

# Programación Avanzada de entornos de escritorio



Escuela Politécnica Superior  
Universidad de Alicante

# Descripción del taller

- ▶ Crearemos un monitor del sistema:
  - primero con Windows Forms
  - después con WPF
    - usaremos las capacidades de Expression Blend para diseñar gráficamente la aplicación



# Índice

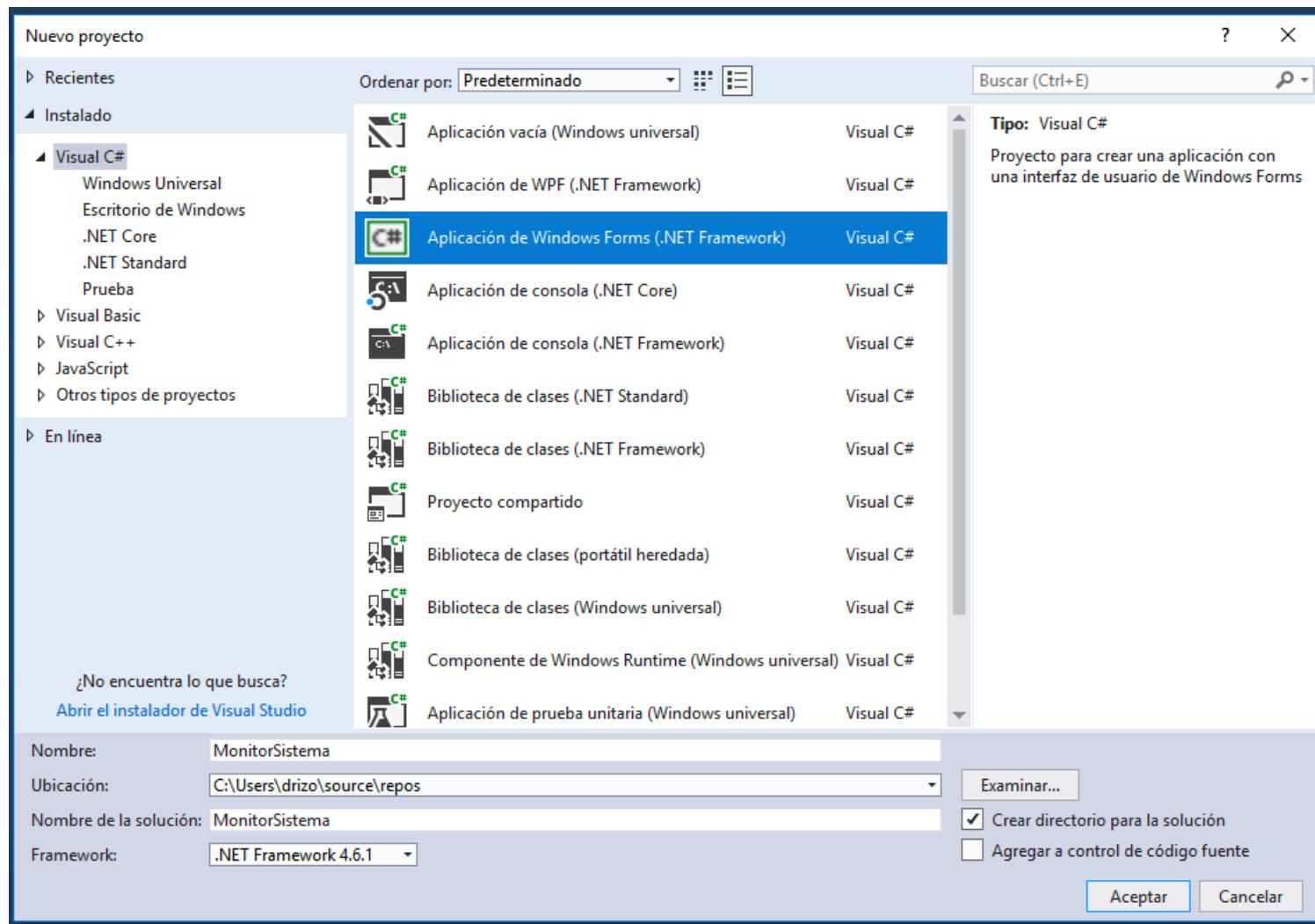
- ▶ Creación de una versión WinForms
- ▶ Migración a Windows Presentation Framework (WPF)

- ▶ Aplicación del diseño con Expression Blend
- ▶ Opcional: ventanas

# NOTA:

- ▶ Se ha subido a Moodle las versiones parciales de la aplicación realizada para por si se quieren consultar DESPUÉS de la clase
- ▶ El código fuente en ocasiones no es el más óptimo: en general se prima la pedagogía cada contexto a la optimización

# Creación de la solución WinForms



# **PRIMERA PARTE: MONITOR CON WINFORMS**

# Librería de monitorización

- ▶ Usaremos la librería de monitorización

System.Diagnostics

- Proporcionamos en Moodle la clase **LectorRecursosSistema** que se encarga de obtener los datos de monitorización
- ▶ [https://docs.microsoft.com/es-es/dotnet/api/  
system.diagnostics](https://docs.microsoft.com/es-es/dotnet/api/system.diagnostics)

# Inclusión de librería

- ▶ Si la clase LectorRecursosSistema proporcionada no compila porque no se encuentran las clases ManagementObjectSearcher y ManagementObject debemos incluir la librería (“referencia”) SystemManagement

```
string str = null;  
ManagementObjectSearcher obj  
foreach (ManagementObject ob  
    -
```

- (ver siguiente diapositiva)

Analizadores

Microsoft.CSharp

System

System.Core

System.Data

System.Data.DataSetExtensions

System.Deployment

System.Drawing

System.Net.Http

Agregar referencia...

Agregar referencia de servicio...

Agregar servicio conectado

Agregar analizador...

Administrar paquetes NuGet...

Limitar el ámbito a esto

Nueva vista de Explorador de soluciones

## Administrador de referencias: MonitorSistema

## Ensamblados

Framework

Extensiones

Resultados de la búsqueda

Reciente

## Proyectos

## Proyectos compartidos

## COM

## Examinar

Nombre

Versión

Microsoft.TeamFoundation.TestManagement.... 15.0.0.0

Microsoft.TeamFoundation.TestManagement.... 15.0.0.0

System.DirectoryServices.AccountManagement 4.0.0.0

 System.Management 4.0.0.0

System.Management.Instrumentation 4.0.0.0

Manage...

Nombre:

System.Management

Creado por:

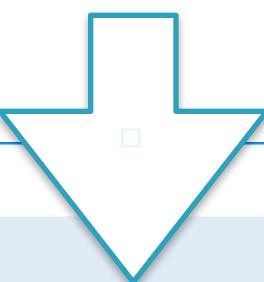
Microsoft Corpora...

Versión:

4.0.0.0

Versión de archiv...

4.6.1055.0 built b...



# Inclusión de librería

- ▶ Si la clase LectorRecursosSistema proporcionada no compila porque el namespace no coincide, corríjase

```
string str = null;  
ManagementObjectSearcher obj  
foreach (ManagementObject ob  
    -
```

# Mostramos datos básicos

- ▶ Creamos ALGUNAS etiquetas (no es necesario hacerlas todas) que nos muestren en un formulario los siguientes datos (de momento los valores de la columna derecha estarán vacíos o con valores literales) - podemos usar un TableLayoutPanel (véase la propiedad Rows para definir la altura de las filas)



# Obtención de datos

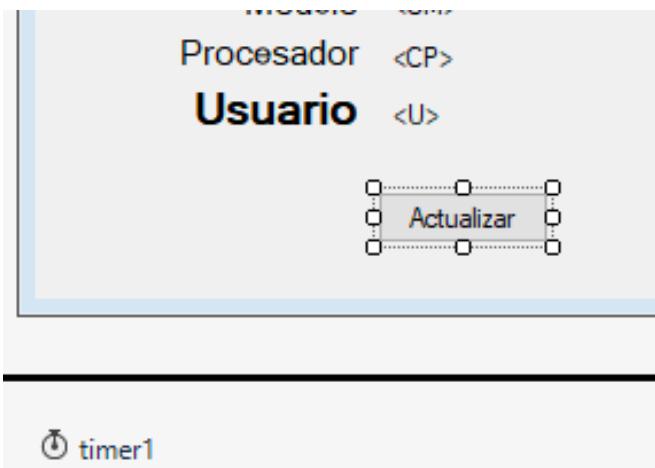
- ▶ Vamos a llenar los valores (columna derecha) bajo demanda pulsando el botón “Actualiza”

```
this.labelCPU.Text = lectorRecursosSistema.getCPU();
this.labelDiscoLectura.Text = lectorRecursosSistema.getDatosDisco(DiskData.Read).ToString();
this.labelDiscoEscritura.Text = lectorRecursosSistema.getDatosDisco(DiskData.Write).ToString();
this.labelDiscoLecturaEscritura.Text = lectorRecursosSistema.getDatosDisco(DiskData.ReadAndWrite).ToString();
this.labelRedRecibidos.Text = lectorRecursosSistema.getDatosRed(NetData.Received).ToString(); ;
this.labelRedEnviados.Text = lectorRecursosSistema.getDatosRed(NetData.Sent).ToString(); ;
this.labelRedRecibidosEnviados.Text = lectorRecursosSistema.getDatosRed(NetData.ReceivedAndSent).ToString();
this.labelDiscosLogicos.Text = lectorRecursosSistema.getDiscosLogicos();
this.labelMemoriaFisica.Text = lectorRecursosSistema.getMemoriaFisica();
this.labelMemoriaVirtual.Text = lectorRecursosSistema.getMemoriaVirtual();
this.labelOrdenadorFabricante.Text = lectorRecursosSistema.getOrdenadorFabricante();
this.labelOrdenadorModelo.Text = lectorRecursosSistema.getOrdenadorModelo();
this.labelOrdenadorProcesador.Text = lectorRecursosSistema.getProcesadores()[0];
this.labelUsuario.Text = lectorRecursosSistema.getUsuario();
```

- Es conveniente usar estos nombres de objetos para seguir las diapositivas

# Lectura continua

- ▶ Cambiamos la actualización manual por una automatizada
  - Para ello añadimos un componente no visible temporizador “Timer”



▫ Al añadirlo se muestra bajo del form

# Lectura continua

- ▶ El botón “Actualizar”
  - Se llamará “Iniciar” / “Parar” según esté activo o no
    - cuando esté inactivo, al pulsar le cambiaremos el nombre a “Parar”
    - cuando esté activo, al posar le cambiaremos el nombre a “Iniciar”

```
if (timer1.Enabled)
{
    timer1.Stop();
    btnActualizar.Text = "Iniciar";
} else
{
    timer1.Interval = 1000;
    timer1.Start();
    btnActualizar.Text = "Parar";
}
```

# Manejador del temporizador

- ▶ Al hacer doble click en el temporizador se abrirá su manejador
- ▶ Movemos el código de actualización de los datos al manejado del temporizador (Comportamiento Tick)

# Actividad final Windows Forms

- ▶ Añadir algún concepto de diseño
  - Alineamientos
  - Colores
  - Tipografía...
- ¿Qué conclusión podemos sacar de esta forma de trabajo?
- ¿Se ha modificado en algún momento el código del formulario que mostramos en la siguiente dispositiva?

```
#region Código generado por el Diseñador de Windows Forms
```

```
/// <summary>
```

```
/// Método necesario para admitir el Diseñador. No se puede modificar  
/// el contenido de este método con el editor de código.
```

```
/// </summary>
```

```
1 referencia
```

```
private void InitializeComponent()
```

```
{
```

```
    this.components = new System.ComponentModel.Container();
```

```
    this.label1 = new System.Windows.Forms.Label();
```

```
//
```

```
// label2
```

```
//
```

```
this.label2.AutoSize = true;
```

```
this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 14F, Sys
```

```
this.label2.Location = new System.Drawing.Point(55, 37);
```

```
this.label2.Name = "label2";
```

```
this.label2.Size = new System.Drawing.Size(91, 24);
```

```
this.label2.TabIndex = 2;
```

```
this.label2.Text = "Memoria";
```

```
//
```

```
// label3
```

```
//
```

```
this.label3.AutoSize = true;
```

# **SEGUNDA PARTE: MONITOR CON WPF**

# Windows Presentation Framework

## ▶ Con Windows Forms:

- es complicado salirse del diseño con estética “Windows” por defecto
- las operaciones de composición de elementos son prácticamente manuales
- Microsoft propone a partir de .NET 3.0 “Windows Presentation Framework” (WPF) como forma más avanzada de desarrollar entornos gráficos

# Windows Presentation Framework

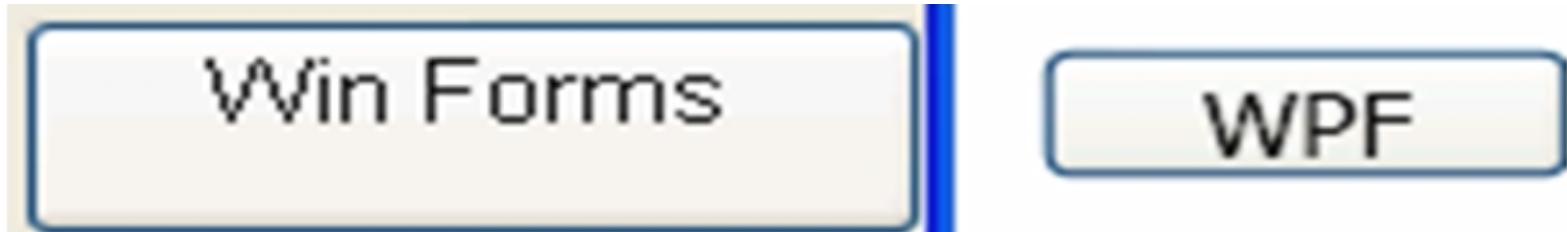
## Características básicas:

