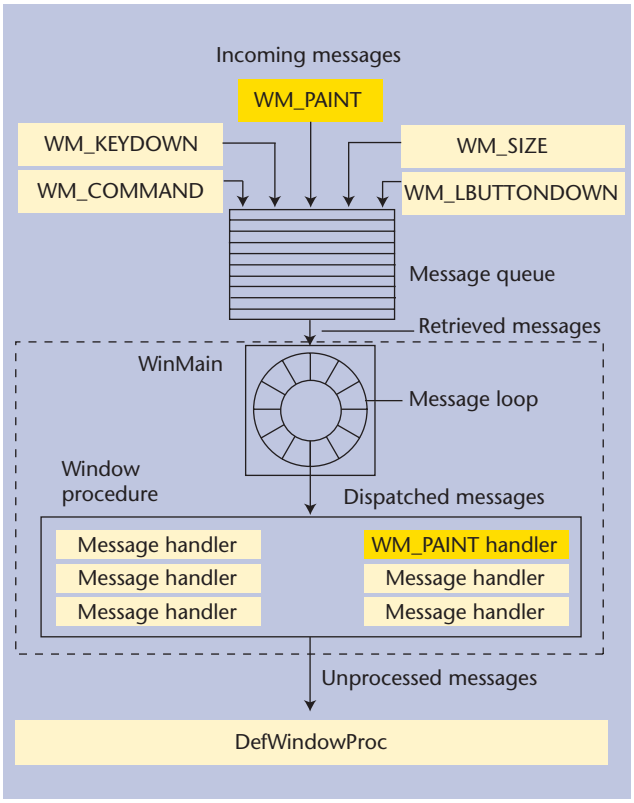
Avalon is de voormalige codenaam voor het nieuwe grafische subsysteem dat onderdeel zal zijn van de eind 2006 aangekondigde Windows Vista (voorheen Longhorn) platform-release. Op de jongste Microsoft PDC (Professional Developer Conference) werd bekend gemaakt dat Avalon voortaan Windows Presentation Foundation (WPF) zal heten, zodat we deze naam verder zullen hanteren. WPF bestaat uit een grafische engine en een managed-code framework. In dit artikel belicht Willem Koppenol de achtergronden en werking van de WPF.

Avalon komt eraan

Grafische hardware beter benut

Met WPF kun je hoogwaardige user interfaces maken waarin applicatie, UI, documenten en media content zijn geïntegreerd. Enerzijds maakt WPF gebruik van de geavanceerde mogelijkheden van de tegenwoordige grafische kaarten. Nieuwe functionaliteit, zoals 3D-graphics en transparante vensters, zijn daardoor mogelijk. Anderzijds introduceert WPF een nieuw programmeermodel waarin user interfaces declaratief in een XML-bestand worden gedefinieerd.



Figuur 1. Windows Programming Model en de WM_PAINT message.

Windows UI Het grafische subsysteem van Windows, gerepresenteerd door de dynamic link library's USER32.DLL en GDI32.DLL, zoals we dat vandaag de dag kennen, is ontstaan in een tijd van CGA- en EGA-beeldschermen en 286 processoren die draaiden op 8 MHz. De hardware-capaciteiten zijn vergeleken met Windows 1.0 enorm toegenomen. Weliswaar zijn in het grafische subsysteem sindsdien grote verbeteringen aangebracht zoals GDI+, toch wordt de huidige kracht van de hardware nauwelijks gebruikt in een gemiddelde Windows-applicatie. Alleen DirectX dat in 1995 werd geïntroduceerd en met name wordt toegepast bij games, maakt gebruikt van de mogelijkheden van de hardware. Standaard Windows-applicaties gebruiken DirectX niet. Met de introductie van WPF gaat dit veranderen, want WPF betekent een volledige herziening van de Windows presentation stack. In WPF komt de kracht van de hardware beschikbaar voor de Windows applicatie-ontwikkelaar.

