Windows Presentation Foundation (WPF)

Anno Accademico 2017-18

C. Barberis, F.Cantavenera, G. Malnati, 2004-18

Applied Pervasive Architectures Lab

Introduzione

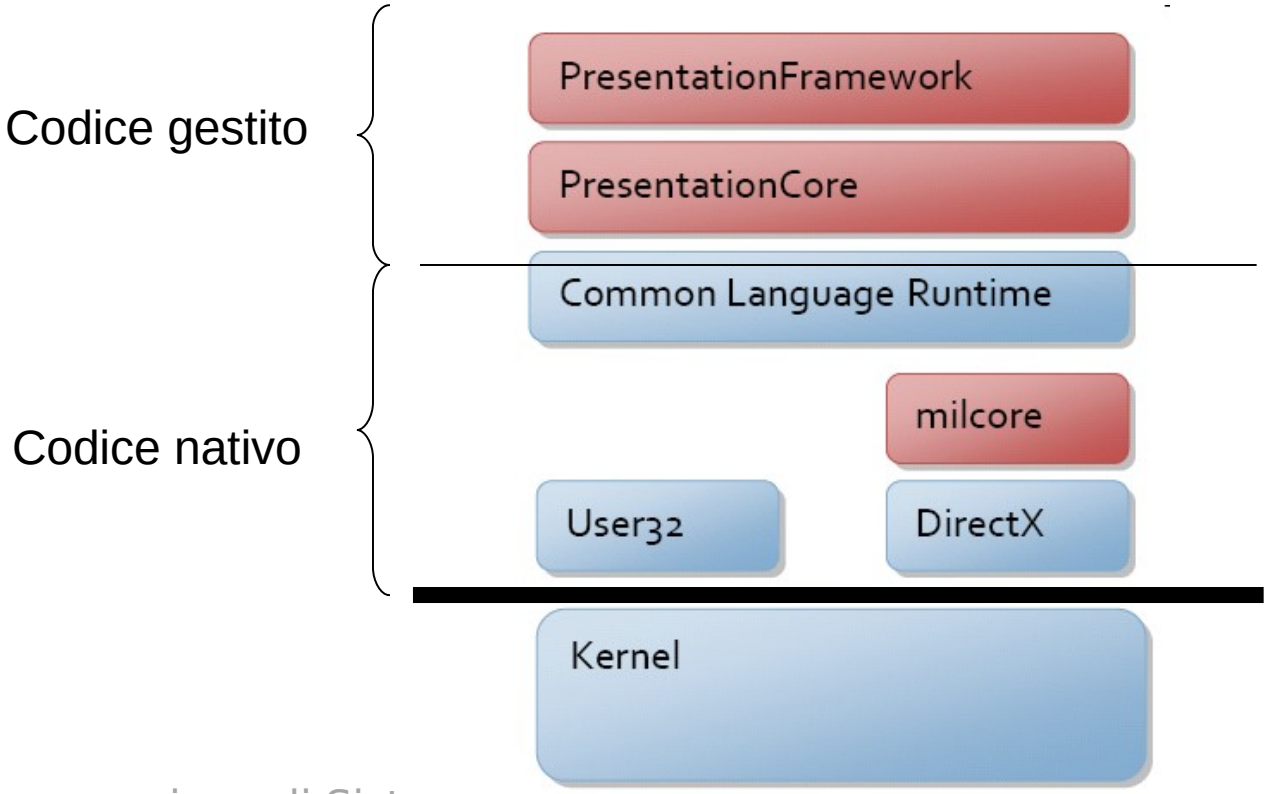
- Uno dei requisiti essenziali per una buona applicazione grafica è avere un “look” accattivante
- Esistono strumenti più potenti ed efficienti di GDI
  - Libreria ormai datata
  - Poco efficiente dal punto di vista della gestione delle risorse

Windows Presentation Foundation (1)

- Offre un approccio unificato per l'integrazione all'interno di applicazioni di componenti di tipo diverso:
  - Documenti
  - Contenuti multimediali (audio, video)
  - Grafica 2D/3D
  - Oggetti MFC, Win32

Windows Presentation Foundation (2)

