XAML Patterns

Michael L Perry

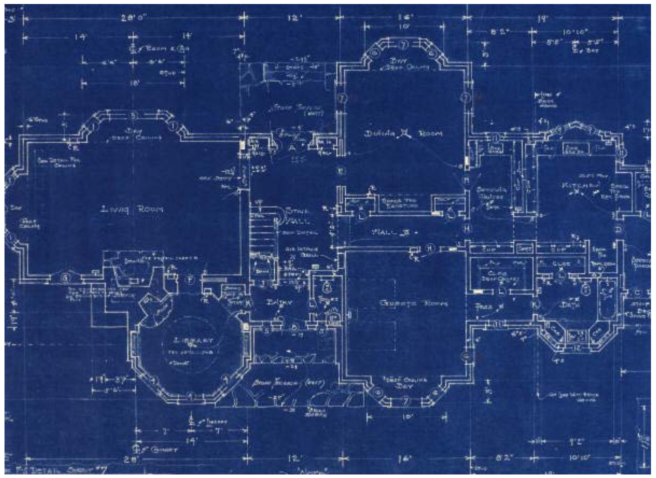
Michael@qedcode.com

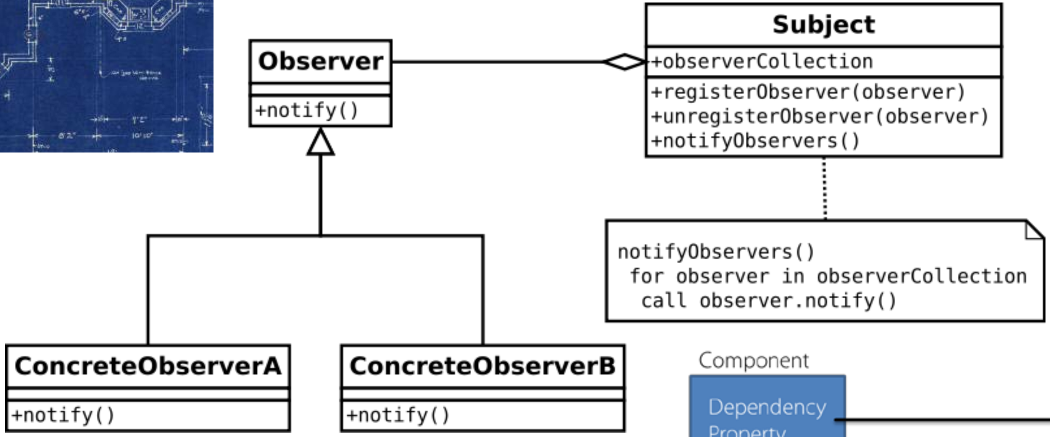
pluralsight
hardcore developer training

Design Patterns

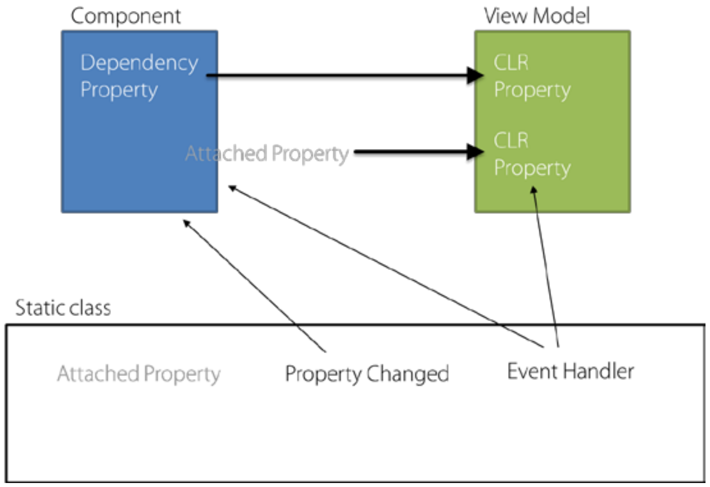
- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

