

► Introduction à l'interopérabilité Utilisation de dll natives en .net

Deux mots sur .net et C++ managé

❖ .net est un framework qui gère

- L'exécution de l'application par le Common Language Runtime (CLR)
 - implémentation de Microsoft de la spécification CLI (common Language Infrastructure) pour exécuter du code CIL (Common Intermediate Language) de façon managé (compilation à la volée : « just in time »)
- Exemple de langage supporté C++/CLI, C#, ...

❖ Le C++ managé (géré par le CLR => garbage collector, etc.)

▪ Classe

- `ref class X { ... }`

▪ Objet

- `String^ s = gcnew String(« bonjour »); // Pas besoin de delete`

▪ Les Handles

- ⇔ à une référence sur un objet managé : `^`

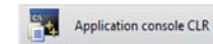
❖ Mixer du code natif C++ avec du code managé (.net)

- Si on dispose des sources natives C++
 - Mixer directement les sources natives C++ avec le C++/CLI.
- Si on veut intégrer une DLL native C++, sans la modifier
 - Création d'un wrapper en C++/CLI
 - Le Wrapper est utilisable dans n'importe quel langage .Net
 - Interface unique en .net d'accès à la bibliothèque
- Utilisation d'outils pour automatiser la création du Wrapper
 - Swig (<http://www.swig.org>).
- Une alternative avec la technologie COM
 - COM interop, le wrapper est généré automatiquement.

❖ Mise en œuvre simple : mécanisme IJW (It Just Works)

- Intégration des sources de classes C++ / méthodes C dans une même application .Net développée en C++/CLI
 - crée d'une application CLR console (C++ managé)
 - en mode de compilation mixte (/clr)
 - chaque transition entre le monde managé et non managé est coûteuse.
 - => limiter les transitions

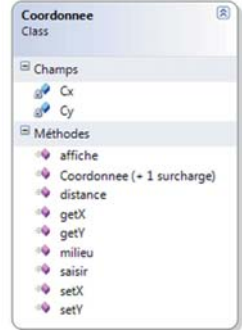
```
class Coordonnee // code C++ « Coordonnee.h"
{
public:
    Coordonnee(int x, int y);
    Coordonnee();
    void setX(int x);
    void setY(int y);
    double getX();
    double getY();
    double distance(const Coordonnee &P);
    Coordonnee milieu(const Coordonnee &P);
    void saisir();
    void affiche();
private:
    int Cx,Cy;
};
```



```
// AppCLR_consol.cpp : fichier projet principal.

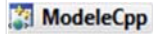
#include "stdafx.h"
#include "Coordonnee.h"
using namespace System;

int main(array<System::String ^> ^args)
{
    Coordonnee c1(10,10);
    c1.affiche();
    return 0;
}
```



❖ Création d'une DLL C++

- Choisir « DLL » comme type d'application
- Ajouter les entêtes des fonctions/classes exportées
 - L'utilisation de « extern "C" » permet d'avoir un nom de fonction non décoré (cf. utilitaire **dumpbin.exe**)
- Après compilation et édition de liens, on obtient :
 - un fichier ModeleCpp.lib et un fichier ModeleCpp.dll



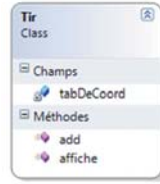
```
#include <vector>
#include <iostream>
#include "Coordonnee.h"
using namespace std;

class __declspec(dllexport) Tir {
private:
    vector<Coordonnee> tabDeCoord;
public:
    void add(int x,int y);
    void affiche();
};

extern "C" __declspec(dllexport) Tir* Tir_New();
extern "C" __declspec(dllexport) void Tir_Delete(Tir* tdC);
```

```
#include "Tir.h"
void Tir::add(int x,int y) { tabDeCoord.push_back(Coordonnee(x,y)); }
void Tir::affiche() {
    vector<Coordonnee>::iterator i;
    for(i=tabDeCoord.begin(); i!=tabDeCoord.end(); i++) {
        ((*i).affiche());
        cout << "|";
    }
    cout << endl;
}

extern "C" __declspec(dllexport) Tir* Tir_New(){ return new Tir();}
extern "C" __declspec(dllexport) void Tir_Delete( Tir* tdC){ delete tdC; }
```