### ► 1. Programación declarativa

```
<Window x:Class="WpfSamples.WrapPanel"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WrapPanel" Height="300" Width="300">
    <Grid Height="44">
    </Grid>
</Window>
```

# Windows Presentation Framework

- ▶ 2. Independencia de la resolución de pantalla
  - Windows forms usaba los componentes User32
  - WPF utiliza componentes DirectX a través de la Media Integration Layer (MIL)
  - Usa vectores en lugar de raster para renderizar los controles (véase w bajo)



# Windows Presentation Framework

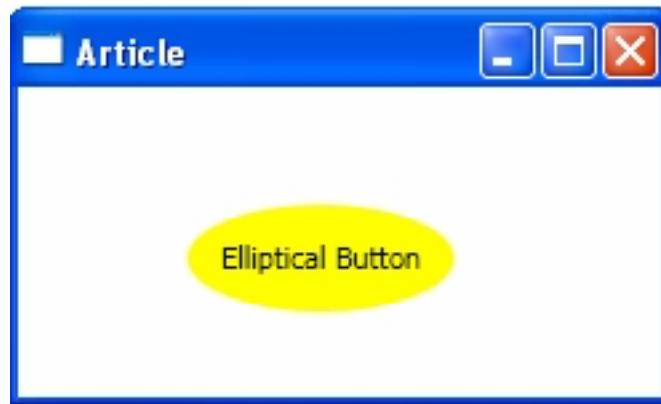
- ▶ 3. Permite poner un control dentro de otro control



```
<Button Margin="90,88,75,124" Name="ButtonWithTextBox">  
    <TextBox Width="75">Enter Text</TextBox>  
</Button>
```

# Windows Presentation Framework

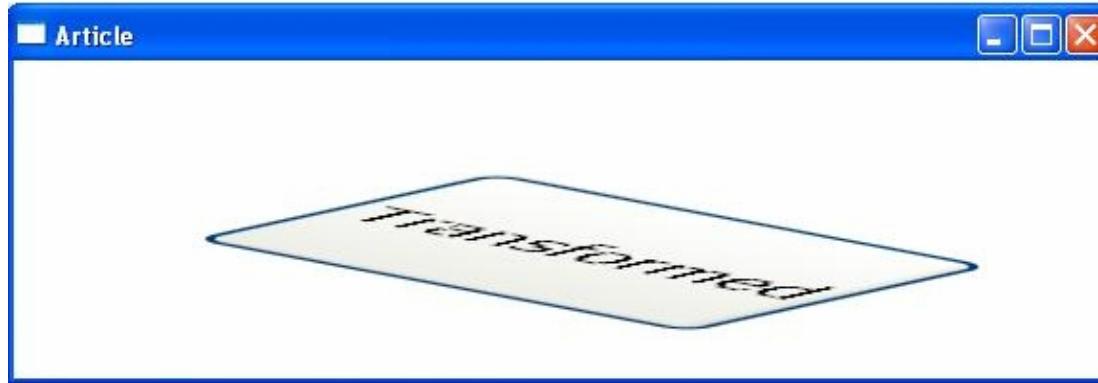
- ▶ 4. Permite cambiar la forma de los controles con ControlTemplate



```
<Button x:Name="EllipticalButton" Margin="30,12,45,0" Content="Elliptical Button">
    <Button.Template>
        <ControlTemplate TargetType="{x:Type Button}">
            <Grid>
                <Ellipse Width="100" Height="40" Fill="Yellow"/>
            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
        </Grid>
    </ControlTemplate>
</Button.Template>
</Button>
```

# Windows Presentation Framework

- ▶ 5. Los controles soportan operaciones de transformación



```
<Button Width="70" Height="50">
    <Button.RenderTransform>
        <TransformGroup>
            <RotateTransform Angle="45" />
            <ScaleTransform ScaleX="5" ScaleY="1" />
            <SkewTransform AngleX="20" />
        </TransformGroup>
    </Button.RenderTransform>
    Transformed
</Button>
```

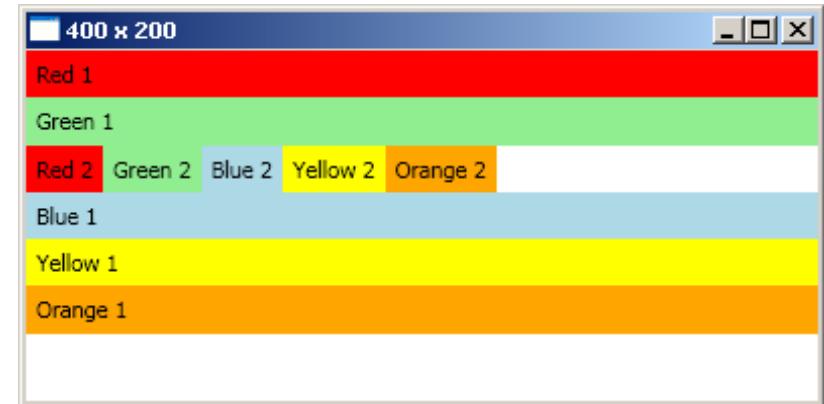
# Windows Presentation Framework

- ▶ 6. Gran cantidad de layouts:
  - Stack panel
  - Wrap Panel
  - Dock Panel
  - Grid
  - Canvas
  - 2D, 3D, animaciones y multimedia

# Windows Presentation Framework

- ▶ StackPanel: organiza controles en una línea, vertical u horizontal

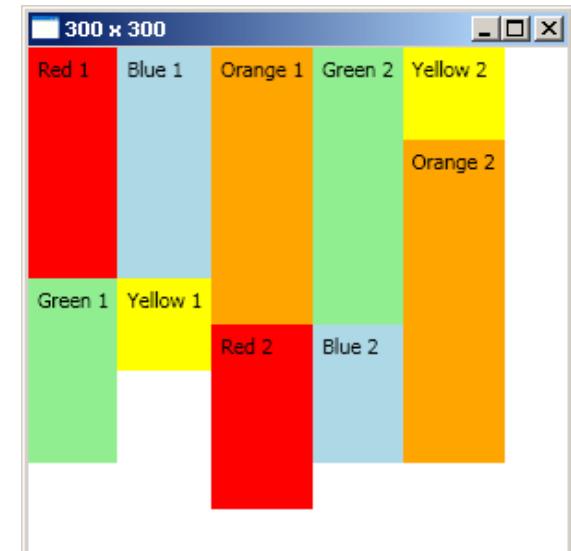
```
<StackPanel Orientation="Vertical"> <!-- Vertical is the default -->
    <Label Background="Red">Red 1</Label>
    <Label Background="LightGreen">Green 1</Label>
    <StackPanel Orientation="Horizontal">
        <Label Background="Red">Red 2</Label>
        <Label Background="LightGreen">Green 2</Label>
        <Label Background="LightBlue">Blue 2</Label>
        <Label Background="Yellow">Yellow 2</Label>
        <Label Background="Orange">Orange 2</Label>
    </StackPanel>
    <Label Background="LightBlue">Blue 1</Label>
    <Label Background="Yellow">Yellow 1</Label>
    <Label Background="Orange">Orange 1</Label>
</StackPanel>
```



# Windows Presentation Framework

- WrapPanel: organiza controles en una línea hasta donde caben, bajando a la siguiente cuando no hay sitio

```
<WrapPanel Orientation="Horizontal"> <!-- Horizontal is the default -->
    <Label Width="125" Background="Red">Red 1</Label>
    <Label Width="100" Background="LightGreen">Green 1</Label>
    <Label Width="125" Background="LightBlue">Blue 1</Label>
    <Label Width="50" Background="Yellow">Yellow 1</Label>
    <Label Width="150" Background="Orange">Orange 1</Label>
    <Label Width="100" Background="Red">Red 2</Label>
    <Label Width="150" Background="LightGreen">Green 2</Label>
    <Label Width="75" Background="LightBlue">Blue 2</Label>
    <Label Width="50" Background="Yellow">Yellow 2</Label>
    <Label Width="175" Background="Orange">Orange 2</Label>
</WrapPanel>
```



# Windows Presentation Framework

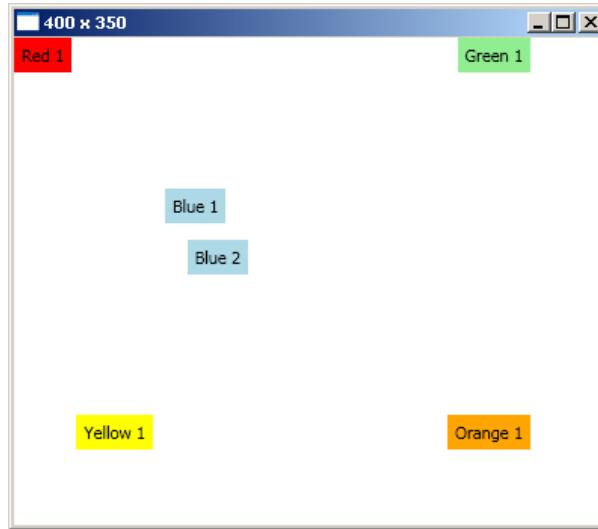
- ▶ DockPanel. Sitúa los controles en 5 regiones in five different regions: top, bottom, left, right, o center (rellena).



```
<DockPanel>
    <Label DockPanel.Dock="Top" Height="100" Background="Red">Red 1</Label>
    <Label DockPanel.Dock="Left" Background="LightGreen">Green 1</Label>
    <Label DockPanel.Dock="Right" Background="LightBlue">Blue 1</Label>
    <Label DockPanel.Dock="Bottom" Background="LightBlue">Blue 2</Label>
    <Label DockPanel.Dock="Bottom" Height="50" Background="Yellow">Yellow 1</Label>
    <Label Width="100" Background="Orange">Orange 1</Label> |!-- default is to fill -->
    <Label Background="LightGreen">Green 2</Label>
</DockPanel>
```

# Windows Presentation Framework

- ▶ Canvas: sitúa elementos en una coordenada explícita

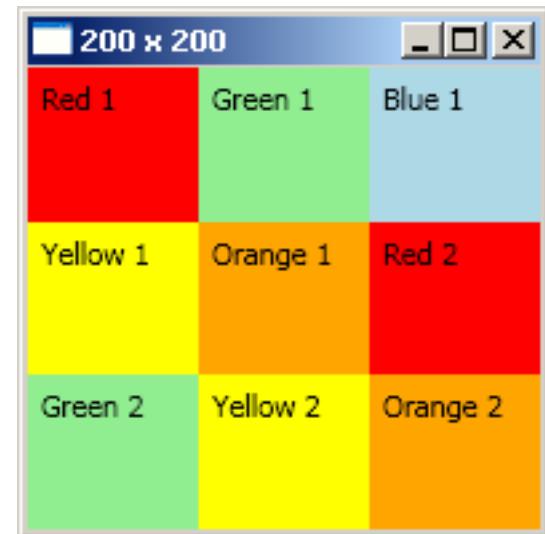


```
<Canvas>
    <!-- default coordinates 0,0 from top left; like WinForms -->
    <Label Background="Red">Red 1</Label>
    <Label Canvas.Right="50" Background="LightGreen" >Green 1</Label>
    <Label Canvas.Top="100" Canvas.Left="100" Background="LightBlue" >Blue 1</Label>
    <Label Canvas.Bottom="166" Canvas.Right="237" Background="LightBlue" >
        Blue 2</Label>
    <Label Canvas.Right="300" Canvas.Top="250" Background="Yellow" >Yellow 1</Label>
    <Label Canvas.Right="50" Canvas.Bottom="50" Background="Orange" >Orange 1</Label>
</Canvas>
```

