FW: MVVM - Silverlight

Arun Manglick <arun.manglick@thedigitalgroup.net> Mon, Sep 6, 2010 at 9:12 AM
To: arunmanglick@gmail.com

The **Model View ViewModel (MVVM)** is an architectural pattern used in software engineering that originated from Microsoft as a specialization of the Presentation Model design pattern introduced by Martin Fowler.

Why MVVM & Why not MVC or MVP:

In the MVC pattern, the Model is the data, the View is the user interface, and the Controller is the programmatic interface between the view, the model, and the user input.
This pattern, however, does not work well in declarative user interfaces like WPF or Silverlight. Reason, the XAML that these technologies uses can define some of the interface between the input and the view (Using Data Binding, Triggers, and states can be declared in XAML).

Model-View-Presenter (MVP) is another common pattern for layering applications.
In the MVP pattern, the presenter is responsible for setting and managing state for a view.
Like MVC, MVP does not quite fit the Silverlight model because the XAML might contain declarative Data Binding, Triggers, & State Management.

So where does that leave us?

Luckily for Silverlight , the WPF community has rallied behind a pattern called Model-View-ViewModel (**MVVM**).
This pattern is an adaptation of the MVC and MVP patterns in which:

- The ViewModel provides a data model and behavior to the View. This allows the View to declaratively bind to the ViewModel.
- The View becomes a mix of XAML and C#.
- The Model represents the data available to the application, and
- The ViewModel prepares the Model in order to bind it to the View.

Note: As the MVVM pattern is designed to support WPF and Silverlight, this pattern is only newly *Available To The Public*, as opposed to MVC or Model View Presenter (MVP).

MVVM:

Largely based on the MVC pattern, MVVM is targeted at modern UI development platforms (WPF & Silverlight).

MVVM pattern designed to gain two things.

- The advantages of separation of functional development provided by MVC
- And leveraging the advantages of XAML and the WPF, by binding data **as close as possible** to the Model while using the XAML. Thus minimizing the need for “code behind,” especially in the View

- MVVM was designed to make use of specific functions in WPF & Silverlight, to better facilitate the separation of View layer development from the rest of the pattern.
- This is done by removing virtually all “code behind” from the View layer.
- Instead of requiring designers to write View code, they can use the native WPF markup language XAML and *Create Bindings To The Viewmodel*, which is written and maintained by application developers.
- This separation of roles allows designers to focus on UX needs rather than programming or business logic, allowing for the layers of an application to be developed in multiple work streams.

Elements of the MVVM pattern are described as below - Also similartiies with MVC.

Model	As in the classic MVC pattern, the model refers to either (a) An object model that represents the real state content (an object-oriented approach), or (b) The data access layer that represents that content (a data-centric approach).

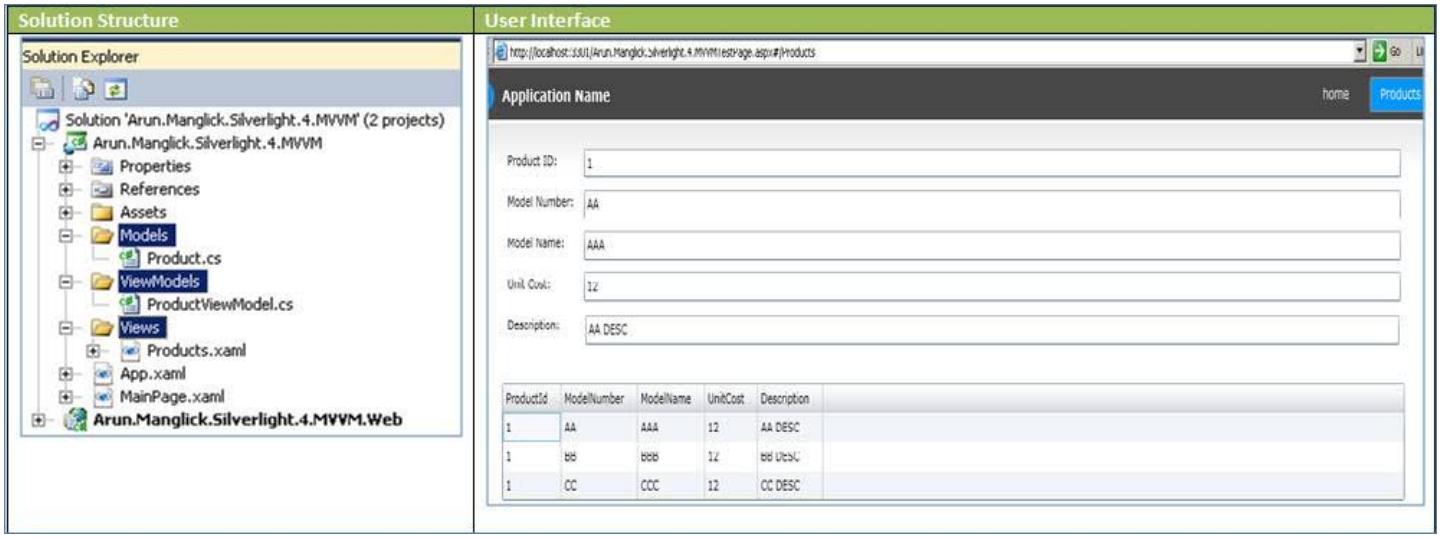
View	As in the classic MVC pattern, the view refers to all elements displayed by the GUI such as buttons, windows, graphics, and other controls.
ViewModel	The ViewModel is a “Model of the View” that serves in data binding between the View and the Model. The View-Model of MVVM is responsible for exposing the data objects from the Model, in such a way that those objects are easily consumed in XAML. It is similar to the Controller (in the MVC pattern) that acts as a Data Binder that changes Model information into View information and passes commands from the View into the Model. The ViewModel exposes public properties, commands, and abstractions.
Controller	some references for MVVM also include a Controller layer or illustrate that the ViewModel is a specialized functional set in parallel with a Controller, while others do not. This difference is an ongoing area of discussion regarding the standardization of the MVVM pattern.