- Componente del framework .NET 3.0
  - disponibile a partire da XP SP2



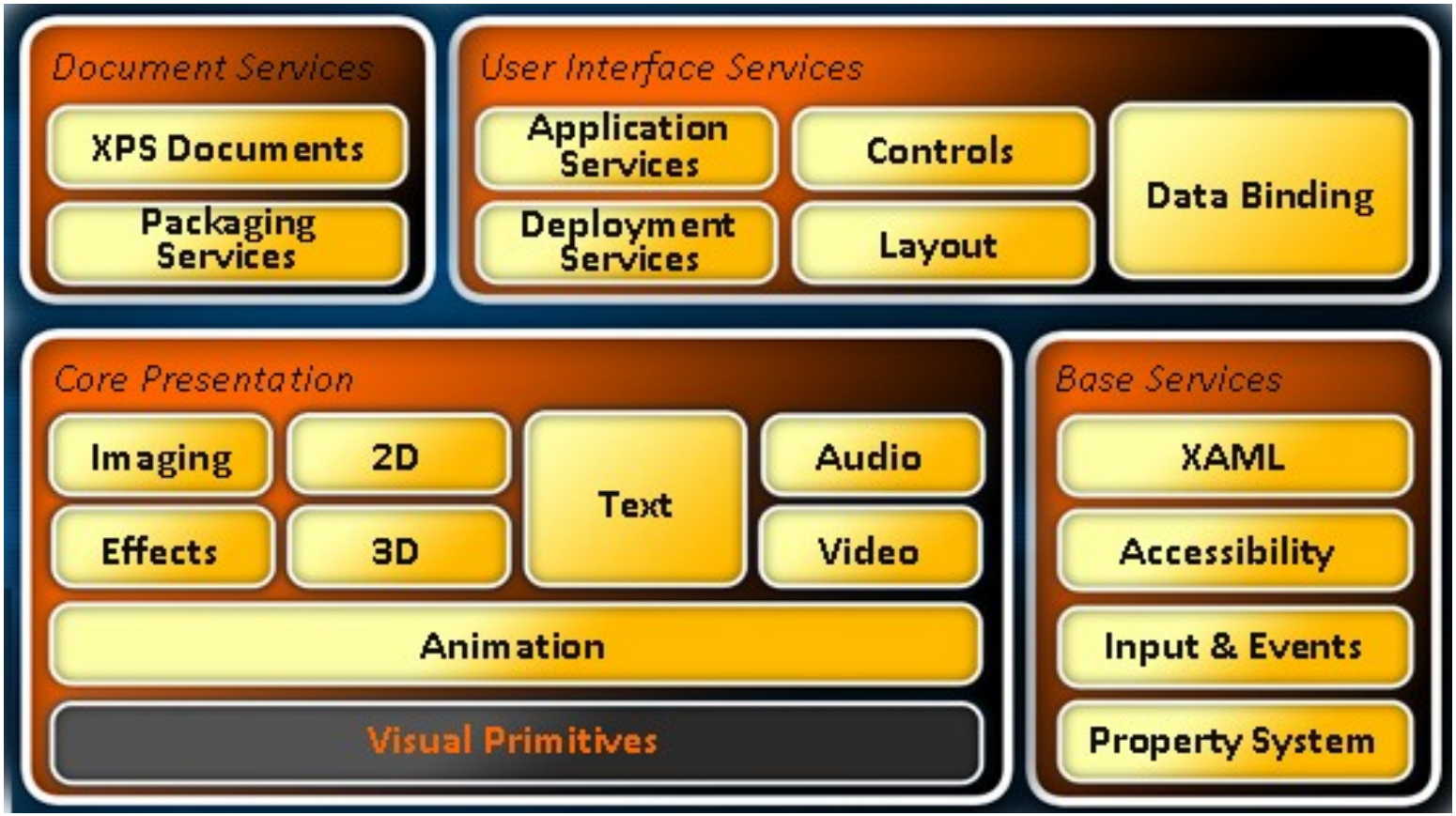
Windows Presentation Foundation (3)

- Interamente basato su grafica vettoriale
  - Sfrutta al meglio le potenzialità dei calcolatori moderni
- Specificamente pensato per la realizzazione di interfacce ad alto impatto visivo
- Basato sul motore grafico DirectX

Windows Presentation Foundation (4)

- Unisce un linguaggio dichiarativo a uno procedurale
  - Separazione esplicita tra logica applicativa e interfaccia grafica
  - Netta suddivisione dei ruoli di grafico e sviluppatore
- Procedura di installazione semplificata
  - Tramite browser
  - Applicazioni stand alone

Architettura WPF



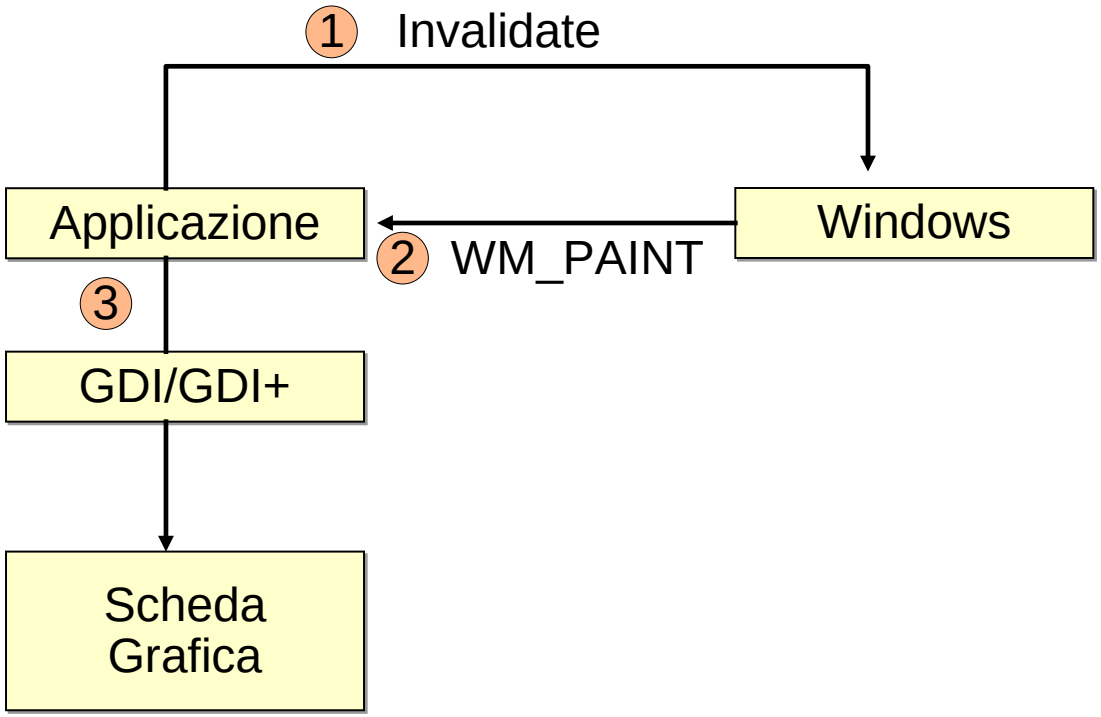
Visual rendering

- WPF introduce diverse innovazioni dal punto di vista del rendering degli oggetti grafici:
  - Modalità “retained”
  - Grafica vettoriale
  - Unità di misura grafica indipendente dal dispositivo

Modalità immediate

- GDI/GDI+ disegna gli oggetti grafici secondo una modalità detta “immediate”
  - Applicazione direttamente responsabile del ridisegno della propria client area
  - Gli oggetti grafici vengono (ri)disegnati invocando una callback dell’applicazione che si occupa del rendering
  - GDI non conserva alcuna informazione sugli oggetti disegnati:
    - Riceve l’immagine dall’applicazione e la copia nel frame buffer della scheda video

Modalità immediate



Modalità retained

- WPF usa una modalità detta “retained”
  - Ciascun oggetto grafico contiene dati sul proprio rendering, che possono essere:
    - Dati vettoriali
    - Immagini
    - Glifi
    - Video
  - Il sistema grafico mantiene in una struttura ad albero i dati sul rendering degli oggetti che compongono la scena da visualizzare

