Microsoft .net

Services réseaux

COURS N°4 – TP N°3 : MVVM
BINDING – EVENTS - COMMANDS

Antony BRUGERE
a.brugere@cs3i.fr

Frédéric CHASSAGNE
frederic.chassagne@capgemini.com

Plan du cours

-
- User Control
- Pattern Model – View – ViewModel
- Binding en WPF
- Events et Command en WPF

User Control

- Contrôle personnalisé
  - Regrouper 1 à n contrôles
  - Définir un comportement pour le groupe de contrôles
  - Dérive de la classe UserControl
- But :
  - Réutilisation de composants
  - TPN°3 : PlanningElementView sera un UserControl

Model – View - ViewModel

- Design Pattern de Présentation
- Similaire au Model - View - Controler
- Adapté aux technologies modernes
- But :
  - Découpler la réalisation de la Vue pour la confier à un Designer
  - Permettre la validation de la PL par des tests unitaires

Model – View - ViewModel

- Model
  - Données métier à manipuler
  - Aucune représentation graphique précise
- Model = Liste d'objets ou d'interfaces devant être affichés dans la vue

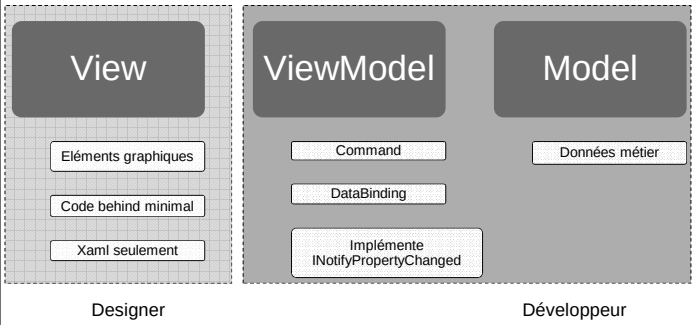
Model – View - ViewModel

- View
  - Ensemble d'éléments graphiques (boutons, fenêtre, listes déroulantes, …)
  - Prend en charge les raccourcis clavier
  - Les contrôles contenu gèrent l'interaction avec les périphériques de capture
  - Contrôles regroupés dans un UserControl
- Dans les cas simple, vue directement liée au modèle. Sinon, utilisation d'un ViewModel

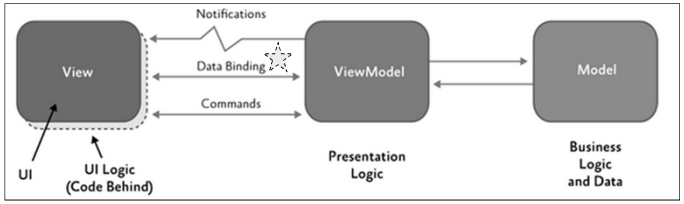
Model – View - ViewModel

- ViewModel = Adaptateur entre Vue et Modèle
  - Adaptation des types du Modèle en type exploitable via databinding par la Vue
  - Exposition de Commandes pour l'interaction entre Vue et Modèle
  - Peut avoir un seul accesseur (binding oneWay) ou 2 (binding TwoWay)
  - Nécessité de l'implémentation de INotifyPropertyChanged
- Elément clé du pattern MVVM, permet d'avoir l'architecture découplée

M-V-VM : Synthèse



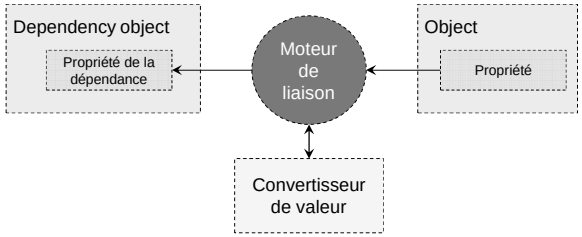
M-V-MV : Interactions



Binding

- Binding = liaison de données entre
  - Une cible
  - Un DependencyProperty d'un DependencyObject
  - Une source de donnée (CLR, ADO.NET, Xml)
- La valeur peut être convertie par un convertisseur de valeur
- La liaison est effectuée par le moteur de liaison (binding engine)
- Liaison soit dans le code C#, soit dans le xaml

