Windows Forms x WPF

Apresentação das principais características das tecnologias Windows Forms e Windows Presentation Foundation. Exposição da comparação entre as mesmas, expondo o que é semelhante e diferente, as vantagens e as desvantagens, e o objetivo, mostrar ao leitor casos em que cada tecnologia é mais bem aplicada.

[lead]**Do que trata o artigo**

Apresentação das principais características das tecnologias Windows Forms e Windows Presentation Foundation. Exposição da comparação entre as mesmas, expondo o que é semelhante e diferente, as vantagens e as desvantagens, e o objetivo, mostrar ao leitor casos em que cada tecnologia é mais bem aplicada.

Para que serve

O Windows Forms é uma tecnologia madura e possui muitas ferramentas, só que com o passar das versões do framework, este não sofreu muitas mudanças. Por exemplo, a customização de controles, por mais que se mudem as características, eles sempre possuem a aparência do padrão Windows. O WPF permite a customização total dos controles, podendo fugir do padrão do sistema operacional. Porém, para tal, requer um poder de processamento maior do computador.

Em que situação o tema é útil

Quando chegar o momento de escolher qual tecnologia utilizar, é fundamental conhecer as principais características de cada uma das plataformas para a aplicação ficar agradável tanto para o desenvolvedor como para o usuário final.

Resumo do DevMan

O Windows Forms foi a primeira solução da Microsoft para aplicações desktop. Posteriormente, no .net framework 3.0, surgiu o Windows Presentation Foundation, o WPF. Dependendo do software a ser desenvolvido, a escolha errada da tecnologia a ser aplicada pode atrapalhar sua produção. Então, é necessário conhecer os pontos fortes e fracos de cada uma das tecnologias para, assim, maximizar o desenvolvimento do software e atender as necessidades do cliente.[/lead]

O .net framework desde a primeira versão permitiu que os desenvolvedores criassem aplicações tanto web como desktop. A versão web trouxe o ASP.NET, uma versão do ASP utilizando as bibliotecas do .NET Framework. A versão desktop trouxe o Windows Forms, que lembra as aplicações desenvolvidas em IDEs como o Delphi, e também trouxe o lançamento do C#, uma nova linguagem derivada do C, totalmente orientada a objetos. Nas últimas versões do .net, especificamente da 3.0 ou superior, a Microsoft trouxe o Windows Presentation Foundation. Uma nova maneira de desenvolver aplicações, tanto para desktop, como para Web. A sua interface é baseada em uma linguagem denominada XAML, um tipo de XML, que faz com que a aplicação desenvolvida possa ser executada tanto no navegador como no Windows.

Atualmente os clientes pedem softwares de qualquer espécie, desde simples calculadoras até programas que execute complexas regras de negócio. Se o software será desktop e será desenvolvido com o .net framework, qual a melhor opção para criar a aplicação? Windows Forms ou WPF? Em seguida surgem várias dúvidas, tais como: com qual das tecnologias será suprida a necessidade do cliente e da melhor forma? Com qual das tecnologias o software será produzido mais rápido? Se for trabalhar em equipe, qual conceito permite melhor o trabalho em conjunto?

O objetivo deste artigo é destacar os pontos fortes e fracos do Windows Forms e do WPF, apresentando um caso onde a aplicação é favorável à utilização do primeiro e outro caso onde a aplicação será mais bem elaborada com a utilização do segundo.

[subtitulo]Windows Forms[/subtitulo]

O Windows Forms existe no .net framework desde a primeira versão. Para a versão desktop, foi a única solução até a versão 3.0, quando surgiu o WPF. Com o objetivo de reproduzir aplicações Windows, o Windows Forms lembra aplicações desenvolvidas em IDEs como Delphi, que também utilizam APIs do sistema operacional.

Uma API (Application Programming Interface, ou em português, Interface de Programação de Aplicativos) é um conjunto de código que está disponível no sistema operacional para realizar as mais diversas funções. Dentre elas criar uma aparência visual dos elementos de interface do usuário, tais como botões, caixas de texto, caixas de seleção e assim por diante. Como resultado, esses componentes são essencialmente não customizáveis, muito pelo contrário, são bem parecidos com o do sistema operacional em questão.

Na questão da customização, os controles que são os componentes de entrada e saída utilizados pelo Windows Forms, possuem características pouco customizáveis, permitem apenas cor, tamanho de fonte, cor de fonte, estilo da fonte entre outras. Por exemplo, se o desenvolvedor precisar deixar um botão com o formato diferente vai precisar fazer ou procurar uma imagem do botão que quiser, e aplicar ao botão na propriedade Image.

Não é uma opção muito viável, já que imagens consomem bastante memória para renderizar. A **Figura 1** ilustra um formulário com um campo desenvolvido em Windows Forms.

Figura 1. Ilustração de um formulário em Windows Forms

Todo formulário quando criado pelo Visual Studio possui três arquivos: um com a extensão .cs, outro com a extensão .Designer.cs e ainda outro com a extensão .resx. O primeiro é a classe que define o comportamento do formulário, ou seja, nesta classe são programados os eventos do próprio formulário e dos controles contidos nele. O segundo, com a extensão .Designer, é a classe que contém os componentes do formulário: os botões, as caixas de textos, os labels entre outros. Além de definir as instâncias dos controles, armazena o valor das propriedades atribuídas em modo design, por exemplo, o texto que estará no label, a posição absoluta na tela, o tamanho, entre outras. Possui também métodos de criação e remoção do formulário na memória, como os métodos InitializeComponent e Dispose, respectivamente. O primeiro e o segundo arquivo contêm classes partial, ou seja, são dois arquivos que contêm o mesmo nome de classe, e eles se completam se transformando em uma classe só quando compiladas. O terceiro arquivo são os resources do formulário, ou seja, um arquivo de recurso que contém informações que podem ser utilizadas por ele, como strings, imagens, ícones, entre outras opções. A **Listagem 1** mostra o código gerado pelo próprio framework para criar o formulário, ou seja, a classe com extensão .Designer.