eric.anquetil@insa.fr

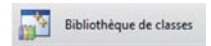
❖ Objectif

- Utiliser des méthodes depuis des dll existantes (C++).
 - ne pas toucher aux bibliothèques existantes
- Créer une assembly en C++/CLI qui englobe la bibliothèque existante
 - le C++/CLI est le langage idéal pour gérer l'interopérabilité
- Wrapper utilisable dans n'importe quel langage .Net
 - interface unique en .net d'accès à la bibliothèque

❖ Comment wrapper la classe *Tir* ?

Créer une interface d'utilisation pour la classe *Tir* de notre DLL C++ (ModeleCpp.dll) en C++ managé (CLR)

- Créer un projet CLR class library (mWrapper.dll)
 - L'assembly wrapper = Classe *WrapperTir*
 - Inclure le .h et le .lib de la DLL dans la classe wrapper (*WrapperTir*)
- La classe wrapper doit être publique
 - pour que les autres langages .net puissent l'utiliser



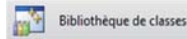
eric.anquetil@insa.fr

Création de la classe Wrapper (*WrapperTir*) en C++ managé

❖ Classe *WrapperTir*

- Un pointeur (membre privé) vers notre objet natif (*Tir* tDCw*).
- Un constructeur : sert à appeler le constructeur de *Tir* dans notre dll.
- Un destructeur et un finalizer : sert à appeler le destructeur dans notre dll.
 - C'est le garbage collector qui s'occupera d'appeler le finalizer ou le destructeur, et permettra ainsi la libération des ressources utilisées.
- Méthodes *add* et *affiche* qui appellent les méthodes de notre classe.

```
// mWrapper.h
#include "Tir.h"
#pragma comment (Lib, "ModeLeCpp.Lib")
using namespace System;
namespace mWrapper {
public ref class WrapperTir {
private:
    Tir* tDCw;
public:
    WrapperTir(){ tDCw=Tir_New(); }
    ~WrapperTir(){ Tir_Delete(tDCw); }
    System::Void add(int x,int y){ tDCw->add(x,y); }
    System::Void affiche(){ tDCw->affiche();}
protected:
    !WrapperTir(){ Tir_Delete(tDCw);}
};
}
```



Bibliothèque de classes

eric.anquell@insa.fr

Utilisation de la classe Wrapper (*WrapperTir*) en C++ managé pure

❖ Création d'un projet de test : *TestWrapperCLR*

- De type: Application console CLR
 - Prise en charge du Common Language Runtime MSIL pure (/clr:pure)
- Référencer l'assembly
 - Propriété → propriété commune → structure et référence →



Application console CLR

```
// TestWrapperCLR.cpp : fichier projet principal.
```

```
#include "stdafx.h"
using namespace System;
using namespace mWrapper;

int main(array<System::String ^> ^args)
{
    WrapperTir^ mtir = gcnew WrapperTir();
    mtir->add(10, 10);
    mtir->affiche();
    return 0;
}
```

eric.anquell@insa.fr

Utilisation de la classe Wrapper (*WrapperTir*) en C#

❖ Création d'un projet de test : *TestWrapperCS*

- De type Application console c#
- Référencer l'assembly
 - Ajouter une référence à mWrapper
 - Il faut que la dll soit accessible à l'exécution du programme