Binding



Exemple :
Textbox1.Text = Evenement.Name

détails sur http://bindings.wpf-france.fr

Binding - Exemple

Binding : liaisons simples

- Propriété **DataContext** de l'élément cible définit la source de données
- Liaisons simples

Liaisons simples	
{Binding}	Liaison sur le DataContext courant
{Binding Nom} ou {Binding Path=Nom}	Liaison sur la propriété « *Nom* » du DataContext courant
{Binding Nom.Propriete}	Liaison sur la propriété « *Propriete* » de la propriété « *Nom* » du DataContext courant
{Binding ElementName=unAutreControle, Path=Propriete }	Liaison sur la propriété « *Propriete* » de l'élément XAML ayant comme nom « *unAutreControle* »
{Binding Source=unObjetSource, Path=Propriete }	Liaison sur la propriété « *Propriete* » de l'objet source « *unObjetSource* »

- Liaison relatives (non abordées)
  - Élément précédents, courants, liaison sur ancêtre…

Binding : exemple de liaison

- Dans le fichier xaml

```
<ListView Height="234" Name="listViewLieu" Width="281" ItemsSource="{Binding }">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" DisplayMemberBinding="{Binding Path=Nom}"/>
            <GridViewColumn Header="Nb places" DisplayMemberBinding="{Binding Path=NombrePlacesTotal}"/>
        </GridView>
    </ListView.View>
</ListView>
```

  - Propriétés liées aux GridViewColumn
  - ItemsSource vide fait référence implicitement au DataContext

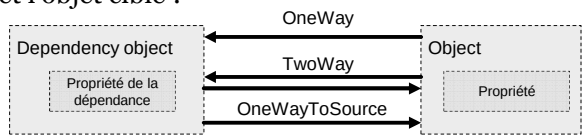
- Dans le fichier cs
  - Définition explicite du DataContext

```
public LieuxWindow(IEnumerable<Lieu> salleList)
{
    InitializeComponent();

    listViewLieu.DataContext = salleList;
}
```

Binding : Mode de liaison

- Dans quel sens est faite la liaison entre l'objet source et l'objet cible ?



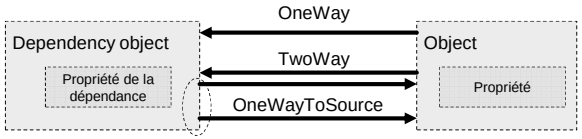
Binding : Mode de liaison

- Propriété **Mode** de l'objet Binding

Mode de liaison : Le mode de liaison définit dans quel sens est faite la liaison entre l'objet source et l'objet cible. Il est utilisé de la façon suivante : **{Binding …, Mode=XXX}** avec XXX prenant comme valeur :	
TwoWay	**La source et la cible reste synchronisées.** La source met à jour la cible lorsqu'elle change et inversement.
OneWay	Seules les **modifications sur la source sont répercutées sur la cible**. Les modifications de la cible ne changent pas les valeurs de la source.
OneWayToSource	C'est le fonctionnement inverse de la valeur précédente. Seules les **modifications sur la cible sont répercutées sur la source.**
OneTime	Les valeurs de la liaison ne sont fournies que de la source vers la cible et cela uniquement **lorsque l'application démarre ou à chaque changement du DataContext.**
Default	Le mode de liaison est définit par l'objet sur lequel il est appliqué.

Binding : Mode de liaison

- Lorsque la source doit être mise à jour comment cela fonctionne-t-il ?



Propriété UpdateSourceTrigger de la liaison détermine ce qui déclenche la mise à jour de la source.

Binding : Mode de liaison

- Dans le cas des liaisons TwoWay ou OneWayToSource

Événement déclencheur de la mise à jour de la liaison : Les liaisons de données ne sont pas totalement synchrones et ce sont certains événements qui déclenchent la synchronisation. On paramètre l'événement de la façon suivante : **{Binding …, UpdateSourceTrigger=XXX}** avec XXX prenant comme valeur :	
PropertyChanged	La mise à jour s'effectue immédiatement à chaque changement de valeur
LostFocus	La mise à jour s'effectue lorsque le contrôle où est faite la liaison perd le focus
Explicit	La mise à jour doit être faite explicitement sur l'objet BindingSource correspond à la liaison via la méthode *UpdateTarget*. Voici un exemple : `BindingExpression be = txtBx.GetBindingExpression(TextBox.TextProperty); be.UpdateTarget();`
Default	La cible de la liaison décide quand elle se met à jour

