L’outil MDE

1. Déroulement de l’outil MDE

La génération de code avec l’outil MDE se fait en trois étapes : spécification de l’architecture, paramétrisation du PIM, et génération du code.

1. Spécification de l’architecture :

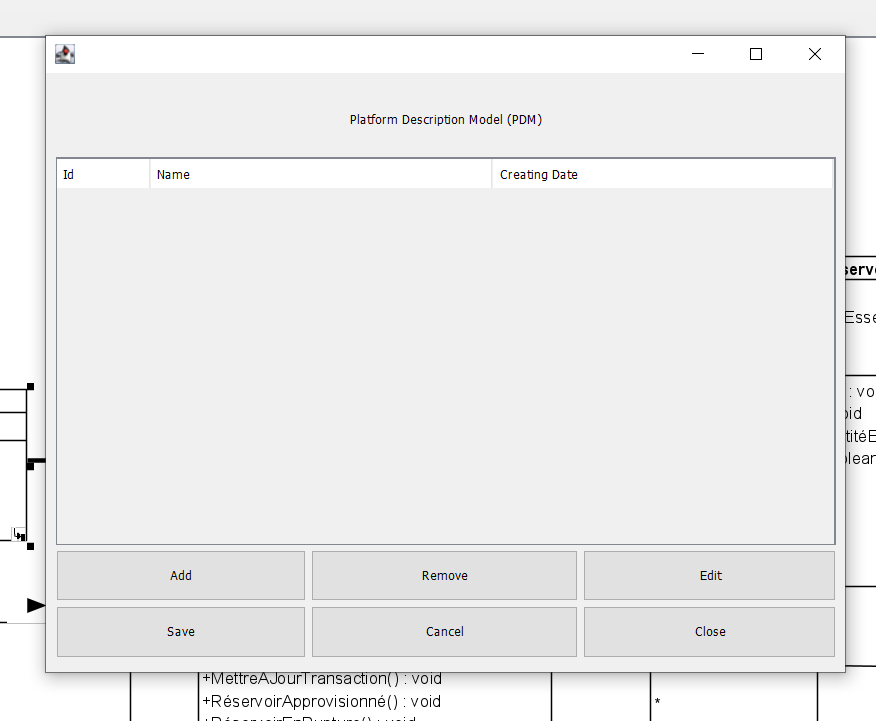
Pour spécifier l’architecture il faut suivre les étapes suivantes :

* Ouvrez Visual Paradigm puis cliquez sur « platform description model » dans l’onglet « MDE ».

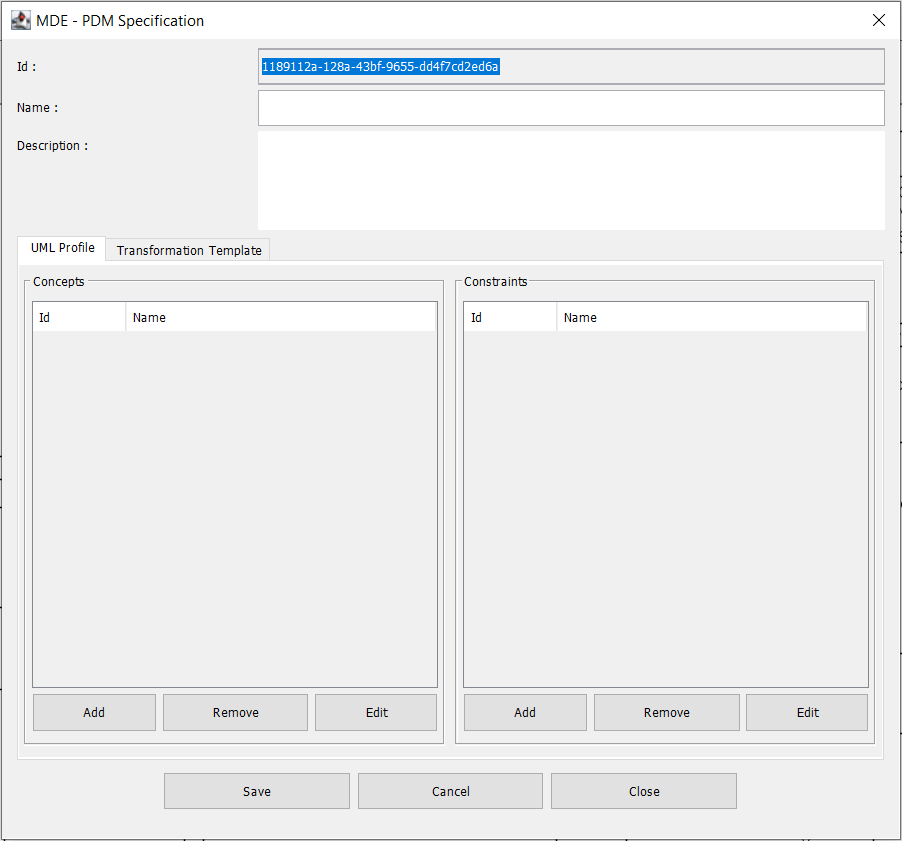
Une image contenant texte

Description générée automatiquement

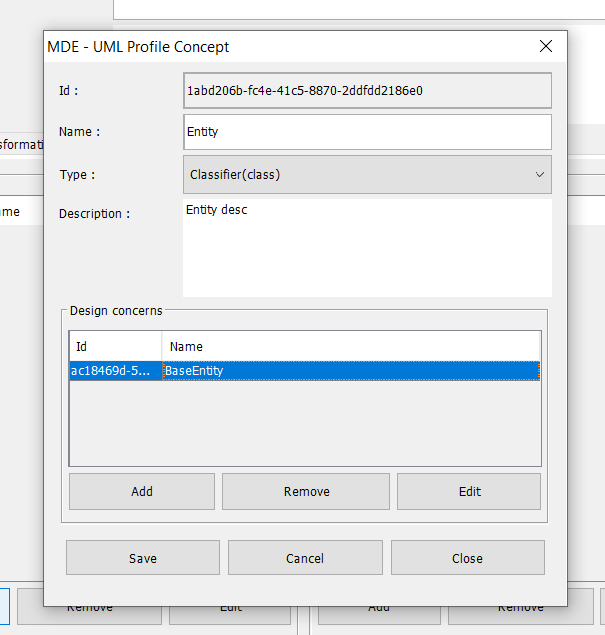
* Cliquez sur « Add » dans la fenêtre principale pour ajouter un PDM.