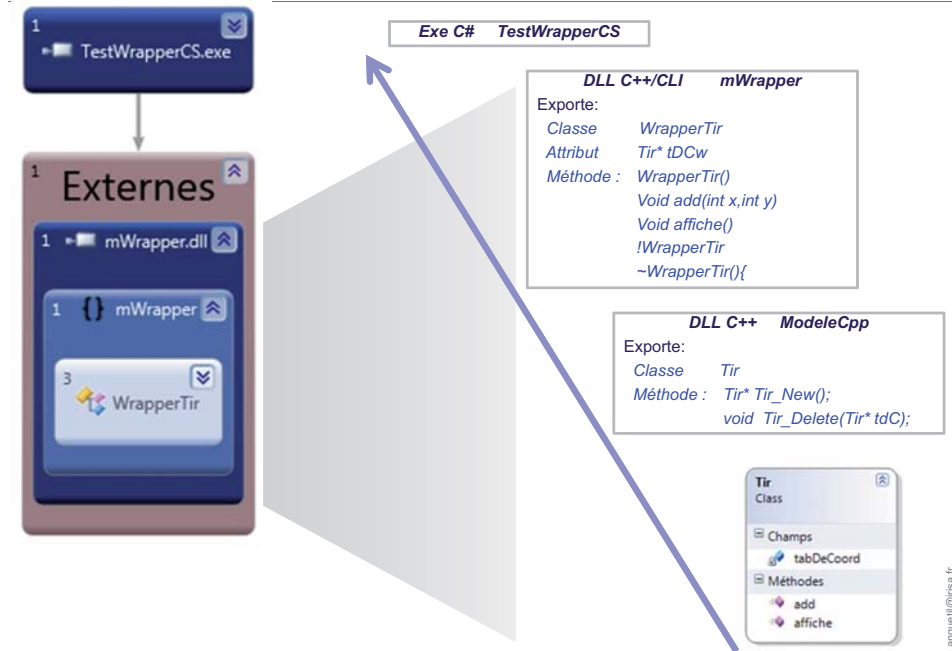


```
// Program.cs.
using System;
using mWrapper;

namespace TestWrapperCS
{
    class Program
    {
        static void Main(string[] args)
        {
            WrapperTir mtir = new WrapperTir();
            mtir.add(10, 10);
            mtir.affiche();
        }
    }
}
```

eric.anquell@insa.fr

Récapitulatif



eric.anquell@insa.fr

Notion sur le Marshalling (conversion de type)

276

❖ Gérer la conversion des types entre le monde managé et non managé

- Pas besoin de conversion pour certains types
 - Types *blittables*
 - types qui ont une représentation commune à la fois dans le monde managé et natif. (Byte, Int16, IntPtr, etc ...)
- Pour les autres, il faut faire la conversion

❖ Exemple

- convertir une `String` ^ en `string` de la STL ?

```
static string convertStringToStlString (String^ chaine)
{
    char * chaineChar;
    pin_ptr<const wchar_t> wch = PtrToStringChars(chaine);
    int taille = (chaine->Length+1) * 2;
    chaineChar = new char[taille];
    int t = WideCharToMultiByte(CP_ACP, 0, wch, taille, NULL, 0, NULL, NULL);
    WideCharToMultiByte(CP_ACP, 0, wch, taille, chaineChar, t, NULL, NULL);
    std::string chaineSTL = chaineChar;
    delete chaineChar;
    return chaineSTL;
}
```

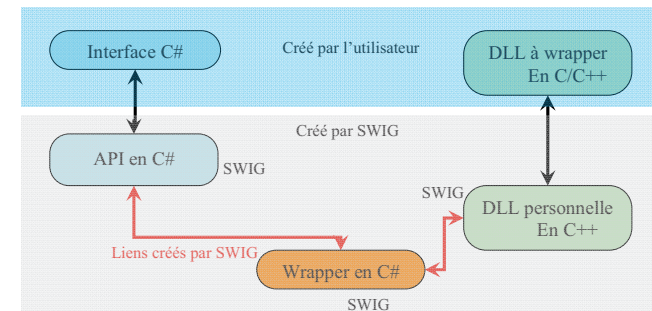
eric.anquell@insa.fr

Outils pour automatiser la création du Wrapper (SWIG)