Listagem 1. Código gerado para criação do formulário da **Figura 1**

```
namespace Listagem1
  {
      partial class Form1
      {
          /// <summary>
          /// Required designer variable.
          /// </summary>
          private System.ComponentModel.IContainer components = null;

          /// <summary>
          /// Clean up any resources being used.
          /// </summary>
          /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
          protected override void Dispose(bool disposing)
          {
              if (disposing && (components != null))
              {
                  components.Dispose();
              }
              base.Dispose(disposing);
          }

          #region Windows Form Designer generated code

          /// <summary>
          /// Required method for Designer support - do not modify
          /// the contents of this method with the code editor.
          /// </summary>
          private void InitializeComponent()
          {
              this.label1 = new System.Windows.Forms.Label();
              this.textBox1 = new System.Windows.Forms.TextBox();
              this.SuspendLayout();
              //
              // label1
              //
              this.label1.AutoSize = true;
              this.label1.Location = new System.Drawing.Point(13, 13);
              this.label1.Name = "label1";
              this.label1.Size = new System.Drawing.Size(84, 13);
              this.label1.TabIndex = 0;
              this.label1.Text = "Nome do cliente";
              //
              // textBox1
              //
              this.textBox1.Location = new System.Drawing.Point(16, 30);
              this.textBox1.Name = "textBox1";
```

```
            this.textBox1.Size = new System.Drawing.Size(256, 20);
            this.textBox1.TabIndex = 1;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 262);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBox1;
    }
}
```

Para mudar as propriedades dos controles é possível selecioná-los e pressionar a tecla F4 para que a guia Properties apareça, como indica a **Figura 2**.

Figura 2. Aba Properties mostrando as propriedades da caixa de texto

É bastante simples montar um formulário com Windows Forms: basta passar o mouse sobre a aba Toolbox e clicar sobre o controle desejado e arrastar para o formulário, como mostra a **Figura 3**.

Figura 3. Inserindo um controle no formulário em Windows Forms

O principal namespace das aplicações Windows Forms é o System.Windows.Forms. Dentro deste namespace estão as classes que permitem a criação de interfaces de usuário.

[subtitulo]Windows Presentation Foundation[/subtitulo]

Segundo o autor Moroney, o Windows Presentation Foundation é um subsistema de exibição gráfica para Windows. O WPF combina o melhor do antigo mundo de desenvolvimento Windows com inovações para construções modernas e interfaces de usuário graficamente ricas.

Sua origem se dá na versão 3.0 do .NET framework 3.0 e constantemente vem sofrendo modificações e novos complementos com novidades para os desenvolvedores. Os recursos estão ficando cada vez mais inovadores. Subjacente a novas funcionalidades, o WPF é uma nova e poderosa infraestrutura baseada em DirectX. O DirectX é uma API multimídia que oferece uma interface padrão para interagir com elementos gráficos, placas de som e dispositivos de entrada, entre outros.

Isto significa que é possível criar efeitos gráficos ricos sem sobrecarregar o desempenho, problema corriqueiro com a utilização de Windows Forms. Na verdade, obtêm-se recursos avançados como suporte para arquivos de vídeo e conteúdo 3D. Usando estes recursos é possível criar interfaces de usuário estilizadas e efeitos visuais que não seriam possíveis utilizando Windows Forms.

O WPF inclui controles padrão para se familiarizar com os desenhos de texto, bordas e preenchimento de fundo. Como resultado, a tecnologia pode proporcionar características mais poderosas que permitem alterar o modo como o conteúdo da tela é exibido. É possível reescrever controles comuns, como botões ou caixas de texto, muitas vezes sem escrever qualquer código, e obter um controle bem mais atraente e que chame a atenção do usuário.

Além disso, é permitido usar transformações de objetos para girar, esticar e deformar alguma coisa na interface de usuário. E como o núcleo WPF renderiza o conteúdo de uma janela como parte de uma única operação, pode-se manipular ilimitadas camadas de sobreposição de controles, mesmo que estes sejam irregularmente desenhados e parcialmente transparentes. A **Figura 4** mostra um simples formulário feito em WPF.

Figura 4. Formulário simples desenvolvido em WPF

Aparentemente o formulário desenvolvido em WPF é pior na questão visual do que o formulário feito em Windows Forms. O que acontece é que a Window, ou seja, o formulário WPF, quando criado, vem com uma aparência padrão não muito atraente. Porém, com leves modificações, como por

exemplo um gradiente no fundo, cor para a fonte do Label e bordas arredondadas para a caixa de texto, pode se obter um formulário muito melhor, como mostra a **Figura 5**.

Figura 5. Window com a aparência padrão modificada

Os formulários em WPF são desenhados a partir de um XAML, abreviação para Extensible Application Markup Language e pronunciado como "zammel". Embora o XAML seja uma tecnologia que pode ser aplicada a muitos problemas de domínios diferentes, o seu papel principal é construir interfaces de usuário com WPF. A **Listagem 2** mostra o documento XAML do formulário da forma simples, sem estilização.

Listagem 2. Window sem estilização

```
<Window x:Class="Listagem2.Window1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Window1" Height="300" Width="300">
      <StackPanel Margin="6" Orientation="Vertical">
          <Label Content="Nome do cliente" />
          <TextBox Width="250" Height="20" HorizontalAlignment="Left" />
      </StackPanel>
  </Window>
```

Em outras palavras, documentos XAML podem definir a disposição dos painéis, botões e controles que compõem as janelas em uma aplicação WPF, como também podem definir estilos para os mesmos. Estes estilos são semelhantes ao CSS (Cascade Style Sheet) da web.

É muito útil a utilização do recurso de estilização, pois ao invés de mudar cada propriedade dos controles para aplicar alguma característica, é somente necessário mudar o estilo definido para o controle e ele automaticamente se aplicará.

