Zbigniew ROSIEK

PROGRAMOWANIE WINDOWS PRESENTATION FOUNDATION (AVALON) W JĘZYKACH XAML ORAZ C#

Streszczenie

W grudniu 2005 roku firma Microsoft zaprezentowała zestaw nowych bibliotek dla systemów operacyjnych rodziny Windows. Zbiór ten zawiera ujednolicony interfejs realizujący wejście i wyjście w aplikacjach dla platformy .NET. Podsystem graficzny Windows Presentation Foundation (nazwa kodowa Avalon) będący następcą bibliotek MFC i Windows Forms ma integrować graficzny interfejs użytkownika, grafikę 2D i 3D, multimedia oraz generowanie/rozpoznawanie mowy. API opiera się o XML, zaś wszystkie operacje graficzne są oparte o mechanizmy grafiki wektorowej, co pozwoli w dużym stopniu wykorzystać sprzęt (np. akceleratory sprzętowe) do wyświetlania obrazu.
Podstawowym pojęciem w WPF są dokumenty. Przypominają one nieco pliki Macromedia Flash. Mogą reprezentować strony WWW, grafikę wektorową dwu i trójwymiarową, dokumenty tekstowe oraz zwykłe formatki Windows. Programista może tworzyć dokumenty WPF bezpośrednio z poziomu kodu aplikacji .NET, może także zdefiniować dokument w specjalnym formacie XAML wywodzącym się z XML-a.
XAML (Extensible Application Markup Language) to deklaratywny język, który pozwala zdefiniować obiekty i ich cechy w formacie XML. Parser języka XAML tworzy instancje obiektów zdefiniowanych w pliku źródłowym i ustala ich cechy. Język XAML został wykorzystany w bibliotece WPF do definiowania interfejsów użytkownika. Można utworzyć plik XAML definiujący interfejs użytkownika i dołączyć go do aplikacji .NET. Pozwala to, w przypadku prawidłowo zaprojektowanej aplikacji, swobodnie dodawać i zmieniać interfejsy niezależnie od istniejącej logiki programu.

Abstract

In December 2005, the Microsoft Company presented a new collection of libraries for the family of Windows operating systems. A unified interface that provides input/output services in .NET platform applications is included in this collection. The aim of the graphical subsystem called Windows Presentation Foundation (codename Avalon), which succeeded the MFC and Windows Forms, is to integrate graphical user interface, 2D and 3D graphics, multimedia and voice recognition/generation. The new API uses XML and all graphical operations are based on vector graphics, which allows to take advantage of hardware graphics accelerators.
WPF library is based on documents. These documents resemble Macromedia Flash files. They can represent WWW pages, two and three-dimensional vector graphics, text documents and plain Windows forms. A software developer can produce WPF documents directly from .NET application code: he/she can also define a document using a special, XML based format called XAML.
XAML (Extensible Application Markup Language) is a declarative language, which allows for defining objects and their properties in XML manner. XAML parser creates object instances defined in a source document and sets values of their properties. XAML language is being used to define user interfaces in WPF library. It is possible to create a XAML document defining user interface and include it in .NET application. It makes it possible to create and modify UIs irrespective of application logic.

1. WSTĘP

W grudniu 2006 roku firma Microsoft wprowadziła na rynek kolejną wersję systemu operacyjnego Windows o nazwie Vista zawierający w stosunku do poprzedniej wersji XP wiele nowych rozwiązań polegających przede wszystkim na opracowaniu zestawu nowych bibliotek. Biblioteki te mają za zadanie unowocześnić i ujednolicić interfejs realizujący wejście i wyście w aplikacjach dla platformy .NET. WinFX (taką nazwę nadano temu zestawowi bibliotek) miał początkowo składać się z trzech części:

- WinFS – nakładka na system plików NTFS realizująca dostęp do danych na dysku. Z punktu widzenia użytkownika dane na dysku mają przypominać raczej bazę danych niż hierarchiczną strukturę folderów, a sam interfejs programistyczny dostępu do danych oparty jest na XML'u.
- Windows Communication Foundation (nazwa kodowa Indigo) – zbiór interfejsów komunikacji sieciowej. WCF integruje wszystkie dotychczasowe technologie sieciowe firmy Microsoft, czyli m.in. COM+, .NET Remoting, kolejki komunikatów MSMQ oraz WebServices. Programowanie w WCF ma być niezależne od wybranego protokołu (potoki, IPC, TCP, http) oraz kompatybilne z dotychczas istniejącymi aplikacjami (stosowanie protokołu SOAP, COM+ oraz WebServices). W chwili obecnej jest dostępna jedynie wersja preview tego pakietu.
- Windows Presentation Foundation (nazwa kodowa Avalon) – podsystem graficzny będący następcą bibliotek MFC i Windows Forms. Zadaniem WPF jest zintegrowanie graficznego interfejsu użytkownika, grafiki 2D i 3D, multimedii oraz generowanie/rozpoznawanie mowy. API opiera się o XML, zaś wszystkie operacje graficzne są oparte o mechanizmy grafiki wektorowej, co pozwola w dużym stopniu wykorzystać sprzęt (np. akceleratory sprzętowe) do wyświetlania obrazu.