277

❖ SWIG (Simplified Wrapper and Interface Generator)

<http://www.swig.org/>



- SWIG utilise un *fichier interface* pour définir
 - les fonctions qui seront wrappées
 - les différentes informations nécessaires comme les types, les structures, etc.
 - Gestion du Marshalling/conversion (de la STL ...)
- NB: le Wrapper sera généré en C#

eric.anquell@insa.fr

► Introduction à WPF Windows Presentation Foudation

Présentation de Wpf (Windows Presentation Foundation)

279

❖ Technologie de développement d'interface utilisateur

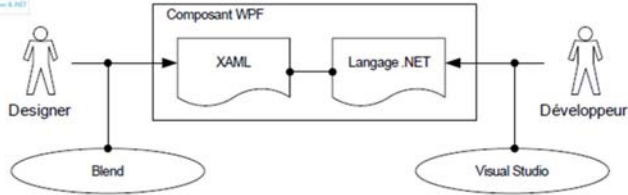
- **Application riche pour Windows**
 - Orientée Objet et basée sur la plateforme .NET
- **Cible aussi les navigateurs internet**
 - XBAP (Xaml Browser Application) → Windows
 - Silverlight → multiplateforme

❖ Caractéristiques

- **Unifier les techniques de développement d'interfaces**
 - Affichage vectoriel, mise à l'échelle sans pixellisation...
- **Composition graphiques vectorielles 2D et 3D**
 - Utilise les processeurs graphiques GPU, basé sur Direct3D (DirectX)
- **Images, animations, flux multimédia, audio et vidéo, document texte...**

❖ Inconvénients : manque encore d'interopérabilité

- **Dépend du portage de .NET sur les autres systèmes: mono**
 - mono ne supporte actuellement pas le WPF...
- **Solution émergente Silverlight, WPF pour les applications web .NET**
 - portage réalisé : Moonlight...



❖ Principe

- **WPF Sépare le code behind (développeur) / du code designer (infographiste)**
 - Fonctionnalités codées en .NET (C#...)
 - Outils : Visual Studio ou ...
 - Présentation Visuel décrite en XAML (eXtensible Application Markup Language)
 - Outils : Expression Blend, Visual Studio, XamlPad.exe, ...
- **Design de l'application**
 - Via un langage commun basé sur du XML qui est le XAML

eric.anquetil@insa.fr



❖ Langage déclaratif basé sur la syntaxe du XML

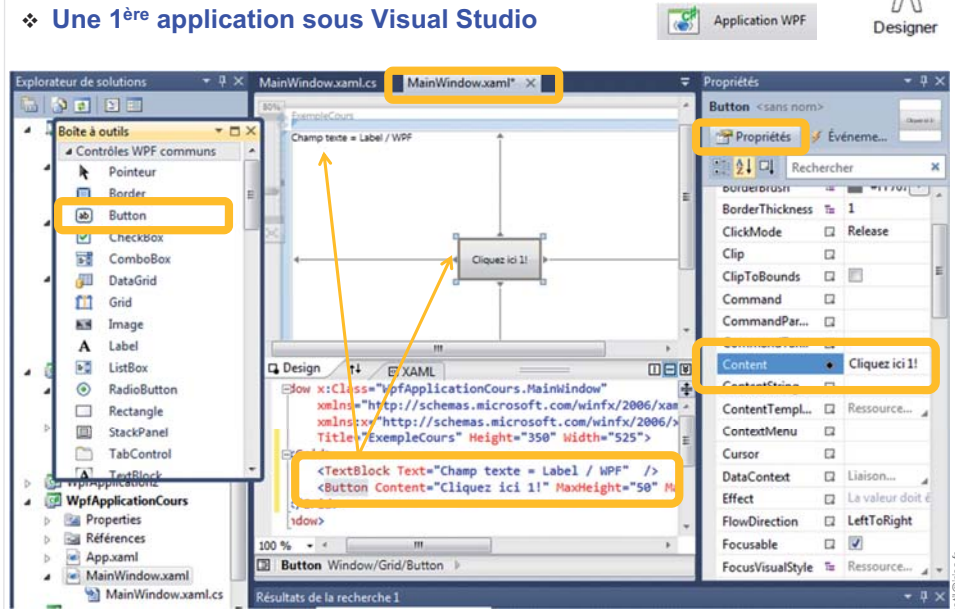
- Permet de déclarer et définir des objets dynamiquement grâce à
 - des balises (équivalent des classes)
 - et aux attributs (équivalents aux propriétés).
- **Restitution de graphiques vectoriels, 2D ou 3D...**
 - Respecter la casse
 - Les balises ouvertes doivent être refermées sans se chevaucher
 - Chaque attribut doit avoir une valeur inscrite entre " ou '
- **Exemple**

```
<Window x:Class="WpfApplicationCours.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ExempleCours" Height="350" Width="525">
  <Grid>
    <TextBlock Text="Champ texte = Label / WPF" />
    <Button Content="Cliquez ici !" MaxHeight="50" MaxWidth="100" />
  </Grid>
</Window>
```