# Windows Presentation Framework

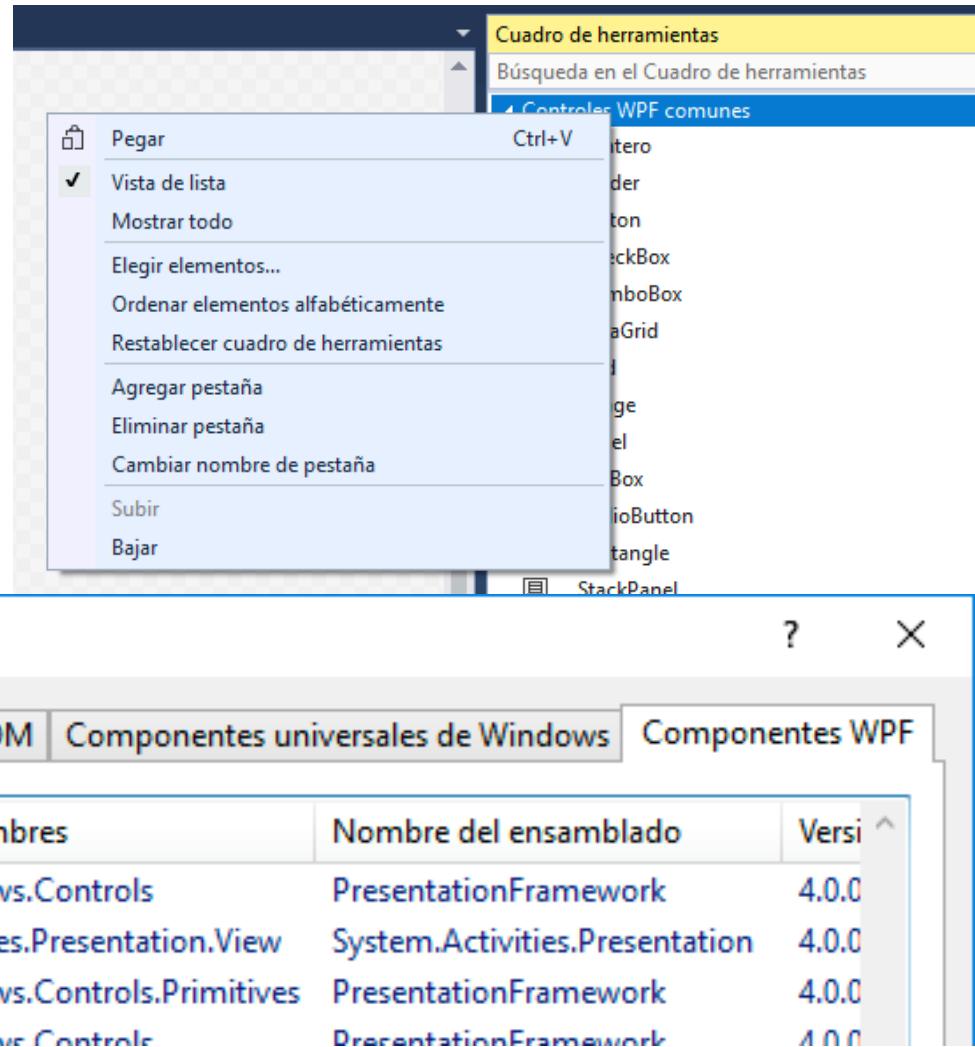
- ▶ UniformGrid: mantiene una serie de celdas de tamaño igual.

```
<UniformGrid>
    <Label Background="Red">Red 1</Label>
    <Label Background="LightGreen">Green 1</Label>
    <Label Background="LightBlue">Blue 1</Label>
    <Label Background="Yellow">Yellow 1</Label>
    <Label Background="Orange">Orange 1</Label>
    <Label Background="Red">Red 2</Label>
    <Label Background="LightGreen">Green 2</Label>
    <Label Background="Yellow">Yellow 2</Label>
    <Label Background="Orange">Orange 2</Label>
</UniformGrid>
```



# Personalización herramientas WPF

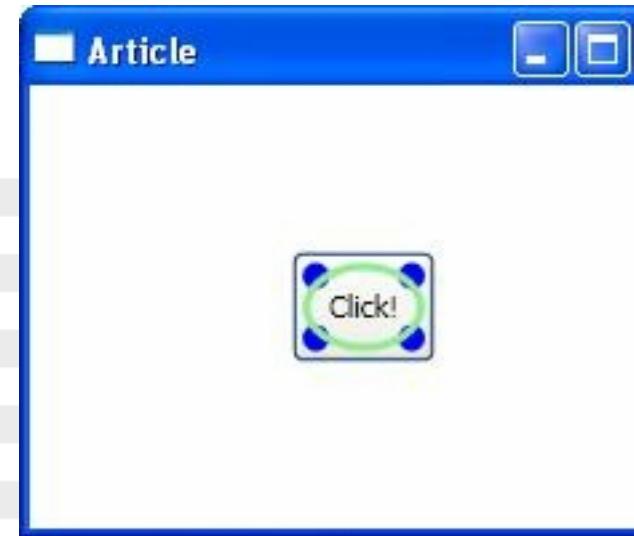
- ▶ Si no aparece UniformGrid debemos pulsar botón derecho sobre la barra de herramientas > Elegir elementos



# Windows Presentation Framework

## ▶ Gráficos 2D en los controles

```
<Button HorizontalAlignment="Center" VerticalAlignment="Center">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Ellipse Grid.Column="0" Grid.Row="0" Fill="Blue" Width="10" Height="10" />
        <Ellipse Grid.Column="2" Grid.Row="0" Fill="Blue" Width="10" Height="10" />
        <Ellipse Grid.Column="0" Grid.Row="2" Fill="Blue" Width="10" Height="10" />
        <Ellipse Grid.Column="2" Grid.Row="2" Fill="Blue" Width="10" Height="10" />
        <Ellipse Grid.ColumnSpan="3" Grid.RowSpan="3" Stroke="LightGreen" StrokeThickness="3" />
        <TextBlock Grid.Column="1" Grid.Row="1" VerticalAlignment="Center" Text="Click!" />
    </Grid>
</Button>
```



# Windows Presentation Framework

## ▶ Renderizaciones 3D

```
<Viewport3D>
    <Viewport3D.Camera>
        <PerspectiveCamera Position="-4,1,10" LookDirection="4,-1,-10"
    UpDirection="0,1,0" FieldOfView="45" />
    </Viewport3D.Camera>
    <ModelVisual3D>
        <ModelVisual3D.Content>
            <Model3DGroup>
                <DirectionalLight Direction="0,0,-1" />
                <GeometryModel3D>
                    <GeometryModel3D.Geometry>
                        <MeshGeometry3D Positions="0,1,0 1,-1,1 -1,-1,1 1,-1,-1 -1,-1,-1"
    Normals="0,1,0 -1,0,1 1,0,1 -1,0,-1 1,0,-1"
    TriangleIndices="0,2,1 0,3,1 0,3,4 0,2,4" />
                    </GeometryModel3D.Geometry>
                    <GeometryModel3D.Material>
                        <DiffuseMaterial Brush="Red" />
                    </GeometryModel3D.Material>
                    <GeometryModel3D.BackMaterial>
                        <DiffuseMaterial Brush="Green" />
                    </GeometryModel3D.BackMaterial>
                </GeometryModel3D>
            </Model3DGroup>
        </ModelVisual3D.Content>
    </ModelVisual3D>
</Viewport3D>
```



# Windows Presentation Framework

## ▶ Animación

```
DoubleAnimation animate = new DoubleAnimation();
animate.To = 300;
animate.Duration = new Duration(TimeSpan.FromSeconds(5));
animate.RepeatBehavior = RepeatBehavior.Forever;
myEllipse.BeginAnimation(Ellipse.WidthProperty, animate);
```

## ▶ Elementos multimedia

```
<MediaElement Source="C:\sample.wmv" />
```

# Windows Presentation Framework

- ▶ 7. Estilos y recursos reutilizables
  - Ejemplo de recurso local (myBrush y text)

```
<Window x:Class="EjemplosWPF"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Article" Height="199" Width="362" Loaded="Window_Loaded"
    xmlns:s="clr-namespace:System;assembly=mscorlib">
    <Window.Resources>
        <SolidColorBrush x:Key="myBrush" Color="LightGreen" />
        <s:String x:Key="text">Hello!</s:String>
    </Window.Resources>
    <Grid>
        <TextBlock FontSize="36" Width="100" Height="50"
            Text="{StaticResource text}" Background="{StaticResource myBrush}">
        </TextBlock>
    </Grid>
</Window>
```

- Algo parecido podemos hacer a nivel global

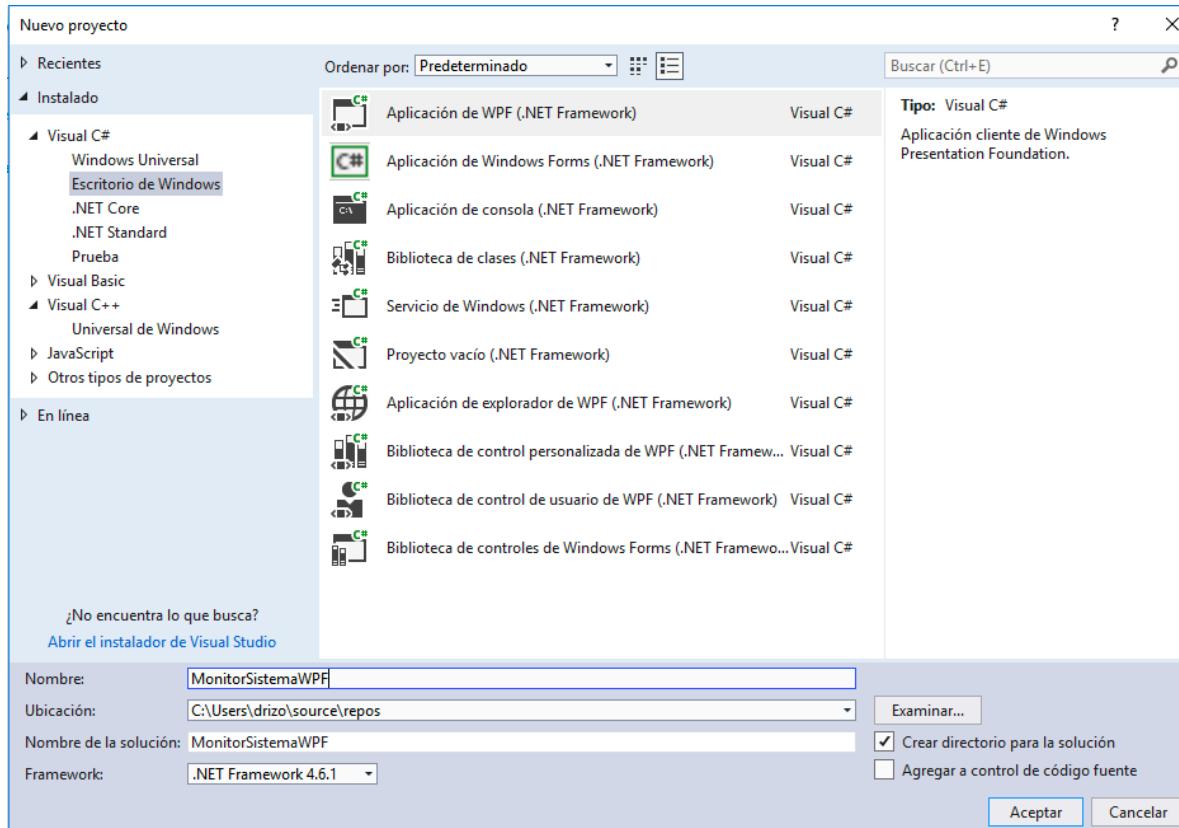
# Windows Presentation Framework

## ► 8. Databinding

```
<Slider x:Name='sizeSlider'  
        Orientation='Vertical'  
        Minimum='10'  
        Maximum='80'  
        Value='25' />  
  
<TextBlock Text='Sample Text - abcde'  
          Margin='5,0'  
          FontSize=  
          '{Binding ElementName=sizeSlider, Path= Value}' />
```

# Monitor de actividad con WPF

- ▶ Vamos a implementar el monitor de actividad como Aplicación de WPF



# MainWindow

□ Vista  
diseño

□ XAML

Code  
behind

The image shows a Windows application development interface with three main panes:

- Top Left (Design View):** Shows the visual representation of the `MainWindow`. It has a title bar labeled "MainWindow" and a single blue-bordered `Grid` element.
- Top Right (Code-Behind):** Shows the `MainWindow.xaml.cs` file. It includes standard using statements for System and various Windows namespaces, followed by the `MonitorSistemaWPF` namespace and the `MainWindow` class definition.
- Bottom (XAML):** Shows the `MainWindow.xaml` XAML code. It defines a `Window` element with `x:Class="MonitorSistemaWPF.MainWindow"`, sets its title to "MainWindow", and specifies dimensions of 450x800 pixels. Inside the window is a `Grid` element.