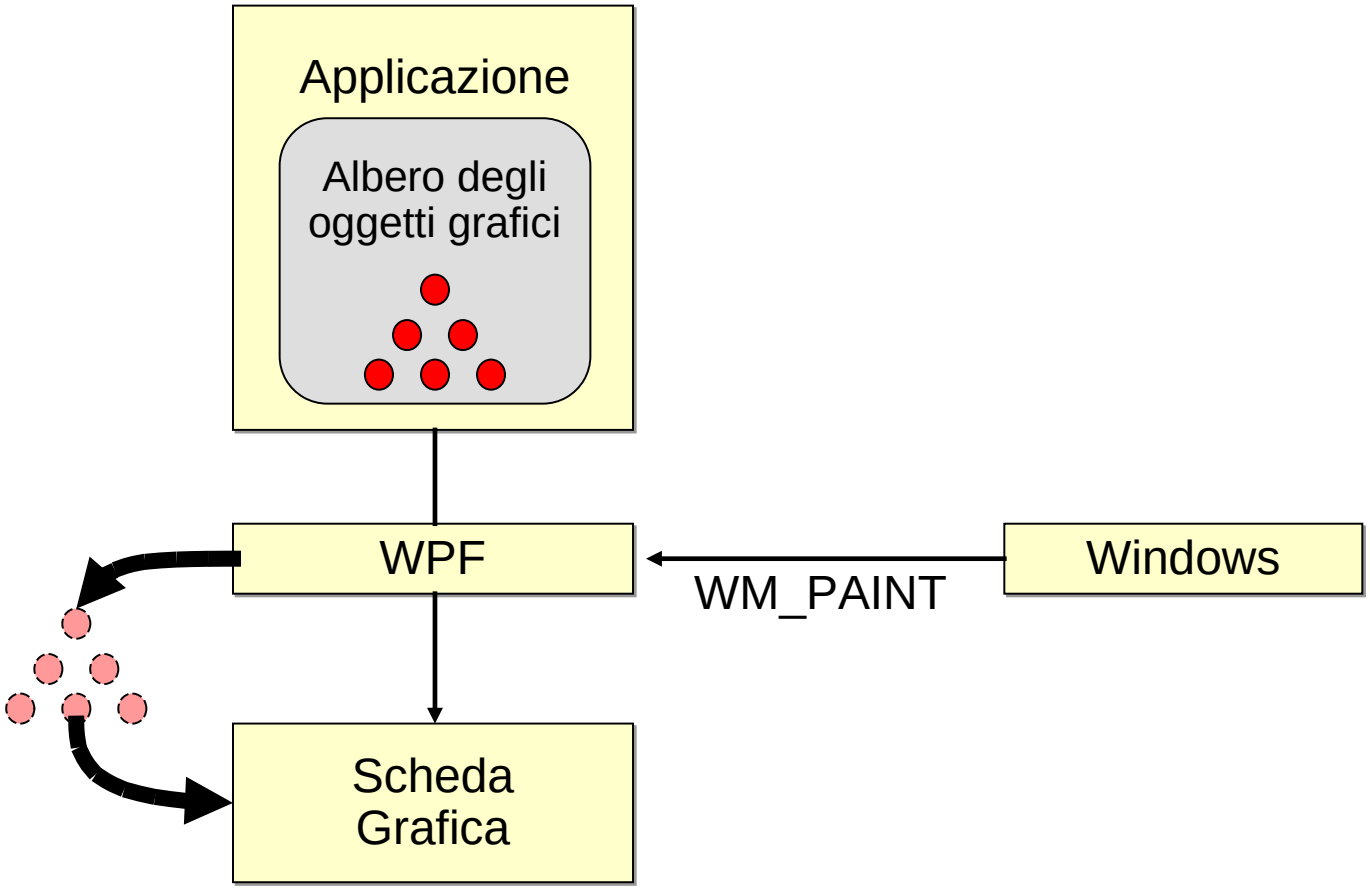
Modalità retained

- WPF traduce questi dati in una serie di istruzioni che passa alla GPU della scheda grafica
  - Quando arriva una richiesta di disegno, tali istruzioni vengono eseguite
- A differenza della modalità “immediate”, è il sistema grafico WPF che gestisce autonomamente le operazioni di ridisegno, non l’applicazione
  - Ottimizzando le operazioni

Modalità retained

- Questo comporta alcuni vantaggi:
  - Lo sviluppo delle applicazioni grafiche viene semplificato
    - Non deve essere implementata la parte di ridisegno degli oggetti grafici
  - Maggiore efficienza
    - Caching delle istruzioni di rendering
    - Vengono sfruttate a pieno le potenzialità della scheda video