eric.anquetil@insa.fr

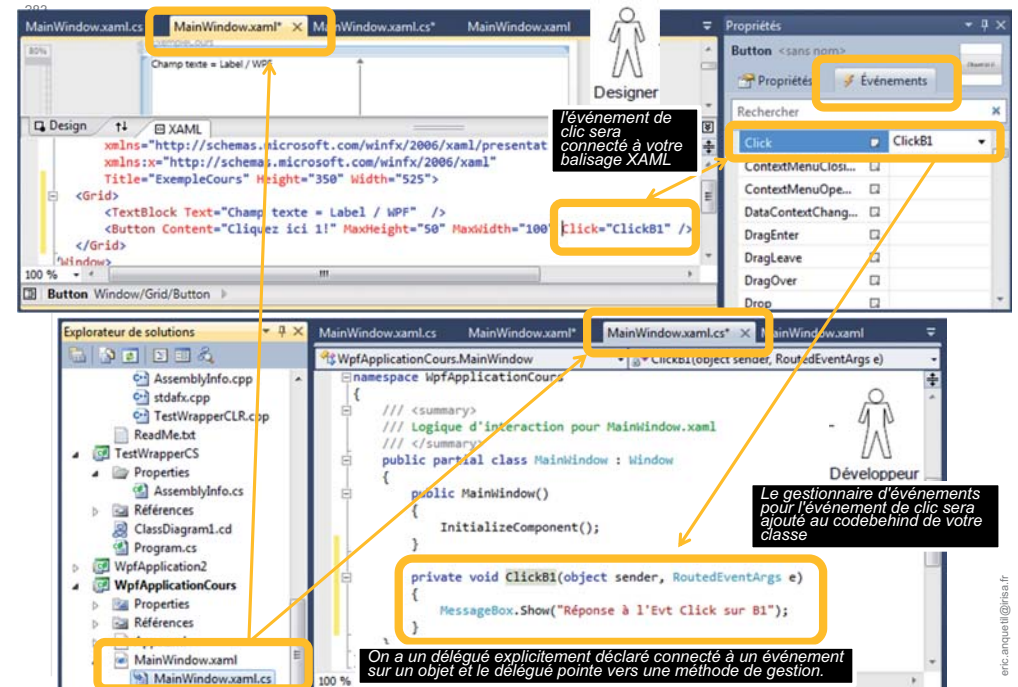
Visual Studio : 1^{ère} application wpf / Xaml

❖ Une 1^{ère} application sous Visual Studio



eric.arqueill@insa.fr

Visual Studio : Xaml/c# - Réalisation d'une action



eric.arqueill@insa.fr

WPF : Disposition des contrôles WPF dans une fenêtre

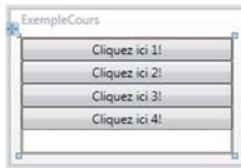
284

❖ Les différents Panels

▪ Le contrôle StackPanel.