Cały pakiet WinFX jest integralną częścią systemu operacyjnego Microsoft Windows Vista, mają również powstać jego wersje (uzupełnienia) dla Windows XP i Windows Server 2003. Oczywiście wszystkie biblioteki składające się na WinFX są w pełni obiektowe i korzystają z dobrodziejstw platformy .NET takich jak garbage collector i kod nadzorowany.

Podstawowym pojęciem, którym będziemy się posługiwali korzystając z WPF jest dokument. Dokumenty WPF przypominają nieco pliki Macromedia Flash. Mogą one reprezentować strony WWW, grafikę wektorową dwu i trójwymiarową, dokumenty tekstowe oraz zwykłe formatki Windows. Programista może tworzyć dokumenty WPF bezpośrednio z poziomu kodu aplikacji .NET, może także zdefiniować dokument w specjalnym formacie XAML wywodzącym się z XML-a.

2. XAML – JĘZYK ZNACZNIKÓW UKIERUNKOWANY NA TWORZENIE GRAFIKI

XAML (Extensible Application Markup Language) to deklaratywny język, który pozwala zdefiniować obiekty i ich cechy w formacie XML. Parser języka XAML tworzy instancje obiektów zdefiniowanych w pliku źródłowym i ustala ich cechy. Język XAML został wykorzystany w bibliotece WPF do definiowania interfejsów użytkownika. Można utworzyć plik XAML definiujący interfejs użytkownika i dołączyć go do aplikacji .NET. Pozwala to, w przypadku prawidłowo zaprojektowanej aplikacji, swobodnie dodawać i zmieniać interfejsy niezależnie od istniejącej logiki programu. Podobne rozwiązanie jest stosowane w przeglądarce internetowej Mozilla. Interfejs użytkownika jest tam tworzony za pomocą plików w formacie XUI. Rozwiązanie to pozwala w przypadku Mozilli na łatwą zmianę kolorystyki i wyglądu interfejsu.

Kilka istotnych cech:

- Warstwa prezentacji oddzielona od warstwy logiki,
- Interfejs stworzony za pomocą XAML'a jest niezależny od platformy systemowej,
- Ponieważ XAML jest tekstowy, możliwe są zmiany w interfejsie bez ingerencji w logikę (rekompilacja jednak konieczna),
- Gdzieś tam „po drodze" aplikacja korzysta z DirectX, a więc jest szybsza niż z GDI+, ale bez akceleratora 3D działa niezadowalająco.

2.1. Budowa pliku

Plik XAML jest typowym plikiem XML'owym. Hierarchia tagów reprezentuje obiekty oraz relacje między nimi, a także atrybuty obiektów. Nazwy tagów odpowiadają nazwom klas, zaś atrybuty tagów określają wartości atrybutów obiektów, hierarchie tagów przekładają się zaś na hierarchie obiektów. W praktyce pliki XAML będą tworzone automatycznie przez zintegrowane środowisko programistyczne (np. Visual Studio), jednak nic nie stoi na przeszkodzie, aby tworzyć je ręcznie lub modyfikować przy użyciu zwykłego edytora tekstów. Poniżej pokazany jest przykładowy skrypt XAML opisujący kontrolkę ListBox zawierającą trzy wiersze do wyboru:

```
<ListBox Name="lbox" Height="127" Width="154">
<ListBoxItem Name="lBoxItem1" Background="Aqua" Width="100">
<ContentControl.Content>
```

```
Item One
</ContentControl.Content>
<ContentControl.Height>
30
</ContentControl.Height>
</ListBoxItem>
<ListBoxItem Name="lBoxItem2">
Item Two
</ListBoxItem>
<ListBoxItem Name="lBoxItem3" Content="Item Three" />
</ListBox>
```

