|  |
| --- |
|  |
| Modélisation UML |
| Patrons MVC pour PowerDesigner |
|  |
| **Thomas Auguey** |
| **20/08/2015** |

|  |
| --- |
| Guide pratique des patrons MVC de modélisation UML applicables aux applications en développement. |

SOMMAIRE

[Squelette d’application 3](#_Toc430174396)

[Evénements 3](#_Toc430174397)

[Entités 3](#_Toc430174398)

[En Pratique 5](#_Toc430174399)

[Sérialisation 5](#_Toc430174400)

[Base de données 5](#_Toc430174401)

[Génération avec PowerDesigner 6](#_Toc430174402)

[Applications cibles 6](#_Toc430174403)

[Stéréotypes 6](#_Toc430174404)

[Options de génération 6](#_Toc430174405)

[Implémentation C# 8](#_Toc430174406)

[Introduction 9](#_Toc430174407)

[Implémentation 9](#_Toc430174408)

[Génération avec PowerDesigner 10](#_Toc430174409)

[Implémentation Visionner 12](#_Toc430174410)

[Introduction 12](#_Toc430174411)

[Implémentation 12](#_Toc430174412)

[Génération avec PowerDesigner 12](#_Toc430174413)

# Squelette d’application

Les applications orientés objets quel que C++ et C# sont basées sur un squelette commun définit par une collection de classes d’interfaces. Ce squelette permet de définir les règles de fonctionnement de toute application basée sur ce patron.

Les éléments représente les concepts les plus communs et indispensables que l’on retrouve dans une application MVC.

## Evénements

Les événements participes à la communication liée aux changements d’état extérieurs et intérieur au processus. L’un des plus courant est par exemple d’avertir une vue d’un changement des données dans le model.

### IEvent

Base de tout événement.

### IEventProcess

Classe d’interface entre les événements est un composant de l’application (par exemple une vue).

## Entités

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

### IEntity

Base de toute entité.

|  |  |
| --- | --- |
| **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 |
| **EntityName** | Nom de l’entité |
| **Clone()** | Duplique les propriétés de l’entité dans une nouvelle instance |

### EntityState

Etat d’une entité.

|  |  |
| --- | --- |
| **Unmodified** | Non-Modifiée  Indique une entité existante et non modifiée |
| **Modified** | Modifiée  Indique une entité existante mais ayant été modifiée |
| **Added** | Ajouté  Indique une entité nouvellement ajoutée n’est pas encore sauvegardée |
| **Deleted** | Supprimée  Indique une entité supprimée qui n’a plus lieu d’être utilisée |

### IEntityFactory

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

### IEntityPersistent

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

### IEntitySerializable

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

### IEntityValidable

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

# En Pratique

Cette section d’écrit les avantages et bonnes pratiques a adopté pour programmer son application.

## Sérialisation

### Utilisation des agrégations

L’utilisation des agrégations dans le model UML permet de définir rapidement les dépendances entres classes et ainsi généralisé la sérialisation des classes enfants.

## Base de données

### Connexions multiples, références multiples

L’utilisation de base de données multiples avec parfois un model identique implique de stocker les références d’objets dans des listes séparés. L’utilisation de 2 classes **IEntityFactory** implémente automatiquement 2 listes d’entités en cache, ainsi chaque entité conserve un lien avec sa base de données ce qui empêche toute collision d’identifiants primaires.

### Base de données piloté par procédures stockées et les vues SQL

### Base de données piloté par requêtes SQL stockées et les tables de données

# Génération avec PowerDesigner

Ce document est un support utilisateur aux Templates de langages PowerDesigner. Ces templates permettent de générer la partie MVC des model de données pour diverses applications.

Le but de ces modèles et de générer une partie du code correspondant au model de données et l’architecture de l’application. Le gain de temps est conséquent et le produit final n’impose aucune contrainte au programmeur. En effet, tout le code généré étant totalement éditable manuellement dans le cas ou PowerDesigner ne serait plus utilisé dans les versions futures du programme.