UI Repainting In essentie is de schermopbouw van Windows-applicaties sinds de eerste release van Windows ongewijzigd en gebaseerd op een reactief repainting model. Het besturingssysteem houdt niet bij wat er op het scherm staat. Als een gedeelte van de applicatie, na op de achtergrond geraakt te zijn, weer zichtbaar wordt, krijgt de applicatie een repaint-opdracht uit het operating systeem. De applicatie moet dan code hebben klaarstaan die in staat is de user interface van de applicatie weer in goede staat te hertekenen. In Visual Basic 6.0 krijgt de applicatie een `Paint` event, in de MFC wordt de `OnDraw` methode aangeroepen en in .NET Windows Forms kennen we de `OnPaint` methode en het `Control.Paint` event. Het zijn allemaal implementaties als reactie op de `WM_PAINT` mes-

sage van het operating systeem:

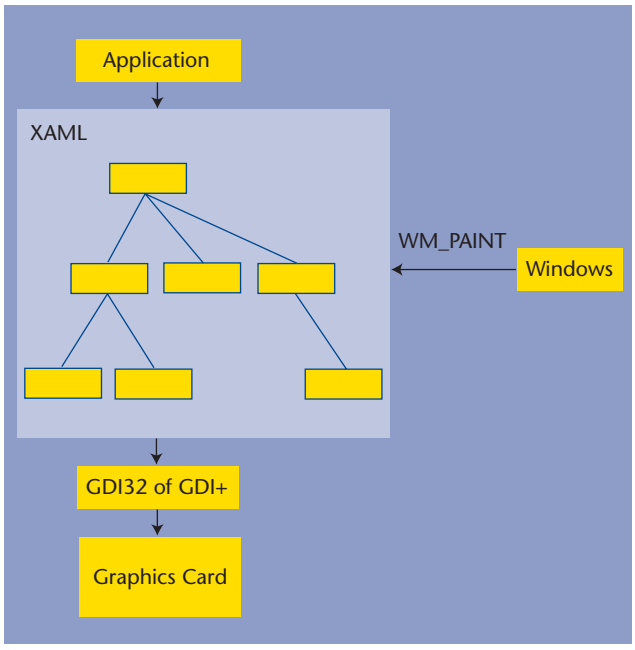
```
case WM_PAINT:
  hDC = BeginPaint(hwnd,&paintStruct);

  TextOut(hDC,10,10,"Goodbye WM_
PAINT",sizeof(string)-1);
  EndPaint(hwnd, &paintStruct);
  return 0;
```

UI BEWAREN Met WPF komt er een einde aan deze traditionele wijze van interactie tussen applicatie en operating systeem bij het opbouwen van de user interface. In WPF wordt de user-interface van een applicatie in essentie bewaard en niet hertekend. Er zijn verschillende technieken voor het bewaren van user interfaces.

Je kunt de user interface simpelweg bewaren als bitmap in het geheugen van de grafische kaart. Deze techniek wordt toegepast bij de gelaagde vensters die zijn geïntroduceerd in Windows 2000. Compositie effecten zoals transparante vensters worden dan mogelijk. Vaak gebeurt dit met hulp van speciale hardware van grafische kaarten.

Je kunt de user-interface ook beschrijven door een markup-taal en toegankelijk maken met een objectmodel. Deze techniek wordt toegepast bij HTML en de DOM (Document Object Model). Dit biedt meer flexibiliteit dan de bitmap-benadering omdat de elementen van de user interface zoals tekst en controls via het objectmodel benaderd kunnen worden en wanneer nodig aangepast. Een nadeel van het HTML-objectmodel is dat het een beperkt aantal primitieven heeft. Voor visueel rijk geschakeerde HTML-interfaces, moet toch een beroep worden gedaan op bitmaps.