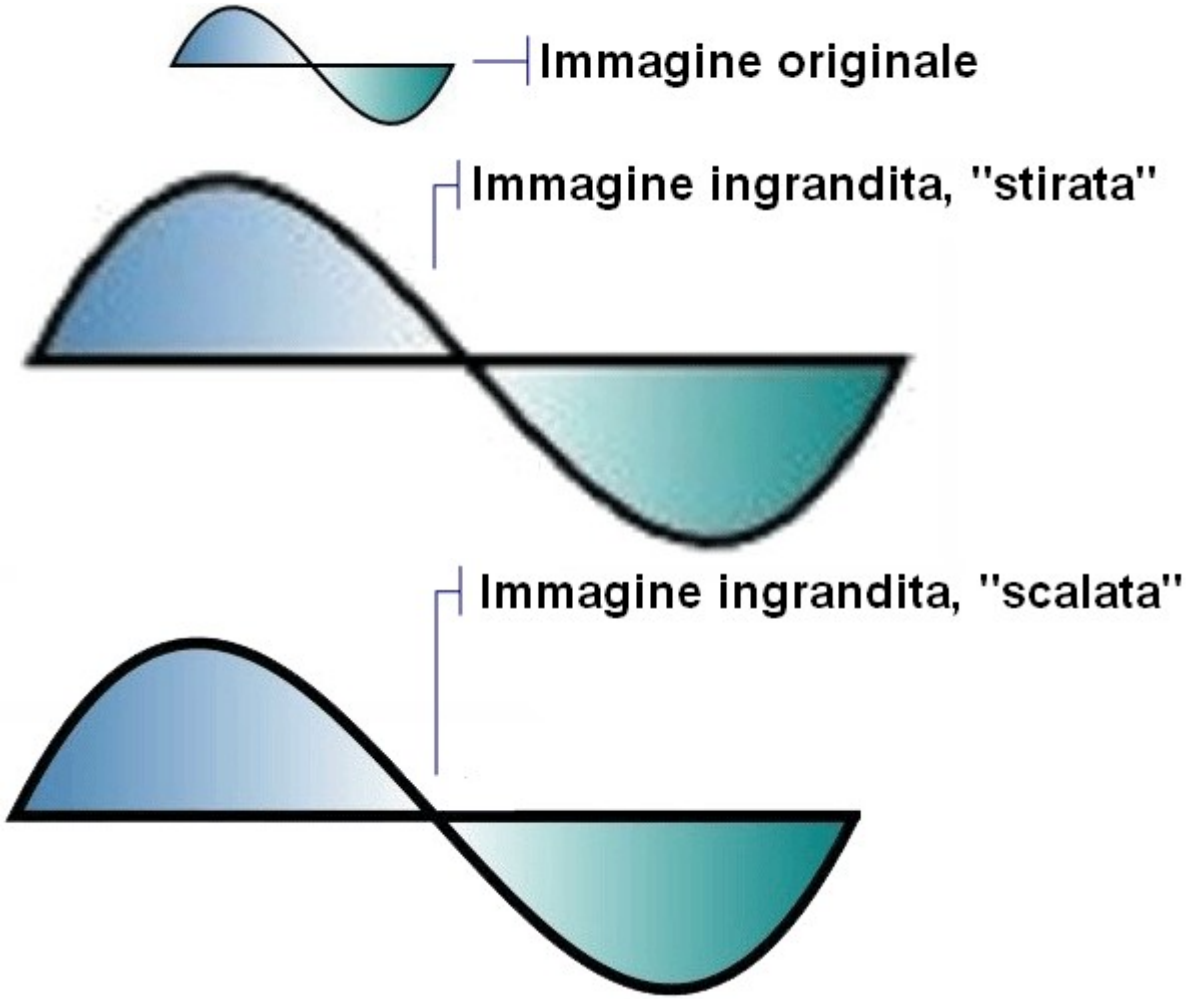
Modalità retained



Grafica vettoriale

- Consente di descrivere un elemento grafico in base ad una serie di primitive grafiche
  - Soluzione più sofisticata della tradizionale grafica “bitmap”
  - Es: font TrueType descritti come insieme di linee, curve e comandi di riempimento
- Un’immagine ridimensionata viene ridisegnata e non perde qualità rispetto all’originale
  - L’immagine viene “scalata” anziché “stirata”

Grafica vettoriale



Unità di misura

- Due fattori determinano la dimensione fisica del testo e oggetti grafici sullo schermo
  - Risoluzione dello schermo: numero di pixel visualizzati
  - Densità dei pixel (DPI, dot per inch): dimensione di un pollice ideale espressa in pixel
- WPF supporta schermi con differenti densità e usa come unità di misura primaria i “device independent pixel” anziché i pixel hardware
  - Consente di riadattare automaticamente testo ed elementi grafici a diverse risoluzioni e DPI mantenendo costante la dimensione

Sviluppo in WPF

- E’ possibile sviluppare applicazioni usando:
  - Codice C# che istanzia gli oggetti e li collega tra loro nel modo voluto
  - XAML (eXtensible Application Markup Language), linguaggio descrittivo basato su XML
  - Un misto dei due linguaggi
    - XAML per la descrizione dell’interfaccia
    - C# per l’implementazione della logica

XAML

- Usato in WPF per creare e inizializzare oggetti secondo una data gerarchia
- Tutte le classi in WPF hanno solo un costruttore privo di argomenti
  - Si utilizzano le proprietà per configurare il comportamento e l'aspetto degli oggetti
  - Questo facilita l'integrazione con un linguaggio descrittivo

Vantaggi di XAML

- Codice compatto
- Istruzioni leggibili, anche per i non programmatori
- Separazione del codice per l'aspetto grafico da quello della logica applicativa
- Le proprietà degli oggetti vengono specificate come proprietà di elementi XML
- La conversione dei valori dei tipi viene fatta in automatico

Esempio in C#

```
public partial class Window1 : Window  {
      public Window1() {
         InitializeComponent();
         StackPanel stackPanel = new StackPanel();
         this.Content = stackPanel;
         this.Background = new LinearGradientBrush(Colors.Wheat,Colors.White,
                                                   new Point(0,0), new Point(1,1));
         TextBlock textBlock = new TextBlock();
         textBlock.Margin = new Thickness(20);
         textBlock.FontSize = 20;
         textBlock.Foreground = Brushes.Blue;
         textBlock.Text = "Hello World";
         stackPanel.Children.Add(textBlock);
         Button button = new Button();
         button.Margin= new Thickness(10);
         button.Height = 25; button.Width = 80;
         button.HorizontalAlignment = HorizontalAlignment.Right;
         button.Content = "OK";
         stackPanel.Children.Add(button);
      }
   }
```

Esempio equivalente in XAML

```
<Window x:Class="hello.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="145" Width="298" >
   <Window.Background>
      <LinearGradientBrush StartPoint="0 0" EndPoint="1 1" >
         <GradientStop Offset="0" Color="Wheat"/>
         <GradientStop Offset="1" Color="White"/>
      </LinearGradientBrush>
   </Window.Background>
   <StackPanel>
      <TextBlock Margin="20" FontSize="20" Foreground="Blue">
         Hello World </TextBlock>
      <Button Margin="10" HorizontalAlignment="Right"
         Height="25" Width="80">OK</Button>
   </StackPanel>
</Window>
```

