|  |
| --- |
|  |
| Application Type |
| Implémentation C# |
|  |
| **Thomas Auguey** |
| **20/08/2015** |

|  |
| --- |
| Implémentation C# basée sur WPF. |

SOMMAIRE

[Introduction 3](#_Toc433911057)

[Avantages 3](#_Toc433911058)

[Inconvénients 3](#_Toc433911059)

[Dépendances 3](#_Toc433911060)

[Lexique 4](#_Toc433911061)

[Interfaces 5](#_Toc433911062)

[Application 7](#_Toc433911063)

[Contrôleur 7](#_Toc433911064)

[Model 7](#_Toc433911065)

[Vue 7](#_Toc433911066)

[Evénement 8](#_Toc433911067)

[Entité 9](#_Toc433911068)

[Etat 11](#_Toc433911069)

[Donnée 11](#_Toc433911070)

[Architecture 12](#_Toc433911071)

[Arborescence 13](#_Toc433911072)

[Implémentation 14](#_Toc433911073)

[PowerDesigner 16](#_Toc433911074)

[Options de génération 16](#_Toc433911075)

[C# (WinForms) 17](#_Toc433911076)

[Architecture 17](#_Toc433911077)

# Introduction

Ce document propose une implémentation C# Type basée sur une interface Utilisateur WPF et une interface de données.

## Avantages

* Temps de compilation rapide
* Code simplifié

## Inconvénients

* Support OS limité (principalement Windows)

## Dépendances

Le code utilise :

* La librairie .Net Framework

# Lexique

ORM : Mapping objet-relationnel. Technique de programmation permettant de faire correspondre des données entre les systèmes de types incompatibles. Par exemple le mapping d’une base de données relationnelle en objets du langage de programmation.

# Interfaces

L’implémentation repose sur un squelette de classes abstraites (interfaces) et des interactions entre-elles.

Le squelette définir la structure de base de l’implémentation, en implémentant l’ensemble de ces interfaces l’architecture suit un patron de programmation permettant une meilleur lisibilité et robustesse.

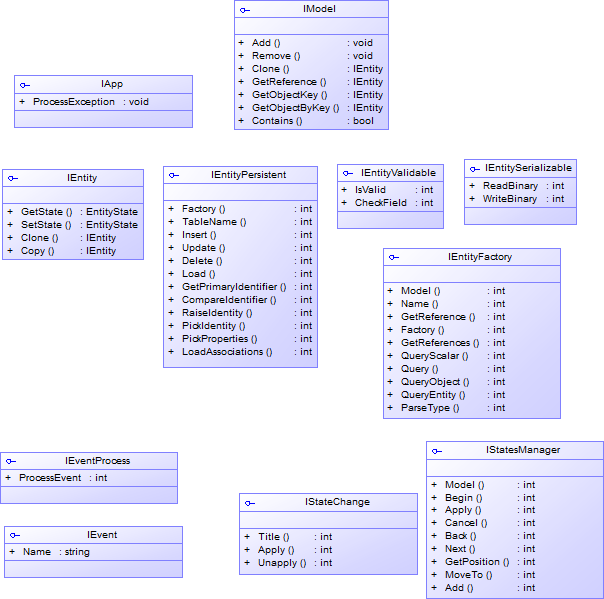
Suivant l’implémentation finale du programmeur certaines de ces interfaces seront remplacées par des classes d’un Framework externe ou des éléments du langage. Dans tous les cas ce squelette a uniquement pour but de poser une base à l’application finale, le programmeur restant maître de son architecture.