# Actividad clase

- ▶ Crear el nuevo proyecto, modificar el título y las dimensiones de la ventana principal
- ▶ Copiar el fichero LectorRecursosSistema.cs
- ▶ Crear un panel e incluir los mismos controles que teníamos en el proyecto anterior hecho en WindowsForms
  - Para que el panel ocupe el 100% de la ventana:

```
1 HorizontalAlignment="Stretch" VerticalAlignment="Stretch">>
```

(ver siguientes diapositivas)

# Panel Grid

- ▶ Podemos organizar los elementos en el panel Grid
- ▶ Añadimos dentro del grid los controles Label
  - lo más cómodo es usar la vista XAML y copiar/pegar

```
<Label Content="Monitor de actividad"/>
<Label Content="Ordenador"/>
<Label Content="Fabricante"/>
<Label Content="ValorFabricante"/>
<Label Content="Modelo"/>
<Label Content="ValorModelo"/>
```

| Monitor de sistema              |                 |
|---------------------------------|-----------------|
| Monitor de actividad            |                 |
| ValorUsuario                    |                 |
| Ordenador                       |                 |
| Fabricante                      | ValorFabricante |
| Modelo                          |                 |
| ValorProcesador                 | ValorModelo     |
| CPU                             | ValorCPU        |
| Memoria                         |                 |
| Física                          | MF              |
| Virtual                         | MV              |
| Disco                           |                 |
| Valor discos lógicos            |                 |
| Lectura                         | DR              |
| Escritura                       | DW              |
| Lectura/Escritura               | DRW             |
| Red                             |                 |
| Paquetes recibidos              | NR              |
| Paquetes enviados               | NPS             |
| Red paquetes recibidos/enviados | NPRS            |

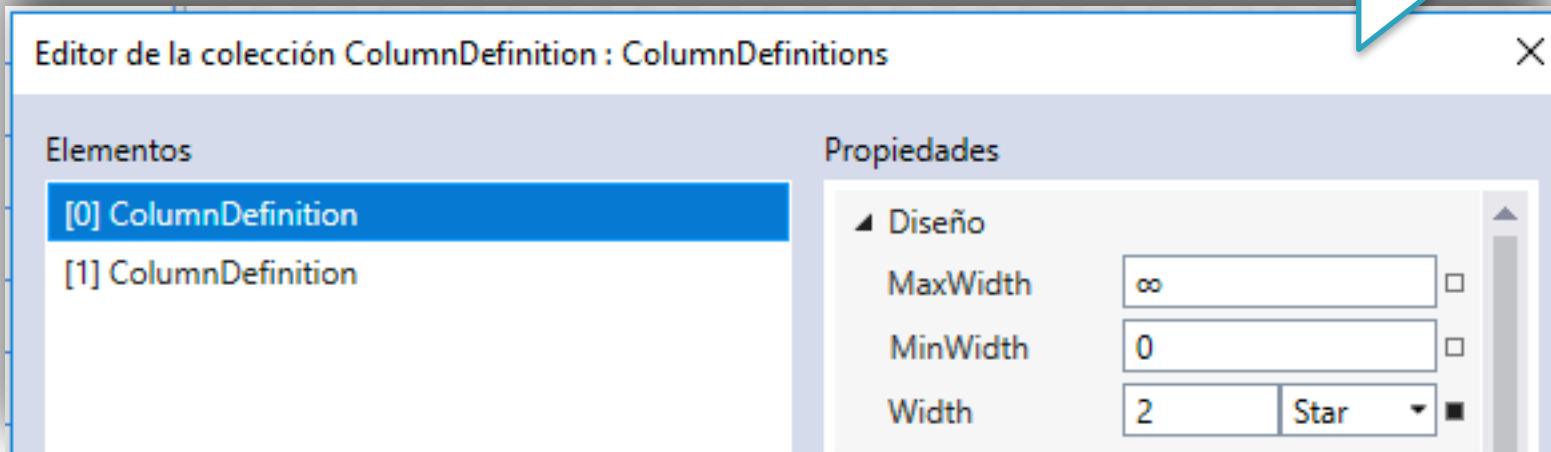
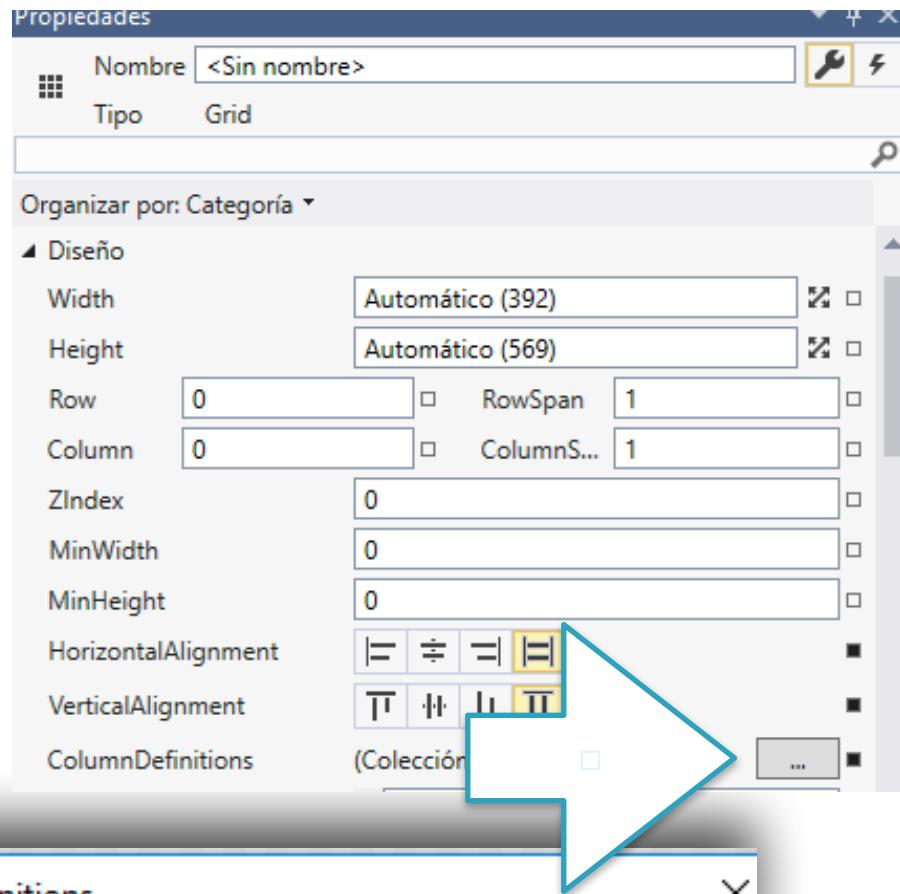
# Panel Grid

- ▶ Las anchuras / alturas se definen en el XAML
  - ▶ Una definición por cada fila / columna
    - ▶ Anchura “star”:  $2* + 1* = 3*$  en total, de forma que  $2* = 66.6\%$  y  $1* = 33.3\%$

```
<Grid HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
```

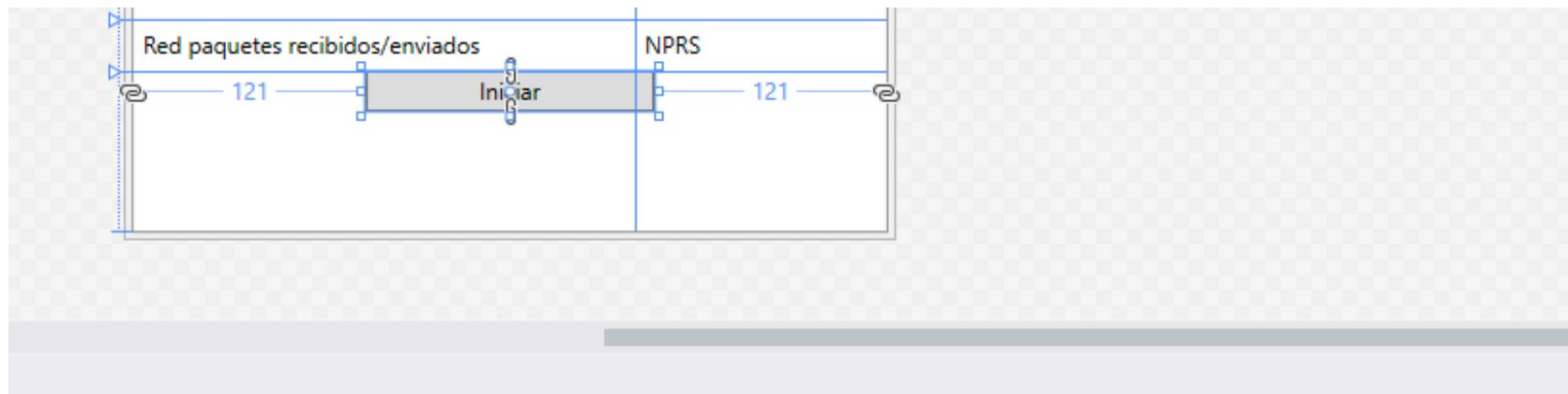
# Panel Grid

- ▶ Las anchuras / alturas se pueden definir también en el panel de propiedades del Grid



# Botón en panel Grid

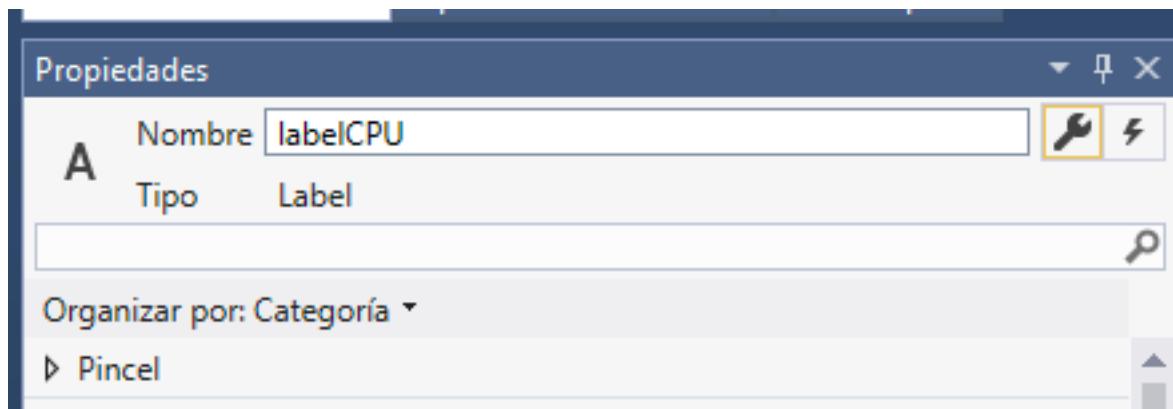
- ▶ Añadimos también el botón y hacemos que ocupe dos columnas con ColumnSpan



```
<Label Content="Paquetes recibidos" Grid.Row="15" />
<Label Content="NR" Grid.Row="15" Grid.Column="1"/>
<Label Content="Paquetes enviados" Grid.Row="16"/>
<Label Content="NPS" Grid.Row="16" Grid.Column="1"/>
<Label Content="Red paquetes recibidos/enviados" Grid.Row="17"/>
<Label Content="NPRS" Grid.Row="17" Grid.Column="1"/>
<Button Content="Iniciar" Grid.Row="18" Grid.ColumnSpan="2" Width="150"/>
```

# Nombre objetos

- ▶ Añadimos el nombre de las Label igual que en el proyecto anterior



Visual Studio - Windows Form Application - Form1.cs

```
<Label x:Name="labelCPU" Content="ValorCPU" Grid.Row="6" Grid.Column="1"/>
```

# Actividad Ampliación

- ▶ ¿Qué pasa si queremos cambiar la posición de elementos en el grid?
  - Hay que cambiar manualmente la asignación de fila / columna
  - O usar en su lugar otro panel como DockPanel

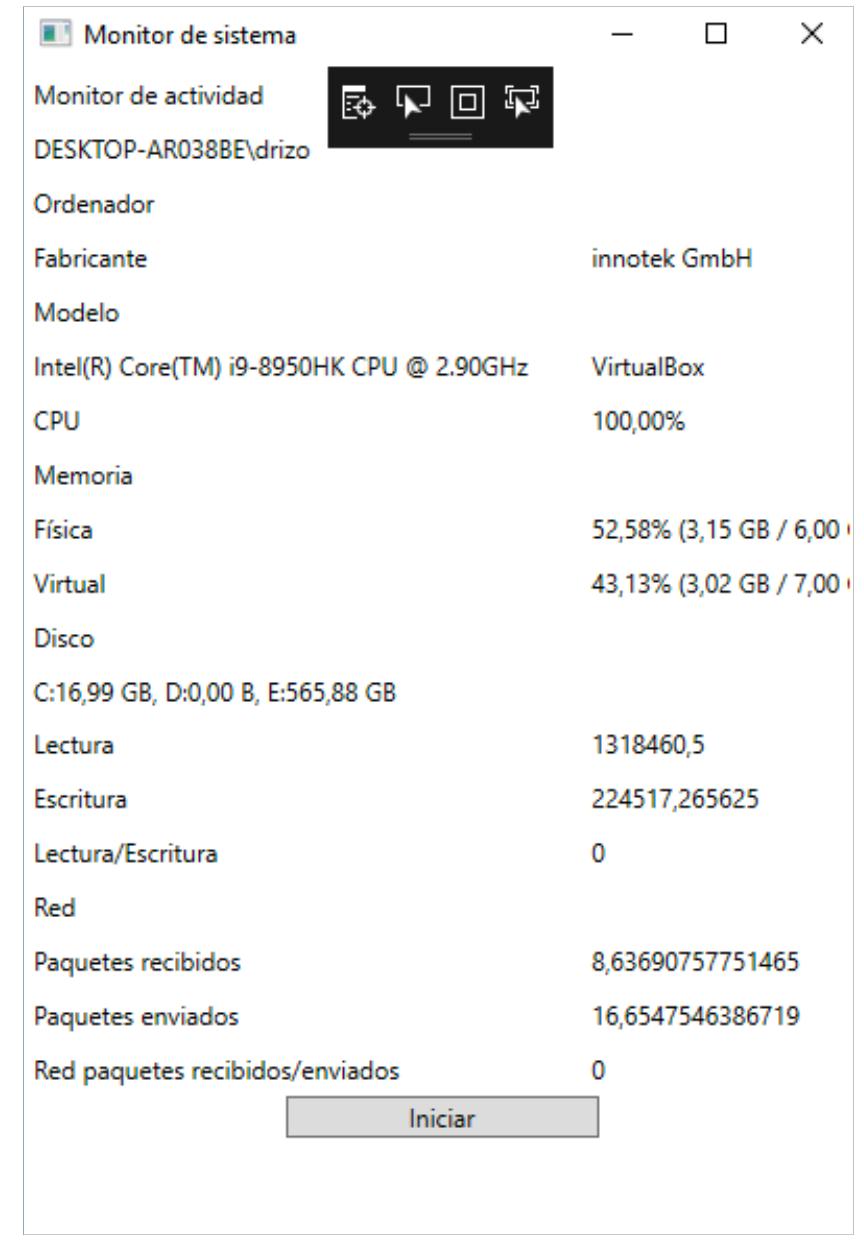
# Obtención de datos

- ▶ El control Timer no existe en WPF
- ▶ Comenzamos leyendo los valores del sistema cuando se pulse el botón en el manejado (click) del botón (sin actualización automática)

```
this.labelCPU.Content = lectorRecursosSistema.getCPU();
this.labelDiscoLectura.Content = lectorRecursosSistema.getDatosDisco(DiskData.Read).ToString();
this.labelDiscoEscritura.Content = lectorRecursosSistema.getDatosDisco(DiskData.Write).ToString();
```

