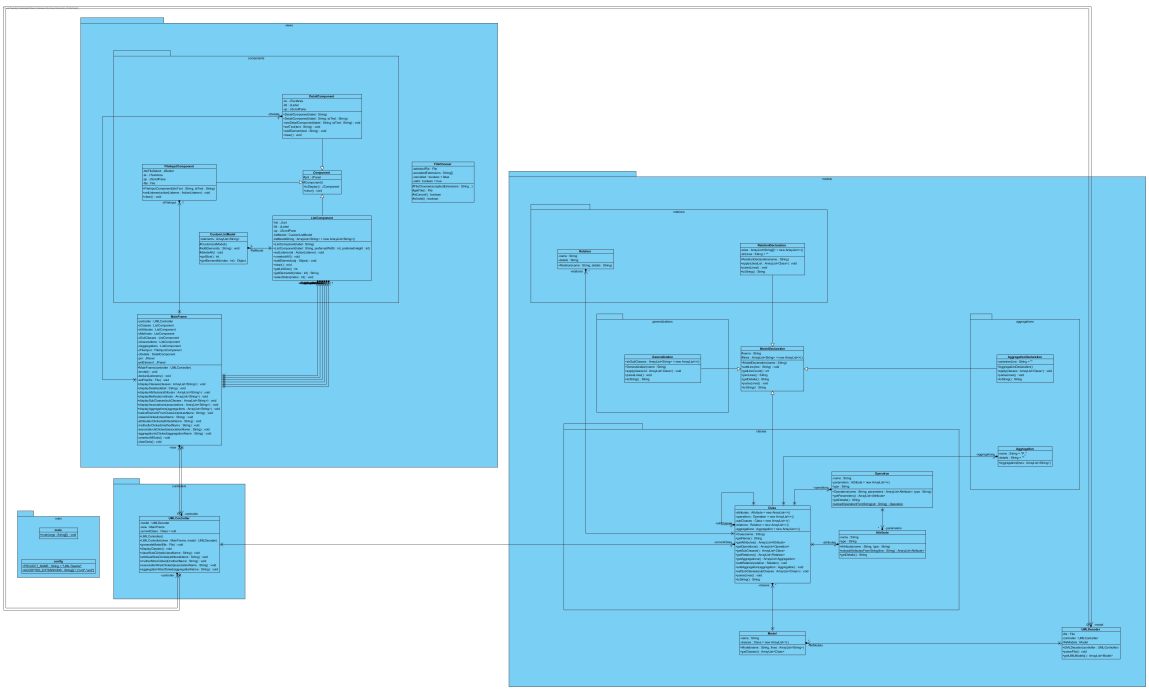
Projet 1
IFT3913

Olivier Hassaoui St-Amour (20078289)
Francis Boulet-Rouleau (20067884)

4 octobre 2018

1 Conception

Pour ce projet, nous avons décidé d'adopter une conception MVC (Modèle-Vue-Controlleur) pour le développement de l'application. Le but était séparer les éléments de l'interface des éléments de logique pour pouvoir gérer les deux de la façon la plus indépendante possible. Le constructeur étant utilisé comme pont entre le modèle et la vue. Ainsi, le constructeur recevrait des requêtes de la vue demandant des informations du modèle. Le constructeur demanderait ensuite au modèle de générer les données demandées et les retournerais à la vue pour les afficher. Voici le diagramme de classe qui illustre la conception.



Les versions aggrandies de chacune des composantes du projet sont inclues dans les pages suivantes. Pour permettre une meilleure visuallisation, un fichier .jpeg est remis avec le projet.