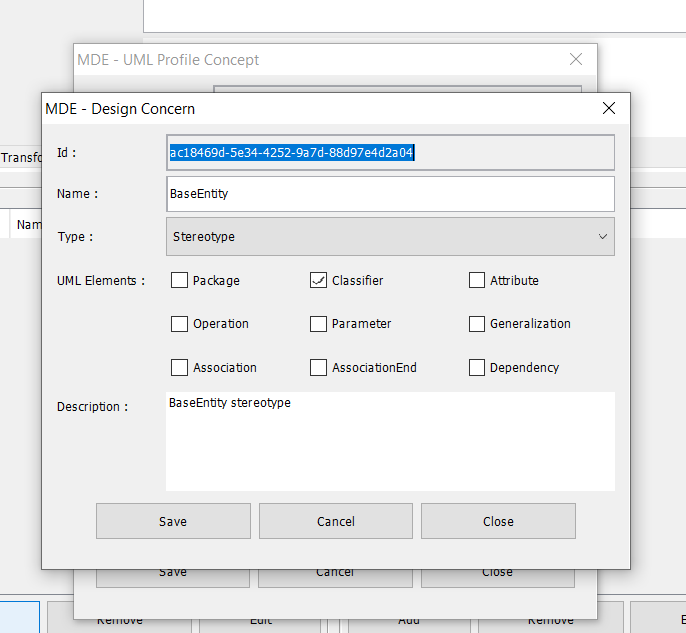
* Un PDM est défini par un profil UML et des modèles de transformation. Cliquez sur « Add » dans chaque section pour ajouter un concept ou une contraint du profil UML, ou un modèle de transformation.



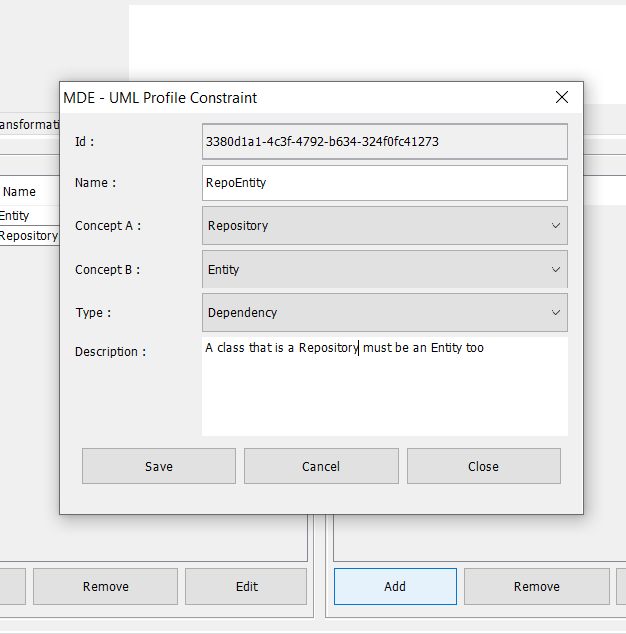
* Pour ajouter un concept il faut saisir : le nom, le type, la description, et les design concerns. Pour ajouter un design concern, cliquez sur « Add » dans la section « Design concerns ».



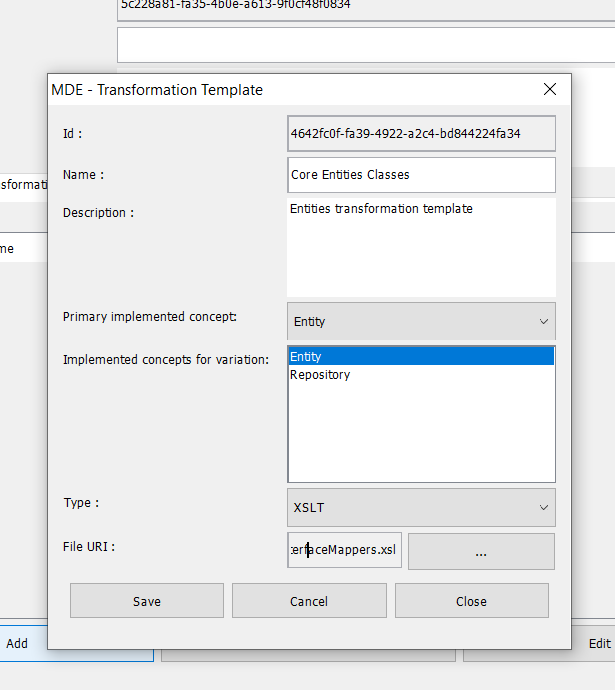
* Pour ajouter un « design concern », saisissez le nom, le type, les éléments UML concernés, et la description.



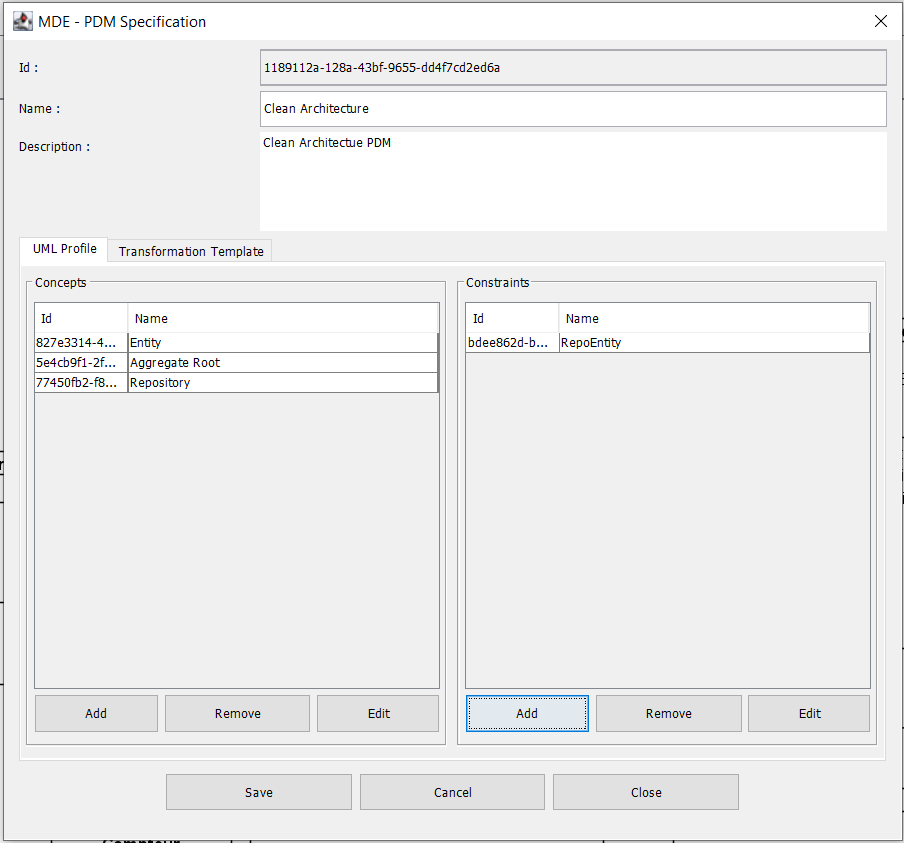
* Après la définition des concepts, il faut définir les contraintes entre eux. Pour ajouter une contrainte saisissez le nom, le premier concept, le deuxième concept, le type, et la description.