Dane opisane w języku XAML są równoważne pewnemu kodowi napisanemu w językach .NET. Kod równoważny powyższemu, napisany w języku C# wyglądałby następująco:

```
private void WindowLoaded(object sender, EventArgs e)
{
lBox = new ListBox();
lBox.Width = 154;
lBox.Height = 127;
lBoxItem1 = new ListBoxItem();
lBoxItem1.Content = "Item One";
lBoxItem1.Background = Brushes.Aqua;
lBoxItem1.Width = 100;
lBoxItem1.Height = 30;
lBox.Items.Add(lBoxItem1);
lBoxItem2 = new ListBoxItem();
lBoxItem2.Content = "Item Two";
lBox.Items.Add(lBoxItem2);
lBoxItem3 = new ListBoxItem();
lBoxItem3.Content = "Item Three";
lBox.Items.Add(lBoxItem3);
myWindow.Children.Add(lBox);
}
```

Jak widać, najpierw wywoływany jest konstruktor obiektu nadrzędnego – w tym przypadku ListBox, następnie ustawiane są jego atrybuty. Później w ten sam sposób są tworzone jego obiekty podrzędne, które z kolei zostają dodane do ich obiektu

nadrzędnego (metoda *Add* atrybutu *Items* w obiekcie nadrzędnym). Warto zauważyć, że wartość atrybutu *Content* możemy ustalić na dwa sposoby – jako wartość atrybutu *Content* w odpowiednim tagu XML (patrz *Item 3)*, albo jako zwykły tekst wewnątrz tego tagu (*Item 1* i *Item 2*).

2.2. Obiektowość XAML

W bibliotece WPF relacje między obiektami przekładają się bezpośrednio na ich prezentacje na ekranie. I tak, w przypadku gdy tworzymy formatkę zawierającą przycisk, na którym jest obrazek i tekst, hierarchia obiektów w pliku XAML wygląda następująco:

```
<Button>
    <Image>
    </Image>
    <TextPanel>
    </TextPanel>
</Button>
```

Możliwe jest praktycznie dowolne zagnieżdżanie obiektów – przypomina to nieco język HTML, w którym można dowolnie zagnieżdżać w sobie tabelki.

W NET.Framework wszystkie klasy są pogrupowane w hierarchiczną strukturę przestrzeni nazw. Każda klasa jest określona przez swoją nazwę oraz nazwę jej przestrzeni nazw. Pozwala to pogrupować klasy w podsystemy i zachować jednoznaczność w przypadku, gdy w różnych podsystemach jest kilka klas o tej samej nazwie. Także biblioteki .NET i WinFX pogrupowane są w przestrzenie nazw. Najczęściej używane przestrzenie nazw to:

- System.Windows – zawierająca podstawowe klasy pozwalające na tworzenie aplikacji,
- System.Windows.Controls – zawierająca podstawowe kontrolki interfejsu użytkownika (takie jak pola tekstowe czy chceckboxy),
- System.Windows.Input – zawierająca podstawowe klasy i interfejsy służące do obsługi wejścia (obsługa klawiatury czy myszy).

Każdy obiekt, który chcielibyśmy umieścić w tworzonym interfejsie musi należeć do klasy dziedziczącej po klasie *UIElement*. Klasa ta definiuje podstawowe atrybuty wszystkich kontrolek wizualnych, takie jak położenie, widoczność oraz podstawowe rodzaje interakcji z użytkownikiem.

2.2.1. Podstawowe kontrolki formularzy WPF

Oczywiście WinFX implementuje wszystkie standardowe kontrolki dostępne w istniejących bibliotekach okienkowych. Możemy, więc tworzyć pola tekstowe, pola edycyjne, pola jednokrotnego i wielokrotnego wyboru, listboxy, comboboxy, menu główne i kontekstowe, paski postępu i przewijania. Skupmy się, zatem na kontrolkach specyficznych dla nowego API. Dosyć ciekawa jest kontrolka Document Viewer. Pozwala ona umieścić na formatce podgląd dokumentu XAML'owego. Kontrolka obsługuje przewijanie dokumentu i pozwala np. na jego powiększanie. Inną ciekawostką, dla której w chwili obecnej trudno znaleźć sensowne zastosowanie, jest kontrolka InkCanvas. Jest to panel, na którym możemy coś narysować korzystając z myszy albo tabletu.