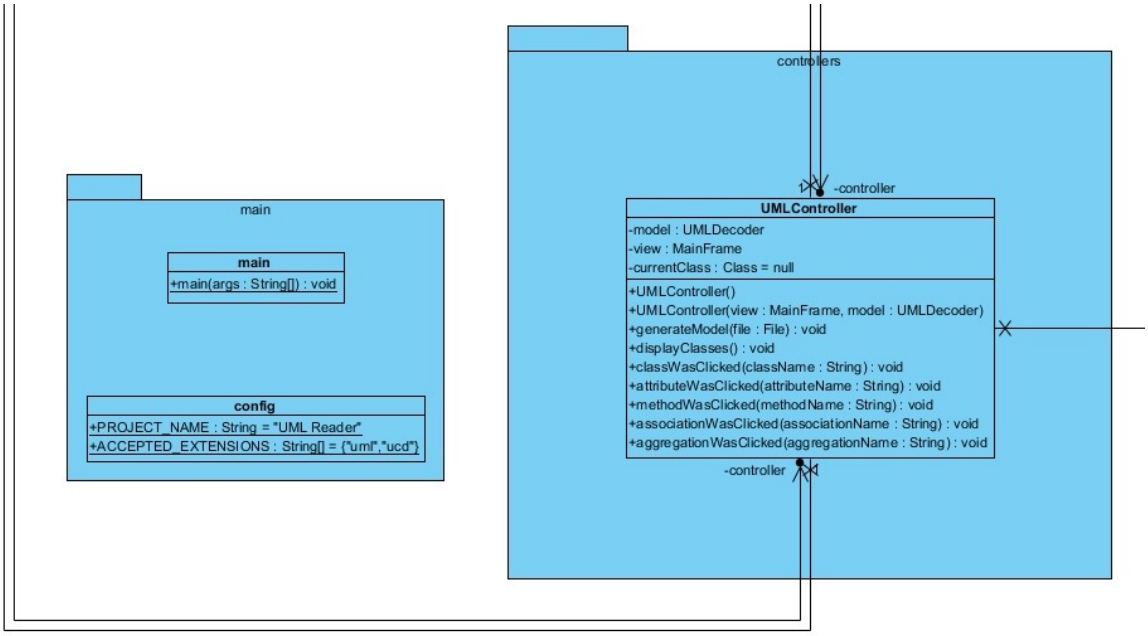


Figure 1: Diagramme de classe partielle représentant le point d'entré et le controlleur.