Reattività

- Le applicazione dotate di interfaccia grafica devono essere reattive
  - Richiedono l'impiego di più di un thread
  - Un thread principale
    - raccoglie continuamente l'input dell'utente
  - Uno o più thread secondari
    - eseguono i compiti in base agli input e comunicano i risultati al thread principale il quale può aggiornare conseguentemente l'interfaccia utente

Thread in WPF

- Ogni applicazione WPF ha associati due thread all'atto della creazione
  - *Rendering thread*
    - Eseguito in background
  - *UI thread*:
    - Raccoglie gli input
    - Aggiorna l'albero degli oggetti grafici
    - ...
- E' possibile creare altri thread per aumentare la reattività del programma
  - Sconsigliato, aumenta la complessità

Dispatcher

- Oggetto che fa da intermediario nello scambio di messaggi tra thread differenti
- Concettualmente analogo al gestore degli eventi in Win32
- Mantiene al suo interno una coda
  - Seleziona ed inoltra i messaggi secondo criteri di priorità
  - Ogni UI Thread deve essere associato ad un Dispatcher

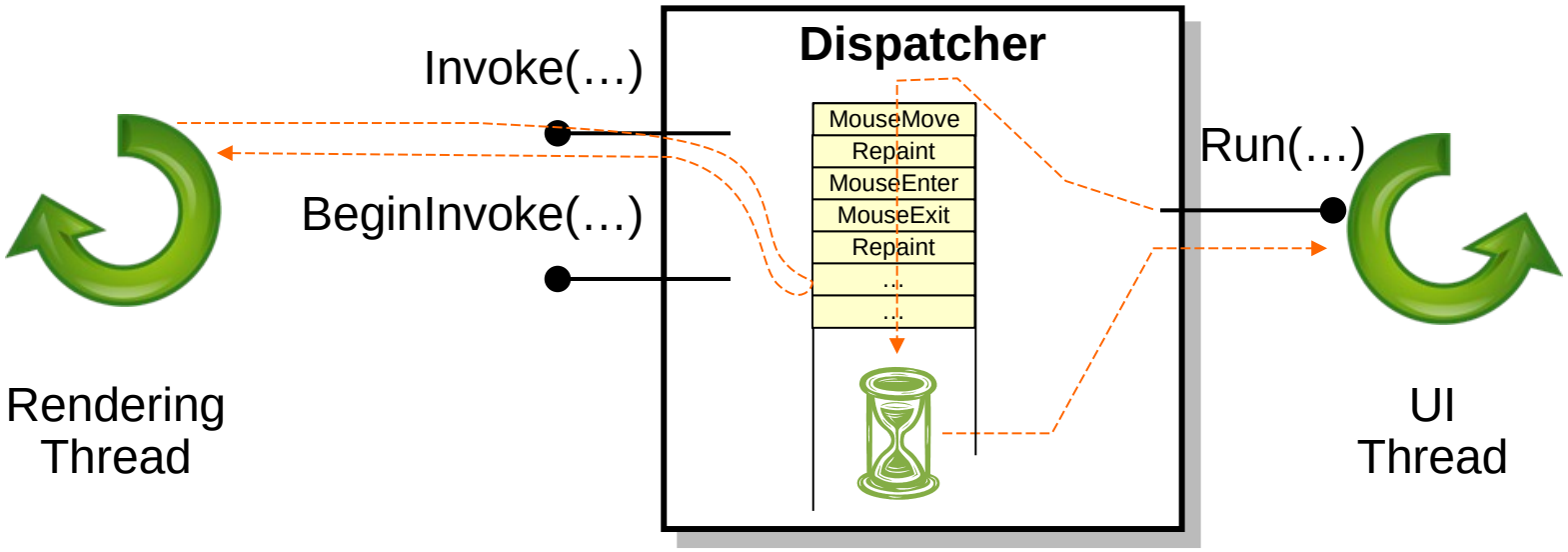
Dispatcher

- UI Thread e Rendering Thread fanno accesso alla coda del Dispatcher
- UI Thread
  - Inserisce richieste a fronte del verificarsi di un evento
  - Legge i messaggi di notifica da parte degli altri thread
- Rendering Thread
  - Riceve richieste di inizio di un'attività
  - Inserisce messaggi sull'esito delle attività svolte

Dispatcher

- Offre due metodi per inserire messaggi nella coda:
  - Invoke(): sincrono, bloccante
  - BeginInvoke() asincrono, non bloccante
- La coda deve essere svuotata velocemente
  - Quanto più velocemente ciò accade tanto più reattiva risulterà l'applicazione

Dispatcher


Dispatcher
Invoke(…)
BeginInvoke(…)
MouseMove
Repaint
MouseEnter
MouseExit
Repaint
…
…
Run(…)
Rendering Thread
UI Thread