Les différents groupes d’interfaces

Les interfaces sont triées par groupe plus ou moins indépendant les uns des autres.

|  |  |
| --- | --- |
| GROUPE | DEFINITION |
| Application | Implémentation principale de l’application |
| ModelVue | Implémente la logique d’interaction entre la vue (l’utilisateur) et le model de données |
| Model | Implémente l’interface avec les données de l’application et les fonctions de la logique métier. Plusieurs modèles peuvent être implémentés |
| Vue | La vue réagit aux interactions de l’utilisateur et communique avec le ModelVue pour affecter les données |
| Evénement | Les événements permettent de diffuser un message à travers les différents éléments de l’application (vue, model, model-vue, entité, …). Son rôle est d’informer grâce à son typage et les données qu’il transporte. |
| Entité | Les entités sont les objets du model de données. Ils définissent les propriétés, les fonctions et les relations d’associations. |
| Etat | Un état définit une donnée à un moment T de sont existence. Les états sont principalement utilisé pour historiser les modifications d’un utilisateur ou d’un thread sur le model de données. |
| Donné | La donnée est l’élément le plus basique de l’application, elle peut être un type de base (entier, booléen, réel, ..) ou un type composé (chaine de caractères). L’intérêt et de pouvoir la transformer et la représenter de différente manières (dans une chaine SQL, dans un fichier XML, dans un flux binaire, …). Cette transformation est la base de l’échange de données entre les applications. |

Représentation UML des interfaces

****

## Application

Point d'entrée du programme et des ressources logicielles.

### Interface IApp

|  |  |
| --- | --- |
| **ProcessException** | Traite une exception retournée par l’un des éléments de l’application. Implémente généralement un message d’erreur ou une voix alternative à un problème. |

## ModelVue

Classe d'interaction entre la vue et le model

### Interface IViewModel

…

## Model

Manage les données et implémente la logique métier

### Interface IModel

Model de données.

|  |  |
| --- | --- |
| **CreateEntity** | Crée une entité.  *Cette méthode n’implémente pas la logique métier.* |
| **Remove** | Supprime une entité des références.  *Cette méthode n’implémente pas la logique métier.* |
| **Add** | Ajoute une entité aux références.  *Cette méthode n’implémente pas la logique métier.* |
| **Clone** | Clone le model et les entités qu'il contient.  *Cette méthode n’implémente pas la logique métier.* |
| **GetReference** | Obtient une référence de l'objet, si celui-ci n'existe pas il est automatiquement ajouté à la liste.  *Cette méthode n’implémente pas la logique métier.* |
| **GetObjectKey** | Obtient la clé d'un objet |
| **GetObjectByKey** | Obtient un objet par sa clé |
| **Contains** | Test si une référence de l’objet existe dans le model. |
| **Update** | Met à jour du model.  *Cette méthode implémente la logique métier (validation, persistance, …).* |
| **Create** | Crée une entité du model.  *Cette méthode implémente la logique métier (validation, persistance, …).* |
| **Delete** | Supprime une entité du model.  *Cette méthode implémente la logique métier (validation, persistance, …).* |

## Vue

Classe d'interaction avec l’utilisateur

### Interface IView

…

## Evénement

Les événements participes à la communication liée aux changements d’état interne et externe à l’application. L’un des plus courant est par exemple d’avertir une vue d’un changement d’état dans le model.

Les événements sont définit par le programmeur selon les besoins de sont application. Certains plus courants sont implémenté par défaut (ex : changement d’une entité)

### Interface IEvent

Evénement de base.

|  |  |
| --- | --- |
| **Name** | Nom de l’événement |

### Interface IEventProcess

Interface de traitement des événements.

|  |  |
| --- | --- |
| **ProcessEvent** | Traite l’événement |

### Interface IEventManager

Manage une collection d’objets notifiables.

|  |  |
| --- | --- |
| **NotifyRegister** | Abonne un objet aux notifications |
| **NotifyUnRegister** | Désabonne un objet notifié |
| **NotifyEvent** | Notifie un événement |

## Entité

Une entité est une composante du model de données, elle représente une structure composée de variables élémentaires.