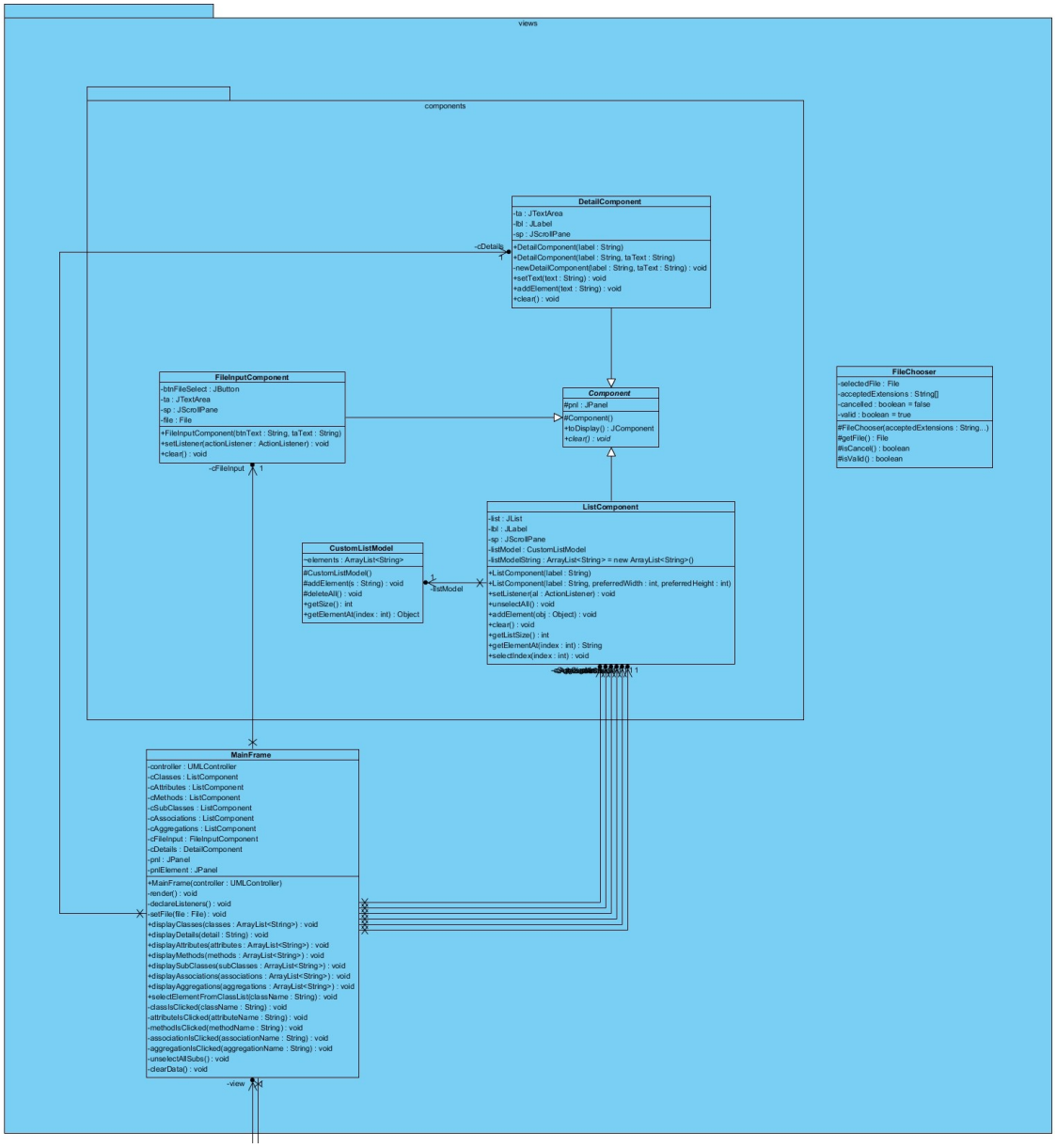


Figure 2: Diagramme de classe partielle représentant la vue.

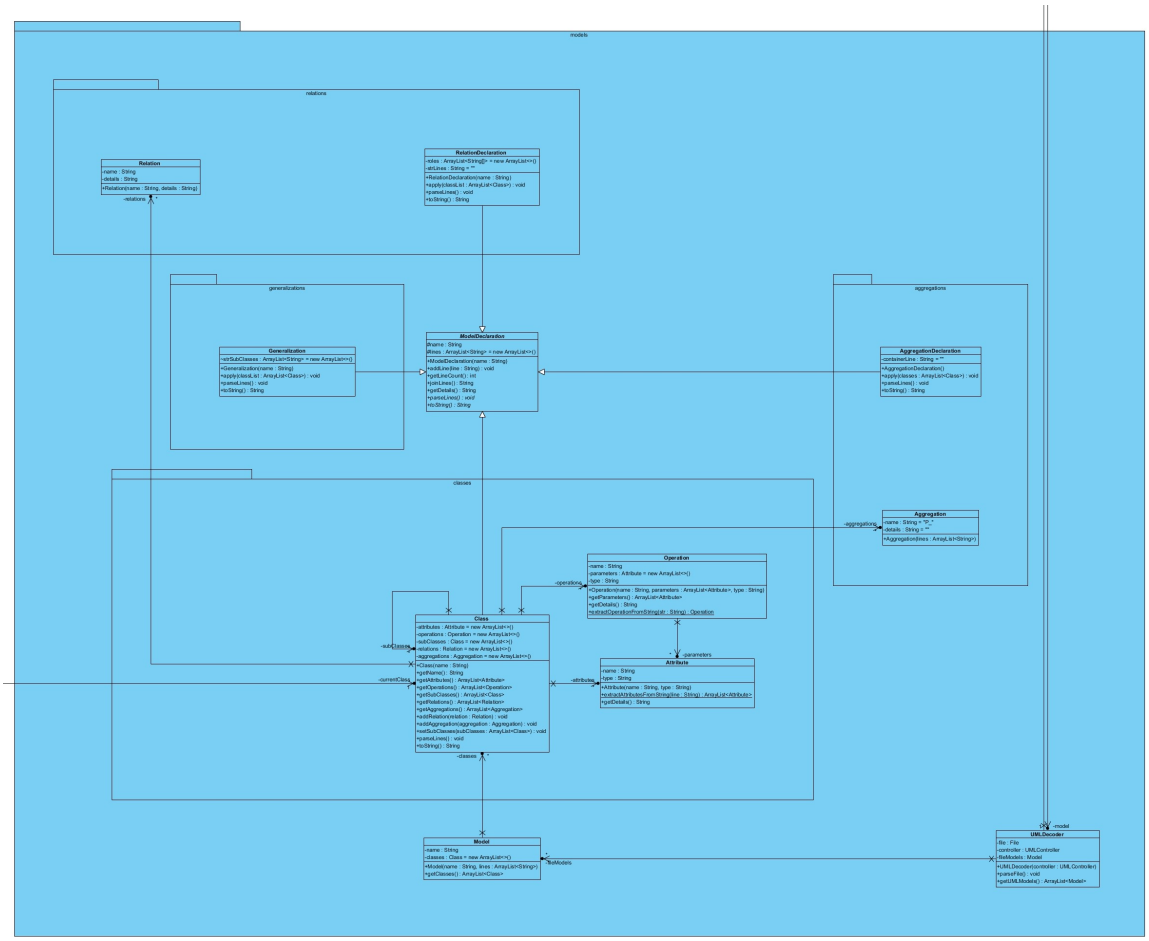


Figure 3: Diagramme de classe partielle représentant le modèle.