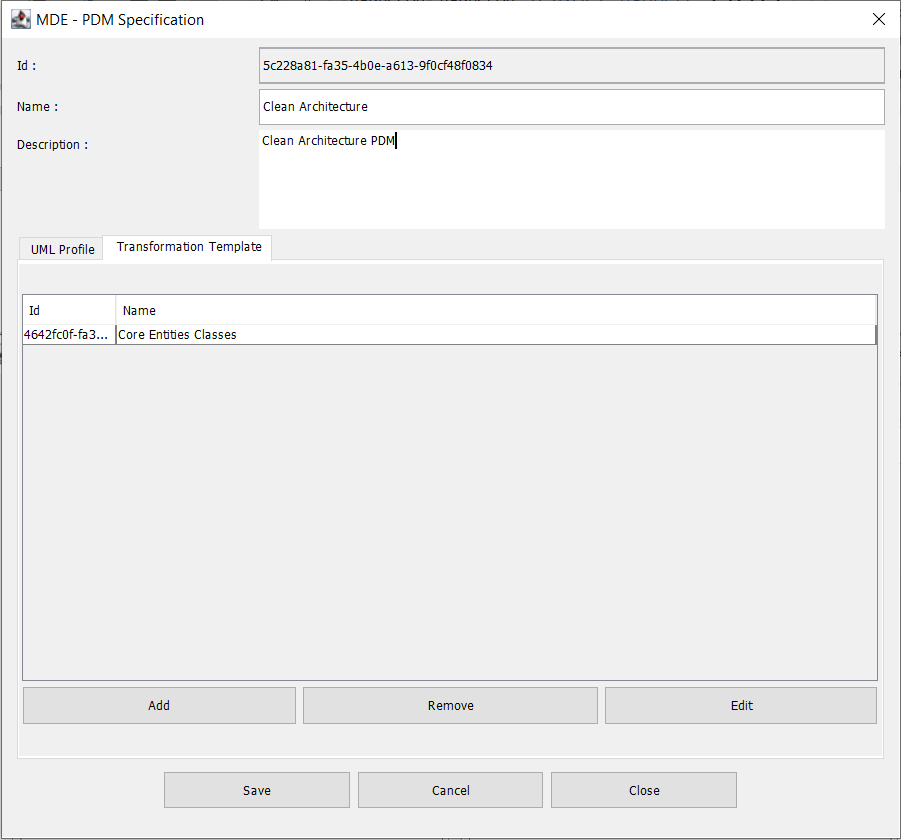
* La deuxième partie de la spécification de l’architecture consiste à définir les modèles de transformation. Pour ajouter un modèle de transformation saisissez le nom, la description, le concept principal implémenté, les concepts de variation implémentés, le type, et le chemin vers le modèle de transformation.



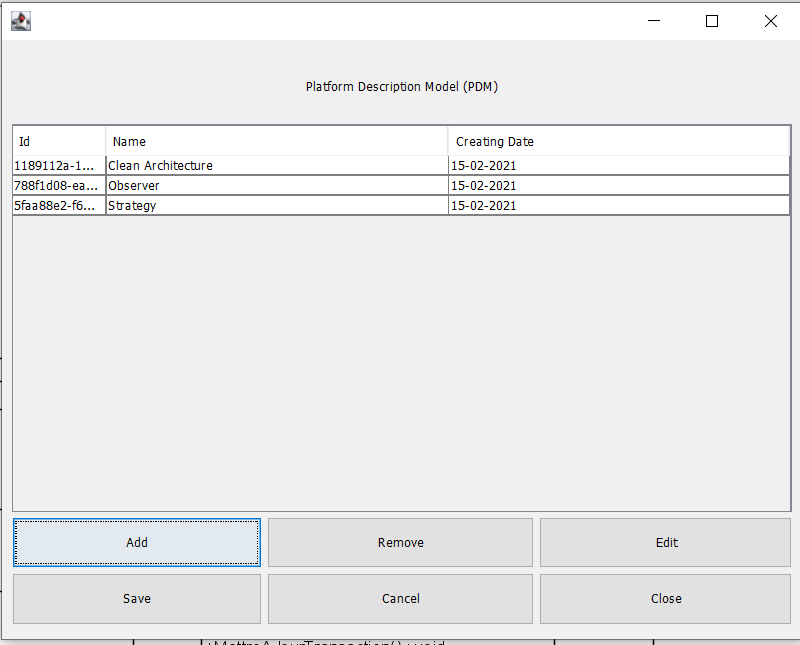
* Voici le profil UML après sa définition. L’utilisateur peut ajouter, modifier, ou supprimer un concept ou une contrainte.



* Voici les modèles de transformation après leur définition. L’utilisateur peut ajouter, modifier, ou supprimer un modèle de transformation.
* Après la spécification de l’architecture, cliquez sur « Save ».



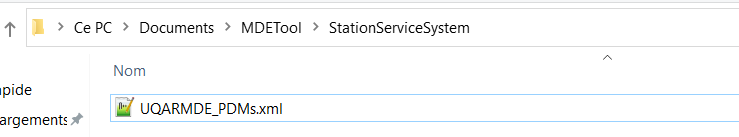
* Après l’ajout du PDM « clean architecture », j’ai ajouté d’autres pour les patrons de conception (observateur et stratégie). L’utilisateur peut ajouter, modifier, ou supprimer un PDM.
* Enfin, cliquez sur « Save » pour sauvegarder les PDMs.



* Les PDMs sont enregistrés au format XML dans le chemin :

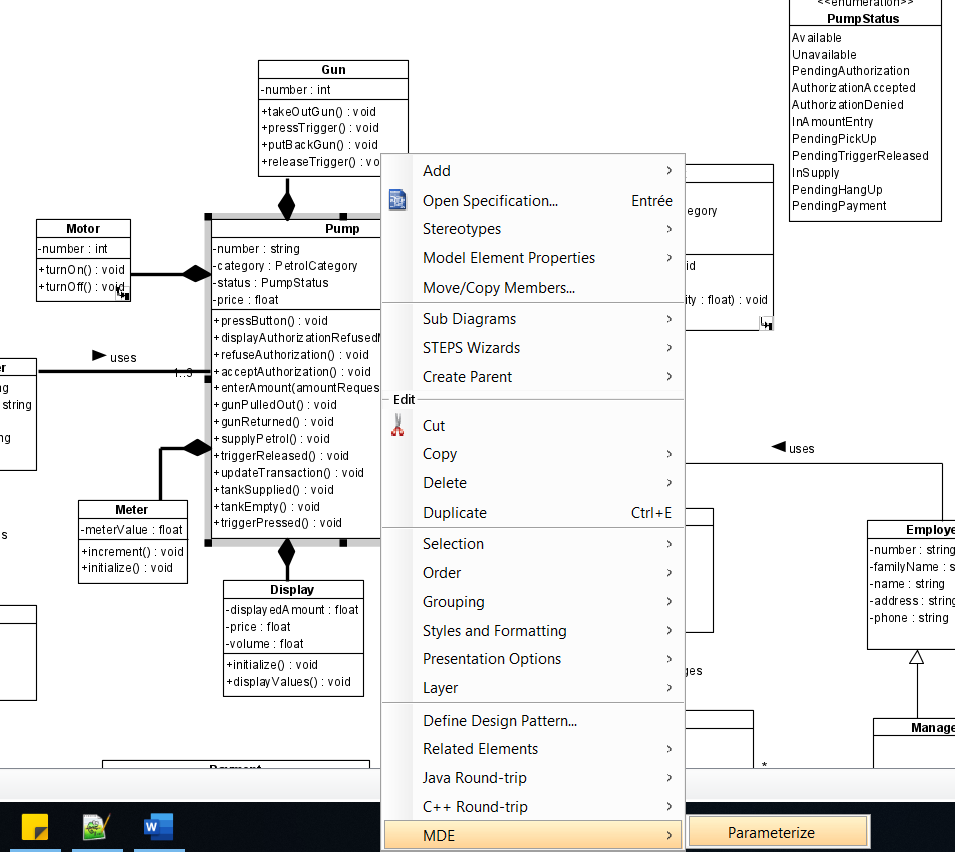
C:\Users\<*Nom\_utilisateur*>\Documents\MDETool\<*Nom\_projet\_VP*>\UQARMDE\_PDMs.xml>

* Voir le fichier joint « UQARMDE\_PDMS.xml ».