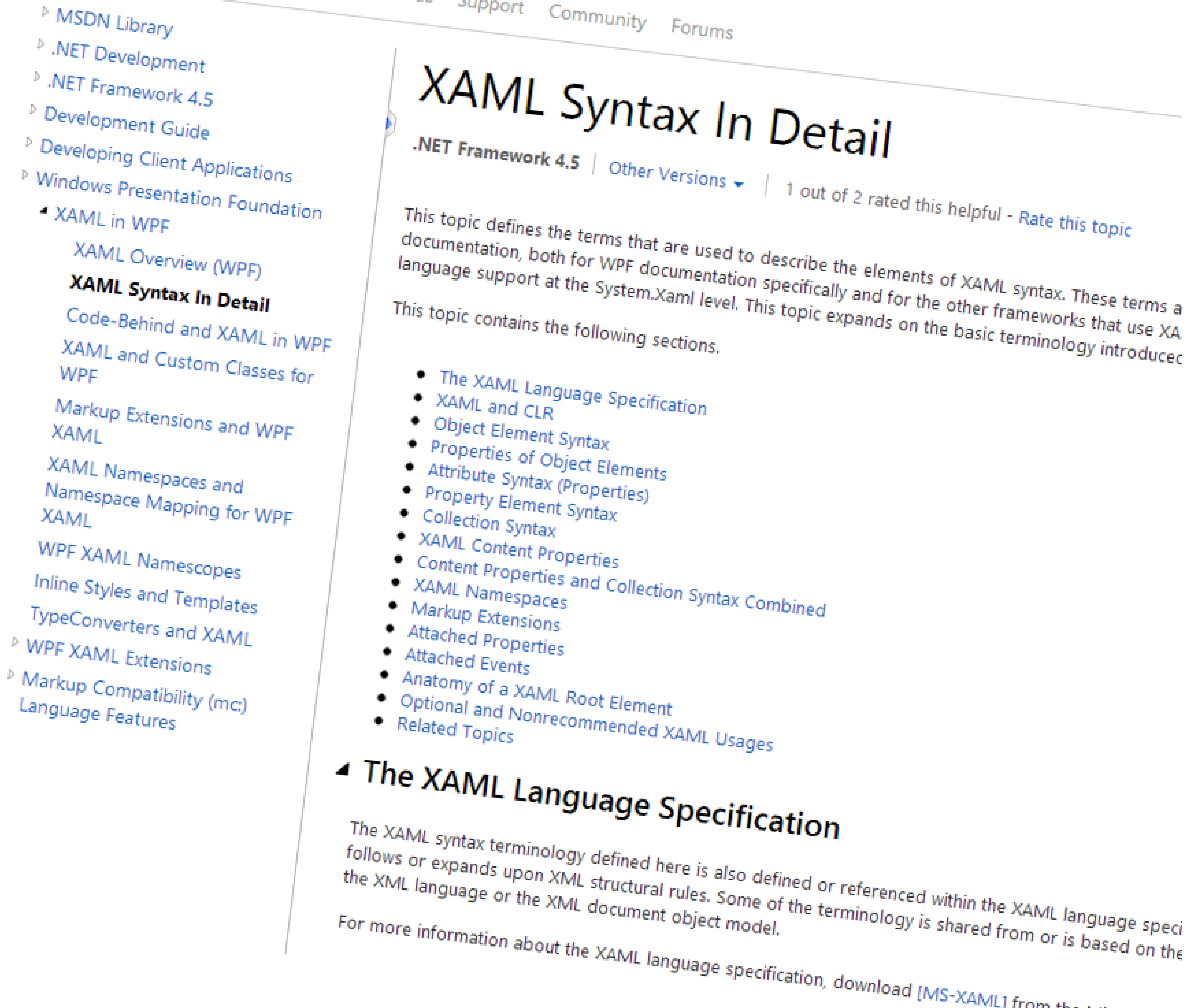
Pattern Languages





```
notifyObservers()
 for observer in observerCollection
  call observer.notify()
```





XAML Syntax In Detail

.NET Framework 4.5 | Other Versions | 1 out of 2 rated this helpful - Rate this topic

This topic defines the terms that are used to describe the elements of XAML syntax. These terms a documentation, both for WPF documentation specifically and for the other frameworks that use XA language support at the System.Xaml level. This topic expands on the basic terminology introduced

This topic contains the following sections.