2 Implémentation

Lors de l'implémentation, nous avons utiliser une approche qui priorisait l'encapsulation pour permettre une meilleure lisibilité de code et une maintenance de celui-ci plus facile.

2.1 Modèle

Pour la partie du modèle, son point d'entrée est la classe *UMLDecoder* qui lit une première fois le fichier donné et y trouve tous les modèles UML qui s'y trouve. Par la suite, il crée le nombre d'instances de *Model* correspondant au nombre de modèles UML lus. La classe *Model* lit les lignes de caractères qui lui sont données et cherche dans ceux-ci les mots-clés qui indiquent la déclaration d'un nouvel élément UML. Chacun des élément a une classe propre et le role de chaque classe est de *parser* les lignes qui lui est donnée et populer ces différents attributs. Une liste de *Class* est ainsi générée pour que le controlleur puisse l'utiliser pour populer la vue.

Nous avons tenté de réduire le plus possible la duplication de code dans ce projet. De ce fait, les classes *Attribute* est utiliser à deux moments différents : pour définir les attributs de *Class* et pour définir les attributs de *Operation*. Dans le même ordre d'idée, la classe *Class* est utilisé premièrement pour instancier les classes UML et deuixièmement pour instancier les sous-classes de chacune des classes UML.

Pour l'instant, les spécifications expliquaient qu'il n'y aurait qu'un modèle par fichier. Cependant, le modèle est fait de sorte à lire plus qu'un modèle s'il y en a plusieurs de définis dans le fichier.

Il est à noté que le *parsing* se base de façon importante sur le format UML. Par exemple, lorsqu'on croise le mot-clé "ATTRIBUTES", le parseur examine la ligne suivante pour trouver les attributs de la classe. Ainsi, si les attributs se trouvent sur la même ligne que le mot-clé, ils sont ignorés par le parseur.

2.2 Vue

L'outil d'interfrace utilisé pour ce projet est la libraire *Swing*.

Le point d'entrée de la partie vue est la classe *MainFrame*. Lors de son instanciation, la vue crée d'abord toutes les composantes requises sans y afficher de données (évidemment). Les composantes sont créé grâce à la classe abstraite *Component*. Cette classe sert à être héritée par les autres composantes d'affichage. C'est composantes sont *FileInputComponent*, *ListComponent* et *DetailsComponent*. Dans l'ordre, ils servent à :

1. Créer un bouton suivi d'un champs texte pour le téléchargement de fichier.
2. Créer une liste dans un *ScrollPane* avec une étiquette au dessus.
3. Créer une zone de texte non-éditable avec une étiquette au dessus.

Chacune de ces classes est requise d'avoir une fonction *toDisplay()* qui peut être appelé lorsque l'on veut ajouter la composante à un *JFrame*.

L'instance de *MainFrame* utilise ensuite ces composantes pour construire l'interface suivante :