Eseguire compiti lenti

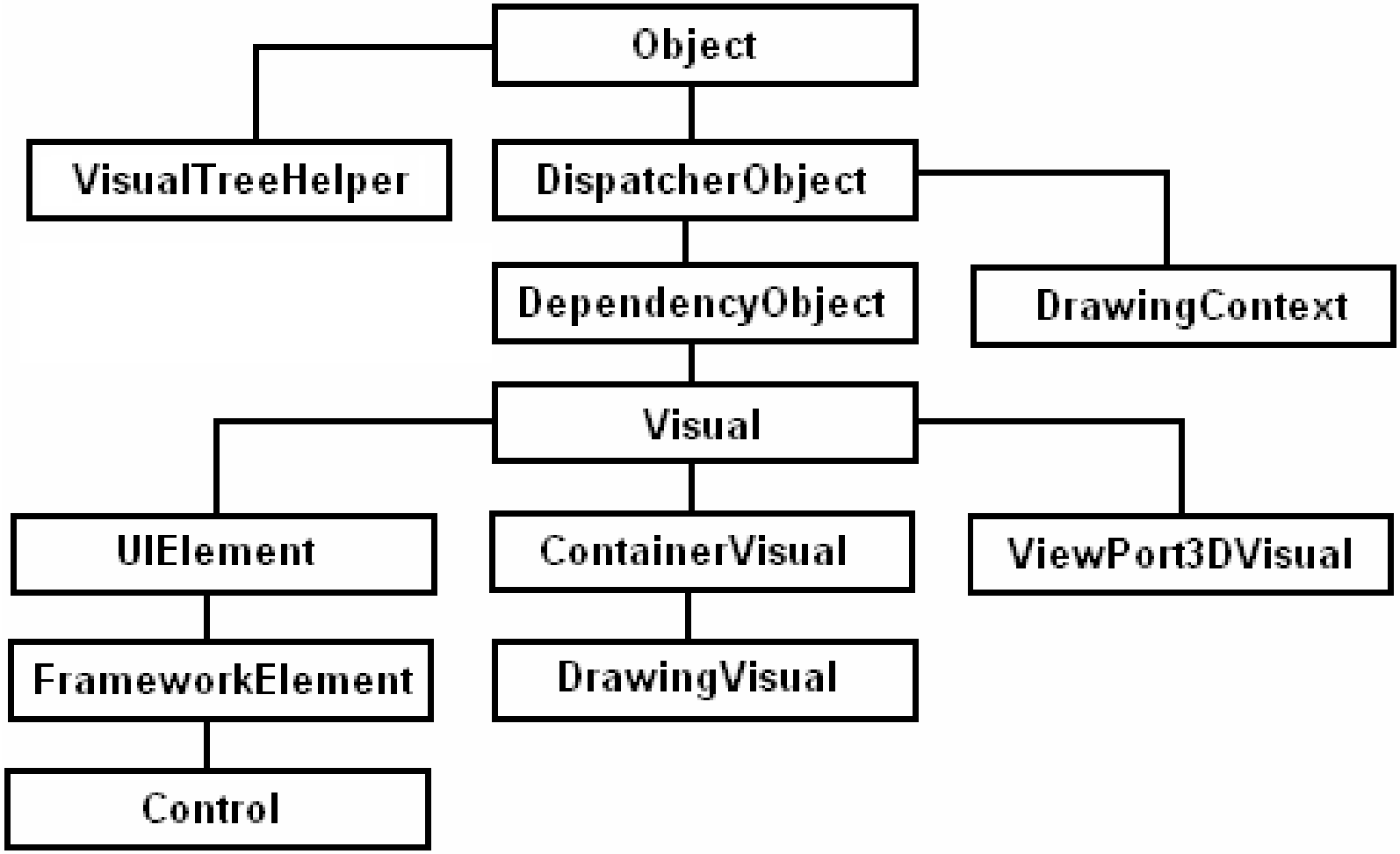
```
private void b1_Click(object sender, RoutedEventArgs e)
{
    b1.IsEnabled = false;

    Task.Run(
        () => {
      DoSlowWork();
      Dispatcher.BeginInvoke(
                    DispatcherPriority.Normal,
          new Action(DoUIUpdate));
    });
}

private void DoSlowWork() {
  //…
}

private void DoUIUpdate() {
  b1.IsEnabled = true;
}
```

Classi base in WPF



System.Threading.DispatcherObject

- Radice di tutti gli oggetti dotati di una presentazione grafica WPF
- Mantiene un riferimento all'oggetto Dispatcher associato al thread in esecuzione
  - Garantisce che tutte le operazioni effettuate dalle proprie istanze avvengano nel contesto del thread responsabile della gestione del Dispatcher cui è associato

DependencyProperties

- Tutte le informazioni relative al contenuto grafico sono memorizzate nelle proprietà dei suoi nodi
  - La classe System.Windows.DependencyObject offre un meccanismo efficiente per collegare tra loro due proprietà, e gestire la percolazione dei valori attraverso l'albero logico che descrive la scena
- Il valore della proprietà non viene duplicato finché non cambia, anche se appartiene a due oggetti diversi di una certa classe
  - Minore occupazione di memoria
- Gli oggetti di questo tipo sono osservabili
  - Questo permette di registrarsi su una proprietà e ricevere notifiche ogni volta che cambia

Implementare una proprietà

- Occorre fare rifierimento ad un'istanza di DependencyProperty dichiarata a livello classe
  - Il framework provvederà ad allocare memoria specifica per l'istanza corrente

```
public partial class MyWindow: Window {

  public static readonly DependencyProperty SizeProperty =
    DependencyProperty.Register(              "Size",
                                              typeof(double),
                                              typeof(MyWindow),
                                              new UIPropertyMetadata(3));


  public double Size {
    get { return (double) GetValue(SizeProperty); }
    set { SetValue(SizeProperty, value); }
  }
  //…

}
```

Registrarsi al cambiamento

```
DependencyPropertyDescriptor sizeDescr =
    DependencyPropertyDescriptor.FromProperty(
   MyWindow.SizeProperty, typeof(MyWindow));

if (sizeDescr!= null)
{
    sizeDescr.AddValueChanged(this,
    delegate
    {
        // Add your propery changed logic here...
    });
}
```

Proprietà aggiunte

- Spesso gli oggetti contenitori hanno bisogno di dati appartenenti agli oggetti contenuti per definire il proprio comportamento
  - Un oggetto contenitore può possedere le definizioni delle proprietà di un altro oggetto in esso contenuto

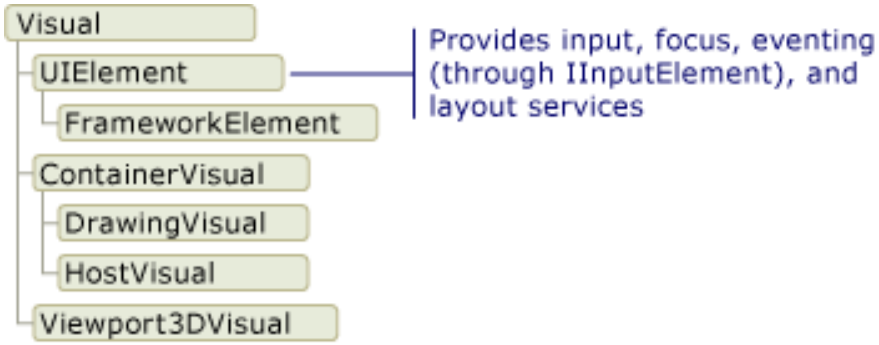
```
<Canvas>
    <Button Canvas.Top="20" Canvas.Left="20"
Content="Ok"/>
</Canvas>
```