Niewątpliwie bardzo ważne są kontrolki związane z Multimediami. Do reprezentowania obrazków służy klasa *Image* pozwalająca dołączyć do dokumentu zawartość pliku graficznego w formacie *.jpg, .gif, .png, .tiff, .bmp* oraz *.ico*. Poniżej przykład użycia tej klasy w dokumencie XAML:

```
<Image Source="myImage.jpg" />
```

Obsługa obrazów i dźwięków została zaimplementowana w klasie *MediaElement*. Odtwarzanie mediów jest realizowane przez podsystem multimedialny systemu operacyjnego. Prosty przykład użycia tej klasy zamieszczony jest poniżej:

```
<StackPanel>
<MediaElement Name="mySound" Source="SoundFile.wma"/>
<MediaElement Name="myVideo" Source="VideoFile.wpl" Stretch="Fill" />
</StackPanel>
```

2.2.2. Definicja układu dokumentu

Realizacja układu dokumentów w bibliotece WPF przypomina rozwiązania stosowane w bibliotekach SWT i Swing języka Java. Nie musimy jawnie określać położenia kontrolek na dokumencie lub formatce – istotne jest wzajemne położenie kontrolek. Typowym obiektem, na którym będziemy zwykle umieszczali kontrolki jest obiekt wywodzący się z klasy *Panel*. Panel kontroluje rozmiar i położenie jego obiektów podrzędnych. Różne odmiany paneli pozwalają nam zastosować różne sposoby ustalania rozmiaru i położenia. Możemy również zagnieżdżać panele uzyskując bardziej skomplikowane warstwy (layout).

Podstawowe rodzaje paneli

- Canvas – dla każdej kontrolki umieszczonej na tym panelu należy jawnie określić współrzędne i rozmiar. Pozwala to na precyzyjne określenie wyglądu np. okna dialogowego, jednak powoduje komplikacje w przypadku, gdy z jakiegoś powodu chcielibyśmy zmieniać rozmiar formularza.
- DockPanel – pierwsza dodawana kontrolka umieszczana jest na wybranym marginesie panelu (do wyboru góra, dół, lewo, prawo). Kolejne kontrolki dodawane są w ten sam sposób na pozostałej wolnej przestrzeni panelu (znane z C# „dokowanie").
- Grid – formularz dzielony jest na ustaloną liczbę wierszy i kolumn. Kontrolki dodawane są w kolejne pola powstałej ramki.
- StackPanel – kolejne kontrolki umieszczane są kolejno od lewej do prawej, albo od góry do dołu.
- BulletPanel – specjalny rodzaj panelu służący do tworzenia radiobuttonów.
- TabPanel – odmiana panelu pozwalający zaimplementować kilka zakładek na formularzu.
- ToolBarOverFlowPanel – rodzaj panelu służący do stworzenia paska narzędziowego, podobnego do tego, który występuje, np. w Microsoft Office.

2.3. Grafika wektorowa XAML

WPF posiada duże możliwości tworzenia grafiki wektorowej w dokumentach. Możemy rysować dowolne krzywe i łamane, umieszczać tekst. Narysowane krzywe zamknięte można wypełniać dowolnym kolorem, gradientem oraz teksturą. Możliwe jest też swobodne korzystanie z przezroczystości. Pod względem możliwości w grafice wektorowej WPF nie ustępuje w niczym narzędziom takim jak Macromedia Flash. Oczywiście w praktyce tworzenie bardziej skomplikowanych rysunków bezpośrednio w kodzie źródłowym byłoby mocno utrudnione, na szczęście jednak istnieją już edytory graficzne pozwalające tworzyć pliki XAML.

2.3.1. Krzywe, łamane, okręgi, wielokąty

W bibliotece WPF wszystkie krzywe są reprezentowane przez obiekty podklas klasy Shape. Najważniejsze atrybuty zdefiniowane w klasie Shape to *Stroke* i *Fill*, definiujące odpowiednio kolor pióra użytego do narysowania krzywej oraz sposób jej wypełnienia. Inne atrybuty służą do określenia kształtu i zachowania pędzla. Sam kształt krzywej jest już zdefiniowany w konkretnej podklasie klasy Shape. W bibliotece WPF dostępne są następujące rodzaje krzywych:

- Linia (Line)
- Elipsa (Ellipse)
- Wielokąt (Poligon)
- Krzywa łamana (Polyline)
- Prostokąt (Rectangle)

Poniżej pokazana jest definicja pewnej elipsy:

```
<Ellipse CenterX="50" CenterY="100" RadiusX="100" RadiusY="50"
Stroke="#000000" StrokeThickness="3">
    <Ellipse.Fill>
        <SolidColorBrush Color="Yellow" Opacity="0.4" />
    </Ellipse.Fill>
</Ellipse>
```