A **Listagem 3** mostra a Window simples, porém, com aplicação de estilos nela mesma e nos controles contidos nela. É importante destacar que uma boa prática na utilização dos estilos é mantê-los no nível mais alto possível, ou seja, nunca dentro do próprio controle, mas sim na Window ou ainda se possível, em um arquivo denominado Resource Dictionary, que é também um arquivo XAML e trabalha arquivo CSS da web, centralizando os estilos dos controles em um único lugar.

Listagem 3. Window e seus controles com estilização

```
<Window x:Class="Listagem2.Window1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Window1" Height="300" Width="300">
      <!--Fundo da janela-->
      <Window.Background>
          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
              <GradientStop Color="#FFEDF0F9" Offset="0.5"/>
              <GradientStop Color="White" Offset="1"/>
          </LinearGradientBrush>
      </Window.Background>
      <!--Local onde são definidos os estilos-->
      <Window.Resources>
          <!--Estilo para o controle TextBox-->
          <Style TargetType="{x:Type TextBox}">
              <Setter Property="KeyboardNavigation.TabNavigation" Value="None"/>
              <Setter Property="FocusVisualStyle" Value="{x:Null}"/>
              <Setter Property="Validation.ErrorTemplate" Value="{x:Null}"/>
              <Setter Property="Margin" Value="3,0,5,0"/>
              <Setter Property="AllowDrop" Value="true"/>
              <Setter Property="CharacterCasing" Value="Upper"/>
              <Setter Property="HorizontalAlignment" Value="Stretch"/>
              <Setter Property="Height" Value="21"/>
              <Setter Property="Foreground" Value="#FF001362"/>
              <Setter Property="UndoLimit" Value="1"/>
              <Setter Property="FontFamily" Value="Tahoma"/>
              <Setter Property="Template">
                  <Setter.Value>
                      <ControlTemplate TargetType="{x:Type TextBox}">
                          <Grid>
                              <Border x:Name="Border" Opacity="1" Background="{TemplateBinding Background}" BorderBrush="{TemplateBinding BorderBrush]
                                  <Grid>
                                      <Border BorderThickness="1" CornerRadius="1,1,1,1">
```

```
                                    <Border.BorderBrush>
                                        <SolidColorBrush Color="Transparent" x:Name="MouseOverColor"/>
                                    </Border.BorderBrush>
                                    <ScrollViewer Margin="0" x:Name="PART_ContentHost" Background="{TemplateBinding Background}"/>
                                </Border>
                            </Grid>
                        </Border>
                        <Border x:Name="HoverBorder" Opacity="0" BorderBrush="#FF749CD3" BorderThickness="2,2,2,2" CornerRadius="2,2,2,2"/>
                        <Border x:Name="DisabledVisualElement" IsHitTestVisible="False" Opacity="0" Background="#FFFFFFFF" BorderBrush="#A5F7F
                        <Border x:Name="FocusVisualElement" IsHitTestVisible="False" Opacity="0" BorderBrush="#FF749CD3" BorderThickness="2.1,2
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
        <Setter Property="BorderBrush" Value="#FF749CD3"/>
        <Setter Property="Background" Value="White" />
    </Style>
    <!--Estilo para o controle Label-->
    <Style TargetType="Label">
        <Setter Property="Foreground" Value="#FF001362" />
        <Setter Property="FontFamily" Value="Verdana" />
    </Style>
</Window.Resources>

<!--Conteúdo da janela-->
<StackPanel Margin="6" Orientation="Vertical">
    <Label Content="Nome do cliente" />
    <TextBox Width="250" Height="20" HorizontalAlignment="Left" />
</StackPanel>
</Window>
```

É pouco provável que se escreva XAML na mão. Ao invés disso, pode-se usar uma ferramenta que gere o XAML. Para um designer gráfico, esta ferramenta pode ser o programa Microsoft Expression Blend, mas para um desenvolvedor, pode-se usar o próprio Visual Studio. Ambas as ferramentas trabalham tranquilamente com o XAML, podendo-se criar uma interface de usuário básica com o Visual Studio e então dar acabamentos com gráficos personalizados no Expression Blend.

Na verdade, esta capacidade de integrar o fluxo de trabalho entre desenvolvedores e designers é uma das principais razões que fez a Microsoft criar o XAML. Tanto o Visual Studio como no Expression Blend, a construção do formulário pode ser do modo “arrastar e soltar”, como no Windows Forms.

Quando uma Window é criada pelo Visual Studio, são gerados dois arquivos: um com a extensão xaml e outro com a extensão .xaml.cs. O primeiro é o conteúdo visual da interface, sendo somente texto XAML. Já o segundo, é o arquivo que controla o comportamento dos controles da tela, semelhante à classe .cs do Windows Forms.

Para se alterar as propriedades dos controles numa aplicação WPF e adicionar controles aos formulários, é o mesmo caminho do Windows Forms. O namespace principal do WPF é o System.Windows. Além dos controles contidos dentro deste namespace, contém classes que auxiliam nas animações e efeitos.

O WPF não é bom somente visual, melhora também a manipulação dos dados, com o avançado Data Binding. Data Binding é o caminho que conecta a interface da aplicação aos dados do sistema, com a atualização automática do valor da sua fonte de dados. Por exemplo, supondo que um TextBox esteja ligado com a propriedade de alguma classe, e, se o usuário mudar o valor no controle, automaticamente o valor da propriedade na classe será alterado, sem necessitar de ficar preenchendo a toda interação.

No Windows Forms também existe o Data Binding, porém, não é tão eficaz e de tão fácil uso como no WPF. Principalmente na questão dos tipos dos dados é bem mais complicado de se tratar no Windows Forms do que no WPF. A **Figura 6** ilustra como funciona a conexão do Data Binding entre o controle de interface (objeto de dependência e propriedade de dependência, por exemplo, controle TextBox e a propriedade Text) e a sua fonte (objeto comum e uma propriedade da classe, por exemplo, classe Pessoa propriedade Nome). Mostra também os tipos de ligação que o WPF disponibiliza, que seria OneWay, ou seja, do objeto de negócio para o controle na interface, TwoWay, que permite a ligação tanto do controle da interface para objeto de negócio e vice-versa, e OneWayToSource, que a direção dos dados se dá entre o controle da interface para o objeto de negócio.