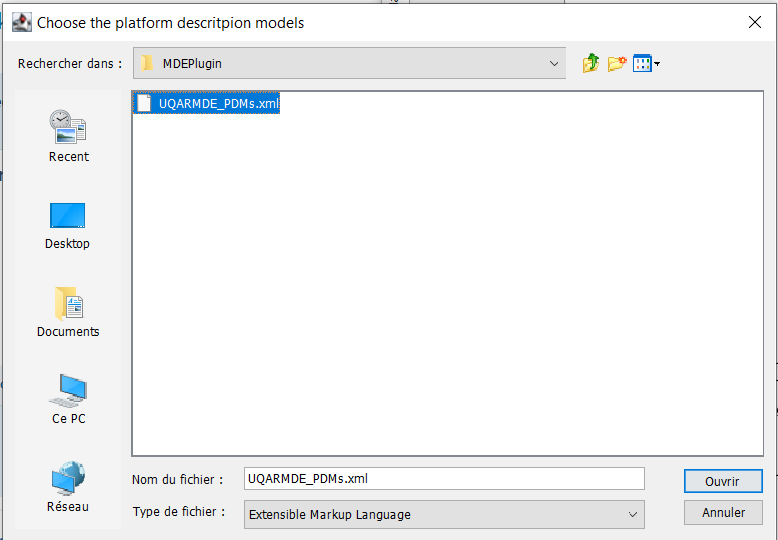


1. Paramétrisation du PIM :

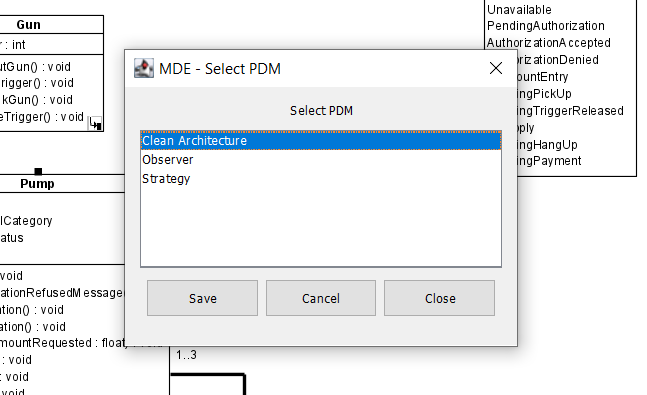
* Sélectionnez les éléments UML à paramétrer, puis faites un clic droit et choisissez « MDE » ensuite cliquez sur « Parameterize ». Dans l’exemple suivant, j’ai sélectionné la classe « Pump ».



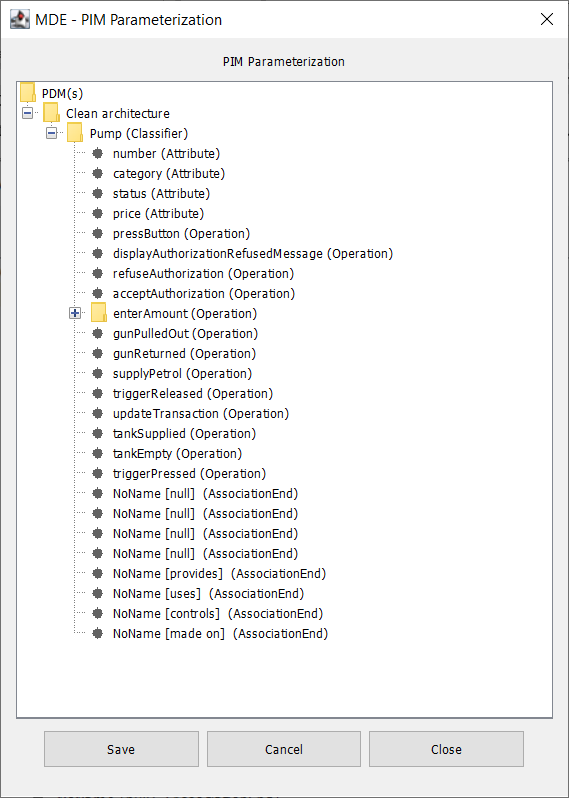
* Choisissez les PDMs déjà définis.



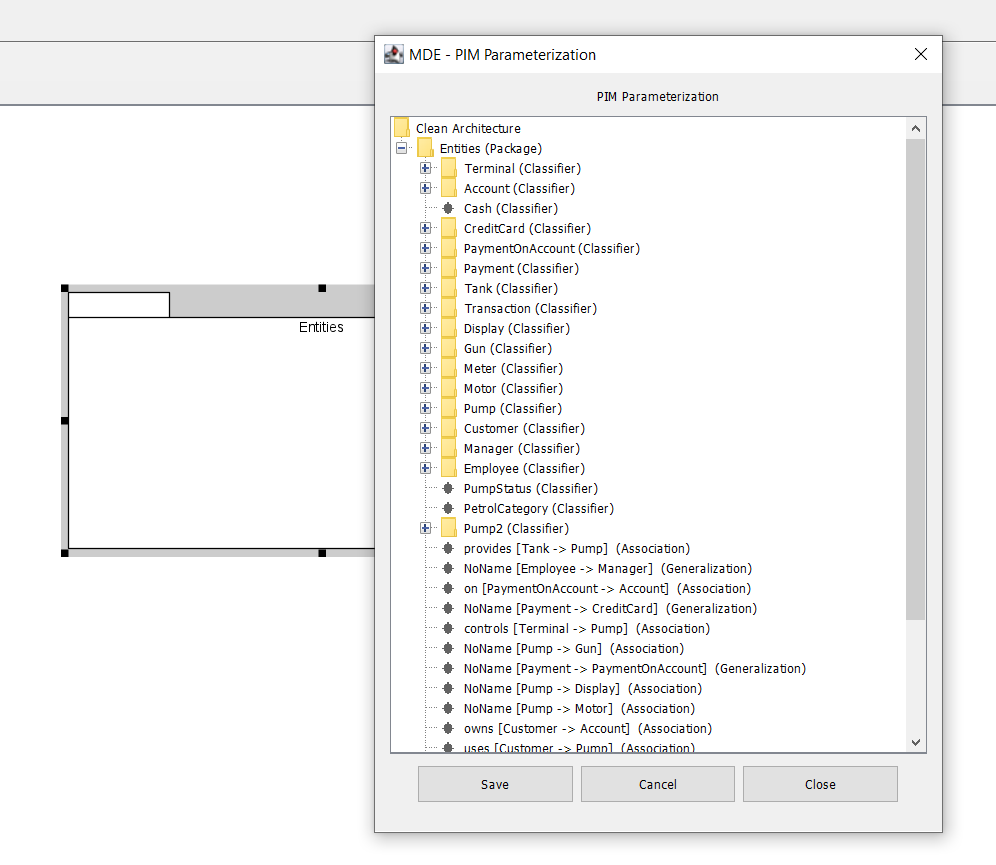
* Sélectionnez un ou plusieurs PDMs pour paramétriser le PIM et cliquez sur « Save ». dans cet exemple j’ai choisi que le PDM « Clean Architecture ».