### Domaine EntityState

Constantes des états possibles d’une entité.

|  |  |
| --- | --- |
| **Unmodified** | Non-Modifiée  Indique une entité existante dont les données ne sont pas modifiées |
| **Modified** | Modifiée  Indique une entité modifiée mais pas encore sauvegardée |
| **Added** | Ajouté  Indique une entité nouvellement ajoutée mais pas encore sauvegardée |
| **Deleted** | Supprimée  Indique une entité supprimée qui n’est plus utilisée |

### Interface IEntity

Entité de base.

|  |  |
| --- | --- |
| **EntityState** | Etat actuel de l’entité, cette valeur est gérée par le programmeur pour identifier l’état actuel de l’entité et donc les opérations à lui appliquer |
| **Model** | Model parent de l’entité |
| **EntityName** | Nom de l’entité |
| **Clone** | Duplique les propriétés de l’entité dans une nouvelle instance |
| **Copy** | Copie les propriétés d’une entité dans une autre |

### Interface IEntityFactory

Classe d’interface entre les entités et une base de données SQL.

|  |  |
| --- | --- |
| **Model** | Obtient le model de données contenant les références des entités |
| **Name** | Obtient le nom du fournisseur d'accès |
| **GetReference** | Obtient une référence d'entité |
| **Factory** | Enumère les références en base de données |
| **GetReferences** | Obtient une liste des références du model de données |
| **QueryScalar** | Exécute une requête SQL et retourne le premier champ de la première colonne trouvée |
| **Query** | Exécute une requête SQL sans retourner de résultat |
| **QueryObject** | Initialise les propriétés d'un objet générique avec les données d'une requête SQL |
| **QueryEntity** | Retourne une entité de référence initialisée avec les données d'une requête SQL |
| **ParseType** | Converti un type natif en type SQL |

### Interface IEntityPersistent

Entité dont les membres sont hébergés dans une base de données SQL.

|  |  |
| --- | --- |
| **Factory** | Obtient l'instance d'interface avec la base de données |
| **TableName** | Nom de la table correspondante à l’entité |
| **Insert** | Insert l'entité en base de données |
| **Update** | Actualise les champs en base de données |
| **Delete** | Supprime l'entité en base de données |
| **Load** | Charge les relations d'entités |
| **GetPrimaryIdentifier** | Obtient l’identificateur primaire |
| **CompareIdentifier** | Compare l'identifiant primaire avec une autre entité |
| **RaiseIdentity** | Rase l'identifiant primaire |
| **PickIdentity** | Extrait l'identifiant primaire d'un résultat de requête |
| **PickProperties** | Extrait les propriétés d'un résultat de requête |
| **LoadAssociations** | Charge une relation d'entités |

### Interface IEntitySerializable

Entité dont les membres sont exportables dans un flux de données (par exemple un fichier).

|  |  |
| --- | --- |
| **ReadBinary** | Lit les propriétés de l'entité depuis un flux de données binaire |
| **WriteBinary** | Ecrit les propriétés de l'entité depuis un flux de données binaire |

### Interface IEntityValidable

Entité dont les membres sont soumis à une validation des données.

|  |  |
| --- | --- |
| **Validate** | Test la validité de tous les champs |
| **CheckField** | Test la validité d'un champ |

## Etat

Les états permettent d’identifier les changements apportés à une application (changement de données dans une entité, exécution d’une commande, …)

Un changement d’état est généralement initié par une action de l’utilisateur (clique, saisie, copier/coller, …) ou une action du système (transformation, importation,..).

Permettre de contrôler ces états donne la possibilité de :

* Naviguer dans l’historique des modifications
* Sauvegarder des états avant de les appliquer au model persistant (mémoire, BDD, Fichiers,…)

### Interface IStateChange

Etat ou action. Il peut s’agir d’une commande, de l’état d’une entité ou d’une donnée quelconque définit par le programmeur.

|  |  |
| --- | --- |
| **Title** | Titre |
| **Apply** | Applique la modification dans le model de données |
| **Unapply** | Annule la modification dans model de données |