- Présente les éléments en les empilant

```
<StackPanel>
  <Button Content="Cliquez ici 1!" />
  <Button Content="Cliquez ici 2!" />
  <Button Content="Cliquez ici 3!" />
  <Button Content="Cliquez ici 4!" />
</StackPanel>
```



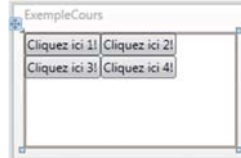
▪ Le contrôle WrapPanel.

- les éléments vont être positionnés les uns après les autres

▪ Le contrôle DockPanel.

- permet de « docker » sur ses bordures, les éléments qui le composent.

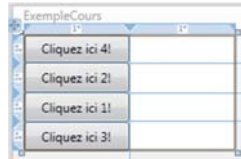
```
<DockPanel>
  <Button DockPanel.Dock="Bottom" Content="Cliquez ici 1!" />
  <Button DockPanel.Dock="Top" Content="Cliquez ici 2!" />
  <Button Content="Cliquez ici 3!" />
  <Button Content="Cliquez ici 4!" />
</DockPanel>
```



▪ Les contrôles Grid et GridSplitter.

- dispose les éléments dans une grille
- GridSplitter : permet en plus de choisir la taille d'une cellule

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button Grid.Row="2" Content="Cliquez ici 1!" />
  <Button Grid.Row="1" Content="Cliquez ici 2!" />
  <Button Grid.Row="3" Content="Cliquez ici 3!" />
  <Button Grid.Row="0" Content="Cliquez ici 4!" />
</Grid>
```



eric.anquetil@insa.fr

WPF: le contrôle Canvas / un petit exemple

285



❖ Le contrôle Canvas

- Permet de positionner un *contrôle* grâce à des coordonnées X et Y en DIP (Device Independent Pixel)

❖ Le code designer Xaml

```
<Window x:Class="WpfJeu.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Canon Noir" Height="650" Width="840">
  <Canvas Name="canvas1" MouseLeftButtonDown="MouseDown">
    <Image Margin="10,10,10,10" Width="800" Stretch="UniformToFill" Name="pPlateau"
      Source="/WpfJeu;component/Resources/pPlateau.jpg" />
    <Button Canvas.Top="70" Canvas.Left="250" Name="Canon" Click="ClickC">
      <StackPanel Orientation="Vertical">
        <Image Height="43" Source="/WpfJeu;component/Resources/canon.jpg" Width="47" />
        <TextBlock Text="Canon" HorizontalAlignment="Center" FontSize="10" />
      </StackPanel>
    </Button>
  </Canvas>
</Window>
```



eric.anquetil@insa.fr

WPF: un petit exemple

❖ Le code behind

```
namespace WpfJeu
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow() {
            InitializeComponent();
        }
        private void ClickC(object sender, RoutedEventArgs e) {
            // Affichage d'une MessageBox avec Les coordonnées du bouton Canon
            MessageBox.Show("BOUMM ! (" + Canvas.GetLeft(Canon) + "," + Canvas.GetTop(Canon) + ")");
        }
        private void MouseDown(object sender, MouseButtonEventArgs e) {
            // On récupère La position du MouseDown
            Point pt = e.GetPosition(canvas1);
            // On déplace Le Bouton à ces coordonnées
            Canvas.SetLeft(Canon, pt.X);
            Canvas.SetTop(Canon, pt.Y);
        }
    }
}
```



eric.arqueill@insa.fr

WPF: liaison avec une DLL C++

❖ Liaison à la DLL mWrapper