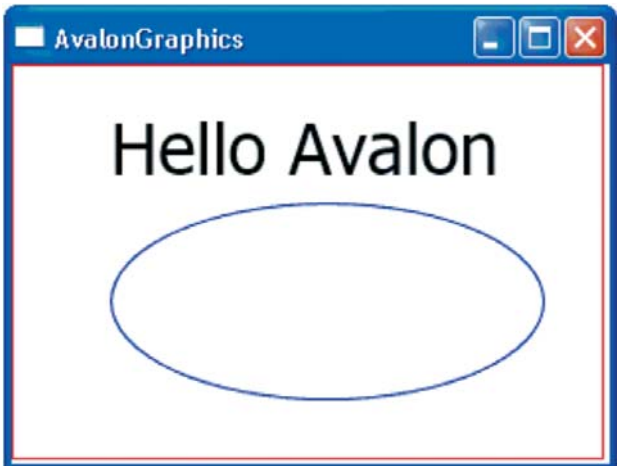
FIGUUR 2. User interface gedefinieerd met XAML markup.

In WPF worden grafische user interfaces gedefinieerd door de XAML markup-taal. De structuur van deze markup is tijdens runtime beschikbaar via een .NET objectmodel. In WPF kun je met behulp van deze techniek dezelfde rijke user interfaces maken als met Windows/Forms-applicaties onder het oude repainting-model. Dit komt doordat de XAML markup-taal bijzonder rijk geschakeerd is. XAML biedt bijvoorbeeld ondersteuning voor allerlei grafische primitieven (ellipsen, polygonen, rechthoeken e.d.), animatie en 3D. Deze grafische primitieven kunnen tijdens runtime worden benaderd en desgewenst gewijzigd. De beschikbare functionaliteit gaat verder dan GDI+. WPF voorziet verder in een krachtig compositiemodel. Ieder user interface-element, inclusief de grafische primitieven, kan voorzien worden van een eigen transparantiegraad.

DECLARATIEF PROGRAMMEER MODEL WPF introduceert een declaratief op XML gebaseerd programmeermodel voor user interfaces. User interfaces worden gedefinieerd met de markup-taal XAML (Extensible Application Markup Language). Het gebruik van XAML voor WPF user interfaces is vergelijkbaar met het gebruik van HTML voor webpagina's. XAML-pagina's moeten alleen aan de veel strengere XML syntax regels voldoen. Element- en attribuut-namen zijn hoofdlettergevoelig. Het gebruik van quotes bij attribuutwaarden is verplicht. XAML kan gebruikt worden om basis zaken als vensters, controls en grafische elementen te maken. Maar XAML is veel rijker geschakeerd dan dat. XAML wordt ook gebruikt bij 2D en 3D imaging, animatie, audio en video. Een user interface met een venster en enkele grafische elementen wordt gecreëerd met:

```
<Window x:Class="AvalonGraphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/
avalon/2005"
  xmlns:x="http://schemas.microsoft.com/
winfx/xaml/2005"
  Title="AvalonGraphics">
  <Canvas>
    <Rectangle Width="300"
         Height="200" Stroke="Red" />
    <TextBlock FontSize="36"
     Canvas.Left="50" Canvas.Top="20">
     Hello Avalon
    </TextBlock>
    <Ellipse  Canvas.Left="50" Canvas.
Top="70"
     Width="220" Height="100" Stroke="Blue"
/>
  </Canvas>
</Window>
```

Het canvas is hier de houder van verschillende elementen. Ieder element heeft eigenschappen die met XAML-attributen worden aangegeven. Sommige zijn generiek en gelden voor alle elementen zoals "Stroke". Anderen zijn specifiek voor een bepaald element. Het resultaat van de XAML-markup is te zien in figuur 3.