Oprócz podstawowych figur geometrycznych można oczywiście tworzyć bardziej skomplikowane krzywe poprzez zdefiniowanie tak zwanej ścieżki, składającej się z fragmentów takich jak odcinek, wycinek elipsy albo krzywa Beziera. Poniższy przykład pokazuje ścieżki tworzące krzywą Beziera:

```
<Path Stroke="Black" StrokeThickness="1">
    <Path.Data>
        <PathGeometry>
            <PathGeometry.Figures>
                <PathFigureCollection>
                    <PathFigure>
                        <PathFigure.Segments>
                            <PathSegmentCollection>
                                <StartSegment
Point="50,100" />
                                <QuadraticBezierSegment
Point1="250,300" Point2="500,100"/>
                            </PathSegmentCollection>
                        </PathFigure.Segments>
                    </PathFigure>
                </PathFigureCollection>
            </PathGeometry.Figures>
        </PathGeometry>
```

```
    </Path.Data>
</Path>
```

2.3.2. Wypełnienia

Korzystając z WPF możemy stosować kilka różnych rodzajów wypełnień figury dwuwymiarowej. Wnętrze figury może oczywiście zostać przezroczyste, możemy je wypełnić jednolitym kolorem, a także gradientem albo innym rysunkiem. Sposoby wypełniania są reprezentowane przez obiekty klasy *Brush* i jej podklas. Sposób wypełnienia figury ustalamy umieszczając w jej atrybucie *Fill* odpowiedni obiekt klasy *Brush*. W standardowej bibliotece WPF dostępne są następujące wypełnienia:

- SolidColorBrush – wypełnienie jednolitym kolorem - musimy jedynie ustalić kolor i przezroczystość wypełnienia,
- LinearGradientBrush – wypełnienie gradientem liniowym. Gradient może składać się z wielu przejść między kolorami, które to przejścia definiujemy w atrybutach obiektu,
- RadialGradientBrush – wypełnienie gradientem punktowym,
- ImageBrush – wypełnienie obrazkiem. Obrazek może być w jednym z następujących formatów: .jpg, .gif, .tiff, .bmp, .png i .ico,
- DrawingBrush – pozwala rysować obrazy wektorowe na powierzchni figury.

2.3.3. Transformacje

Obraz umieszczony na panelu może zostać poddany różnego rodzaju przekształceniom geometrycznym. Możemy dokonywać przekształceń złożonych z przesunięć, obrotów i przeskalowań. Możemy również zdefiniować dowolne inne przekształcenie określone macierzą 3 na 3. Przykładowa definicja przekształcenia pokazana jest poniżej:

```
<Canvas>
        <Canvas.RenderTransform>
                <TransformGroup>
                        <SkewTransform AngleX=”10” AngleY=”10” />
                        <RotateTransform Center=”0 0” Angle=”-30” />
                        <TranslateTransform X=”100” Y=”50” />
                <ScaleTransform ScaleX=”1.5” ScaleY=”1.5” />
                </TransformGroup>
        </Canvas.RenderTransform>
```

```
        <Polygon Fill=”Pink” Stroke=”#FF000000” StrokeThickness=”3”
        Points=”65,150.238333333333 10,64.2383333333333 61,9.23833333333332
        149,62.2383333333333 67,35.2383333333333 38,67.2383333333333”/>
        <Ellipse CenterX=”245” CenterY=”101.23833333333332” Fill=”#FFFF0000”
                RadiusX=”88” RadiusY=”44” Stroke=”#FF000000”/>
        <Button Width=”100” Height=”30”>Translated Button</Button>
</Canvas>
```

2.3.4. Animacja

W formacie XAML istnieje możliwość zdefiniowania animacji stworzonych wcześniej obrazków. W praktyce sprowadza się to do zdefiniowania scenariusza animacji i określenia, w jakich momentach zmieniają się atrybuty narysowanych przez nas obiektów, zatem możemy przesunąć obiekt, zmieniać jego kolor lub przemodelować kształt. Klasy definiujące przebieg animacji pochodzą od klasy bazowej *Timeline*. Wewnątrz scenariusza umieszczamy definicje przekształceń odbywających się podczas animacji. W poniższym przykładzie zdefiniowana jest animowana elipsa, której kolor wypełnienia zmienia się od czerwonego do niebieskiego.

```
<StackPanel xmlns=”http://schemas.microsoft.com/winfx/avalon/2005”>
        <StackPanel.Storyboards>
                <SetterTimeline TargetName=”myEllipse” Path=”(Ellipse.Fill).(So-
lidColorBrush.Color)”>
                        <!-- Animate from Red to Blue. -->
                <ColorAnimation From=”Red” To=”Blue” Duration=”0:0:001”
RepeatBehavior=”3x” AutoReverse=”True” />
                </SetterTimeline>
        </StackPanel.Storyboards>
        <Ellipse Name=”myEllipse” Fill=”Pink” CenterX=”200” CenterY=”200”/>
</StackPanel>
```