```
using mWrapper;
namespace WpfJeu
{
    public partial class MainWindow : Window {
        WrapperTir tDC;
        public MainWindow() {
            InitializeComponent();
            // Création de notre Objet Tir via Le Wrapper
            tDC = new WrapperTir();
        }
        private void ClickC(object sender, RoutedEventArgs e) {
            // Affichage d'une MessageBox avec Les coordonnées du bouton Canon
            MessageBox.Show("BOUMM ! (" + Canvas.GetLeft(Canon) + "," + Canvas.GetTop(Canon) + ")");
            // affichage dans La console des données enregistrées
            tDC.affiche();
        }
        private void MouseDown(object sender, MouseButtonEventArgs e) {
            // On récupère La position du MouseDown
            Point pt = e.GetPosition(canvas1);
            // On ajoute ces coordonnées dans notre structure (DLL C++)
            tDC.add((int)pt.X, (int)pt.Y);
            // On déplace Le Bouton à ces coordonnées
            Canvas.SetLeft(Canon, pt.X);
            Canvas.SetTop(Canon, pt.Y);
        }
    }
}
```



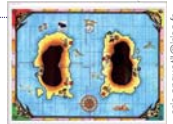
eric.arqueill@insa.fr

WPF: Une 1^{ère} idée sur le data binding

❖ But du data binding

- permettre de lier un *contrôle* avec une ou plusieurs *sources* de données.
 - le *contrôle* va définir le binding = *la cible*
 - la *source* de données = *la source*
- Plusieurs configurations possibles pour la mise à jour automatique
 - méthodes dites OneWay, OneTime, TwoWay...
 - mettre à jour le contrôle et la source dès qu'un des deux est modifié, ou seulement un seul...

```
<Canvas Name="canvas1" MouseLeftButtonDown="MouseDown">
  <TextBox Text="{Binding ElementName=Canon, Path=IsMouseOver, Mode=OneWay}" IsReadOnly="True" />
  <Image Margin="20,30,10,10" Width="800" Stretch="UniformToFill" Name="plateau"
    Source="/WpfJeu;component/Resources/plateau.jpg" />
  <Button Canvas.Top="70" Canvas.Left="250" Name="Canon" Click="ClickC">
    <StackPanel Orientation="Vertical">
      <Image Height="43" Source="/WpfJeu;component/Resources/canon.jpg" Width="47" />
      <TextBlock Text="Canon" HorizontalAlignment="Center" FontSize="10" />
    </StackPanel>
  </Button>
</Canvas>
```



❖ C'est un vaste sujet qu'il faut approfondir ...

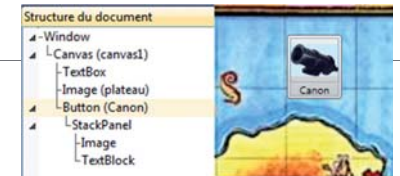
- Cf. <http://www.dotnet-france.com/Cours/70-502.html>



Aperçu des événements routés

❖ Arborescences d'éléments WPF

- Arborecence d'éléments logiques et visuels (souvent bien plus compliquée)



❖ Routage d'événements

```
private void ClickC(object sender, RoutedEventArgs e) {
```


- Stratégies de routage à partir de l'arborescence visuelle.

- Les événements bulle (Buddling)
 - Un événement monte le long de l'arborescence visuelle à partir de l'élément source tant qu'il n'est pas traité ou n'a pas atteint la racine.
 - Le nom des événements Bubble indique simplement leur action (MouseDown).
- Les événements tunnel (Tunneling)
 - Procèdent dans le sens inverse, depuis l'élément racine jusqu'à ce qu'ils soient traités ou jusqu'à ce qu'ils atteignent l'élément source.
 - Permet à des éléments en amont d'intercepter l'événement et de le traiter avant qu'il n'atteigne l'élément source.
 - Le nom des événements tunnel comprend le préfixe Preview (PreviewMouseDown).
- Les événements directs
 - Le seul gestionnaire pour l'événement est un délégué connecté à l'événement.

❖ Illustration du routage d'évènements

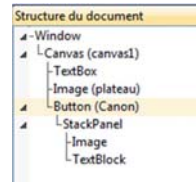
```
private void MouseRightDown_canvas(object sender, MouseButtonEventArgs e)
{
    // Affichage du routage provenant du Canvas
    routage.Items.Add("MouseRD_canvas");
}
```