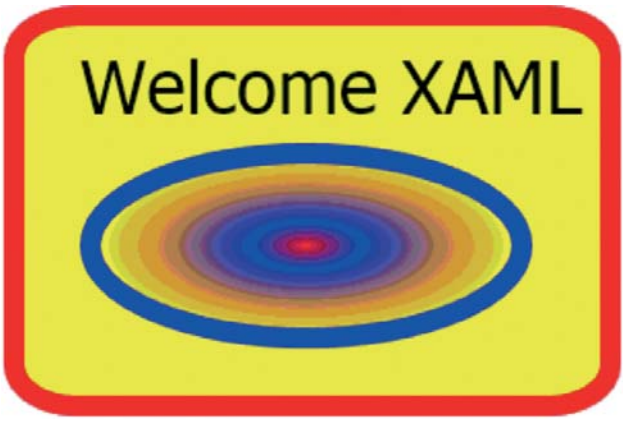


FIGUUR 3. WPF user interface beschreven in XAML.

DECORATIES Extra attributen zorgen voor decoraties op de grafische elementen. Met de attributen `StrokeThickness` en `RadiusX/Y` kunnen we bijvoorbeeld de lijndikte van de `Rectangle` en `Ellipse` veranderen en de hoeken afronden. Door het element `<Ellipse.Fill>` in het `<Ellipse>` element op te nemen, kunnen we de kleur van de `Ellipse` aanpassen. In de onderstaande XAML-markup gebruiken we een `<RadialGradientBrush>` element om een verlopende kleur te krijgen. Het resultaat is te zien in figuur 4.

```
<Rectangle Width="300" Height="200"
   Fill="Yellow" Stroke="Red"
StrokeThickness="10"
   Canvas.Left="3" Canvas.Top="3"
RadiusX="30" RadiusY="20"/>

<Ellipse  Canvas.Left="40" Canvas.Top="70"
     Width="220" Height="100"
StrokeThickness="10" Stroke="Blue">
  <Ellipse.Fill>
    <RadialGradientBrush>
      <RadialGradientBrush.GradientStops>
    <GradientStop Color="Red" Offset="0" />
    <GradientStop Color="Blue" Offset="0.25"/>
    <GradientStop Color="Orange"
Offset="0.75"/>
    <GradientStop Color="Yellow" Offset="1"/>
  </RadialGradientBrush.GradientStops>
    </RadialGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```



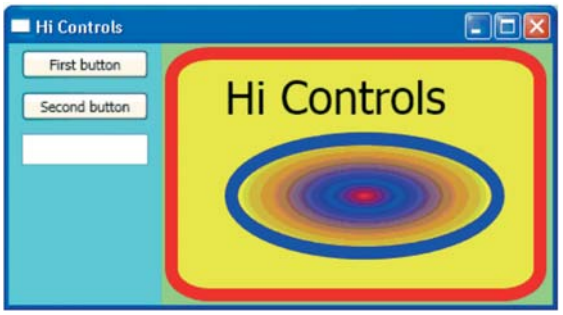
FIGUUR 4. Gradiënten en andere decoraties.

COMPLEXE PROPERTY'S XAML kent het concept van de complexe property's. Niet alle property's kunnen worden gerepresenteerd door een string. Sommige property's hebben een interne structuur die bestaat een aantal geneste objecten. XAML heeft een speciale syntax voor deze complexe property's. Ze zijn te herkennen aan de "X.Y" element-declaratie. Deze syntax betekent dat Y een alias is voor een XML-fragment dat ter plekke wordt gedeclareerd en dat Y een property is van X. In zekere zin is dit een praktische workaround voor het feit dat je geen XML-fragmenten aan XML-attributen kunt toekennen.

In de bovenstaande XAML-markup komen een tweetal complexe property's namelijk `<Ellipse.Fill>` en `<RadialGradientBrush.GradientStops>` voor. Het `<RadialGradientBrush.GradientStops>` element zet de GradientStops property van de `RadialGradientBrush`. Aan de GradientStops property wordt vervolgens een XML-fragment met de lijst van `<GradientStop>` elementen toegekend.

CONTROLS IN WPF Ook controls zoals buttons, labels, textboxen en listboxen worden in WPF declaratief aangemaakt. Je kunt de controls op een `StackPanel` plaatsen waarmee je ervoor zorgt dat ze netjes onder of naast elkaar gepositioneerd worden. Attributen kunnen de afstand tot de rand van het venster aangeven en specificeren welke uitlijning moet worden toegepast. Behalve een `StackPanel` kun je ook een `DockPanel` gebruiken om elementen en andere panels te groeperen. Met het `DockPanel.dock` element geef je aan dat elementen aan een bepaalde kant van het `DockPanel` moeten worden gepositioneerd. De volgende XAML-code creëert een venster met een `DockPanel` waarop een `StackPanel` en een `Canvas` zijn geplaatst. Het `StackPanel` bevat twee buttons en een textbox. Het canvas kennen we al uit eerdere voorbeeldcode. Het resultaat is te zien in figuur 5.