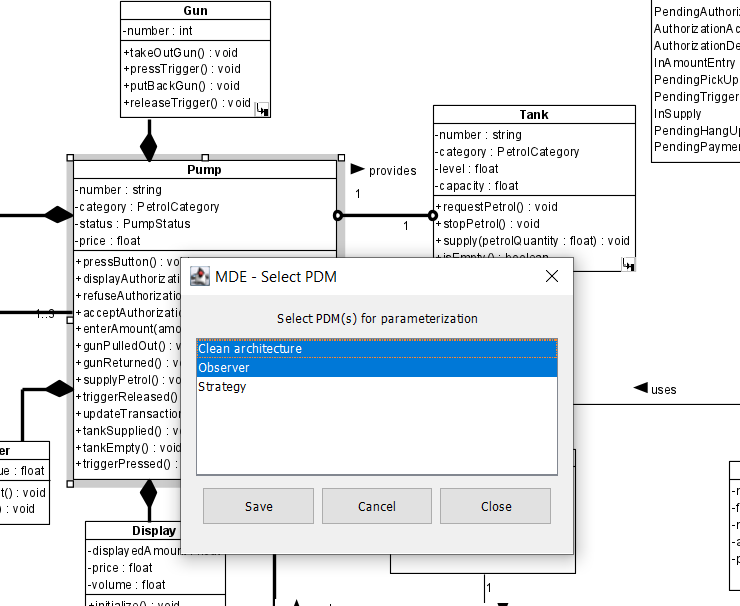
* La fenêtre qui s’ouvre consiste en une arborescence d’éléments. Les racines de l’arborescence sont les PDMs choisis. les branches des racines sont les éléments UML sélectionnés dans Visual Paradigm. En outre, certains éléments UML sélectionnés ils vont avoir des sous-branches, donc on trouve :
* Si l’élément UML sélectionné est un package ses branches seront : des classes et des associations.
* Si l’élément UML sélectionné est une classe ses branches seront : des attributs, des opérations, et des « associationEnds ».
* Si l’élément UML sélectionné est une opération ses branches seront : des paramètres.



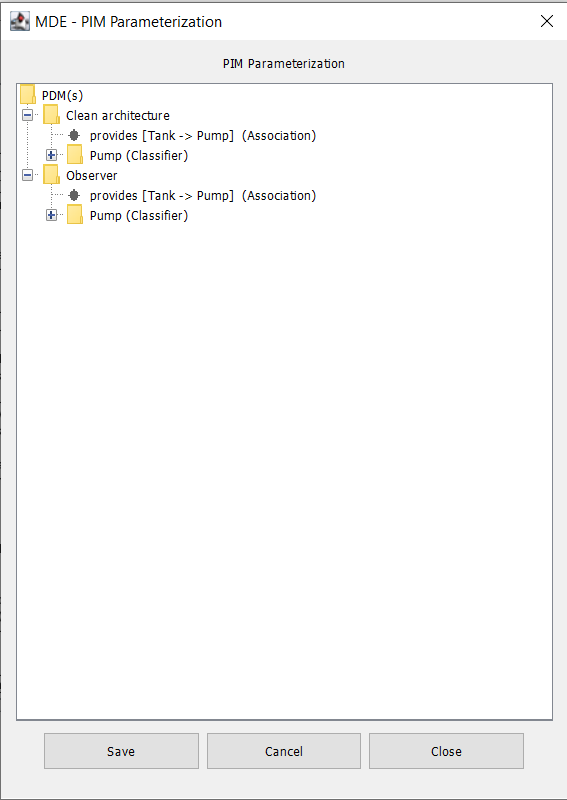
* Voici l’arborescence des éléments quand j’ai sélectionné un package pour le paramétrage.



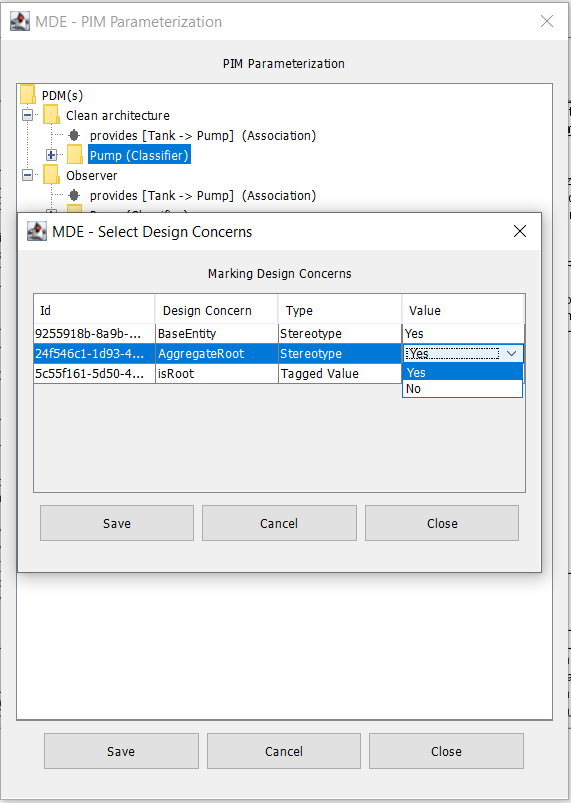
* L’utilisateur peut choisir plusieurs PDM et sélectionner plusieurs éléments UML en même temps. Par exemple, j’ai choisi les PDMs « Clean architecture » et « Observer », et j’ai sélectionné la classe « Pump » et l’association « provide » (entre « Pump » et « Tank »).