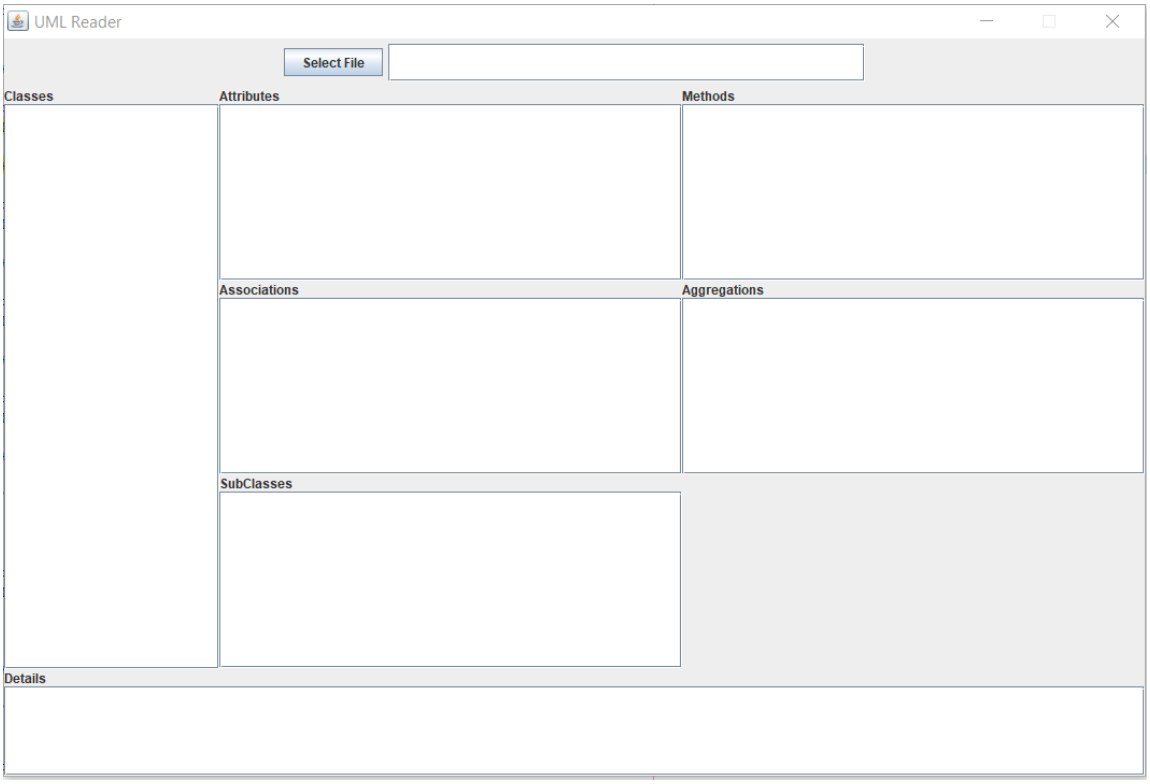


Figure 4: Interface du projet de parseur UML

Il est à noter que les *Component* ne sont pas responsables de déclarer les actions que des interactions avec leurs éléments graphique produisent. Ainsi, si on ajoute une composante *FileInputComponent* à la fenêtre sans rien faire d'autre et qu'on clique ensuite sur le bouton de la composante, rien ne se produit, il faut spécifier que l'action effectuer lors du clique est la création d'une fenêtre de *FileChooser*. De ce fait, c'est la classe *MainFrame* qui, lors de l'instanciation des *Component*, leur lie explicitement des *ActionListenent*, *KeyListener*, etc.

2.3 Gestion des erreurs

La plupart des erreurs que le système est capable d'intercepter sont géré par la vue et le controlleur. La vue s'occupe de s'assurer que le fichier fournit à la bonne extension et est non vide et le controlleur vérifie que le *Model* généré n'a pas d'erreur. En effet, si le fichier à la bonne extension et est non-vide, mais tout de même invalide, il est envoyé malgré tout à la classe *UMLDecoder* qui, en cas de malformation, ne générera pas de donnés. C'est ensuite au controlleur à vérifier que des données ont bel-et-bien été générées, sans quoi, il ordonne à la vue d'afficher un message d'erreur.

3 Conclusion

En bref, nous avons opté pour une conception MVC qui facilite la maintenance du projet en permettant une meilleure lecture du code et en permettant d'ajouter des fonctionnalités de façon relativement facile. En effet, si un nouvel élément descriptif UML s'ajoute, il sera assez simple de l'inclure dans la partie Modèle et également simple dans la partie Vue. Dans le Modèle, l'ajout demanderait la création d'une nouvelle classe pour décrire le nouvel élément et dans la Vue, il suffirait d'ajouter un *Component* à l'interface.

Aussi, le couplage entre les composantes du projet est faible. En effet, on peut modifier la Vue sans devoir changer quoi que ce soit au Modèle. On peut aussi faire l'inverse avec modification relativement minime de la Vue. Également, les éléments du Modèle sont également relativement indépendants entre eux, permettant de changer leur implementation individuelle sans affecter le comportement du programme en général.

4 Manuel d'utilisateur

L'interface montrée à la figure 4 est relativement instinctive à naviguer. D'abord, il faut selectionner un fichier, puis, si le fichier est valide, une liste de classes s'affichera (sinon, un message d'erreur). On peut ensuite naviguer à travers le modèle UML est double cliquant sur une classes, ce qui affichera ces attributs et autres éléments descriptifs dans les bons champs. Chaque champs peut être double-cliqué pour afficher ces détails (s'il y en a).

L'utilisation a été testée avec la version 1.8.0_181 de Java (JRE) et le Java Development Kit (JDK) 1.8.0_144 Les lignes de commandes pour démarer le projet

sont les suivantes.

```
cd ./$DIRECTORY/dist
java -jar UML_reader.jar
```