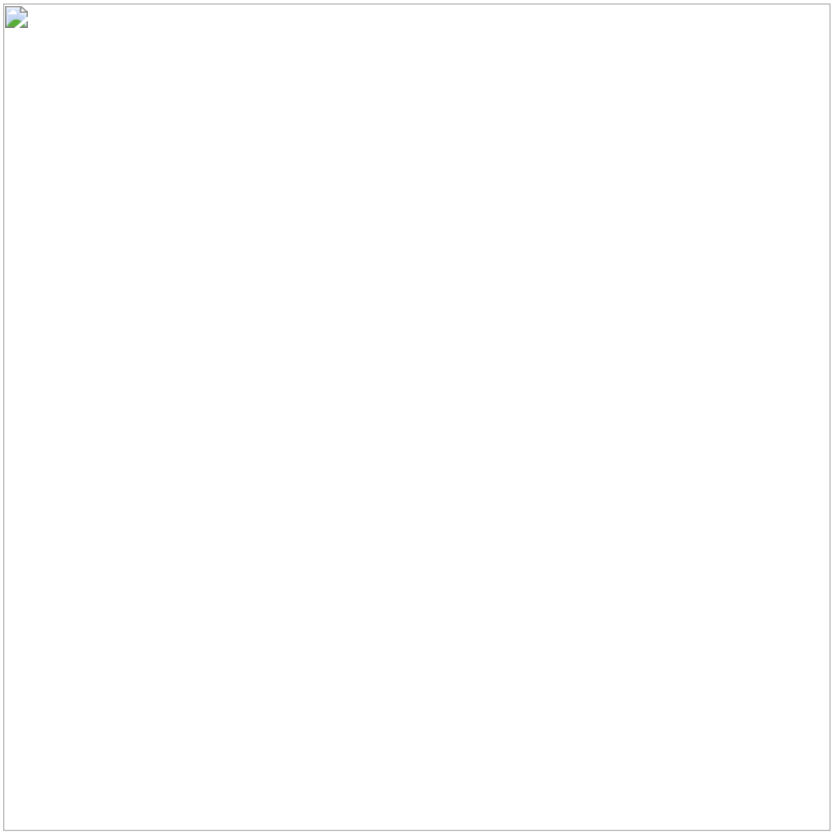

Figura 6. Ilustração do Data Binding no WPF

[subtitulo]Comparação entre WPF e Windows Forms[/subtitulo]

Ambas as tecnologias em alguns aspectos levam vantagem uma sobre a outra, assim como também levam desvantagens. Enquanto o WPF fornece uma aplicação mais bonita visivelmente e, em algumas situações, permite que o fluxo de dados trafegue de maneira fácil para o desenvolvedor, o Windows Forms permite que uma aplicação seja criada rapidamente, talvez não muito chamativa no critério visual, mas de desempenho maior em algumas situações e que exige um computador mais simples que uma aplicação WPF.

Porém, nem tudo é feito de vantagens. O WPF exige um computador mais robusto para a aplicação rodar mais tranquila, principalmente quando os efeitos visuais são pesados, e, como utiliza recursos do hardware, como a placa de vídeo, é necessário um hardware melhor para a aplicação corresponder em termos de desempenho. Não que uma aplicação não rodará em computadores de placa de vídeo onboard, mas, seu desempenho não será o mesmo do que instalado em um computador com hardware melhor.

Já o Windows Forms, apesar de não necessitar de um computador tão robusto, não oferece grandes recursos visuais e de manipulação de dados atraentes para o usuário, como o WPF. Então, uma aplicação Windows Forms é mais trabalhosa em termos de negócio do que a aplicação WPF. Porém, a aplicação WPF é bem mais trabalhosa na questão visual do que a aplicação Windows Forms e isso implica no tempo de desenvolvimento da aplicação, tornando a construção do software um pouco mais demorada. A **Tabela 1** mostra as mesmas características para as tecnologias e indicando se é bem aproveitada ou não por cada uma delas.

Característica	WPF	Windows Forms
Necessita de um hardware mais atual/robusto	Sim	Não
Visual sempre com características do sistema operacional	Não	Sim
Facilidade de manipulação dos dados	Sim	Não
Criação fácil e rápida de formulários	Não	Sim
Desenvolvimento rápido	Não	Sim

Tabela 1. Comparativo de recursos

[subtitulo]Comparação entre WPF e Windows Forms em uma aplicação prática[/subtitulo]

Somente comparações conceituais não são suficientes para deixar claras as diferenças entre as tecnologias desktop da Microsoft. A ideia é mostrar um formulário feito em Windows Forms e o mesmo formulário desenvolvido em WPF. O contexto é um cadastro de clientes simples, com ligações nos campos por Data Binding. O tempo de construção de cada formulário será levado em conta também.

A **Figura 7** ilustra a estrutura do formulário desenvolvido. A **Figura 8** ilustra o diagrama de classes da aplicação, no caso, somente a classe cliente. A **Listagem 4** mostra o código da classe cliente, com as propriedades criadas.