# Estado actual

- Debemos tener la aplicación funcionando hasta este punto



# Timer

```
private void manejadorDispatcherTimer(object sender, EventArgs e)
{
    actualizaValores();
}
```

- ▶ En WPF el control Timer ha desaparecido
- ▶ Podemos usar la clase

System.Windows.Threading.DispatcherTimer dispatcherTimer  
= new ...;

```
public MainWindow()
{
    InitializeComponent();
    dispatcherTimer.Tick += new EventHandler(manejadorDispatcherTimer);
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    if (dispatcherTimer.IsEnabled)
    {
        dispatcherTimer.Stop();
        btnActualizar.Content = "Iniciar";
    }
    else
    {
        dispatcherTimer.Interval = new TimeSpan(0, 0, 1); // se actualiza cada segundo
        dispatcherTimer.Start();
        btnActualizar.Content = "Parar";
    }
}
```

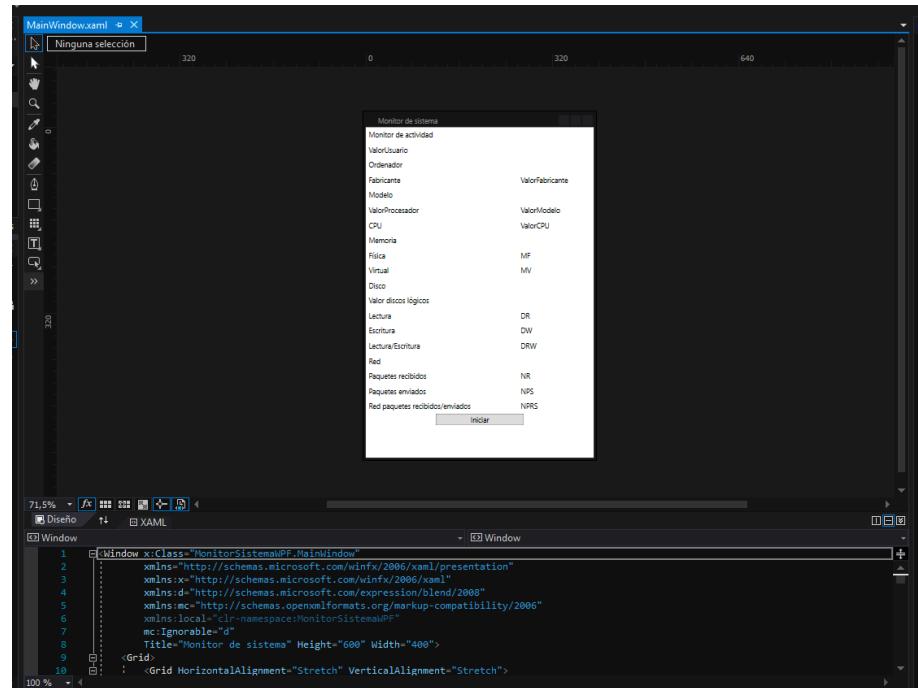
# Ampliación

- ▶ Como ampliación podemos poner un deslizador (Slider) que permita cambiar la frecuencia de actualización

# **TERCERA PARTE: EXPRESSION BLEND**

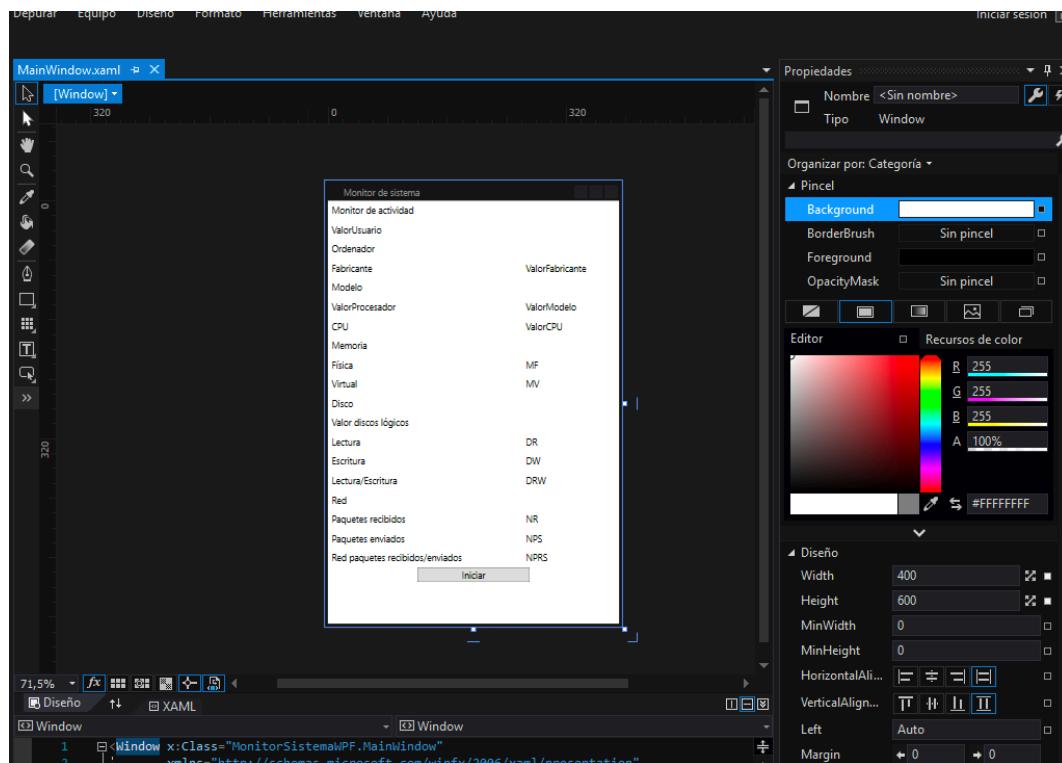
# Diseño visual con Expression Blend

- ▶ Abrimos la solución con Microsoft Blend para Visual Studio
- ▶ Nos interesa trabajar con el XAML
  - MainWindow.xaml



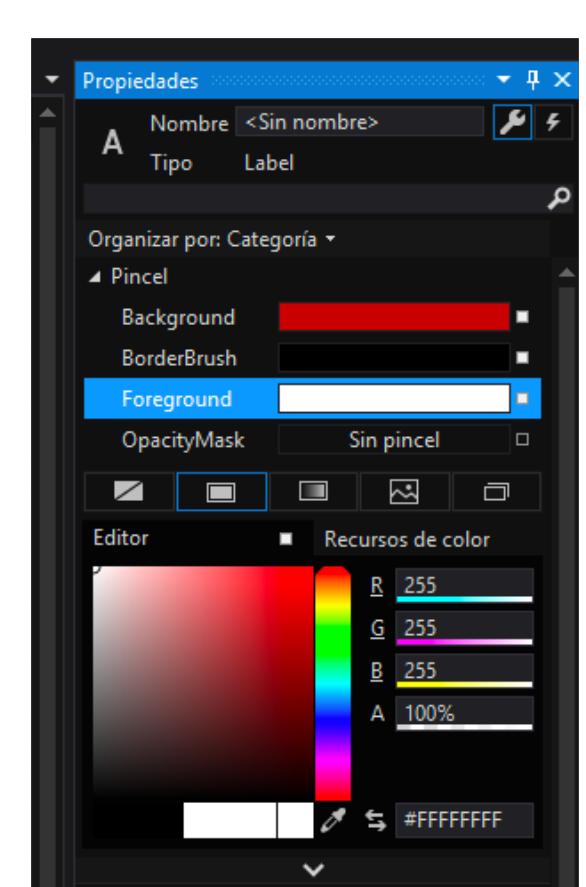
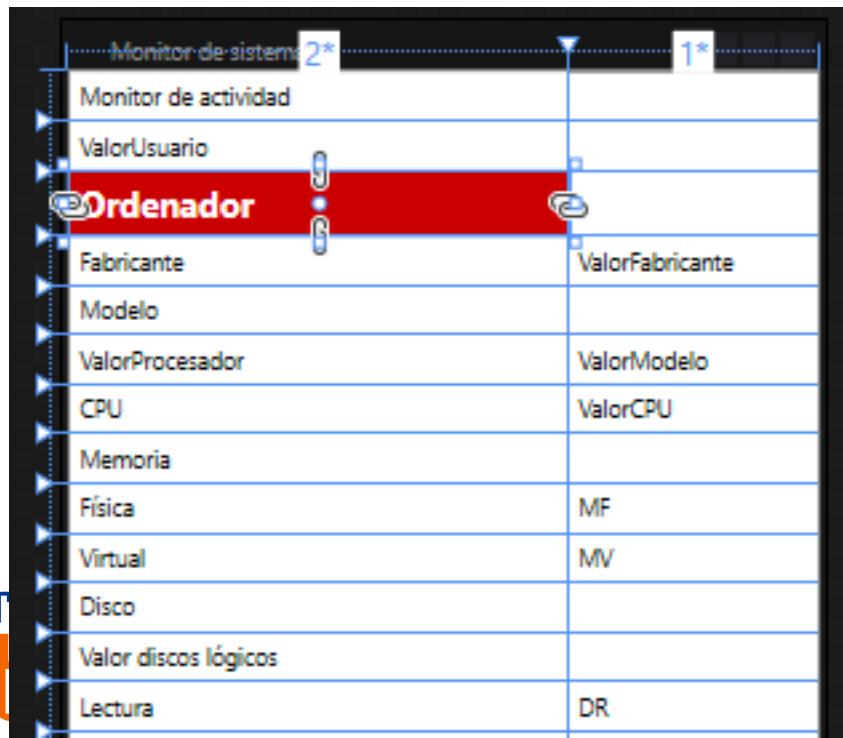
# Diseño visual con Expression Blend

- ▶ Véase cómo esta herramienta nos proporciona mejor control sobre los aspectos visuales que Visual Studio



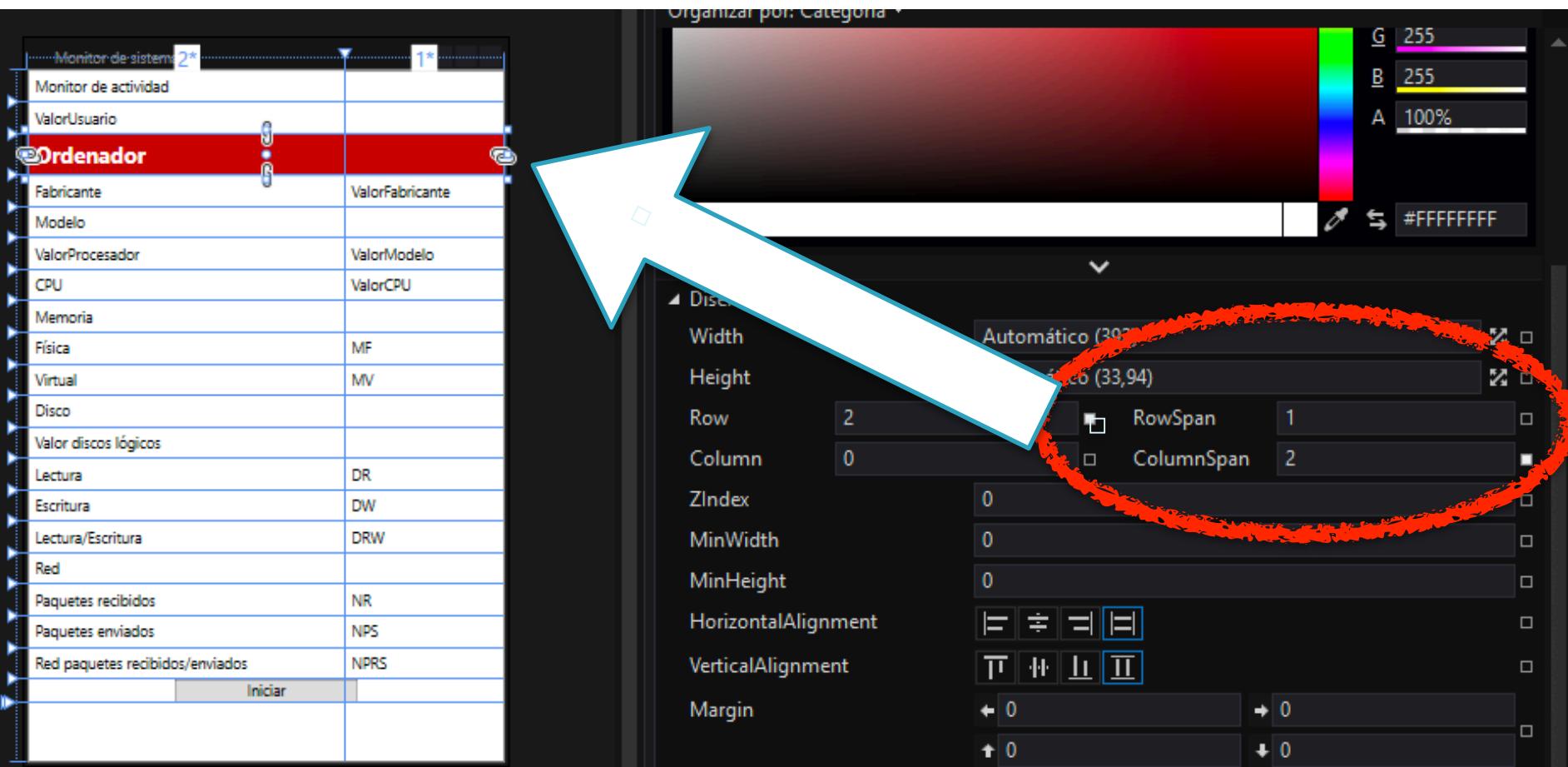
# Blend

- ▶ Cambiamos las propiedades de una etiqueta de título con el panel de propiedades