Model – View – ViewModel

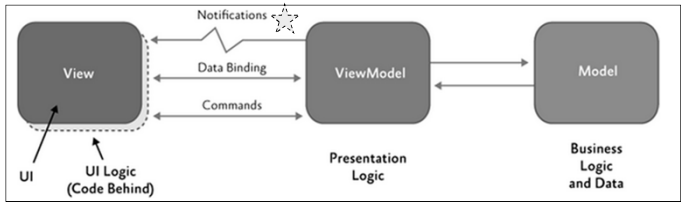
- DataBinding
  - Côté ViewModel : nécessité de l'implémentation de
    - INotifyPropertyChanged ou
    - INotifyCollectionChanged ou
    - ICollectionView (pour ObservableCollection<T>) sur la collection

```
class EvenementViewModel : ViewModelBase
{
    private readonly Evenement evenement;

    public string Titre
    {
        get { return this.evenement.Titre; }
        set
        {
            this.evenement.Titre = value;
            OnPropertyChanged("Titre");
        }
    }
```

  - L'événement levé dans le ViewModel → mise à jour de la View
  - ViewModel : ViewModelBase

M-V-MV : Interactions

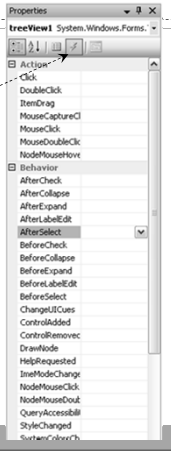


Events

- Event : Code déclenché par une action de l'utilisateur
- Exemple : Click() sur un bouton
- Déclaration en C#
  - Inscription :
    - this.button1.Click += this.button1_Click;
  - Désinscription :
    - this.button1.Click -= this.button1_Click;

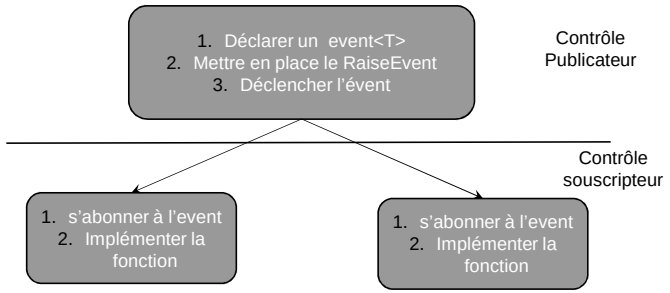
Events

- VS.Net
  - Panneau d'accès
  - Ajout automatique



Events

- Mécanisme = Mécanisme d'abonnement



Warning : Exécution synchrone !

Events – Exemple étape par étape

- Etape 1 : Définir quoi envoyer
  - Créer un objet
  - Le Suffixer EventArgs
  - Hériter de System.EventArgs
  - Définir les propriétés à afficher (accès en lecture seule)
  - Ajouter un constructeur avec l'ensemble des paramètres

```
class ScoreChangedEventArgs : EventArgs
  {
     private int _currentScore;

     public int CurrentScore
     {
        get { return _currentScore;}
     }

     public ScoreChangedEventArgs(int score)
     {
        _currentScore = score;
     }
  }
```

Events – Exemple étape par étape

- Etape 2 : Dans l'objet déclenchant
  - Définir l'event pour être visible de l'extérieur
    - Suffixer par ed = Non annulable
    - Suffixer par ing = Annulable (l'eventArgs hérite de CancelEventArgs au lieu de EventArgs)
  - Ecrire une méthode protected pour déclencher l'event
    - La préfixer On
  - Utiliser la méthode protected pour déclencher les events dans le code
    - Ne jamais déclencher directement un event

```
public class Game
{
    public event
EventHandler<ScoreChangedEventArgs>
CurrentScoreChanged;

    protected void OnCurrentScoreChanged
(ScoreChangedEventArgs e)
    {
        CurrentScoreChanged(this, e);
    }

    private int _score;

    public void DoSomething()
    {
        _score++;
        OnCurrentScoreChanged(
new ScoreChangedEventArgs(_score));
    }

    public Game()
    {
        _score = 0;
    }
}
```