The XAML Language Specification

The XAML syntax terminology defined here is also defined or referenced within the XAML language speci follows or expands upon XML structural rules. Some of the terminology is shared from or is based on the the XML language or the XML document object model.

For more information about the XAML language specification, download [MS-XAML] from the

Patterns

- **Composition**
  - Layout
    - Grid Layout
    - Balanced Whitespace
    - Overflow
    - Extension Grid
  - Organization
    - Assets
    - Control Templates
    - Control Extension
    - Implicit Data Templates

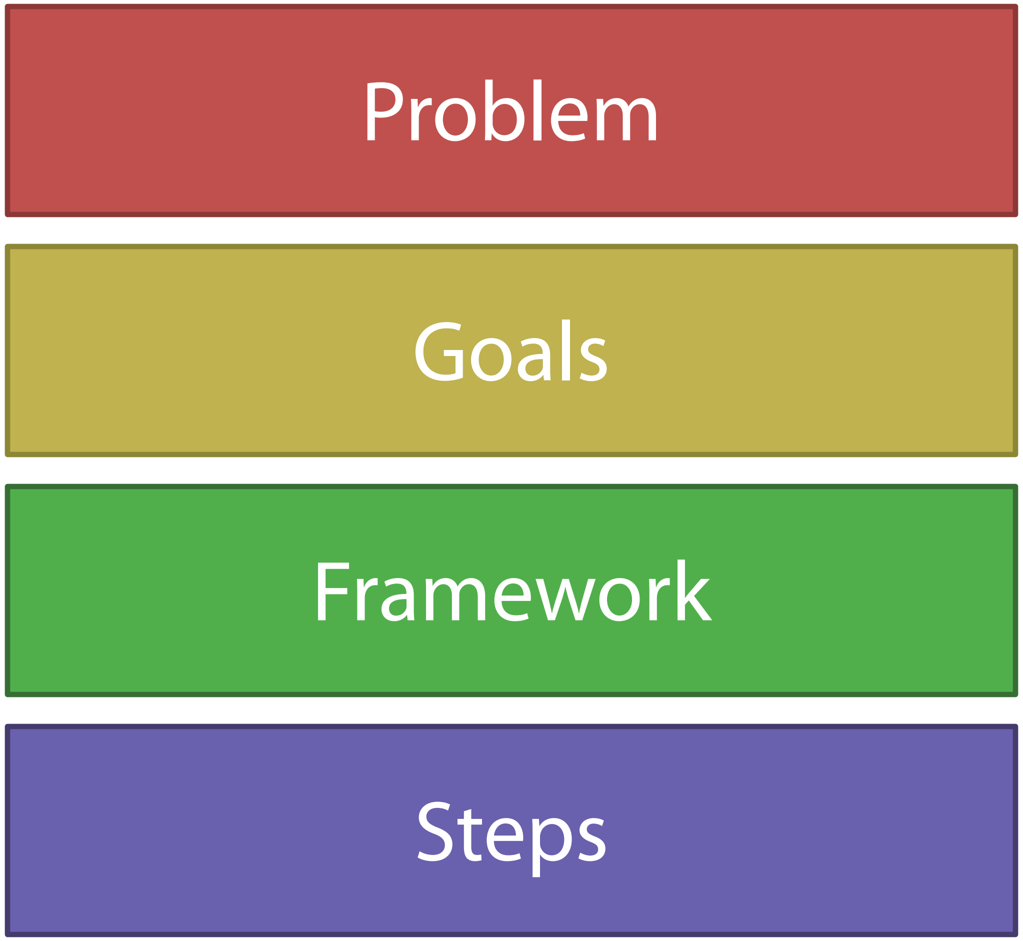
- **View Models**
  - State Management
    - Stateful
    - Stateless
    - Reactive
  - Interaction
    - Message Bus
    - Selection Model
  - Association
    - View Model Locator
    - View Model First
  - Integration
    - View Services
    - View Model Events

- **Behavioral**
  - Commands
    - Relay Command
    - Reactive Command
    - Dependent Command
  - Designer
    - Attached Behaviors
    - Blend Behaviors
    - Trigger Actions
    - Targeted Trigger Actions