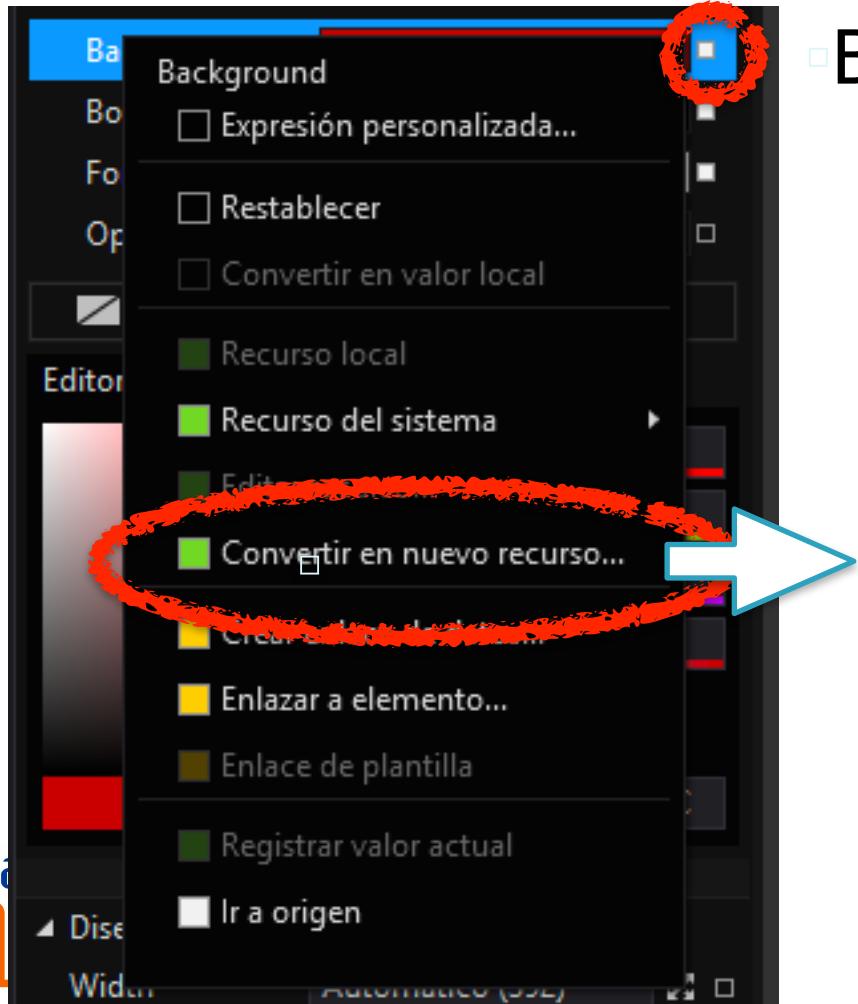
# Blend

- ▶ Cambiamos la estructura desde el propio Blend

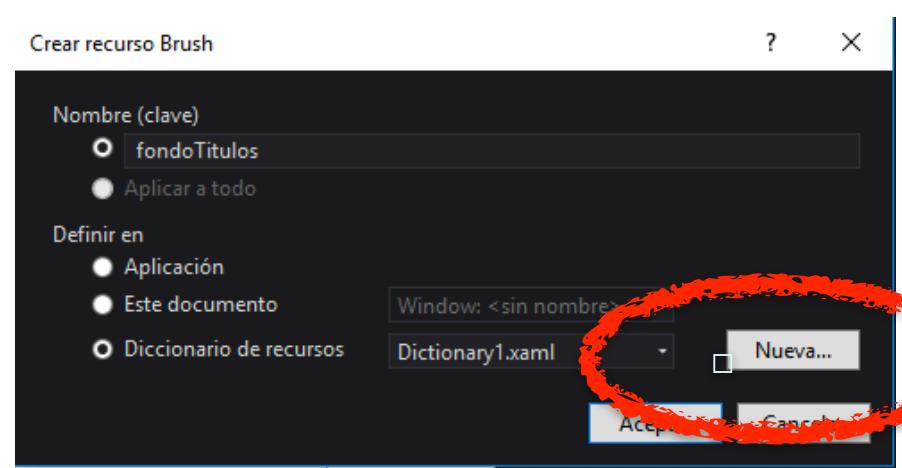


# Blend

- Convertimos las propiedades en estilos propios



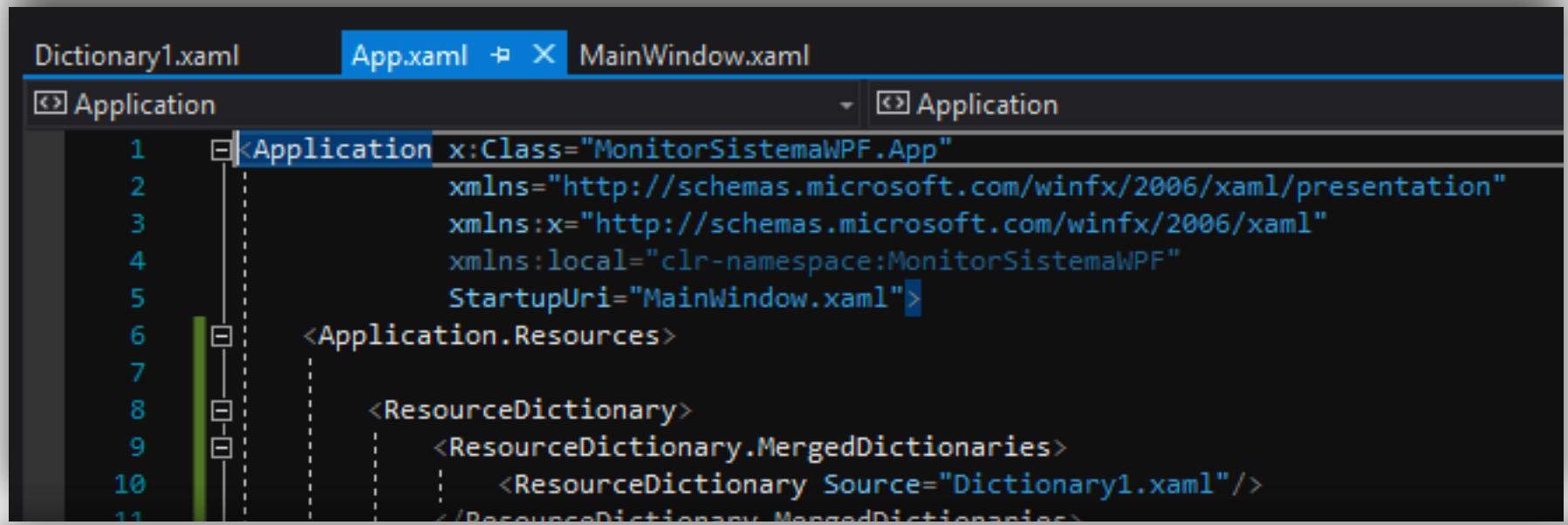
Botón derecho



(siguiente diapositiva)

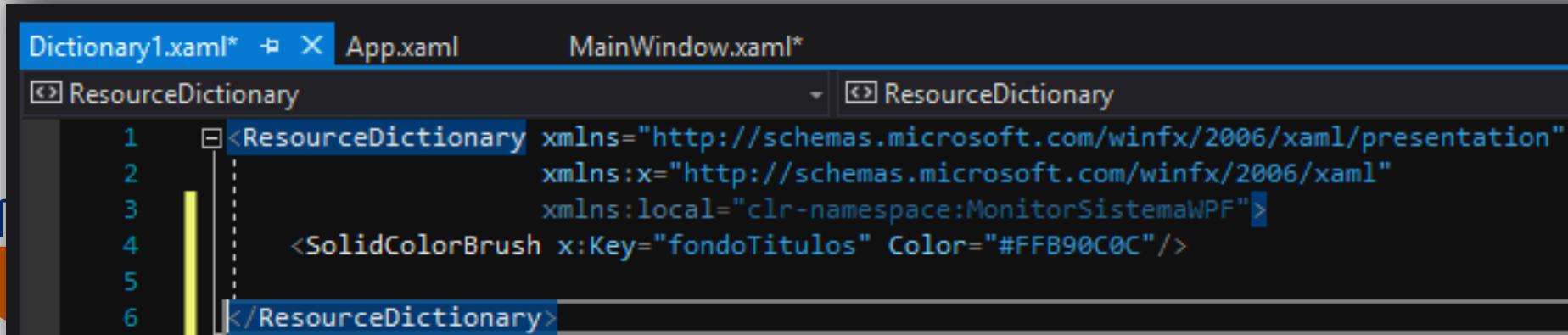
# Blend: diccionario recursos

La aplicación incluye el diccionario de recursos



```
Dictionary1.xaml      App.xaml  ✖  MainWindow.xaml
Application
1  <Application x:Class="MonitorSistemaWPF.App"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:local="clr-namespace:MonitorSistemaWPF"
5      StartupUri="MainWindow.xaml">
6      <Application.Resources>
7          <ResourceDictionary>
8              <ResourceDictionary.MergedDictionaries>
9                  <ResourceDictionary Source="Dictionary1.xaml"/>
10             </ResourceDictionary.MergedDictionaries>
11         </ResourceDictionary>

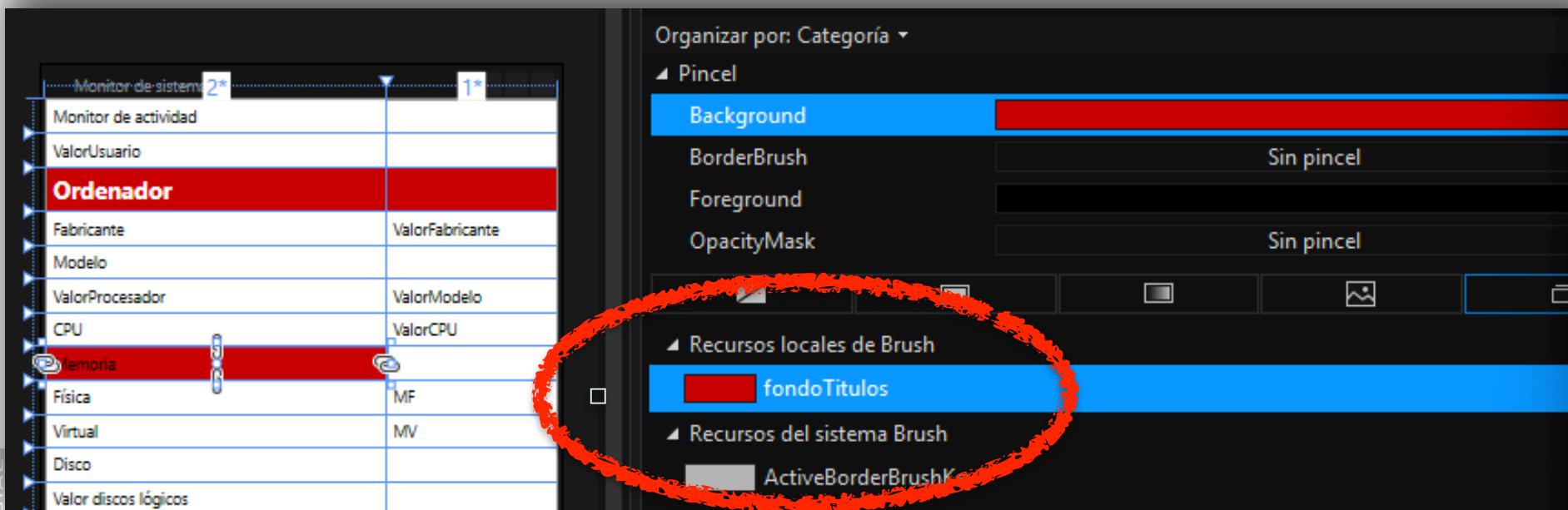
```



```
Dictionary1.xaml*  ✖  App.xaml      MainWindow.xaml*
ResourceDictionary
1  <ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
2      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
3      xmlns:local="clr-namespace:MonitorSistemaWPF">
4      <SolidColorBrush x:Key="fondoTitulos" Color="#FFB90C0C"/>
5
6  </ResourceDictionary>
```

# Blend: aplicación del estilo

- ▶ A partir de ahora lo tenemos como recurso
  - ▶ Pulsamos cuadrado Background > Recurso local > fondoTitulos



# Actividad

- ▶ Crear estilos por este orden:
  - Títulos de sección:
    - ✓ Colores de fondo
    - ✓ Tipografía
  - Etiquetas (Paquetes recibidos, enviados....)
    - Tipografía cambiando propiedades como la alineación
    - Tipografía Valores
- ¿Qué conclusión podemos sacar de esta forma de crear estilos y aplicarlos? ¿Mejor crear estilos no tan específicos?