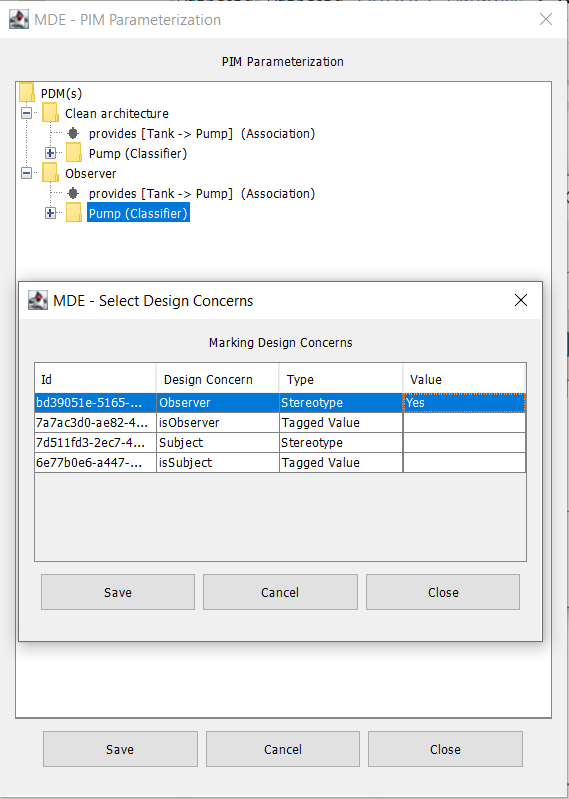
* L’outil va lister chaque PDM choisi, et pour chaque PDM il va lister les éléments UML sélectionnés. L’utilisateur peut donc paramétriser un élément UML avec plusieurs PDM en même temps.



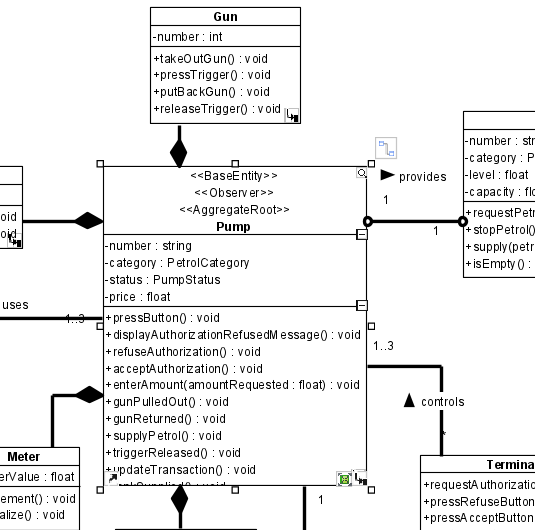
* Cliquez sur un élément UML pour le paramétrer.
* Dans l’exemple suivant, j’ai cliqué sur la classe « Pump » dans le PDM « Clean architecture ». Ensuite une fenêtre s’ouvre contient les design concerns appliqués aux éléments UML de type « Classifier ».
* Pour les stéréotypes sélectionnez « oui » ou « non » comme valeur. Et pour les tagged value saisissez la valeur souhaitée.
* A la fin, cliquez sur « Save ».



* L’utilisateur peut faire la même chose pour la classe « Pump », mais avec les design concerns du PDM « observer ».



* Après l’enregistrement de la paramétrisation, l’utilisateur peut visualiser le résultat dans le PIM.



* Le fichier XML du résultat de la paramétrisation est enregistré dans le chemin :

C:\Users\<*Nom\_utilisateur*>\Documents\MDETool\<*Nom\_projet\_VP*>\<*Nom\_projet\_VP*>\_PIMParametrization.xml ».

* Voir le fichier joint « StationService\_PIMParametrization ».