- **Animations**
  - Visual States
    - Visual State Binding
    - Circular Animations
    - Control States
  - Continuity
    - List Item Animations
    - Theme Transitions

- **Design-Time Data**
  - Designer
    - In-Place Data
    - In-Place Child Items
    - Sample Data
  - Developer
    - Design Mode View Models
    - Design Mode Models
    - Shadow Types

Pattern Structure



Prerequisites

Basic understanding of XAML

Jesse Liberty

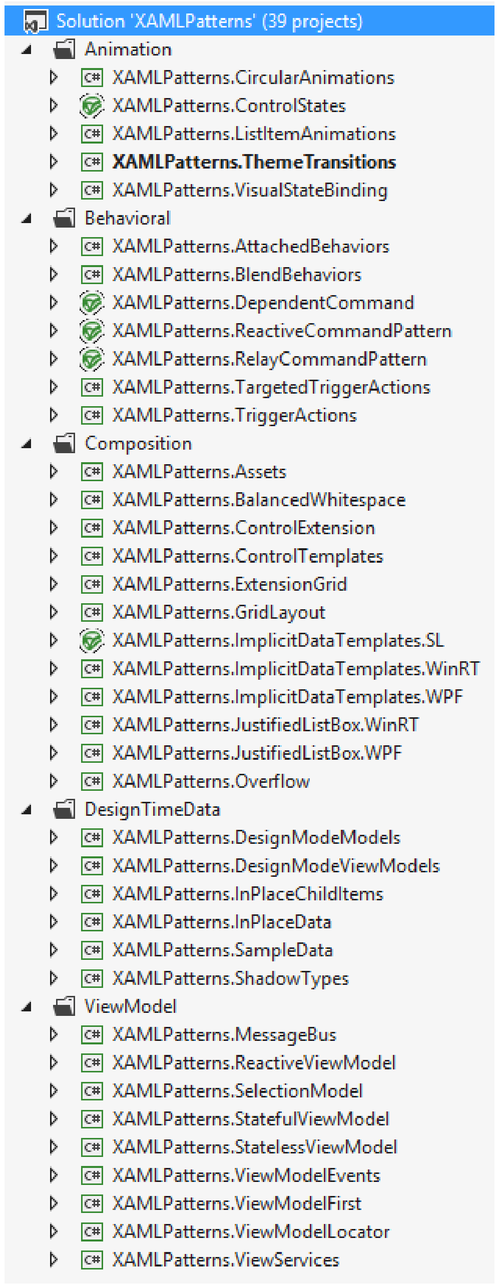
Enterprise WPF
with XAML and C#
From Scratch

Ian Griffiths

WPF and XAML
Fundamentals

Course Structure

- **Pattern modules (3 – 7)**
  - Beginning to end
  - Skip around
  - Refer back
  - Grouped by usage
  - Source code available
- **Introductory modules (1 and 2)**
  - Deconstruct your basic understanding
  - Building blocks
  - What they do



```
Solution 'XAMLPatterns' (39 projects)
◢ Animation
    ▷ XAMLPatterns.CircularAnimations
    ▷ XAMLPatterns.ControlStates
    ▷ XAMLPatterns.ListItemAnimations
    ▷ XAMLPatterns.ThemeTransitions
    ▷ XAMLPatterns.VisualStateBinding
◢ Behavioral
    ▷ XAMLPatterns.AttachedBehaviors
    ▷ XAMLPatterns.BlendBehaviors
    ▷ XAMLPatterns.DependentCommand
    ▷ XAMLPatterns.ReactiveCommandPattern
    ▷ XAMLPatterns.RelayCommandPattern
    ▷ XAMLPatterns.TargetedTriggerActions
    ▷ XAMLPatterns.TriggerActions
◢ Composition
    ▷ XAMLPatterns.Assets
    ▷ XAMLPatterns.BalancedWhitespace
    ▷ XAMLPatterns.ControlExtension
    ▷ XAMLPatterns.ControlTemplates
    ▷ XAMLPatterns.ExtensionGrid
    ▷ XAMLPatterns.GridLayout
    ▷ XAMLPatterns.ImplicitDataTemplates.SL
    ▷ XAMLPatterns.ImplicitDataTemplates.WinRT
    ▷ XAMLPatterns.ImplicitDataTemplates.WPF
    ▷ XAMLPatterns.JustifiedListBox.WinRT
    ▷ XAMLPatterns.JustifiedListBox.WPF
    ▷ XAMLPatterns.Overflow
◢ DesignTimeData
    ▷ XAMLPatterns.DesignModeModels
    ▷ XAMLPatterns.DesignModeViewModels
    ▷ XAMLPatterns.InPlaceChildItems
    ▷ XAMLPatterns.InPlaceData
    ▷ XAMLPatterns.SampleData
    ▷ XAMLPatterns.ShadowTypes
◢ ViewModel
    ▷ XAMLPatterns.MessageBus
    ▷ XAMLPatterns.ReactiveViewModel
    ▷ XAMLPatterns.SelectionModel
    ▷ XAMLPatterns.StatefulViewModel
    ▷ XAMLPatterns.StatelessViewModel
    ▷ XAMLPatterns.ViewModelEvents
    ▷ XAMLPatterns.ViewModelFirst
    ▷ XAMLPatterns.ViewModelLocator
    ▷ XAMLPatterns.ViewServices
```