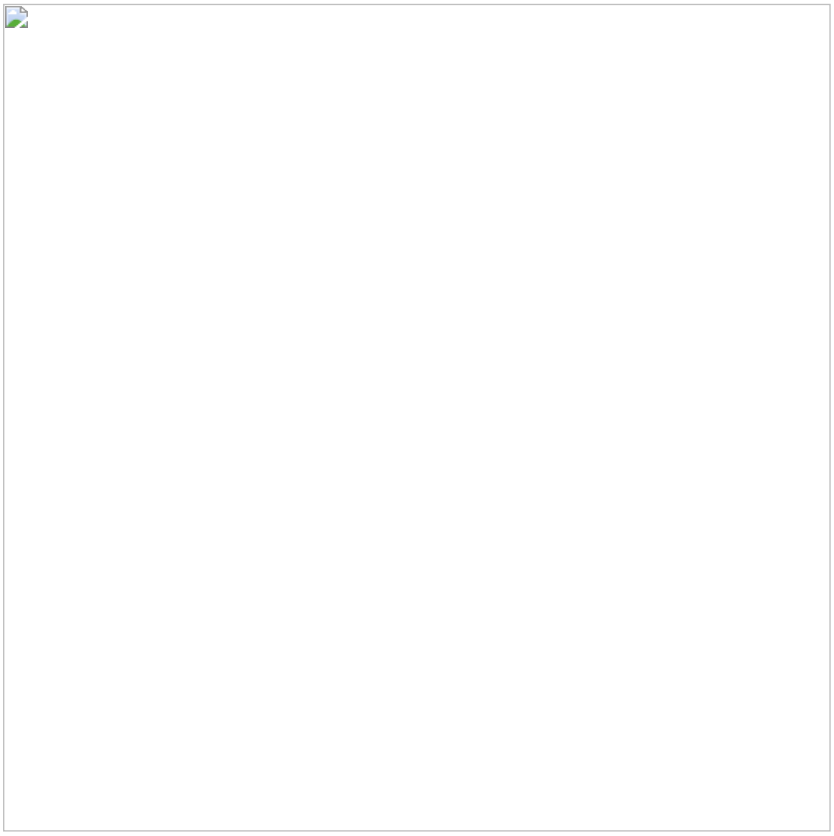

Figura 7. Estrutura do formulário a ser construído em Windows Forms e WPF

Figura 8. Diagrama de classes da aplicação

Listagem 4. Classe cliente e suas propriedades

```
using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Text;

  namespace CadastroCliente.Modelo
  {
      public class Cliente
      {
          public string Nome { get; set; }
          public string CPF { get; set; }
          public DateTime DataNascimento { get; set; }
          public string Endereco { get; set; }
          public int Numero { get; set; }
          public string Complemento { get; set; }
          public string Cidade { get; set; }
          public string Bairro { get; set; }
      }
  }
```

O Data Binding por padrão notifica a alteração do objeto após o foco sair do controle. Por exemplo, a propriedade nome do cliente só receberá o valor do TextBox que representa o nome do cliente somente quando este perder o foco. Para um melhor controle, é interessante o objeto ser preenchido conforme o conteúdo do controle for alterado, ou seja, conforme o usuário digite o nome do cliente, o objeto já vai sendo preenchido.

Para isto é necessário implementar a interface INotifyPropertyChanged do .Net Framework na classe cliente e, em cada set das propriedades, chamar o evento implementado pela interface. A **Listagem 5** mostra a classe cliente modificada, com a definição de propriedades no método antigo, ou seja, sem ser autoproperty, e com o evento da interface e o método que notifica a mudança de valor da propriedade.

Listagem 5. Classe cliente implementando o recurso de notificação de alteração de valor das propriedades

```
using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Text;
  using System.ComponentModel;

  namespace CadastroCliente.Modelo
  {
      public class Cliente : INotifyPropertyChanged
      {
          private string _nome;
          private string _cpf;
          private DateTime _dataNascimento;
          private string _endereco;
          private int _numero;
          private string _complemento;
          private string _bairro;
          private string _cidade;

          public string Nome
          {
              get { return _nome; }
              set
              {
                  _nome = value;
                  OnPropertyChanged("Nome");
              }
          }

          public string CPF
          {
              get { return _cpf; }
              set
              {
                  _cpf = value;
                  OnPropertyChanged("CPF");
              }
          }

          public DateTime DataNascimento
          {
              get { return _dataNascimento; }
              set
              {
                  _dataNascimento = value;
                  OnPropertyChanged("DataNascimento");
              }
          }

          public string Endereco
          {
              get { return _endereco; }
              set
              {
                  _endereco = value;
                  OnPropertyChanged("Endereco");
              }
          }

          public int Numero
          {
              get { return _numero; }
              set
              {
                  _numero = value;
```

```
                OnPropertyChanged("Numero");
            }
        }

        public string Complemento
        {
            get { return _complemento; }
            set
            {
                _complemento = value;
                OnPropertyChanged("Complemento");
            }
        }

        public string Cidade
        {
            get { return _cidade; }
            set
            {
                _cidade = value;
                OnPropertyChanged("Cidade");
            }
        }

        public string Bairro
        {
            get { return _bairro; }
            set
            {
                _bairro = value;
                OnPropertyChanged("Bairro");
            }
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

        // Método que notifica a alteração da propriedade pelo evento
        protected void OnPropertyChanged(string name)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null)
            {
                handler(this, new PropertyChangedEventArgs(name));
            }
        }
        #endregion
    }
}
```

[subtitulo]Construção da interface do formulário[/subtitulo]

Depois de definida a estrutura do objeto que representará o conceito a ser tratado, no caso, o cliente, é só desenhar o formulário em cada uma das tecnologias. Lembrando que no Windows Forms o Data Binding é realizado no code-behind do formulário. Já no WPF, a ligação é definida no próprio XAML do controle. A **Figura 9** ilustra o formulário em Windows Forms e a **Figura 10** mostra o formulário criado em WPF, ambos não estilizados, ou seja, sem aplicação de efeitos visuais.

Figura 9. Formulário de cadastro de clientes em Windows Forms

Figura 10. Window representando o cadastro de clientes desenvolvida em WPF

Como o formulário é bem pequeno, a construção foi bem rápida, sendo aproximadamente 6 minutos para Windows Forms e 10 minutos aproximadamente para o WPF. Uma estilização das aplicações será feita, somente para tirar o padrão do sistema operacional dos controles, porém,

utilizando os recursos de cada tecnologia.