### Interface IStatesManager

Gestionnaire des états. Consiste en une liste ordonnée d’états. Le gestionnaire permet d’appliquer ou annuler les modifications apportées à un modèle de données.

|  |  |
| --- | --- |
| **Model** | Obtient le model de données |
| **Begin** | Débute l’enregistrement des modifications sur un model |
| **Apply** | Applique toutes les modifications apportées depuis l’appel à Begin |
| **Cancel** | Annule toutes les modifications apportées depuis l’appel à Begin |
| **Back** | Annule la modification et recule le curseur de modification |
| **Next** | Applique la modification et avance le curseur de modification |
| **GetPosition** | Obtient la position du curseur de modification |
| **MoveTo** | Déplace le curseur de modification |
| **Add** | Ajoute un changement à la liste |

## Donnée

Données

### Interface ITypedData

Représente une donnée typée

|  |  |
| --- | --- |
| **ToString** | Convertie le type en chaine de caractères |
| **FromString** | Convertie la chaine de caractères en type |
| **IsInitialised** | Indique si la valeur est initialisée |
| **GetData** | Obtient la donnée |
| **GetType** | Obtient le type de la donnée |

# Architecture

L’implémentation est basée sur le model MVVM.



## Arborescence

Ci-dessous l’arborescence des répertoires du code source :

|  |  |
| --- | --- |
| + RootDir  + [%namespace%]  + Domain  + Format  + Entity  + Association  + Interface  + View  + Event  + Lib  + ModelView  + View  + Control | Dossier racine  Dossier correspondant à l’espace de nom du model cible  Classes des domaines de valeurs  Classes de validation des formats de données  Classes des entités de données  Classes des associations d’entités  Interfaces additionnelles du model  Vues du model  Classes d’événements  Interfaces et librairies  Contrôleurs  Vues  Contrôles de vue |

## Packages NUGET

Installation sous Visual Studio :

1. Ouvrir la console nuget (Outils > Gestionnaire de package nuget > Console du Gestionnaire de package nuget) :

> install-package extended.wpf.toolkit

> install-package WPFFolderBrowser

> install-package npgsql

1. Ouvrir le gestionnaire nuget (Outils > Gestionnaire de package nuget > Gérer les packages Nuget pour la solution…) :

Pour chaque package cliquer sur Gérer et associer au projet WpfApp

# Implémentation

## Attributs

|  |  |
| --- | --- |
| UML | C# |
| A  Integer Name  … | Class A{  protected  int name ;  public  int Name{ get ; set ; } ;  } |

## Relations

|  |  |
| --- | --- |
| UML | C# |
| Relation Multiples-Multiples | Class Avion : IEntity{  …  public  IEnumerable<Aeroport> Atterrissages;  public  AddAtterrissages(Aeroport e) ;  public  RemoveAtterrissages(Aeroport e) ;  }  Class Aeroport : IEntity {  …  public  IEnumerable<Avion> Flotte;  public  AddFlotte (Avion e) ;  public  RemoveFlotte(Avion e) ;  } |
| Relation Multiples-Simple | Class Employeur : IEntity{  …  public  IEnumerable<Employe> Employes;  public  AddEmployes(Employe e) ;  public  RemoveEmployes(Employe e) ;  }  Class Employe : IEntity {  …  public Employeur patron;  public  Patron{ get; set; }  } |
| A1 <-> B1  (RoleA)<--> (RoleB) | Class Employeur : IEntity{  …  public Employeur patron;  public  Employeur Patron{ get; set; }  }  Class Employe: IEntity {  …  public Employeur patron;  public  Employeur Patron{ get; set; }  } |
| Navigabilité restreinte | Class Avion : IEntity{  …  public  Aeroport Destination{ get; set; }  }  Class Aeroport : IEntity {  …  // Aucune implémentation  } |