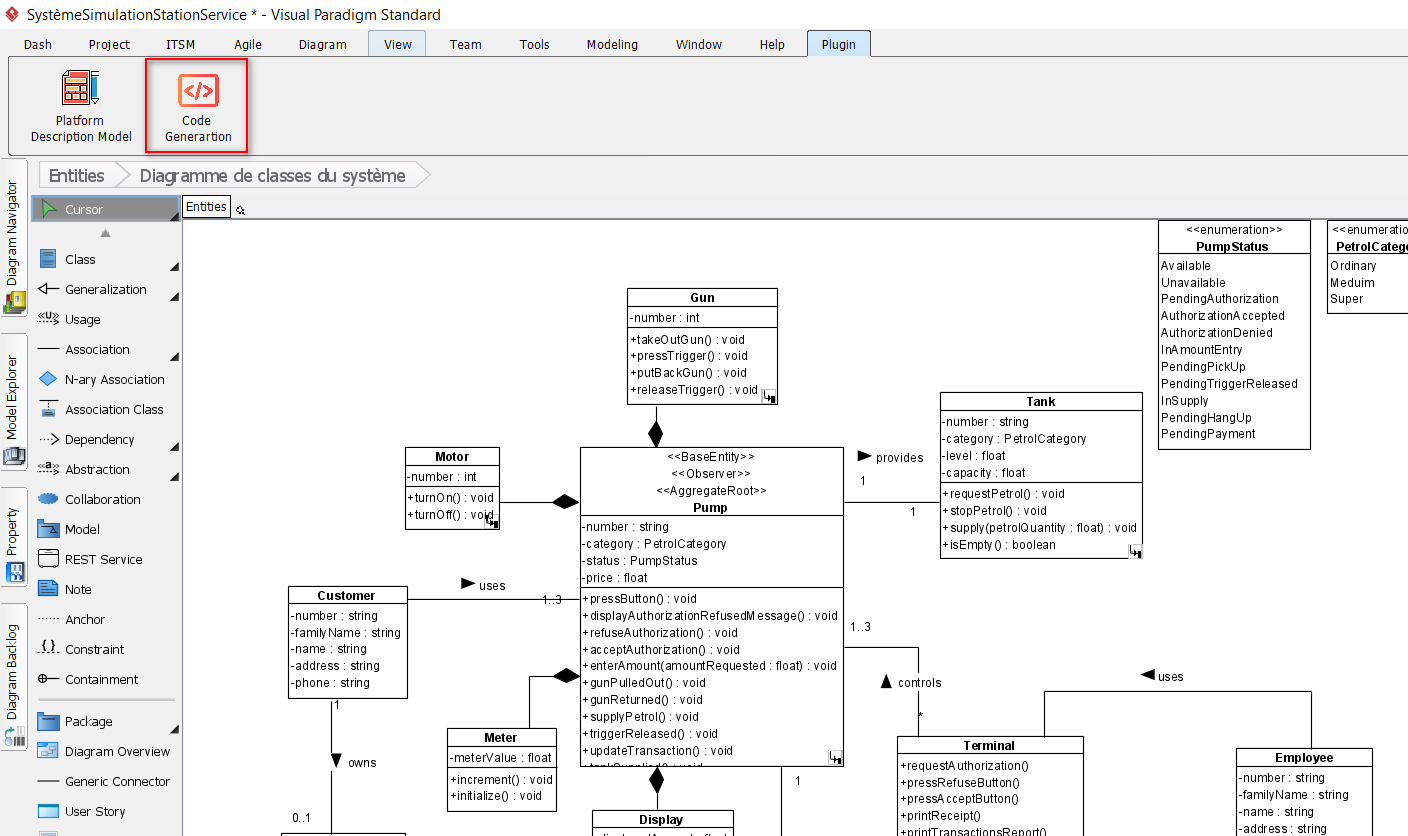
Une image contenant texte

Description générée automatiquement

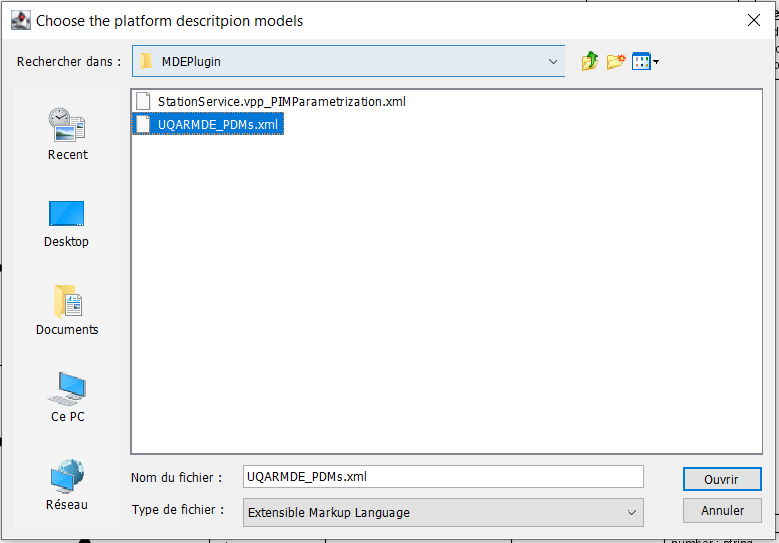
1. Génération de code :

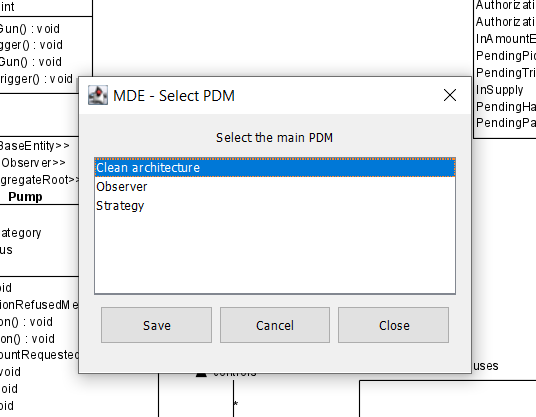
Avant de générer le code il faut exporter le PIM paramétrisé en format XML (n’est pas encore supporter par VP open api) et le déplacer vers le chemin « C:\Users\<*Nom\_utilisateur*>\Documents\MDETool\<*Nom\_projet\_VP*>\Generated code », ainsi que les modèles de transformation.

* Pour générer le code, cliquez sur « Code Generation » dans l’onglet « Plugin ».

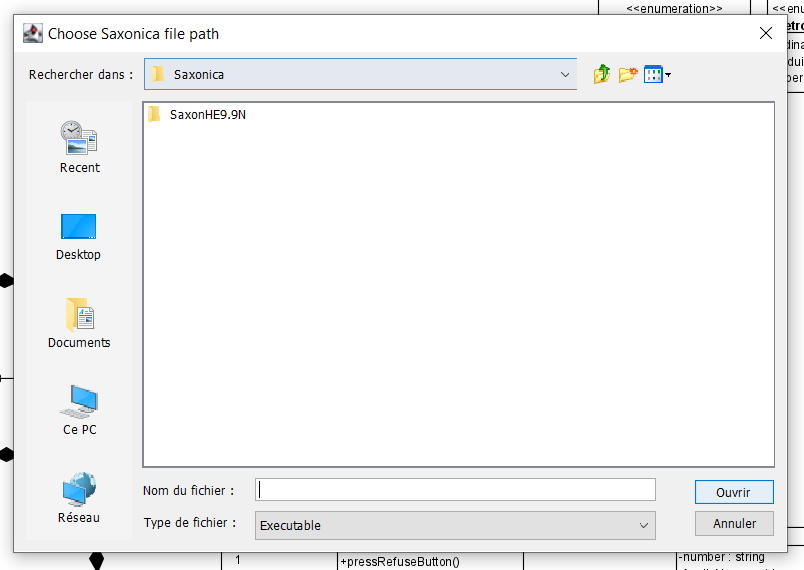


* Sélectionnez le PDM principal qui contient les modèles de transformation.

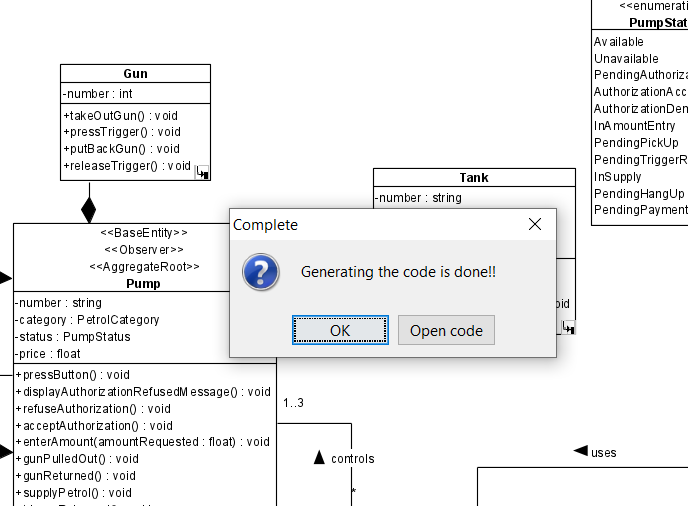




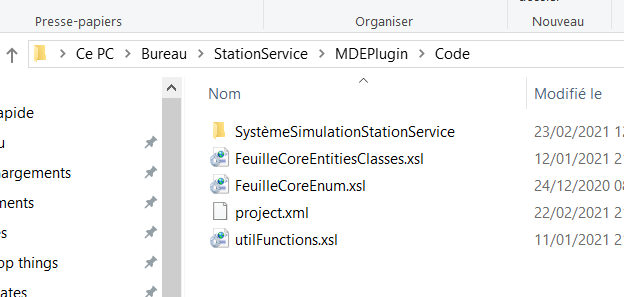
* Sélectionnez le chemin vers l’outil de transformation « Saxonica ». Dans ma machine se trouve dans « C:\Program Files\Saxonica\SaxonHE9.9N\bin\Transform.exe ».



* Attendez la génération de code et cliquez sur « Open code » pour ouvrir l’emplacement du code.



* Le code est généré dans le dossier « Code ».



1. Tester le plugin

Pour tester le plugin dans Visual Paradigm (version 15.2) :

Copier le dossier «Code Generation» dans le dossier «Plugins» de Visual Paradigm. Dans ma machine il est dans le chemin suivant : « **C:/Users/hamza/AppData/Roaming/VisualParadigm/plugins** »

1. Configuration de Visual Studio code pour développer le plugin.

* <https://forums.visual-paradigm.com/t/using-vs-code-to-develop-vp-plugins/14987>
* Installer VS Code.
* Installer java JDK (jdk-11.0.3) et ajouter le chemin (ex : C:\Program Files\Java\jdk-11.0.3\bin) dans les variables d’environnement.
* Ajouter le chemin vers Apache Ant(dossier : apache-ant-1.10.7-bin) dans les variables d’environnement.