No Windows Forms é difícil a estilização pois cada propriedade a ser modificada tem que ser atribuída para cada controle, a não ser que vários controles do mesmo tipo sejam selecionados ou os efeitos sejam atribuídos via código. Outro caminho é a criação de um Custom Control, que é uma classe que herda de um controle especifico e lá o desenvolvedor trata a parte de estilização via código. Para projetos grandes, esta seria a melhor solução, mas como não é o caso atual, a customização será feita nos controles. A **Figura 11** mostra o formulário depois de aplicadas cores e fontes aos controles. O tempo médio levado foi de 15 minutos para realizar a tarefa.

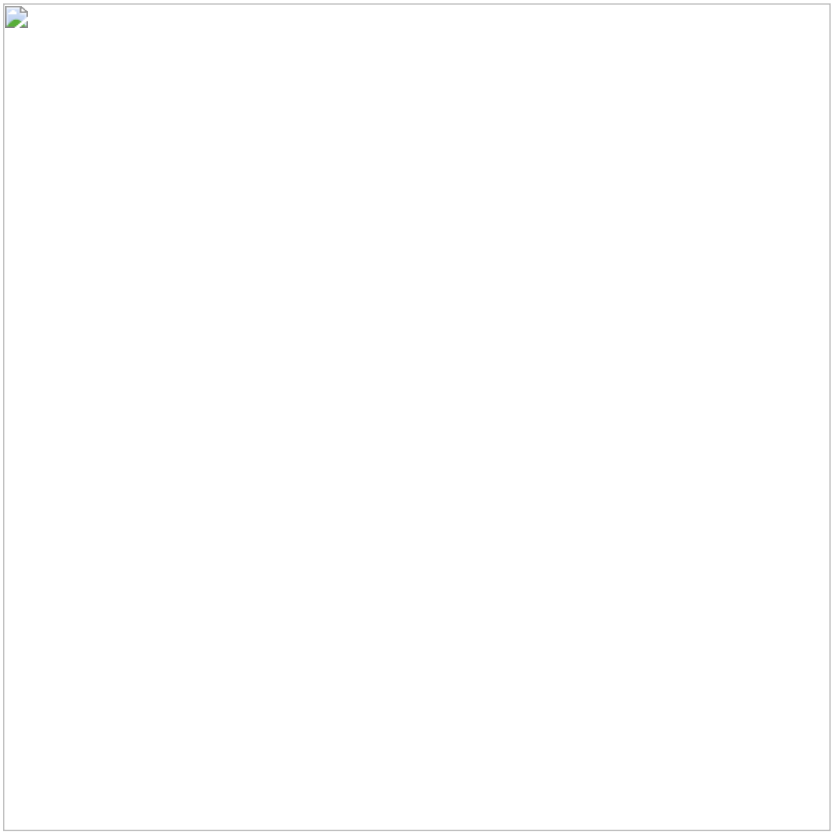

Figura 11. Formulário de cadastro de clientes com customizações na interface em Windows Forms

No WPF dependendo do efeito a ser aplicado, é difícil a estilização também. Porém, para pequenos efeitos, é muito útil, principalmente pela parte que os estilos podem ser atribuídos por tipo de controle e não necessita ir de controle a controle atribuindo cada propriedade para modificar o estilo do mesmo.

A **Figura 12** apresenta a Window que representa o formulário de clientes desenvolvido em WPF estilizada. O tempo aproximado utilizado foi de 40 minutos. Os detalhes são bem mais reais em relação ao Windows Forms, como por exemplo, a borda arredondada, utilização de transparência e gradientes. Porém, o custo disso foi uma maior demora na construção da estilização.

Figura 12. Window do cadastro de clientes estilizada, em WPF

[subtitulo]Ligação e manipulação de informações[/subtitulo]

Com a interface pronta, a última parte da comparação é a ligação dos dados via Data Binding em ambas as tecnologias. No Windows Forms a ligação é feita no code-behind do formulário, em qualquer evento, mas no exemplo, será implementado no Load, e a declaração é semelhante ao código a seguir.

```
controle.DataBindings.Add(new Binding("PropriedadeDoControle", objetoFonte,
  "Nome da propriedade no objeto", true, DataSourceUpdateMode.OnPropertyChanged);
```

A ligação é feita no controle a partir da propriedade DataBindings, que é uma coleção da classe Binding. O método que adiciona o binding nesta coleção possui vários overloads, o mais apropriado é o apresentado, pois é o que permite alterar o modo de notificação.

Na sobrecarga utilizada, os parâmetros na sequência são: a propriedade que vai ser utilizada no controle, o objeto que será a fonte da informação, a propriedade do objeto fonte que será preenchida/recuperada, permissão de formatação de dados e o enumerador que representa o tipo de notificação de alteração.

Já no WPF, a ligação dos controles com a fonte de dados pode ser tanto no XAML como no code-behind. No exemplo, será utilizado o primeiro método, ou seja, definição no próprio XAML. A declaração segue no código a seguir.

```
<TextBox Text="{Binding Path=NomeDaPropriedadeNoObjeto, UpdateSourceTrigger=PropertyChanged}" />
```

A definição da ligação é realizada no atributo da tag do controle desejado. Semelhante ao Windows Forms, é informado o nome da propriedade do objeto que será fonte de dados e o modo da notificação de atualização da informação.

Porém no WPF a definição do objeto fonte de dados é a partir da propriedade Data Context. Essa propriedade existe na maioria dos componentes, mas comumente, é atribuída na tag Window. O DataContext não é o objeto fonte de dados em si, mas sim um objeto que contém este objeto fonte de dados, inclusive, é necessário especificar o nome da instância do objeto. Por exemplo, na classe FonteDados tem uma propriedade ObjetoFonte, do tipo cliente. Na definição do binding, será necessário colocar ObjetoFonte.Nome, caso, por exemplo, deseje-se pegar o nome do cliente.