## Applications cibles

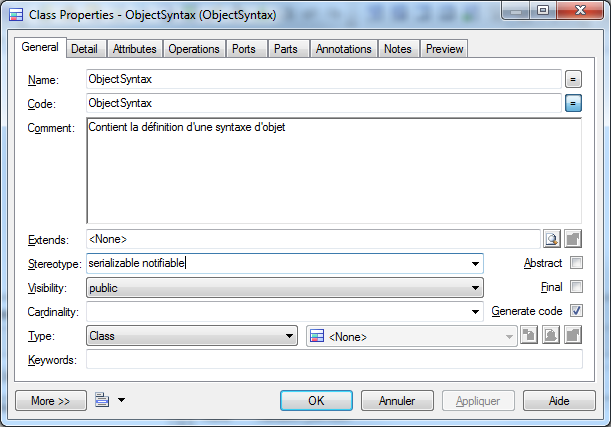
Ce tableau résume les modèles cibles d’applications :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Langage | Version | Framework | Fichier | Modèles cible |
| Visionner – Basic | 3.x | - |  | OOM et PDM |
| C# | - | - | Tpl\_CSharpApp.xol | OOM |
| C# | - | .Net | Tpl\_CSharpApp.xol | OOM |
| C++ | - | - | Tpl\_CPlusPlus.xol | OOM |
| C++ | - | wxWidget | Tpl\_CPlusPlus.xol | OOM |
| SQL Sever | 2012+ | - | Tpl\_SQLServer2012.xdb | PDM |
| Postgres SQL | 9 | - | Tpl\_pgsql9.xdb | PDM |

## Stéréotypes

Les stéréotypes sont utilisés dans les objets pour définir l’usage final dans le code. Par exemple, si les données d’une classes sont persistantes, serializables, validables, etc…

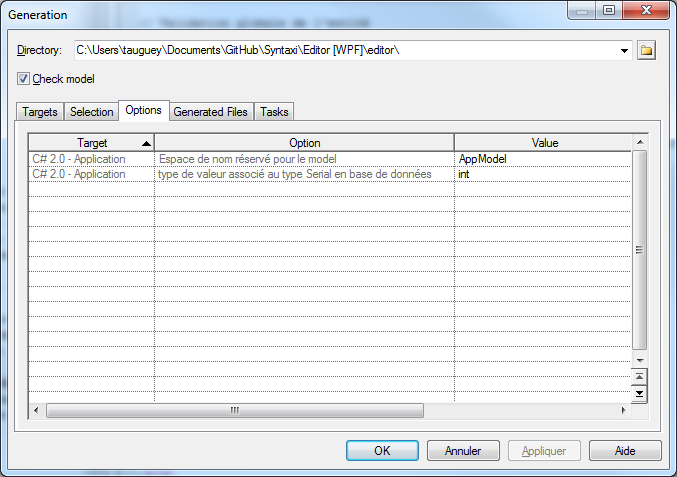
Les stéréotypes sont définissables dans les propriétés d’objets :



***Propriété d’une classe contenant les stéréotypes « serializable » et « validable »***

## Options de génération

Les options de générations sont spécifiques au langage cible, elles permettent d’ajuster la génération du code. Chaque option est définit plus loin dans son model.



***Options de génération dans le dialogue « Générer »***

# Implémentation C#

Le model C# orienté objet permet la génération du model dans une application MVC type. Tout le code généré est indépendant de toutes librairies externes. Le code est seulement soumit aux contraintes du langage. Certaines implémentations additionnelles telles que WPF peut venir compléter l’architecture de l’application cible.

## Introduction

Le model exporte les classes de bases extraites du model UML.

L’intégration implémente :

* Les classes d’entités
* La validation des propriétés d’entités
* Les domaines de valeurs
* Les vues d’éditions
* Le squelette d’interfaces

Le schéma ci-dessous montre les répertoires cibles de l’arborescence générée.

|  |  |
| --- | --- |
| + RootDir  + %model namespace%  + Domain  + Format  + Entity  + Interface  + View  + View  + Lib | Dossier racine  Dossier correspondant à l’espace de nom du model cible  Enumération des domaines de valeurs  Classes de validation des formats de données  Classes des entités de données  Interfaces additionnelles du model  Vues du model (XAML)  Vues globales (XAML)  Interfaces et classes génériques du model de données |

## Implémentation

### Entités

Chaque entité définit une classe de données dans l’espace de nom qui lui est attribué.

### Base de données (.Net)

Le programmeur peut implémenter sa propre interface **SGBD** basée sur l’interface **IEntityFactory**.

Les implémentations fournies sont :

|  |  |
| --- | --- |
| Fichier | Fournisseur |
| Lib/SqlODBCFactory.cs | Passerelle ODBC |
| Lib/SqlPostgresFactory.cs | Postgre SQL |
| Lib/SqlServerFactory.cs | SQL Server |

### Interface utilisateur (WPF)

Les librairies d’exemples implémentent des contrôles utilisateurs dérivées de WPF.