# Ampliación: creación de recursos reutilizables

- Propiedades

- Content = Memoria
- Background color = #F00
- Font size = 18



□ Label1

- Content = Disco
- Background color = #F00
- Font size = 18



□ Label2

- Content = Red
- Background color = #F00
- Font size = 18

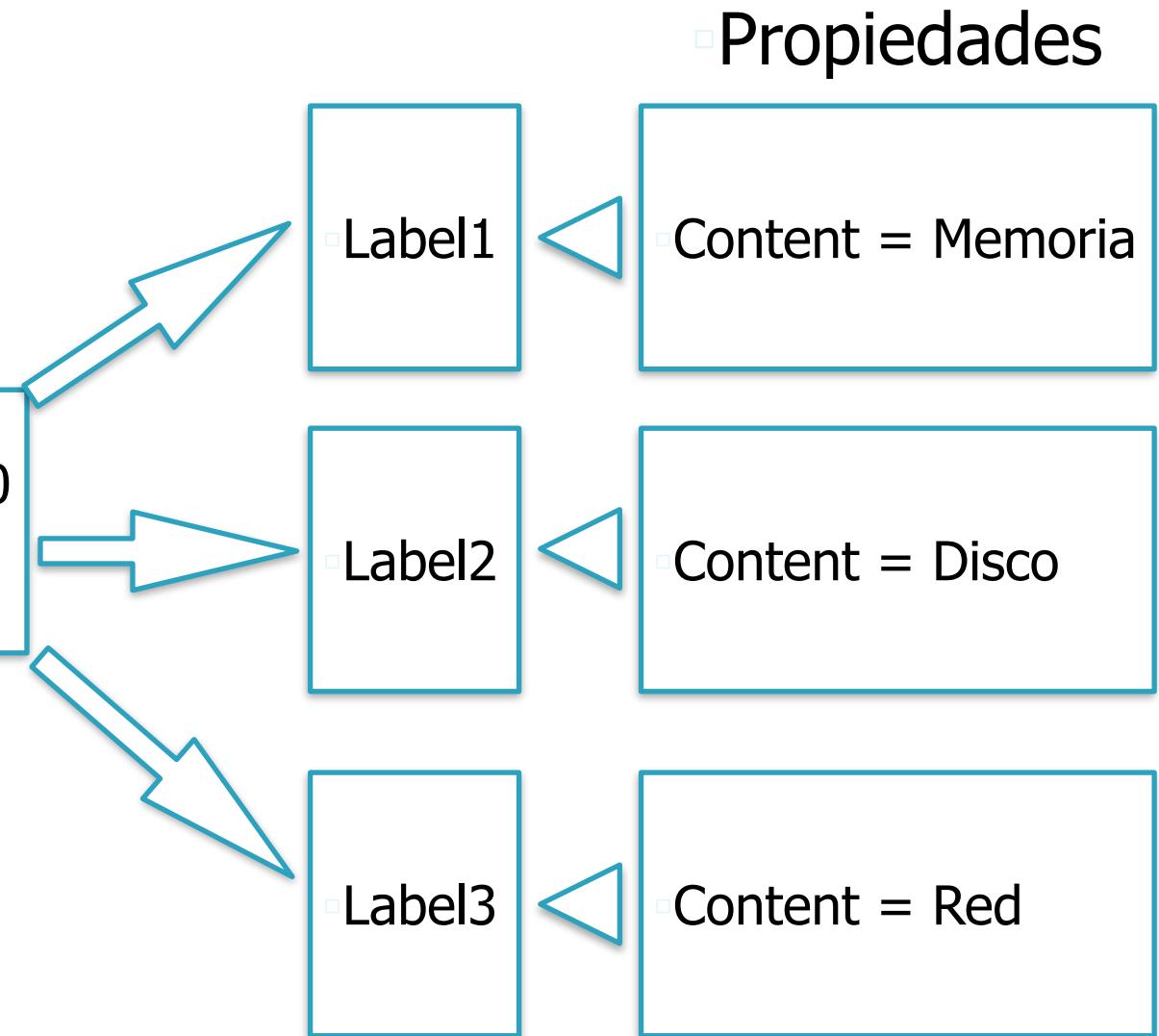


□ Label3

# Creación de recursos reutilizables

## Definimos estilo

- Background color = #F00
- Font size = 18



# Recursos reutilizables

- ▶ En el diccionario (también se puede hacer a nivel de ventana o de aplicación)

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:MonitorSistemaWPF">
    <SolidColorBrush x:Key="fondoTitulos" Color="#FFB90C0C"/>
    <SolidColorBrush x:Key="colorFuenteTitulo" Color="White"/>

    <Style x:Key = "etiquetaTituloSeccion" TargetType = "Label">
        <Setter Property = "Background" Value = "#FFB90C0C" />
        <Setter Property = "Foreground" Value = "white" />
        <Setter Property = "FontSize" Value = "18" />
    </Style>

</ResourceDictionary>
```

- En VisualStudio también podemos hacer estos cambios

# Recursos reutilizables

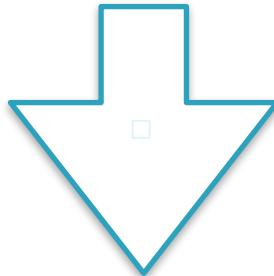
- ▶ Debemos añadir el diccionario (si no está ya) en la aplicación

```
<Application x:Class="MonitorSistemaWPF.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:MonitorSistemaWPF"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

# Recursos reutilizables

## ► Aplicación en formulario XAML

```
<Label Content="Ordenador" Grid.Row="2" Grid.ColumnSpan="2"  
FontSize="18"  
Background="{DynamicResource fondoTitulos}" BorderBrush="Black"  
Foreground="{DynamicResource colorFuenteTitulo}" Style="{DynamicResource tituloSeccion}"/>
```



```
<Label Content="Ordenador" Grid.Row="2" Grid.ColumnSpan="2"  
Style="{StaticResource etiquetaTituloSeccion}"/>
```

# **CUARTA PARTE (OPCIONAL): VENTANAS**

# Ventanas / Páginas

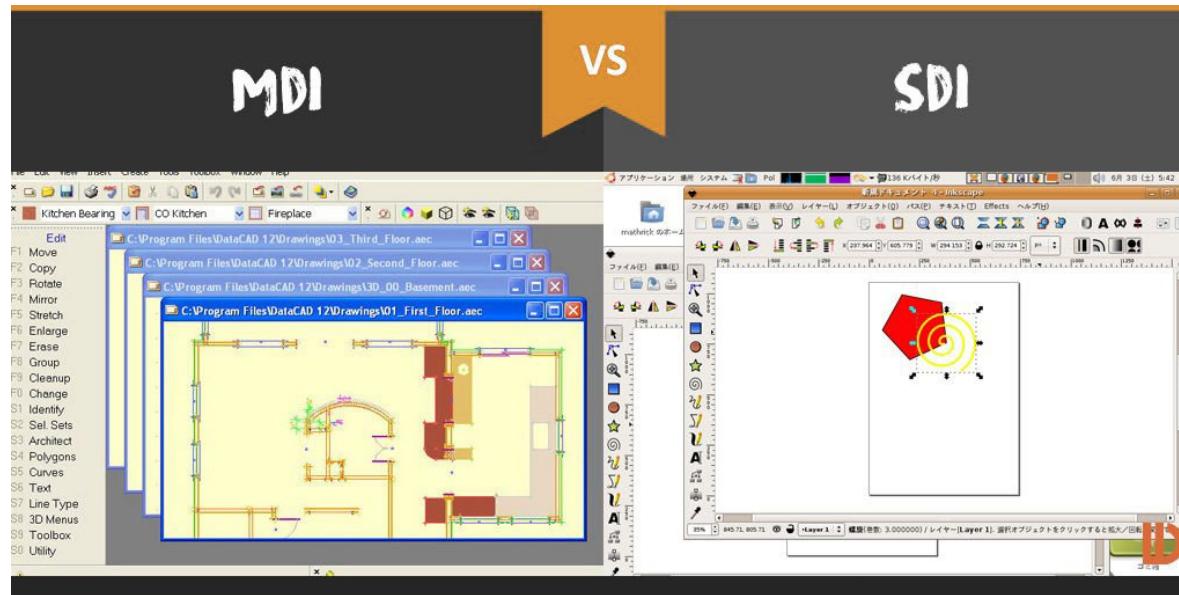
- ▶ Actualmente usamos Single Document Interface (SDI): usamos clases Frame y Page
- ▶ Multiple Document Interface: usamos clase Window

① Note

Under Windows 95 and later, applications are commonly SDI because the operating system has adopted a "document-centered" view.

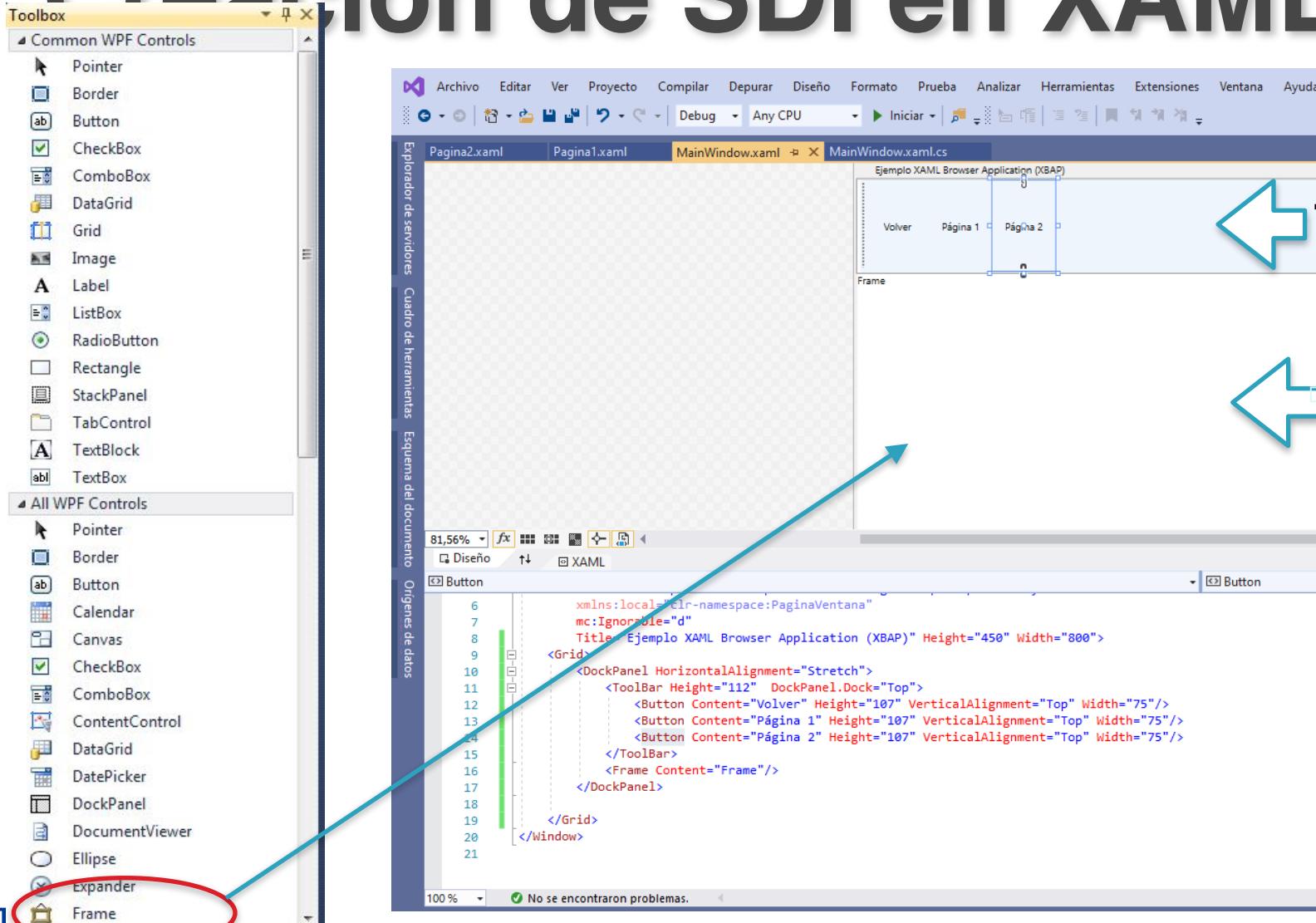
# Ventanas / Páginas

- ▶ Actualmente usamos Single Document Interface (SDI)
- ▶ Multiple Document Interface (MDI)