Um dos atributos do Binding no WPF, é o ElementName. Este atributo aponta como objeto fonte outro elemento definido no XAML. Por exemplo, o TextBox nome só ficará habilitado caso o CheckBox estiver marcado. Isto também é possível no Windows Forms, é só colocar como objeto fonte o controle desejado. A **Listagem 6** mostra a definição do DataContext, em sublinhado, na Window da aplicação. Nesta situação, é apontado que o DataContext da Window é ela mesma. É necessário dar um nome para os controles, na necessidade de utilizar o atributo ElementName.

Listagem 6. Ligação do DataBinding no WPF

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
      ...
      x:Name="CadastroCliente"
      DataContext="{Binding ElementName=CadastroCliente,
      UpdateSourceTrigger=PropertyChanged}">
```

Tanto no Windows Forms como no WPF, o objeto que será a fonte das informações necessitará notificar a alteração de valores da propriedade. Então, a classe onde estará este objeto deverá implementar a interface INotifyPropertyChanged, igual à classe Cliente. No exemplo, o objeto fonte será do tipo Cliente, então, a **Listagem 7** mostra a definição da propriedade ObjetoFonte.

Listagem 7. Propriedade que servirá de fonte de dados para o DataBinding

```
private Cliente _objetoFonte;

  public Cliente ObjetoFonte
  {
          get { return _ objetoFonte; }
          set
          {
                  _ objetoFonte = value;
                  OnPropertyChanged("ObjetoFonte");
          }
  }
```

Com estes passos realizados, é só codificar a ligação nos controles com as propriedades do objeto fonte e o Data Binding entra em ação. A **Listagem 8** mostra as ligações dos TextBoxs nome e cidade no Windows Forms. A **Listagem 9** mostra o XAML da definição das mesmas propriedades de cliente para o WPF.

Listagem 8. Definição do DataBinding no Windows Forms

```
txtNome.DataBindings.Add(new Binding(
           "Text", ObjetoFonte, "Nome", true,
           DataSourceUpdateMode.OnPropertyChanged));
  txtCidade.DataBindings.Add(new Binding(
           "Text", ObjetoFonte, "Cidade", true,
            DataSourceUpdateMode.OnPropertyChanged));
```

Listagem 9. Propriedade cliente que servirá de fonte de dados para o DataBinding

```
<TextBox Text="{Binding Path=ObjetoFonte.Nome, UpdateSourceTrigger=PropertyChanged}" />
  ...
  <TextBox Text="{Binding Path=ObjetoFonte.Cidade, UpdateSourceTrigger=PropertyChanged}" />
```

O uso do Data Binding independente da tecnologia utilizada, é muito útil pois evita ficar preenchendo objetos e controles manualmente. O tempo de desenvolvimento cai significativamente e o código fica bem mais enxuto. Tanto no Windows Forms como no WPF, a definição do Data Binding é bem simples.

O que diferencia uma tecnologia da outra é o tratamento quanto aos tipos de objeto. No WPF, por exemplo, o texto da propriedade foi digitado errado, ele não gera nenhum erro, simplesmente não realiza a ligação. Já no Windows Forms ele gera o erro. Estes erros poderão ser tratados em ambas as tecnologias, porém, não é este o objetivo por agora.

[subtitulo]Análise de casos[/subtitulo]

É importante levar em conta também que o tipo do software a ser produzido é essencial na escolha da tecnologia a ser utilizada, para assim se obter uma melhor vantagem na construção do mesmo. Então, para deixar bem claro ao leitor, serão apresentados alguns casos onde é possível aplicar as tecnologias, porém, analisando caso a caso qual delas é a melhor para determinada situação.

Caso 1: Necessidade de alto processamento

O objetivo do projeto é criar um sistema simples de prazo curto, sem se importar muito com o visual para o usuário e precisa ter um bom desempenho, pois o sistema será utilizado para converter uma base de dados do banco Firebird para o SQL Server. Isto significa que o software terá de ser simples e funcional. Logo, não é necessário fazer grandes interfaces, com animações, efeitos e brilhos. Então neste caso antes de escolher a tecnologia a ser utilizada, é necessário por na balança alguns itens para análise.

Se a aplicação precisa de um alto desempenho, pode ser utilizado WPF, já que este utiliza recursos de hardware multimídia, como da placa de vídeo, para processar as interfaces com o usuário, liberando um pouco o trabalho do processador. Porém, o software pode demorar um pouco mais para ser desenvolvido, pois é mais trabalhosa a construção de formulários com o WPF, ainda mais se as telas forem grandes.

Por outro lado, a utilização de Windows Forms não pode ser descartada já que é bem simples a construção de formulários e diminui significativamente a construção do software. A desvantagem desta tecnologia é que somente o processador irá trabalhar na execução, já que o Windows Forms não utiliza outros hardwares para auxiliá-lo no processamento do sistema.

Para este caso, a sugestão melhor seria utilizar o Windows Forms, pois, neste tipo de software, a maioria das vezes, o computador fica dedicado somente para a execução do sistema, mantendo um bom desempenho da aplicação, e não necessitando de grandes efeitos na interface com o usuário.

[subtitulo]Caso 2: Exibição de dados complexos[/subtitulo]

O objetivo do projeto é apresentar os dados que o cliente deseja de forma agrupada, semelhante a um relatório, porém, como o cliente não possui e não vai adquirir a licença de um framework de relatórios como o Crystal Reports, a apresentação dos dados será desenvolvida em formulários mesmo.

O cliente deseja também que certos dados de alguns conceitos, como por exemplo, cliente e fornecedor, sejam apresentados em cartões, pois como contém muitas informações de um mesmo registro, como nome, CPF/CNPJ, endereço, etc., os dados em modo cartão facilitam a visualização.

Nesta aplicação, a interface é importante para o objetivo do desenvolvimento, já que o cliente solicita uma maneira não muito comum para a visualização dos dados. Então, quando se trata de efeitos visuais, a melhor opção é o WPF em relação ao Windows Forms.

Para desenvolver o que o cliente precisa no Windows Forms demandaria muito mais tempo que o WPF. O WPF oferece um recurso chamado Data Template, que é uma forma de modelar a apresentação dos dados a partir de objetos. A **Listagem 10** mostra um exemplo de Data Template, concatenando propriedades de um único objeto de uma maneira simples, versátil e de fácil manutenção de dados de endereço, como logradouro, número e bairro.

Listagem 10. Exemplo de Data Template modelando dados de endereço