Les implémentations fournies sont :

|  |  |  |
| --- | --- | --- |
| Classe | Classe de base | Implémentation |
| EditableDatagrid | DataGrid | Fournit les commandes de bases pour les entités : **Copier, Coller, Changements d’état** |

**Note : La génération des vues éditables requière une référence à la librairie « Xceed.WPF.Toolkit »**

## Génération avec PowerDesigner

Ce model est peut être générés depuis un « model conceptuel de données » ou directement édité.

### 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 **int** ou **long** suivant l’architecture de la BDD). |
| useDotNetValidation | Implémenter l’interface de validation .Net | Implémente également l’interface standard **IDataErrorInfo** pour les entités validable. |
| useWPF | Exporter le code pour Windows presentation fundation | Exporte les vues, convertisseurs et autres code pour les applications Windows basé sur WPF. |

### Stéréotypes

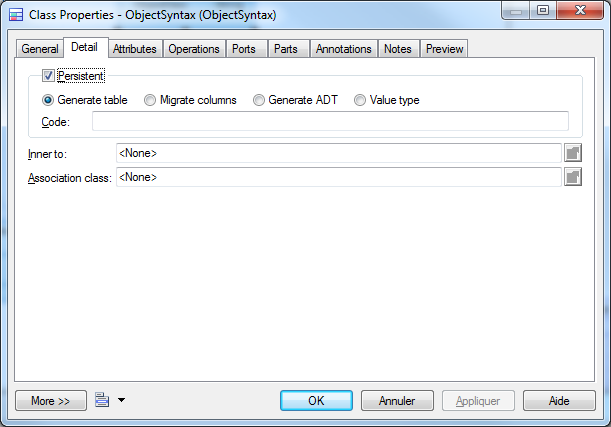
|  |  |
| --- | --- |
| Mot clé | Description |
| persistant | Implémente l’interface **IEntity** avec les méthodes d’accès à la base de données |
| notifiable | Implémente l’interface **IEntitySerializable** avec les méthodes de conversions binaire et XML |
| serializable | Implémente l’interface **INotifyPropertyChanged** avec les méthodes de notification des champs |
| validable | Implémente l’interface **IEntityValidable** avec les méthodes de validation des champs.  Utilisable avec l’option de génération **« useDotNetValidation»**. |

### Persistance

Il est possible de faire spécifier explicitement les noms des tables et champs en correspondance avec la classe généré. Par défaut le nom des tables est hérité des classes.

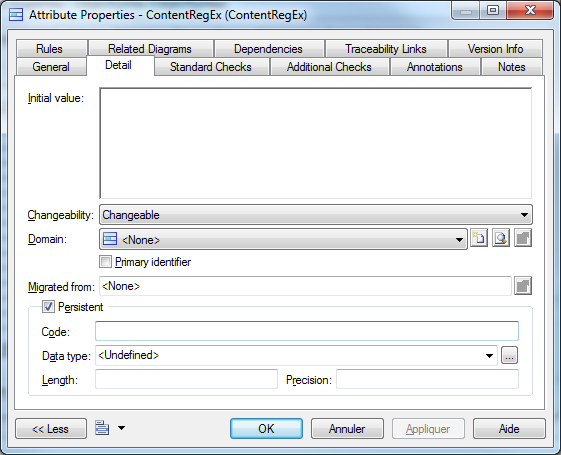
Spécifier le nom d’une table :

Rendez-vous dans le dialogue de propriété de classe sous l’onglet « **Détail** » et éditer le champ « **Code** » sous le groupe « **Persistent** ».



Spécifier le nom d’un champ :

Rendez-vous dans le dialogue de propriété de l’attribut sous l’onglet « Détail » et éditer le champ « Code » sous le groupe « Persistant ».



# Implémentation Visionner

## Introduction

Le model exporte la structure des tables et les fonctions d’accès génériques aux données SQL.

Le schéma ci-dessous montre les répertoires cibles de l’arborescence générée.

|  |  |
| --- | --- |
| + RootDir  + Model.txt  + Model.bas | Dossier racine  Model des structures de données  Fonctions « Basic » d’accès aux données |

## Implémentation

### Tables

Chaque table définit un type de données structuré sous Visionner.

### Fonctions d’interface avec la base de données

Chaque table définit une fonction de sélection pour chaque clé primaire et alternative.

## Génération avec PowerDesigner

Ce model est divisé en 2 module de langage « le model orienté objet » et « le model physique de données ». Ces 2 modèles peuvent être générés depuis un « model conceptuel de données » ou directement édité.