2.4. Zdarzenia

Nawet najładniejsza formatka pozostanie bezużyteczna dopóki za jej pomocą nie będzie można wpływać na pracę programu. WPF proponuje nam typowy mechanizm obsługi zdarzeń wywoływanych przez określone akcje użytkownika. Jeśli tworzymy aplikację łączącą logikę w języku C# i interfejs XAML, wówczas możemy w XAML’u określić metody aplikacji, jakie mają być wywoływane po zajściu zdarzenia. Bardzo często chcielibyśmy jednak, aby obsługa zdarzeń była częścią de-

finicji interfejsu, a nie logiki. Język XAML dopuszcza w takiej sytuacji na dołączenie metod napisanych w języku C# bezpośrednio do definicji interfejsu. Wygląda to tak:

```
<TextPanel xmlns=”http://schemas.microsoft.com/2003/xaml”
        xmlns:def=”Definition” def:Language=”C#”
         HorizontalAlignment=”Center”
        Background=”LightCyan”>
    <Button Click=”ButtonClick” FontSize=”72”>
        Push Me!
    </Button>
    <def:Code>
        <![CDATA[
        void ButtonClick(object el, ClickEventArgs args)
        {
            MessageBox.Show(„The button has been clicked.”);
        }
    ]]>
    </def:Code>
</TextPanel>
```

Możemy, zatem zaimplementować prostą interakcję między kontrolkami wewnątrz pliku z definicją interfejsu, zaś resztę w kodzie aplikacji.

2.5. Style

Style w WPF są rozwiązaniem bardzo podobnym do Cascading Style Sheets (*Kaskadowych Arkuszy Stylów*) w języku HTML. Pozwalają one na określenie atrybutów poszczególnych kontrolek przed ich późniejszym użyciem. Jest to dodatkowa warstwa pośrednia – przydatna w przypadku, gdy chcemy zastosować niestandardową kolorystykę lub wygląd formatki. Jak to wygląda w praktyce? Poniższy kod definiuje formatkę z trzema przyciskami. W formatce zdefiniowaliśmy styl, który ustala we wszystkich przyciskach czerwone tło.

```
<Window x:Class=”Styles.Window1”
xmlns=”http://schemas.microsoft.com/winfx/avalon/2005”
xmlns:x=”http://schemas.microsoft.com/winfx/xaml/2005”
Text=”Implicit Styling”>
```

```
        <Window.Resources>
                <Style TargetType=”{x:Type Button}”>
                        <Setter Property=”Control.Background” Value=”Red” />
                </Style>
        </Window.Resources>
<StackPanel>
        <Button Name=”Button1” Height=”30”>This button is Red.</Button>
        <Button Name=”Button2” Height=”30”>This button is also Red.</Button>
        <Button Name=”Button3” Height=”30”>This button is Red too.</Button>
</StackPanel>
</Window>
```