<https://diffzi.com/mdi-vs-sdi/>

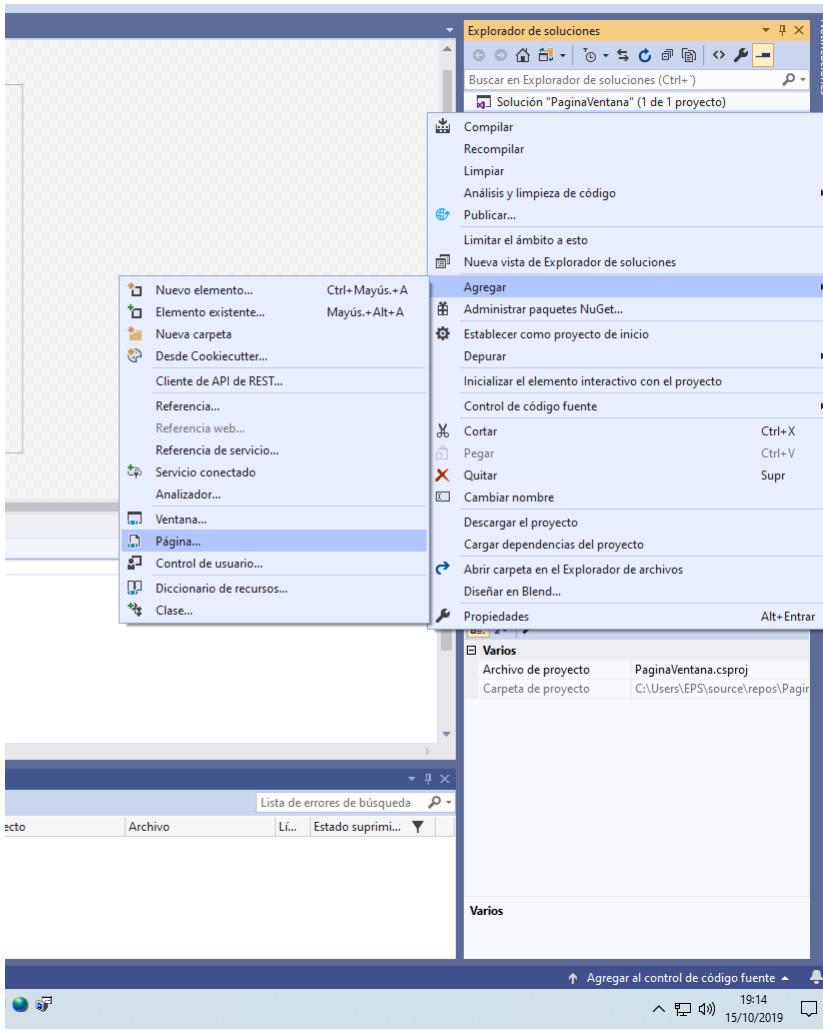
# Creación de SDI en XAML



ToolBar

Frame

# Creación de SDI en XAML



# Creación de SDI en XAML

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays the menu: Archivo, Editar, Ver, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, and Buscar en Visual S. The toolbar below the menu bar includes icons for file operations like Open, Save, and Print, as well as Debug and Run buttons.

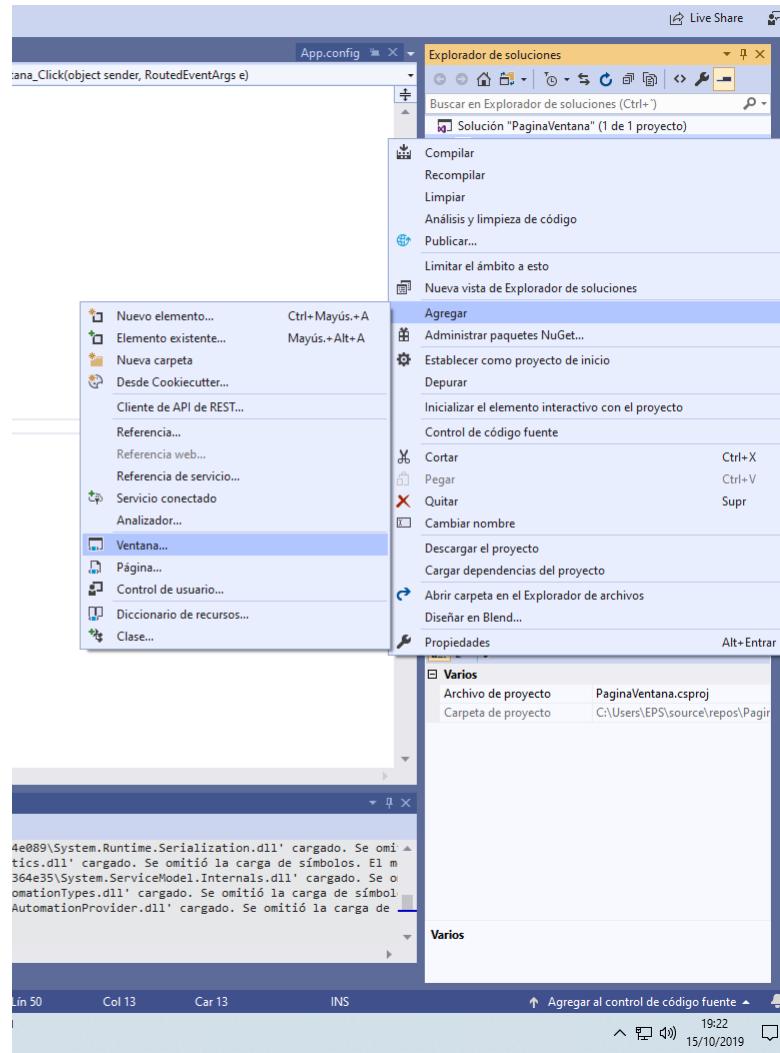
The tabs at the top of the code editor show several files: Pagina3.xaml, Pagina2.xaml, Pagina1.xaml, MainWindow.xaml\*, MainWindow.xaml.cs\*, and PaginaVentana.MainWindow. The MainWindow.xaml.cs\* tab is currently active, displaying C# code.

The code in the editor is as follows:

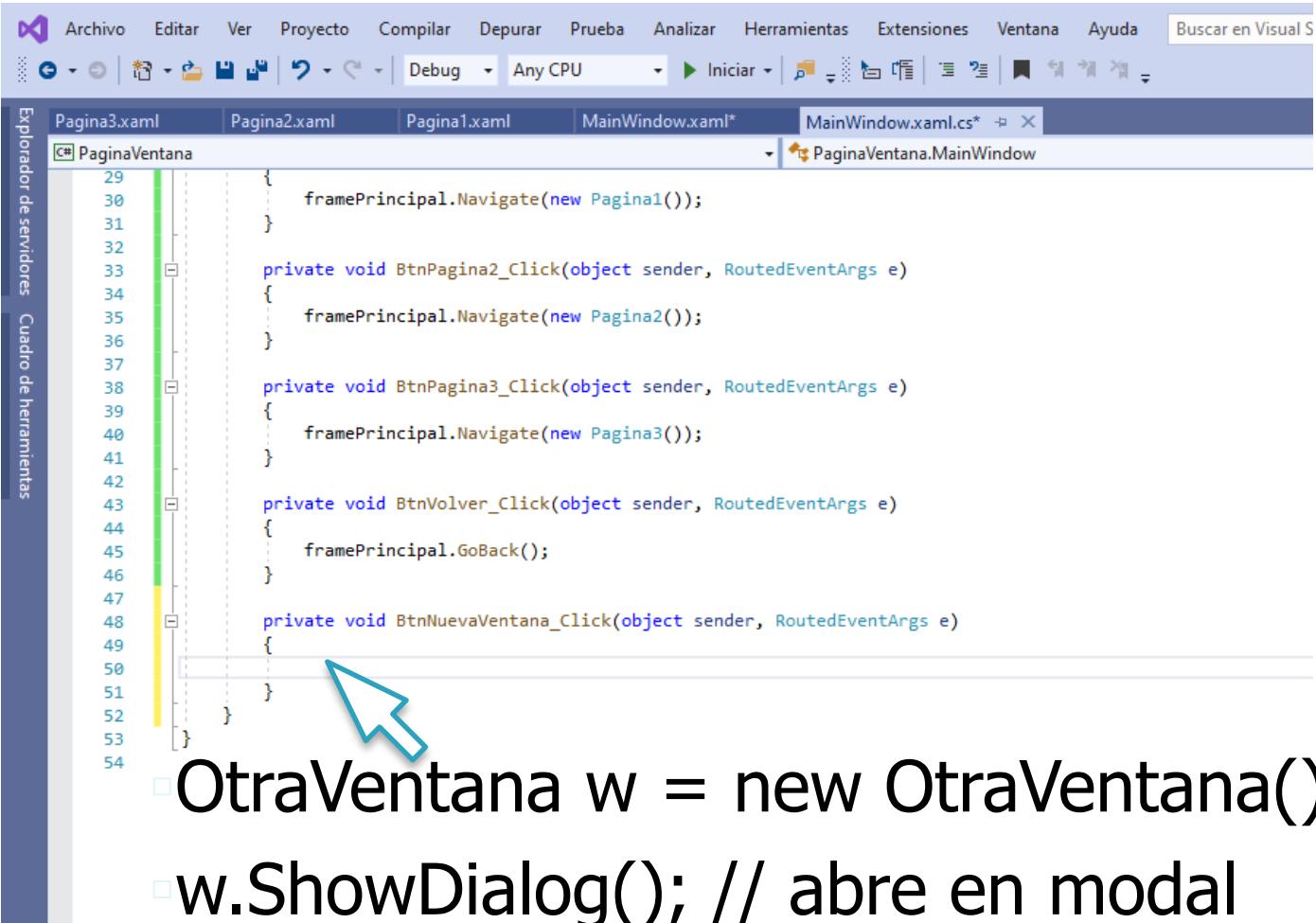
```
29     framePrincipal.Navigate(new Pagina1());
30 }
31
32 private void BtnPagina2_Click(object sender, RoutedEventArgs e)
33 {
34     framePrincipal.Navigate(new Pagina2());
35 }
36
37 private void BtnPagina3_Click(object sender, RoutedEventArgs e)
38 {
39     framePrincipal.Navigate(new Pagina3());
40 }
41
42 private void BtnVolver_Click(object sender, RoutedEventArgs e)
43 {
44     framePrincipal.GoBack();
45 }
46
47 private void BtnNuevaVentana_Click(object sender, RoutedEventArgs e)
48 {
49 }
50 }
51 }
52 }
53 }
54 }
```

The code implements event handlers for four buttons: BtnPagina1, BtnPagina2, BtnPagina3, and BtnVolver. The BtnPagina1 handler navigates to a new instance of Pagina1. The BtnPagina2, BtnPagina3, and BtnVolver handlers navigate to instances of Pagina2 and Pagina3 respectively. The BtnVolver handler uses the GoBack method of the framePrincipal to return to the previous page. The BtnNuevaVentana\_Click handler is currently empty.

# Creación de SDI en XAML



# Creación de SDI en XAML



The screenshot shows the Visual Studio IDE with the following details:

- Menu Bar:** Archivo, Editar, Ver, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, Buscar en Visual S.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and Build.
- Tab Bar:** Pagina3.xaml, Pagina2.xaml, Pagina1.xaml, MainWindow.xaml\*, MainWindow.xaml.cs\* (active tab).
- Solution Explorer:** Shows files PaginaVentana.cs and PaginaVentana.MainWindow.
- Code Editor:** Displays C# code for a window with three buttons:

```
29     framePrincipal.Navigate(new Pagina1());
30 }
31
32 private void BtnPagina2_Click(object sender, RoutedEventArgs e)
33 {
34     framePrincipal.Navigate(new Pagina2());
35 }
36
37 private void BtnPagina3_Click(object sender, RoutedEventArgs e)
38 {
39     framePrincipal.Navigate(new Pagina3());
40 }
41
42 private void BtnVolver_Click(object sender, RoutedEventArgs e)
43 {
44     framePrincipal.GoBack();
45 }
46
47 private void BtnNuevaVentana_Click(object sender, RoutedEventArgs e)
48 {
49     OtraVentana w = new OtraVentana();
50     w.ShowDialog(); // abre en modal
51 }
52 }
53 }
```

A large blue arrow points from the text "OtraVentana w = new OtraVentana(); w.ShowDialog(); // abre en modal" down to the corresponding line in the code editor.

OtraVentana w = new OtraVentana();  
w.ShowDialog(); // abre en modal  
// o w.Show(); // no bloquea - no modal

# Ampliación

- ▶ Distribuir las distintas propiedades del monitor de sistema en páginas
- ▶ Mostrar una ventana “Acerca de...”