Object Graphs

```xml
xmlns:hw="clr-namespace:HelloWorld"
```

```csharp
using hw = HelloWorld;
```

```xml
<TextBlock
    Text="Hello" />
```

```csharp
var textBlock = new TextBlock();
textBlock.Text = "Hello";
```

```xml
<Border>
    <Border.Background>
        <LinearGradientBrush>
            <!-- -->
        </LinearGradientBrush>
    </Border.Background>
</Border>
```

```csharp
var border = new Border();
border.Background =
    new LinearGradientBrush();
```

```xml
<Border>
    <TextBlock />
</Border>
```

```csharp
border.Background = brush;
border.Child = new TextBlock();
```

```xml
<StackPanel>
    <Border />
    <TextBlock />
</StackPanel>
```

```csharp
var stackPanel = new StackPanel();
stackPanel.Children.Add(new Border());
stackPanel.Children.Add(new TextBlock());
```

Dependency Properties

- **Set by external forces:**
  - Data binding
  - Styles
  - Animations

```
public static DependencyProperty AwesomenessProperty =
    DependencyProperty.Register(
        "Awesomeness",
        typeof(double),
        typeof(AwesomeControl));

public double Awesomeness
{
    get { return (double)GetValue(AwesomenessProperty); }
    set { SetValue(AwesomenessProperty, value); }
}
```

Misnomer

- **No dependency tracking**
  - See Stateless View Models
- **Better names**
  - Bindable properties
  - Stylable properties
  - Animatable properties

Styles

XAML ≠ *CSS*

Selectors

```
span,
.my-class,
#my-id,
div .inner-class,
a:hover
{
}
```

```
<Style
    x:Key="MyStyle">
</Style>
```

Cascading

```
div
{
    background-color: blue;
}

.my-class
{
    background-color: green;
}
```

```
<div class="my-class">
    What color am I?
</div>
```

```
<Style x:Key="BlueBorder"
       TargetType="Border">
    <Setter Property="Background"
            Value="Blue" />
</Style>
<Style x:Key="GreenBorder"
       TargetType="Border">
    <Setter Property="Background"
            Value="Green" />
</Style>
```

```
<Border
    Style="{StaticResource BlueBorder}">
    <TextBlock
        Text="What color am I?" />
</Border>
```

Properties

```css
.my-class
{
    background-color: green;
    margin: 4px;
    color: white;
    font-size: 15px;
    border-style: solid;
    text-decoration: none;
}
```

```xml
<Style
    x:Key="MyStyle"
    TargetType="AwesomeButton">
    <Setter
        Property="Template">
        <Setter.Value>
            <ControlTemplate>
                <!-- -->
            </ControlTemplate>
        </Setter.Value>
    </Setter>
    <Setter
        Property="Awesomeness"
        Value="11.0" />
</Style>
```

Summary

- **Design Patterns**
  - Higher level of abstraction
  - Goal-oriented
- **Building blocks:**
  - Object graphs
  - Dependency properties
  - Styles