Visualizzazione di oggetti

- Un oggetto graficamente rappresentabile deve essere istanza di una sottoclasse di System.Windows.Media.Visual
- Base per implementare nuovi controlli
  - Paragonabile a WindowHandle in Win32
- Introduce un meccanismo di “caching” delle istruzioni di ridisegno per massimizzare le prestazioni

Classe Visual

- Offre supporto per:
  - Output display
  - Trasformazioni
  - Clipping
  - Hit test
  - Calcolo dei margini
- Non offre supporto per:
  - Event handling
  - Layout e stili
  - Data binding



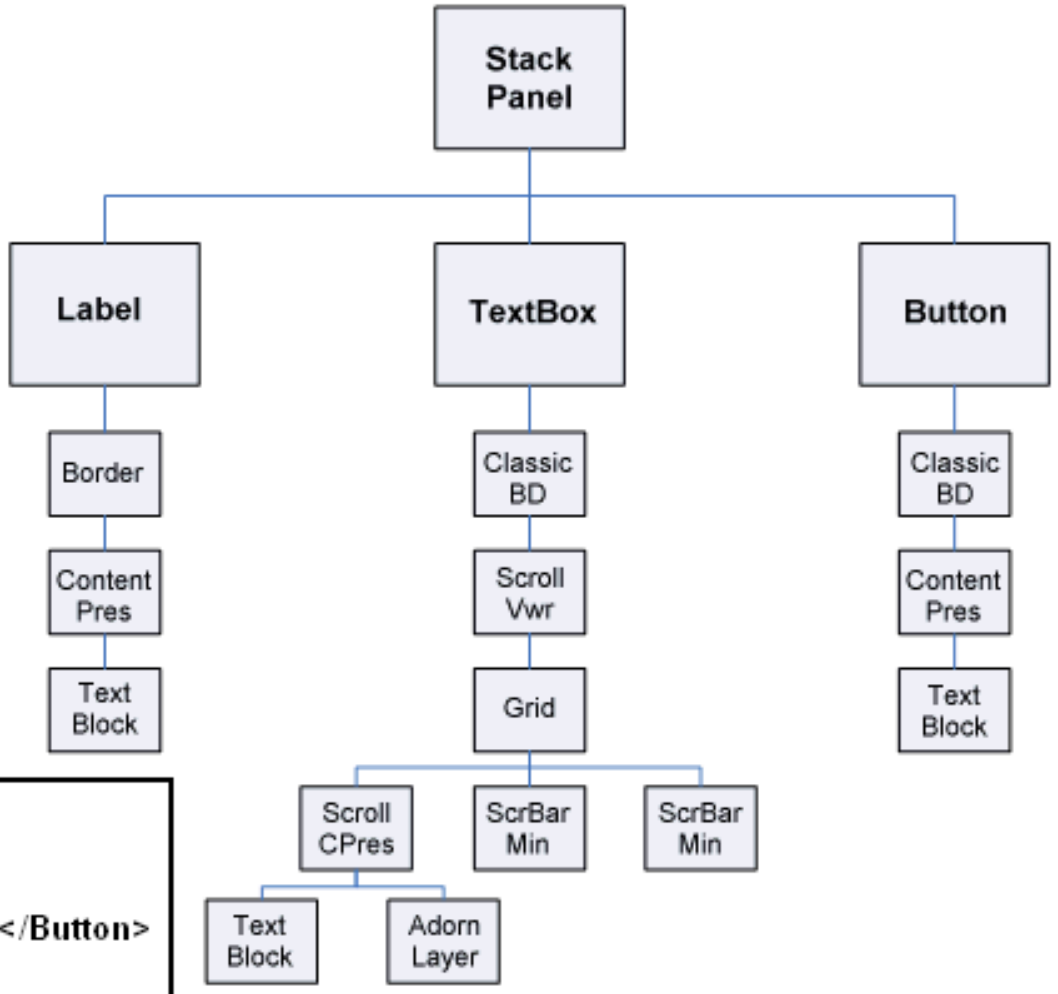
Gerarchie di elementi grafici

- La classe Visual consente di organizzare gli oggetti da visualizzare in una struttura ad albero
  - Ciascun oggetto può contenere le istruzioni e i metadati circa la propria visualizzazione
- Ciascun elemento Visual comunica con gli altri dello stesso albero tramite un protocollo specifico
  - Basato sullo scambio di messaggi

Gerarchie di elementi grafici

- La gerarchia tra elementi grafici può essere considerata sotto due punti di vista:
  - “Visual Tree”: gerarchia degli elementi visualizzati su schermo
  - “Logical Tree”: rappresenta la gerarchia degli elementi dell’applicazione in fase di esecuzione
    - Normalmente non serve manipolarlo direttamente
    - Può rappresentare elementi non strettamente visuali
      - Ad es: “ListItem”
    - Concetto utile per comprendere meccanismi di ereditarietà e gestione degli eventi