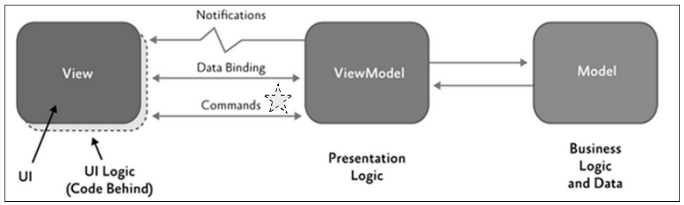
Events – Exemple étape par étape

- Etape 3 : Dans l'objet inscrit
  - S'abonner à l'event
  - Coder quoi faire lorsqu'il se déclenche
  - Se désabonner

```
public class GameViewer
{
    private Game _game;
    public GameViewer()
    {
        _game = new Game();
        _game.CurrentScoreChanged += CurrentScoreChangedNotify;
        _game.DoSomething();
        _game.CurrentScoreChanged -= CurrentScoreChangedNotify
    }

    private void CurrentScoreChangedNotify(object sender, ScoreChangedEventArgs e)
    {
        Console.WriteLine(e.Score);
    }
}
```

M-V-MV : Interactions



Commands

- Command = routage d'évènement
- Implémentation de ICommand

```
public interface ICommand {
   event EventHandler CanExecuteChanged;
   bool CanExecute(object parameter);
   void Execute(object parameter);
}
```

- Permet de lier des actions sur la vue, en déportant le code ailleurs

  → découplage des couches

Commands

- Commands
  - Intérêt = Remplacer le code câblé dans les button_click
  - Côté View

```
<UniformGrid Grid.Row="2" Columns="2">
    <Button Content="Add" Command="{Binding AddCommand}"/>
    <Button Content="Remove" Command="{Binding RemoveCommand}"/>
    <Button Content="Save" Command="{Binding SaveCommand}"/>
    <Button Content="Load" Command="{Binding LoadCommand}"/>
</UniformGrid>
```

  - Côté ViewModel

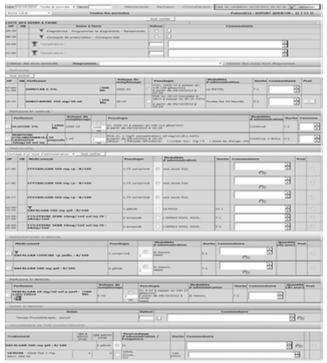
```
public ICommand SaveCommand
{
    get
    {
        if (this.saveCommand == null)
            this.saveCommand = new RelayCommand(() => this.SaveEvenement(), () => this.CanSaveEvenement());
        return this.saveCommand;
    }
}
```

Model – View - ViewModel

- Pattern très utilisés dans les développements WPF
- Des Framework existent pour faciliter la tâche :
  - MVVM Light Toolkit
  - PRISM Microsoft
  - WAF (WPF Application Framework)
- Séparation développeurs/graphistes
- Pour les besoins (limités) du tp, on utilisera 2 classes
  - ViewModelBase.cs
  - RelayCommand.cs

Exemple d'ergonomie

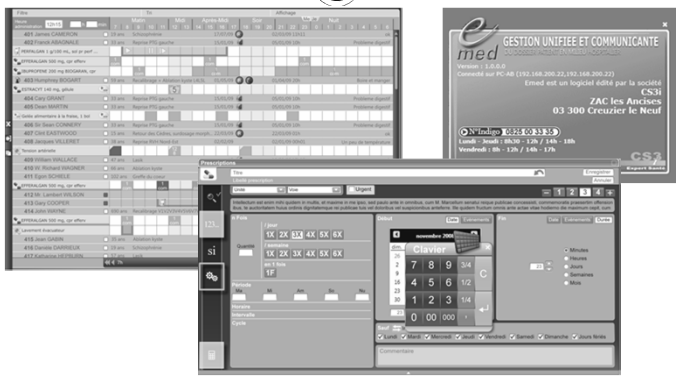
Avant



Après

Exemple d'ergonomie

Exemple d'ergonomie



TP 3

- Smart Application avancée
- But
  - Enrichir l'application WPF du TP2
  - Créer un userControl
  - Utiliser le design pattern MVVM.
    - Binding
    - Commands

→ Pouvoir visualiser, ajouter et supprimer des évènements