```
<DockPanel Background="LightGreen">
   <StackPanel Name="StackPanel2"
Background="Aqua">
  <Button Margin="10,5,10,5"
    HorizontalAlignment="Left" Width="100">
    First button
  </Button>
  <Button Margin="10,5,10,5"
    HorizontalAlignment="Left" Width="100">
    Second button
  </Button>
  <TextBox Name="Textbox1" Width="100"
    HorizontalAlignment="Left"
    Margin="10,5,10,5">
  </TextBox>
   </StackPanel>
   <Canvas>
  <Rectangle .. />
  <TextBlock .. />
  <Ellipse .. />
   </Canvas>
</DockPanel>
```



FIGUUR 5. Controls in WPF.

EVENT HANDLING De structuur van deze markup kan tijdens runtime benaderd worden via een .NET-objectmodel. Dit is vergelijkbaar met de wijze waarop in bijvoorbeeld JavaScript het HTML DOM-objectmodel van een webpagina benaderd kan worden. Events op user interface elementen kunnen gekoppeld worden aan .NET-code door de naam van de functie die het event afhandelt als property in XAML op te geven bij het control dat het event genereert. Button controls genereren click events en bij een button control zetten we de Click property:

```
<Button Click="Button_Click1" .. </Button>
<Button Click="Button_Click2" .. </Button>
```

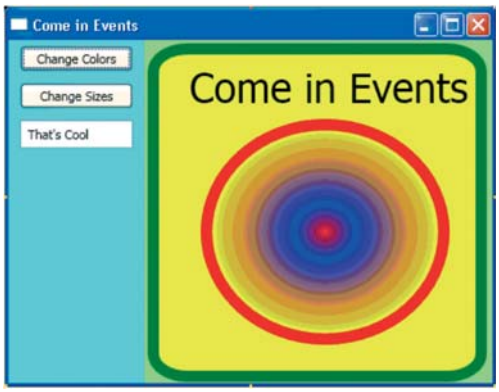
Property's van controls en andere XAML elementen kunnen we in code benaderen en aanpassen als we ze in de XAML-pagina een naam hebben gegeven:

```
<TextBox Name="Textbox1" .. >
<Rectangle Name="r1" .. >
<Ellipse Name="e1" .. >
```

De functie die het event afhandelt zouden we onder een <code> element in een CDATA sectie in dezelfde XAML-pagina kunnen opnemen. Dit is echter niet erg elegant. Gebruikelijker is het de event-functies net als in ASP.NET op te nemen in een code behind-bestand. Dit code behind bestand is een partiële class die tijdens compilatie samen met de XAML-pagina tot een complete class wordt gecompileerd. In Visual Studio wordt een code behind-bestand voor iedere XAML-pagina automatisch aangemaakt. Het deel van het code behind-bestand met de eventfuncties is te zien in onderstaande code. Als de click events hebben plaats gevonden is het resultaat te zien in figuur 6.