Dans ma machine : C:\Users\hamza\Downloads\apache-ant-1.10.7-bin\apache-ant-1.10.7\bin

* On doit ajouter quatre extensions à Visual Studio Code :
* Java Extension Pack : <https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack>
* XML : <https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml>
* Ant Target Runner : <https://marketplace.visualstudio.com/items?itemName=nickheap.vscode-ant>
* Ant Tree Viewer : <https://marketplace.visualstudio.com/items?itemName=pedroguerra.ant-tree-viewer>
* Ouvrir le dossier «CodeGenerationPlugin» avec VS Code
* Changer le chemin vers « Open API » dans le fichier «.classpath ». Dans ma machine il existe dans : «C:/Program Files/Visual Paradigm 15.2/lib/openapi.jar».
* Pour le « Build », Il faut changer les deux propriétés « vp » et « openapi » dans le fichier « build.xml ». Dans ma machine :

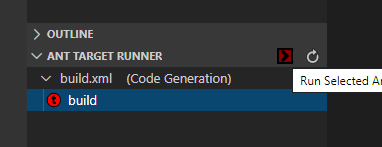
Openapi (emplacement d’open api) :

« C:/Program Files/Visual Paradigm 15.2/lib/openapi.jar »

Vp (destination de l’exécutable du plugin après le build):

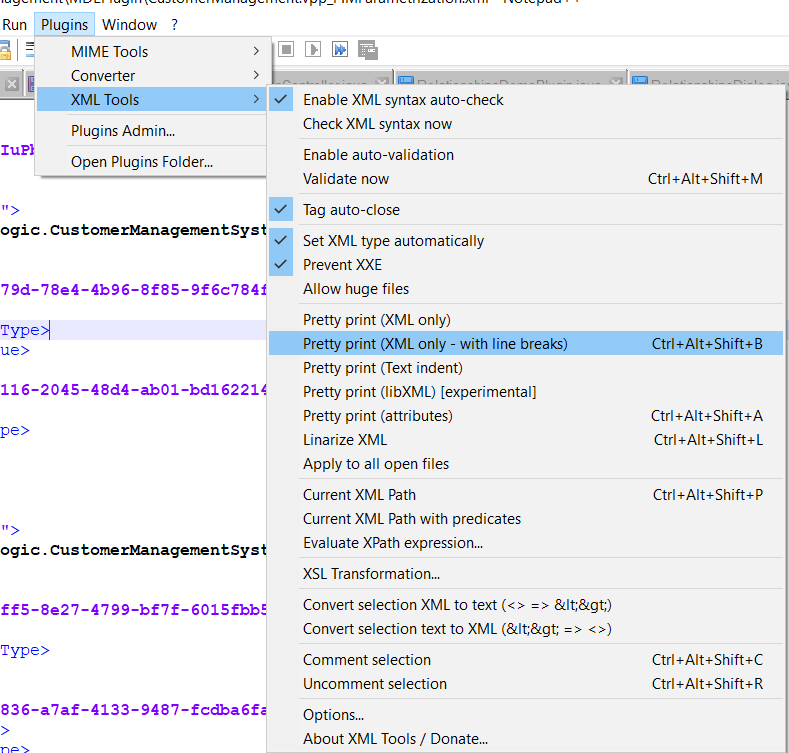
 « C:/Users/hamza/AppData/Roaming/VisualParadigm/plugins »

* Pour faire le build et le déploiement du plugin après chaque changement dans le code. Il faut cliquer sur la flèche rouge dans l’onglet « ANT TARGET RUNNER »



1. Remarques:

* Pour visualiser les fichiers XML j’ai utilisé « Notepad++ ». Le plugin sauvegarde les fichiers XML dans une seule ligne, pour les rendre lisible il faut installer le plugin «XML Tools » dans Notepad++.



1. Debugging:

* XML : problème de création de fichiers XML dans certains projets.
* UI : Problème dans les champs « description » lorsque le texte dépasse la largeur du « textbox ».
* UI : bouton pour choisir le chemin vers le modèle de transformation.
* Diviser le tableau « DesignConcernMarkingHandler » en deux : un tableau pour les stéréotypes et l’autre pour tagged value.
* Dans le tableau « DesignConcernMarkingHandler », ajouter le champ « Concept ».
* Vérifier le nom de « AssociationEnd » dans l’arborescence.
* Déplacer les modèles de transformation vers le dossier « Code » manuellement ou automatiquement?
* Lors la parameterisation d’un élément UML avec deux PDMs, les valeurs sont perdues dans la première « Save »
* Paramétrisation de PIM : Les énumérations sont considérées comme des classificateurs dans l’arborescence des éléments UML.

1. Questions :

----------------------------------------------------------------------------------------------------------------------

Rencontre avec prof 25 décembre – Questions et explication de l’outil MDE

1. Constructeur dans le modèle UML ou les templates de transformation ?

**Réponse** : généralement on ne fait pas les constructeurs dans le modèle UML, mais si on a besoin des constructeurs spéciaux on va les faire dans le modèle UML. Donc on va les faire dans les templates de transformation selon l’architecture.

Pour l’architecture clean on va faire constructeur par défaut (exigé par EF Core) et constructeur pour initialiser les propriétés de la classe.

1. Navigation entre deux classes et le choix ou mettre la clé étrangère ?

**Réponse** : On va faire la navigation dans les deux sens.

Le choix de la clé étrangère c’est une décision de conception donc c’est la paramétrisation (stereotype). On cas d’une relation 1—1 on doit choisir ou mettre la clé étrangère. Donc dans une association on va choisir « associationEnd » qui va avoir le stereotype « HasForeignKey ».

1. Processus de la spécification de l’architecture :

* Dans chaque projet (ex : Core) voir les « stereotypes » ou les « tagged values » qu’on est besoin. (Ex core : Entity, aggregateRoot, valueObject)
* La définition ou la réutilisation des bibliothèques (d’autres noms : accélérateurs, sous architecture…) par exemple accélérateur des design patterns.
* Donc la possibilité d’implémenter plusieurs spécifications d’architecture dans une seule solution. Par exemple on a l’architecture principale et l’architecture clean et on peut réutiliser d’autres spécifications d’architecture (biblio, accélérateur, PDM, profil, morceaux…) par exemple pour les design pattern.
* Donc profil (clean) est composé de sous profils (design pattern, cas des test).
* Exemple d’accelereteur : design pattern observer je vais l’appliqué. Contient les stereotype « subject », « observer ».
* Why concept has design concerns ??
* The concept specify which compnent will be generated. For example if concept has type of classifier(class) it will generate a class, but design concerns will tell where the mark will be done in the UML element in VP. For example FK (foreign key) will be marked in association end.

----------------------------------------------------------------------------------------------------------------------------

Use saxon as the transformer in java (Doesn’t work for now):

1. Add saxon.jar to classpath

<https://stackoverflow.com/questions/50232557/visual-studio-code-java-extension-how-to-add-a-jar-to-classpath>

1. Use saxon in Java

<https://stackoverflow.com/questions/2968190/how-to-select-saxon-transformerfactory-in-java>

Create custom ribbon

<https://knowhow.visual-paradigm.com/openapi/custom-ribbon/>