```
<DataTemplate x:Key="DataTemplateEndereco">
          <StackPanel Orientation="Horizontal">
          <TextBlock Text="{Binding Path=Logradouro}" />
          <TextBlock Text=", " />
          <TextBlock Text="{Binding Path=Numero}" />
          <TextBlock Text=" - "/>
          <TextBlock Text="{Binding Path=Bairro}" />
      </StackPanel>
  </DataTemplate>
```

Com o DataTemplate do WPF, modelar a apresentação de dados fica bem mais fácil, eficaz e apresentável que no Windows Forms. Porém, não é só na modelagem dos dados que o WPF permite uma melhor apresentação dos dados. Por ser estruturado em XAML, ele permite uma hierarquia de controles mais abrangente que o Windows Forms. Isso significa que, por exemplo, é possivel colocar um TextBox dentro de um Botão, ou o contrário. Essa hierarquia permite um agrupamento de controles que revoluciona a apresentação dos dados dentro dos controles. É possível fazer um DataGrid agrupar e filtrar os registros já buscados no banco de dados sem precisar realizar outra consulta para isto, mostrar gráficos, mais dados agrupados, entre outras informações.

As **Figuras 13**, **14**, **15** e **16** ilustram o grande poder de manipular os dados dos DataGrids do WPF, como detalhamento, inserção de objetos como gráficos, agrupamento de registros com os mesmos que já estão no Grid e filtragem de dados, respectivamente.

Figura 13. DataGrid com detalhamento de registros. Fonte: www.telerik.com

Figura 14. DataGrid com gráfico no detalhamento do registro. Fonte: www.telerik.com

Figura 15. Agrupamento dos registros que foram consultados, sem necessitar de um novo acesso a fonte de dados. Fonte: www.telerik.com

Figura 16. Filtragem dos registros, realizado no próprio DataGrid. Fonte: www.telerik.com

[nota]**Nota**

O DataGrid nativo do .net não possui um grande leque de recursos para o desenvolvimento. Ele não é pobre, mas também não é o melhor. Por isso, grandes empresas especializadas em desenvolvimento de controles, como a Telerik e a Xceed, produzem um framework de controles e comercializam, muitas vezes por preços medianos, não tão caros, mas também, não muito baratos.

Em algumas situações é bem melhor adquirir estes produtos, que facilitam o desenvolvimento do software, do que desenvolver um desde o início, com a incerteza se ele atenderá todas as necessidades ou não.[/nota]

[subtitulo]Caso 3: Aplicação simples, extensa e será executada em hardware defasado[/subtitulo]

O objetivo do projeto é fazer um sistema simples, como fluxo de caixa, cadastro de clientes, fornecedores, produtos e relatórios. Este sistema será utilizado em várias máquinas e, muitas delas, já possuem muito tempo de uso e não foi realizado upgrade de hardware.

Esta última informação é chave para a seleção da tecnologia a ser utilizada. Se a escolha for WPF, um dia após o uso, o cliente ligará para a empresa que desenvolveu o software reclamando da lentidão do sistema. O WPF é uma excelente tecnologia, porém, como toda tecnologia, tem vantagens e desvantagens, e sua principal desvantagem é ter um desempenho inferior na renderização das telas em máquinas com hardware defasado. Para a aplicação executar sem problemas, é necessário um hardware melhor.

Então como as configurações de hardware não se enquadram na tecnologia mais nova, a opção é utilizar Windows Forms, com pouco recurso visual, porém, bem eficiente neste tipo de situação, onde o hardware é um problema. O sistema poderá satisfazer a necessidade do cliente facilmente, mesmo desenvolvido com o Windows Forms.

[subtitulo]Conclusão[/subtitulo]

O objetivo deste artigo foi ajudar o leitor a identificar algumas situações onde é melhor a aplicação do Windows Forms e onde é melhor a aplicação do WPF. Muitos desenvolvedores nunca trabalharam com WPF, então só trabalham com o Windows Forms.

Quando o cliente necessita de um visual melhor trabalhado ou visualização dos dados de maneira específica, o programador pode não oferecer o melhor produto utilizando Windows Forms. Às vezes isto acontece por que quem vai desenvolver o software não conhece o WPF, já que é uma tecnologia mais nova, e não são todos que possuem o privilégio de trabalhar com uma tecnologia destas no emprego, e não possuem tempo de estudar isso fora do expediente do trabalho.

De alguma forma, também é objetivo deste artigo encorajar os desenvolvedores Windows Forms a produzir seus programas em WPF, comparando a tecnologia que ele utiliza atualmente e mostrando como uma aplicação pode ser desenvolvida em WPF, com grandes recursos que podem ser um diferencial comercial para seus sistemas que pretenderão desenvolver.

[nota]**Links**

Introdução ao desenvolvimento de Windows Forms
http://msdn.microsoft.com/pt-br/vbasic/ms789117.aspx

Introdução ao WPF
http://msdn.microsoft.com/pt-br/library/cc564903.aspx

Windows Presentation Foundation
http://windowsclient.net/WPF/

Introdução ao DirectX.
http://msdn.microsoft.com/pt-br/library/cc518041.aspx

Introdução ao Microsoft Windows Workflow Foundation
http://msdn.microsoft.com/pt-br/library/aa480214.aspx

Introdução ao Windows Presentation Foundation
http://msdn.microsoft.com/pt-br/library/aa970268.aspx

.NET Framework 3.5 Commonly Used Types and Namespaces Poster
http://windev.wordpress.com/2007/11/18/net-framework-35-commonly-used-types-and-namespaces-poster/[/nota]