```
private void Button_Click1(object sender,
                                RoutedEventArgs
e) {
    Textbox1.Text = "That's Cool";
    e1.Stroke = Brushes.Red;
    r1.Stroke = Brushes.Green;
}

private void Button_Click2(object sender,
                                RoutedEventArgs
e){
    Textbox1.Text = "Super Cool";
    e1.Height = 200;
    r1.Height = 300;
}
```



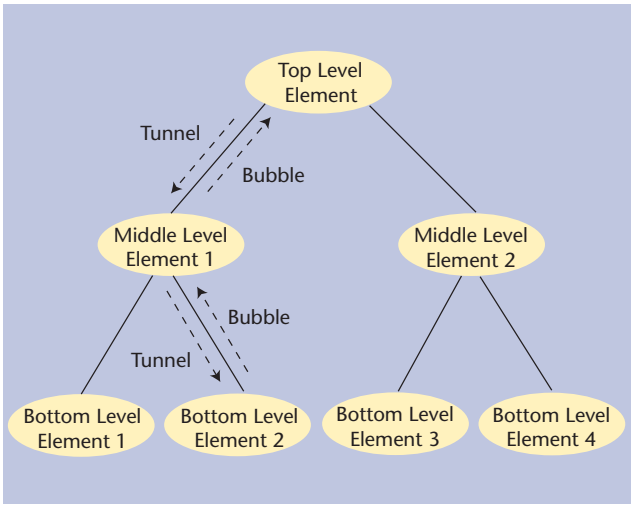
FIGUUR 6. Controls en WPF events.

EVENT ROUTING In een WPF user interface kan het ene element weer andere elementen bevatten. Er kan een boomstructuur bestaan die verscheidene lagen diep is. Events die als doelwit één van de dieper geneste elementen hebben, kunnen ook worden opgepakt en afgehandeld door elementen op hogere niveaus. WPF gebruikt hiervoor een event routing-mechanisme. Bij event routing wordt een event afgeleverd bij verscheidene elementen, tot één van deze het event als afgehandeld markeert. Events kunnen drie routing mechanismen hanteren: Direct-Only, Bubbling en Tunneling.

Direct Only betekent dat alleen het element waar het event direct op is gericht van het event op de hoogte wordt gesteld. Dit mechanisme wordt gehanteerd door Windows Forms en andere .NET-library's.

Event Bubbling wil zeggen dat eerst het element dat het directe doelwit is van het event wordt gewaarschuwd. Vervolgens het element dat hier direct boven ligt en zo verder. Bij Bubbling gaan events van onder in de boomstructuur naar boven.

Event Tunneling is het tegenovergestelde proces. Events beginnen boven in de boomstructuur en dalen af naar diepere lagen. Alle elementen die het doelwit omvatten te beginnen met het element op het hoogste niveau krijgen het event aangeboden.



FIGUUR 7. Event Routing.

RESOURCES XAML kent ook resources zoals brushes of styles. Je kunt deze resources ergens in de applicatie definiëren en er vervolgens op andere plekken aan refereren. Ieder element heeft een resources-collectie. Je kunt de resources dus bij ieder element definiëren, maar meestal worden ze bij het top level-element gedefinieerd zoals bij het onderstaande `StackPanel`:

```
<StackPanel
  xmlns:x="http://schemas.microsoft.com/
winfx/xaml/2005">
  <StackPanel.Resources>
 <SolidColorBrush x:Key="MyBrush"
Color="gold"/>
  </StackPanel.Resources>
  <Button Background="{StaticResource
MyBrush}" .. />
  <Ellipse Fill="{StaticResource MyBrush}" ..
/>
</StackPanel>
```

Als de resource is gedefinieerd, kun je er bij het zetten van een property aan refereren. De definitie namespace met de x prefix bindt de declaratie van `SolidColorBrush` aan het `Resources` attribuut van het `StackPanel`. Als de `Button` of de `Ellipse` refereert aan `{StaticResource MyBrush}`, zal de runtime proberen de reference op te lossen in de huidige scope. De `SolidColorBrush` wordt gevonden, deze wordt geïnstantieerd en een referentie ernaar wordt teruggegeven.

APPLICATIE TYPEN De WPF kent twee typen applicaties: op zichzelf staande applicaties en navigatie-applicaties. Op zichzelf staande applicaties zijn vergelijkbaar met conventionele Win32 Window/Form applicaties. Navigatie-applicaties bestaan uit een collectie van verschillende pagina's met links naar elkaar en zijn conceptueel vergelijkbaar met webapplicaties. Gebruikers kunnen via de links tussen de pagina's navigeren. Een subtype van de navigatie-applicatie is nog de document-applicatie die uitsluitend bedoeld is om informatie te presenteren Een document-applicatie kan geen code bevatten.

APPLICATIE OBJECTEN Window/Form-applicaties worden gerepresenteerd door een `Application`-object dat voorziet in basis- applicatieservices. Navigatie-applicaties worden gerepresenteerd door het zwaardere `NavigationApplication` object dat ook navigatie-ondersteuning biedt. Het startpunt van een WPF-applicatie is de `AppStartup` methode die door het besturingssysteem wordt aangeroepen en waarin vaak het hoofdvenster wordt opgeroepen:

```
namespace AvalonGraphics {
  public partial class MyApp : Application {
    void AppStartup(object sender,
StartupEventArgs args){
          Window1 mainWindow = new Window1();
          mainWindow.Show();
    }
  }
}
```

HOSTING AVALON APPLICATIONS Avalon-applicaties kunnen op verschillende manieren worden gehost. 'Express'-applicaties worden gehost in de browser en worden niet op de client computer geïnstalleerd. Zij draaien in een 'sandbox' en hebben weinig rechten om systeem-resources te benaderen. 'Installed''-applicaties worden gehost in een venster of een navigatievenster. Zij hebben volledige toegang tot systeemresources en kunnen de gehele WPF-API gebruiken.

WPF UITPROBEREN Ontwikkelaars hoeven niet te wachten op de release van Vista en kunnen nu al experimenteren met WPF en XAML. Als de WinFX-SDK wordt geïnstalleerd is de WPF-functionaliteit beschikbaar en kun je WPF ook draaien onder Windows XP en Windows 2003 Server. De WinFX SDK kan vrij worden

gedownload en bevat veel democode. Het XAML-ontwikkeltool Xamlon wordt bij de SDK geleverd en je kunt dus direct met XAML aan de slag. Als je echter liever met Visual Studio 2005 Bèta 2 werkt, is dit ook een optie. Na installatie van de SDK verschijnen er diverse Avalon-templates in Visual Studio. Voor dit artikel werd gebruikt gemaakt van Visual Studio. Hoewel Visual Studio 2005 goede intellisense levert bij het invoeren van XAML, ontbreekt vooralsnog een echte Visuele Editor met drag en drop mogelijkheden voor controls en grafische elementen. Dit heeft ongetwijfeld te maken met de premature staat waarin WPF verkeert, waarschijnlijk zal dit nog wel veranderen.

SLOTWOORD WPF belooft een fikse verandering te worden in de wijze waarop user interfaces worden ontworpen. Het is het definitieve einde van reactieve repainting model en betekent een betere benutting van de mogelijkheden van de huidige grafische hardware. Een centraal element van WPF is de rijk geschakeerde XAML markup-taal. Ontwikkelaars zullen WPF user interfaces met XAML-markup gaan ontwerpen. XAML is als markup-taal voor GUI's op zich niet uniek. XAML heeft overlap met de XUL-taal van Mozilla en de SVG (Standardized Vector Graphics) van het W3C. XAML biedt echter een betere integratie met het .NET platform. De contouren van WPF lijken wel duidelijk, maar voor de definitieve release kan door het team van 400 ontwikkelaars dat aan WPF werkt nog veel worden veranderd.

drs. Willem Koppenol, Senior Trainer en Product Specialist Software Development Twice IT Training (e-mail: wkoppenol@twice.nl).