```
private void MouseRightDown_Button(object sender, MouseButtonEventArgs e) {
    // Affichage du routage provenant du Button
    routage.Items.Add("MouseRD_Button");
}
private void MouseRightDown_Image(object sender, MouseButtonEventArgs e) {
    // Affichage du routage provenant du Image
    routage.Items.Add("MouseRD_Image");
} ...
```



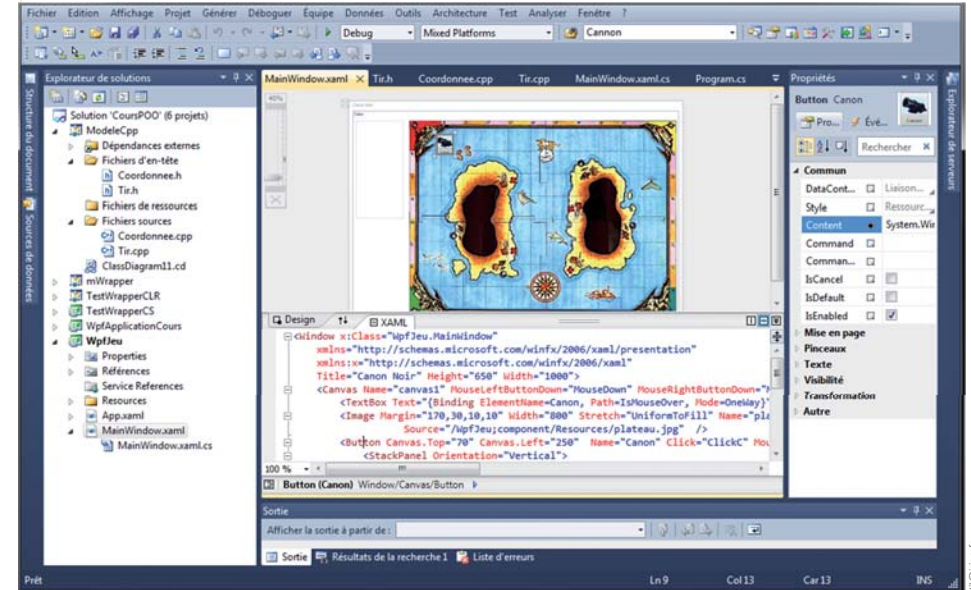
Drivvel

```
<Canvas Name="canvas1" MouseLeftButtonDown="MouseDown"
    MouseRightButtonDown="MouseRightDown_canvas"
    PreviewMouseRightButtonDown="PreviewMouseRightDown_canvas">
    <TextBox Text="(Binding ElementName=Canon, Path=IsMouseOver, Mode=OneWay).IsReadOnly=True" />
    <Image Margin="170,30,10,10" Width="800" Stretch="UniformToFill" Name="plateau"
        Source="/WpfJeu;component/Resources/plateau.jpg" />
    <Button Canvas.Top="70" Canvas.Left="250" Name="Canon" Click="ClickC"
        MouseRightButtonDown="MouseRightDown_Button"
        PreviewMouseRightButtonDown="PreviewMouseRightDown_Button">
        <StackPanel Orientation="Vertical">
            <Image Height="43" Source="/WpfJeu;component/Resources/canon.jpg" Width="47"
                MouseRightButtonDown="MouseRightDown_Image"
                PreviewMouseRightButtonDown="PreviewMouseRightDown_Image" />
            <TextBlock Text="Canon" HorizontalAlignment="Center" FontSize="10" />
        </StackPanel>
    </Button>
    <ListBox Canvas.Left="10" Canvas.Top="30" Height="293" Name="routage" Width="143" />
</Canvas>
```



eric anandtil@icloud.com

Visual Studio : Solution complète (cf. code)



11. anupama@iitb.ac.in