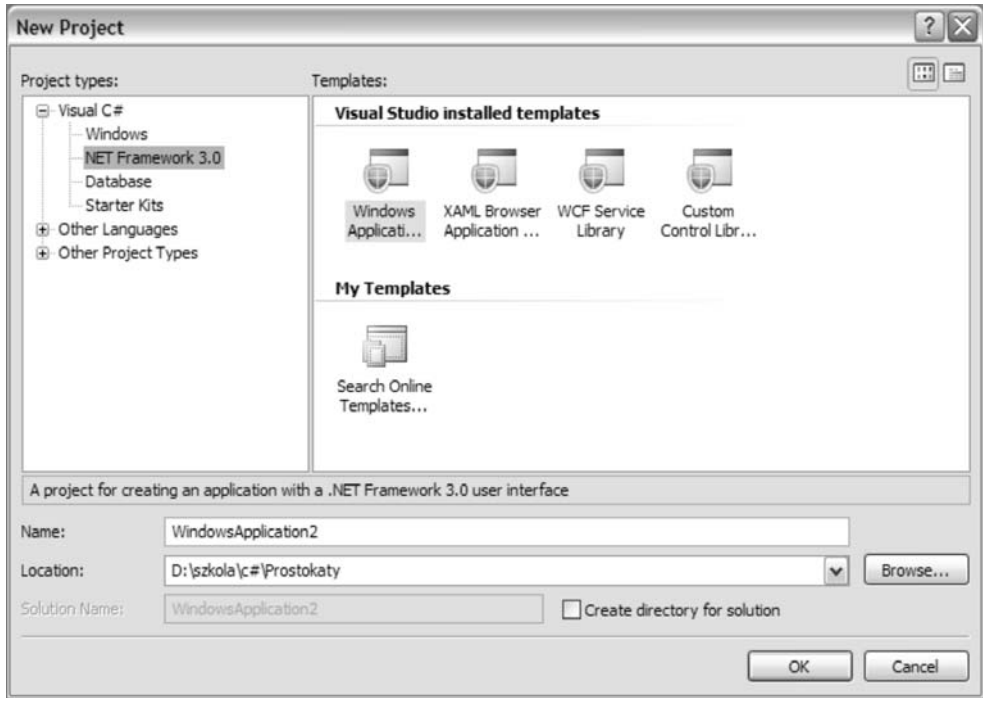
Mechanizm styli w WPF jest dosyć rozbudowany, posiada dziedziczenie oraz zmianę zdarzeń.

3. ŚRODOWISKA DO TWORZENIA OKIENEK W XAML

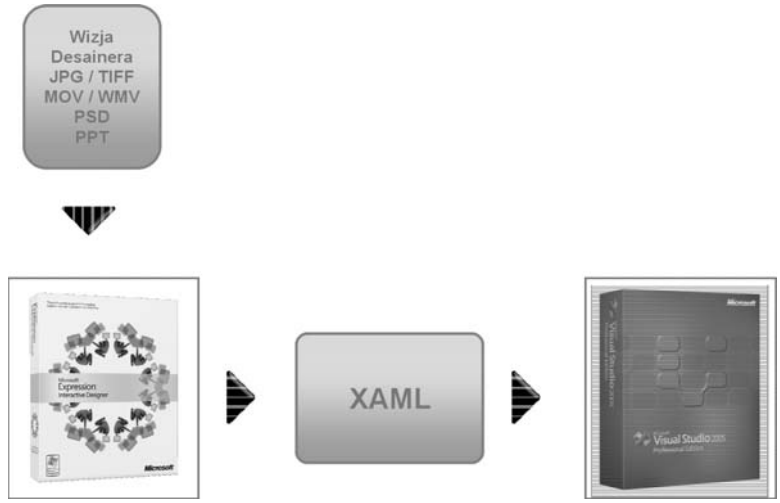
Jeżeli czytelnik ma jakieś doświadczenie w tworzeniu stron internetowych z pewnością zdaje sobie sprawę z faktu, że stworzenie okienka zawierającego wiele kontrolek i do tego ładnego będzie w przypadku braku jakichś narzędzi wspomagających zadaniem trudnym i czasochłonnym. Oczywiście firma Microsoft oraz inne firmy specjalizujące się w tworzeniu narzędzi wspomagających tworzenie grafiki pomyślały o tym zawczasu. Zadanie miały trochę ułatwione dzięki przedłużaniu się procesu tworzenia nowego systemu operacyjnego Windows. Mówiąc o tworzeniu okienek należy podkreślić znaczenie oddzielenia interfejsu graficznego od jego logiki. Interfejs tworzy raczej plastyk, który ma prawo niezbyt dobrze poruszać się w zagadnieniach logiki i ogólnie rzecz biorąc programowania, zaś logiką powinien zając się programista. I tak to jest właśnie robione w nowoczesnych narzędziach wspomagających tworzenie interfejsów graficznych.