## Relation multiples

|  |
| --- |
| Model conceptuel |
|  |
| Diagramme de classes |
|  |
| Code Objet |
| public class Avion : IEntity{  …  // Relations  public IEnumerable< Aeroport > Atterrissages;  public AddAtterrissages (Aeroport e) ;  public RemoveAtterrissages (Aeroport e) ;  public IEnumerable< Aeroport > Itineraire;  public AddItineraire(Aeroport e) ;  public RemoveItineraire(Aeroport e) ;  // Persistence  public LoadAtterrissages() ;  public LoadItineraire() ;  }  public class Aeroport : IEntity {  …  } |

## Evénements

Les événements sont définit par l’utilisateur selon les besoins, les plus communs sont déclarés ainsi :

|  |  |  |
| --- | --- | --- |
| Classe | Base | Usage |
| Event.EntityUpdateEvent | Lib.IEvent | Lorsqu’une entité est modifiée |
| Event.EntityDeleteEvent | Lib.IEvent | Lorsqu’une entité est supprimée |
| Event.EntityCreateEvent | Lib.IEvent | Lorsqu’une entité est créée |
| Event.EntityPreCreateEvent | Lib.IEvent | Lorsqu’une entité doit être créée |
| Event.EntityPreUpdateEvent | Lib.IEvent | Lorsqu’une entité doit être modifiée |
| Event.EntityChangeEvent | Lib.IEvent | Lorsqu’une entité a changée |
| Event.EntityCopyPasteEvent | Lib.IEvent | Lorsqu’une entité est copiée/collée dans/depuis le presse-papier |

## Librairies

|  |  |  |
| --- | --- | --- |
| Classe | Base | Description |
| Lib.SqlODBCFactory | Lib.IEntityFactory | Interface avec une base de données ODBC |
| Lib.SqlPostgresFactory | Lib.IEntityFactory | Interface avec une base de données PostgreSQL |
| Lib.SqlServerFactory | Lib.IEntityFactory | Interface avec une base de données SQL Server |
| Lib.EditableDatagrid | System.Windows.Controls .DataGrid | Fournit les commandes de bases pour éditer un tableau d’entités : **Copier, Coller, Créer, Modifier, Supprimer**  **Note :** Définir la propriété de Binding **UpdateSourceTrigger** à **PropertieChanged** ou **LostFocus** pour permettre la modification des données dans l’entité avant le traitement par le model. |
| Lib.EditWindow | System.Windows.Window | Fenêtre générique encapsulant la vue EditView |
| View.EditView | System.Windows.Controls.Grid | Vue générique utilisée pour générer rapidement une vue de contrôles utilisateur. |
| Lib.EntitiesModel | Lib.IModel | Fournit une implémentation générique d’un model de données |
| Lib.EventProcess | **-** | Fournit des implémentations type des événements pour les contrôleurs de données (copier/coller, création, suppression, modification, …) |
| Lib.StatesManager | Lib.ViewModelBase, Lib.IStateManager | Fournit une implémentation générique d’un gestionnaire d’état (historique des modifications) |

# PowerDesigner

Le Template de langage **Tpl\_CSharp.xol** propose la génération du diagramme de classes en code source exploitable par le compilateur C#.

## Options de génération

|  |  |  |
| --- | --- | --- |
| Nom | Option | Description |
| namespace | Espace de nom réservé pour le model | Indique le « **namespace** » de base pour toutes les classes générées |
| serial\_datatype | Type de valeur associée au type Serial en base de données | Type de donnée C# a faire correspondre avec le type Serial du model de données conceptuel. Il s’agit des index auto incrémenté en base de données (généralement du type **System.Int32** ou **System.Int64** suivant l’architecture de la base de données). |
| useDotNetValidation | Implémenter l’interface de validation .Net | Implémente également l’interface standard **IDataErrorInfo** pour les entités validable (stéréotype). |
| useWPF | Exporter le code pour Windows presentation fundation | Exporte les vues, convertisseurs et autres code pour les applications Windows basé sur WPF.  **Note : La génération des vues éditables requière une référence à la librairie « Xceed.WPF.Toolkit »** |