales como ya se comentó anteriormente es el encargado de manejar las conversiones de coordenadas, de cartesianas a geográficas para posicionar los blancos en el mapa geográfico cada vez que se dibuja un blanco. Este proyecto también genera las coordenadas para los polígonos dibujados en el mapa geográfico como el cono del área de vigilancia y el rectángulo del área objetivo. Una última aplicación de este proyecto es calcular distancias de las mediciones generadas por la herramienta regla sobre el mapa geográfico.