Criticism

There are currently three main areas of criticism regarding the MVVM pattern.

1. MVVM currently lacks standardization from Microsoft both in implementation and in toolsets
2. The overhead in implementing MVVM is “overkill” for simple UI operations. Also for larger applications, generalizing the View layer becomes more difficult.
3. Moreover, data binding, if not managed well, can result in considerable memory consumption in an application.
4. Exercise in creating large numbers of data bindings to the ViewModel results in duplicate code and maintenance problems.

Implementation:



	Model

```
namespace Arun.Manglick.Silverlight.Model

{

    public class Product : INotifyPropertyChanged

    {

        #region Private Variables


        private int productId;

        private string modelNumber;

        private string modelName;

        private double unitCost;

        private string description;
```

```
#endregion

#region Constructor

public Product()
{
}

public Product(string modelNumber, string modelName,double unitCost, string description)
{
    ModelNumber = modelNumber;
    ModelName = modelName;
    UnitCost = unitCost;
    Description = description;
}

#endregion

#region Properties

public int ProductId
{
    get { return productId; }
    set
    {
        if (value < 0)
        {
            throw new ArgumentException("Product Id - Can't be less than 0.");
        }
        else
        {
            productId = value;
            OnPropertyChanged(new PropertyChangedEventArgs("ProductId"));
        }
    }
}
public string ModelNumber
{
    get { return modelNumber; }
    set {
        modelNumber = value;
        OnPropertyChanged(new PropertyChangedEventArgs("ModelNumber"));
    }
}
public string ModelName
{
    get { return modelName; }
    set {
```

```
            modelName = value;
        OnPropertyChanged(new PropertyChangedEventArgs("ModelName"));
        }
    }
    public double UnitCost
    {
        get { return unitCost; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Can't be less than 0.");
            }
            else
            {
                unitCost = value;
                OnPropertyChanged(new PropertyChangedEventArgs("UnitCost"));
            }
        }
    }
    public string Description
    {
        get { return description; }
        set {
            description = value;
            OnPropertyChanged(new PropertyChangedEventArgs("Description"));
        }
    }

    #endregion

    #region INotifyPropertyChanged Members

    /// <summary>
    ///
    /// </summary>
    public event PropertyChangedEventHandler PropertyChanged;

    /// <summary>
    ///
    /// </summary>
    /// <param name="e"></param>
    public void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, e);
    }

    #endregion
```

```
    }
}
```

ViewModel

```
namespace Arun.Manglick.Silverlight.ViewModels
{
    public class ProductViewModel : INotifyPropertyChanged
    {
        #region Variables

        Product theProduct;
        ObservableCollection<Product> theProducts = new ObservableCollection<Product>();
        public event EventHandler LoadComplete;

        #endregion

        #region .ctor
        public ProductViewModel()
        {
        }
        #endregion

        #region Properties

        public ObservableCollection<Product> AllProducts
        {
            get
            {
                return theProducts;
            }
            set
            {
                theProducts = value;
                this.NotifyPropertyChanged("AllProducts");
            }
        }

        public Product SingleProduct
        {
            get
            {
                if (theProducts.Count > 0)
                    return theProducts[0] as Product;

                return null;
            }
```

```
            set
            {
                theProduct = value;
                this.NotifyPropertyChanged("SingleProduct");
            }
        }

        #endregion

        #region Methods

        public void Refresh()
        {
            ObservableCollection<Product> lst = GetData();
            AllProducts = lst;
            SingleProduct = theProducts[0] as Product;
            if (LoadComplete != null) LoadComplete(this, null);
        }

        private ObservableCollection<Product> GetData()
        {
            // This could be a Service Call.
            ObservableCollection<Product> lst = new ObservableCollection<Product> {
                new Product{ ProductId=1, ModelNumber="AA", ModelName="AAA", UnitCost=12, Description= "AA DESC"},
                new Product{ ProductId=1, ModelNumber="BB", ModelName="BBB", UnitCost=12, Description= "BB DESC"},
                new Product{ ProductId=1, ModelNumber="CC", ModelName="CCC", UnitCost=12, Description= "CC DESC"}
            };
            return lst;
        }

        #endregion

        #region Property Changed Implementation

        public event PropertyChangedEventHandler PropertyChanged;

        protected void NotifyPropertyChanged(String info)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(info));
            }
        }

        #endregion
    }
}
```