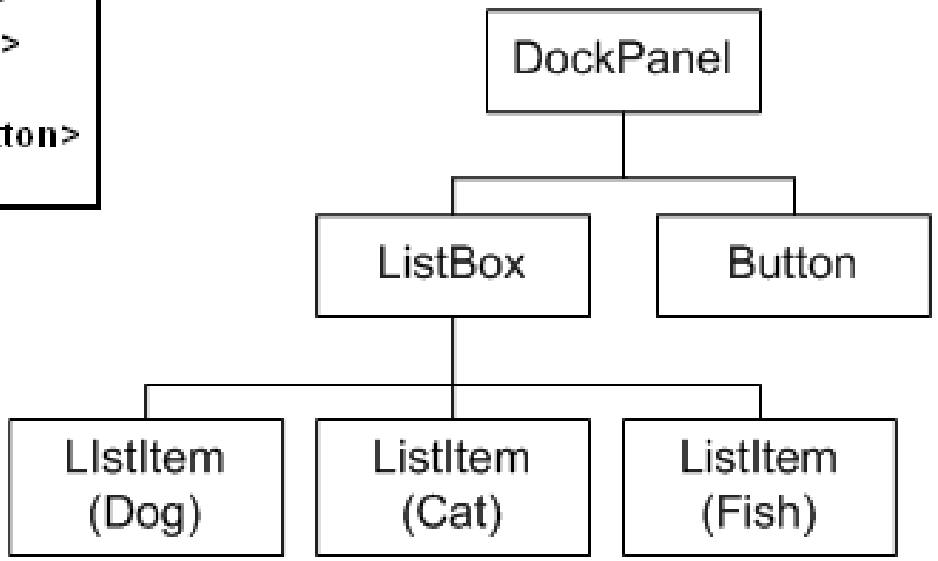
Esempio: Visual Tree




```
<StackPanel>
  <Label>User name:</Label>
  <TextBox />
  <Button Click="OnClick">OK</Button>
</StackPanel>
```

Esempio: Logical Tree

```
<DockPanel>
  <ListBox>
    <ListBoxItem>Dog</ListBoxItem>
    <ListBoxItem>Cat</ListBoxItem>
    <ListBoxItem>Fish</ListBoxItem>
  </ListBox>
  <Button Click="OnClick">OK</Button>
</DockPanel>
```



VisualTreeHelper

- Classe helper statica, deriva direttamente da Object
- Fornisce funzionalità di basso livello
- Utile in casi molto specifici
  - Es. sviluppo di controlli personalizzati
- Consente di esplorare l'albero visuale degli elementi
- Fornisce metodi per effettuare "hit test"
  - Controllare che un determinato punto appartenga a un certo elemento grafico

Classe DrawingContext

- Permette di popolare un Visual con un contenuto grafico vero e proprio
- Tramite una serie di metodi:
  - DrawLine(), DrawImage(), DrawText(), DrawVideo(), …
  - Non disegna in realtime ma memorizza un insieme di informazioni sul disegno utilizzate in seguito
- Non viene instanziato direttamente ma viene acquisito
  - Metodo Open() di DrawingGroup
  - Metodo RenderOpen() di DrawingVisual

Esempio

```
// Crea un oggetto DrawingVisual che conterrà un rettangolo
private DrawingVisual CreateDrawingVisualRectangle(Point p, Size s) {
  DrawingVisual drawingVisual = new DrawingVisual();
  // Si richiede il DrawingContext associato
  DrawingContext drawingContext = drawingVisual.RenderOpen();
  // Crea un rettangolo e lo inserisce nel DrawingContext.
  Rect rect = new Rect(p, s);
  drawingContext.DrawRectangle(Brushes.LightBlue, (Pen)null, rect);
  //Affida il contenuto del DrawingContext al sotto-sistema grafico
  drawingContext.Close();
  return drawingVisual;
}
```

Layout

- Gli elementi grafici dotati di layout sono quelli derivati da UIElement
- La posizione e dimensione di ciascun oggetto viene determinata in due fasi
  - Measure: l'elemento dichiara la dimensione ottimale al contenitore
  - Arrange: il contenitore ricalcola la posizione e la dimensione degli elementi contenuti secondo i propri criteri

Input ed eventi

- Gli input generati dalle periferiche vengono instradati attraverso il kernel di Windows e il sistema User32 fino al processo e al thread opportuno
- Un messaggio User32 viene convertito da WPF secondo il proprio formato e inoltrato al Dispatcher

Routing degli eventi

- Ciascun input è convertito da WPF in almeno due eventi
  - Evento “preview”
  - Evento “effettivo”
- Ad ogni evento è associato il concetto di routing
  - Viene instradato attraverso l’albero logico degli elementi
  - Due possibili direzioni

Eventi preview

- Un evento preview viene generato dall'elemento root e percorre verso il basso l'albero fino all'elemento target
  - Operazione detta "tunnel", consente a tutti gli elementi intermedi dell'albero di filtrare o reagire all'evento

Eventi effettivi

- Un evento effettivo viene inoltrato dall'elemento che lo ha generato all'elemento root
  - Generato dopo la ricezione di un evento "preview"
  - Operazione detta "bubble"

Binding dei comandi

- Consente a un elemento di definire un'associazione tra un input e un comando da eseguire
  - Es: tasti di scelta rapida
- Così come il Layout, la gestione degli eventi e il binding dei comandi sono supportati dalle sottoclassi di
  - System.Windows.UIElement

FrameworkElement

- Gli elementi che discendono da questa classe possono essere visti sotto due punti di vista:
  - Come ulteriore specializzazione di UIElement
    - Regole più stringenti per oggetti visivi con un layout consistente
  - Come base per un insieme di ulteriori sottosistemi di elementi grafici

FrameworkElement

- Fornisce un supporto per la creazione di animazioni
  - Semplicemente specificandone le proprietà
- Introduce i concetti di data binding
  - Simile a quello dei Windows Forms
- e di stile
  - Una forma di data binding per definire l'aspetto di uno o più elementi

Templating

- Concetto introdotto dalla classe System.Windows.Controls.Control
- Consente di stabilire in modo dichiarativo il rendering di un oggetto specificando alcuni parametri
  - Background
  - Foreground
  - Padding
  - ...

Controlli

- WPF consente la personalizzazione totale del comportamento e dell'aspetto di ciascun controllo
  - Ad esempio ad un bottone si può associare un contenuto di tipo stringa...
  - ...ma anche un elemento (o albero di elementi) "disegnabile"
    - Ossia, a cui è associato un Template

Contenitori

- WrapPanel
  - Gli elementi vengono posti uno dopo l'altro, andando a capo se necessario
- DockPanel
  - Disposizione del contenuto seguendo un layout basato sui bordi e sul centro
- StackPanel
  - Lo spazio è diviso in “strisce” orizzontali o verticali: ogni componente ne occupa una
- Canvas
  - Posizionamento assoluto dei componenti
- Grid
  - Suddivisione dello spazio in celle, di dimensioni variabili

Contenitori