Dla Visual Studio 2005 do tworzenia okienek w XAML dostępna jest w MSDN wtyczka o nazwie kodowej ***„Cider”,*** docelowo stanowić będzie integralną część kolejnej wersji VS, która posiada roboczą nazwę ***„Orcas”.***

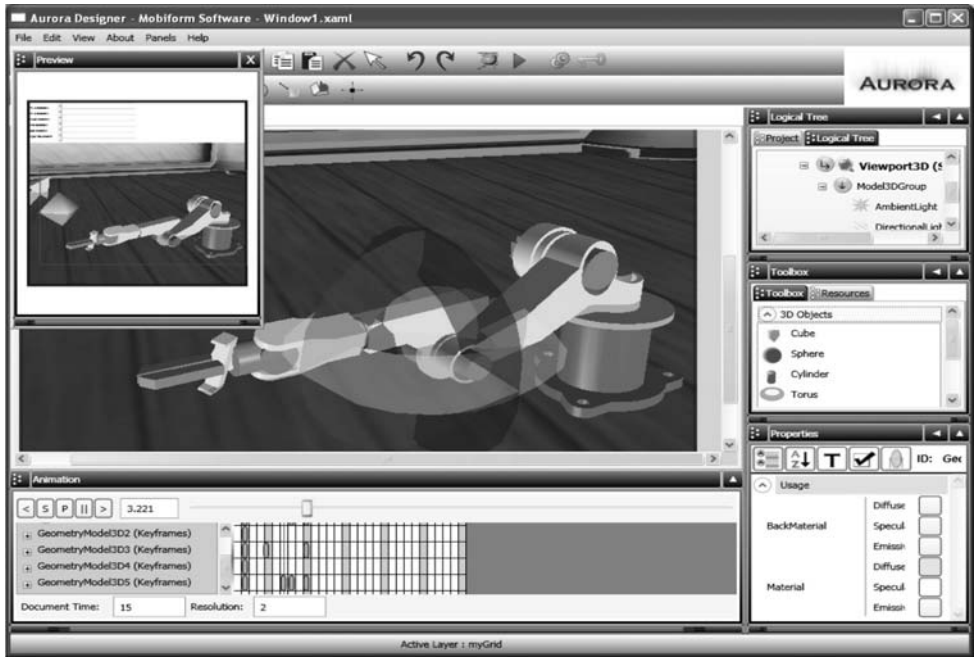
Narzędzie to umożliwia tworzenie zarówno kodu interfejsu graficznego w XAML jak również kodu do obsługi logiki w językach dostępnych w środowisku .NET. Należy jednak podkreślić, że interfejs programisty do tworzenia okienka, podobny zresztą do typowego interfejsu VS 2005 do tworzenia okienek nie jest w wielu przypadkach wystarczający. Może on dobrze służyć do tworzenia poprawek w istniejącym już okienku, jednak do stworzenia złożonego okienka od podstaw nie bardzo



się nadaje. Do tego firma Microsoft opracowała **Microsoft Expression Blend –** narzędzie dla grafików (o funkcjonalności przypominającej Flash) do przygotowania atrakcyjnego interfejsu graficznego w XAML, nadającego się bez żadnych korekt do wykorzystania w .NET 3.0.



Poniżej zaprezentowany jest wygląd głównego okna programu AURORA firmy MOBIFORM posiadający z grubsza funkcjonalność Microsoft Expression Blend.



4. PODSUMOWANIE

W artykule spróbowałem krótko scharakteryzować Windows Presentation Foundation - podsystem graficzny przewidziany do zastosowania w nowej wersji systemu operacyjnego MS Windows. Z prostych przykładów zamieszczonych w artykule widać, że WPF jest narzędziem o logice i składni bazującej na nowoczesnych i popularnych standardach i narzędziach programowania. Można z tego wysnuć wniosek, że będzie stosunkowo łatwy do opanowania i jeżeli tylko nie będzie miał ujemnego wpływu na wydajność aplikacji, stanie się z pewnością narzędziem chętnie i powszechnie stosowanym. Przyglądając się ożywionej dyskusji na temat XAML na specjalistycznych forach można stwierdzić, że o jego przydatności w tworzeniu grafiki opinie są podzielone, jednak biorąc pod uwagę ilość tworzonych przez różne firmy dość złożonych, a więc kosztownych, narzędzi do „generowania" okienek w formacie XAML należy sądzić, że ten sposób opisu obrazu przyjmie się. Tak jak wcześniej HTML, potem XML język, XAML stanie się ogólnie akceptowanym i używanym standardem wykraczającym poza środowisko MS Windows.

Literatura

[1] Zasoby MSDN, Windows Presentation Fundation
[2] Zasoby MSDN, Windows SDK for Vista
[3] Chaster Petzold Applicationa = Code + Markup: A Guide to the Microsoft ® Windows® Presentation Fundation , wyd. Microsoft 2006
[4] Artur Żarski Windows Presentation Fundation, wykład na II Spotkaniu Poznańskiej Grupy .NET (30.01.2007)