View

```
<navigation:Page x:Class="Arun.Manglick.Silverlight.Views.SimpleBindingProducts"
                 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                 xmlns:navigation="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigation"
                 xmlns:ViewModel="clr-namespace:Arun.Manglick.Silverlight.ViewModels"
                 xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">
    <UserControl.Resources>
        <ViewModel:ProductViewModel x:Key="TheProductViewModel"
                                    LoadComplete="viewModel_LoadComplete"
                                    d:IsDataSource="True"></ViewModel:ProductViewModel>
    </UserControl.Resources>
    <StackPanel>
        <Grid Name="gridProductDetails"
              DataContext="{Binding Path=SingleProduct, Mode=TwoWay, Source={StaticResource TheProductViewModel}}">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"></ColumnDefinition>
                <ColumnDefinition></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
            </Grid.RowDefinitions>
            <TextBlock Margin="7">Product ID:</TextBlock>
            <TextBox Margin="5" Grid.Column="1" x:Name="txtProductId" Text="{Binding ProductId, Mode=TwoWay}"></TextBox>
            <TextBlock Margin="7" Grid.Row="1">Model Number:</TextBlock>
            <TextBox Margin="5"   Grid.Row="1" Grid.Column="1" Text="{Binding ModelNumber, Mode=OneWay}"></TextBox>
            <TextBlock Margin="7" Grid.Row="2">Model Name:</TextBlock>
            <TextBox Margin="5"   Grid.Row="2" Grid.Column="1" Text="{Binding ModelName,Mode=TwoWay}"></TextBox>
            <TextBlock Margin="7" Grid.Row="3">Unit Cost:</TextBlock>
            <TextBox Margin="5" Grid.Row="3" Grid.Column="1" Text="{Binding UnitCost, Mode=TwoWay}"></TextBox>
            <TextBlock Margin="7,7,7,0" Grid.Row="4">Description:</TextBlock>
            <TextBox Margin="7" Grid.Row="4" Grid.Column="1" Text="{Binding Description , Mode=TwoWay}"></TextBox>
            <StackPanel Grid.Row="6" Grid.Column="0" Grid.ColumnSpan="2" Orientation="Vertical">
                <ProgressBar Height="20" Visibility="Collapsed" IsIndeterminate="True" x:Name="loadingBar" />
            </StackPanel>
        </Grid>
        <sdk:DataGrid AutoGenerateColumns="True" Name="dataGrid1"
                      ItemsSource="{Binding Path=AllProducts, Mode=TwoWay, Source={StaticResource TheProductViewModel}}" />
    </StackPanel>
```

```
</navigation:Page>
```

```
namespace Arun.Manglick.Silverlight.Views
{
    public partial class SimpleBindingProducts : Page
    {
        ProductViewModel viewModel = null;

        #region Constructor

        /// <summary>
        ///
        /// </summary>
        public SimpleBindingProducts()
        {
            InitializeComponent();
            this.Loaded += new RoutedEventHandler(SimpleBindingProducts_Loaded);
        }

        #endregion

        #region Events

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SimpleBindingProducts_Loaded(object sender, RoutedEventArgs e)
        {
            loadingBar.Visibility = Visibility.Visible;
            viewModel = Resources["TheProductViewModel"] as ProductViewModel;
            viewModel.Refresh();
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void viewModel_LoadComplete(object sender, System.EventArgs e)
        {
            if (loadingBar != null) loadingBar.Visibility = Visibility.Collapsed;
        }

        #endregion
    }
```

```
}
```

Hope this helps.

Thanks & Regards,

T/DG
THE DIGITAL GROUP

Arun Manglick
Project Lead

Digital Group InfoTech Pvt. Ltd.

Pyramid building, Plot No.5

Rajeev Gandhi InfoTech Park, Phase I,

Hinjewadi, Pune - 411057, India

Office: +91 20 66532084

Fax: +91 20 66532052

Mobile: +91 9850901262