Devoir 2 IFT 3913

Ludovic André et Gevrai Jodoin-Tremblay

Remis le 2 Novembre 2017

Introduction

Description du logiciel

Ce logiciel affiche une interface usager simple, très semblable à l'exemple d'interface usager de l'énoncé du travail pratique. Le bouton "Charger fichier" ouvre une fenêtre système permettant de sélectionner un fichier *.ucd*. Lorsqu'un fichier est choisi, il est *parsé* et ses éléments sont affichés dans les champs correspondants de l'interface. L'usager peut selectionner les différents éléments de l'interface pour avoir plus de détails sur ceux-ci, qui s'afficheront dans la boîte *détails* de l'interface.

Ainsi, en sélectionnant une classe dans la boîte de gauche, le programme affiche toutes les métriques pour cette classe dans la boîte de droite. Il est aussi possible d'afficher la définition d'une métrique dans la boîte de détails simplement en la sélectionnant.

Il y a certaines métriques qui peuvent retourner une valeur négative qui vaut -1. Ces métriques sont *DIT*,*CLD* et *NOD*. Cela veut dire que lors du calcul le système a détecté qu'une classe A à un enfant classe B et cette Classe B à un enfant A. À ce moment, les métriques sont automatiquement mises à -1, car il ne peut pas dire avec certitude s'il se situe à la racine ou à la feuille.

Finalement, l'utilisateur peut exporter toutes les métriques de toutes les classes du modèle courant en appuyant sur le bouton *Calculer métriques*. Une boîte de dialogue permettra ainsi de sélectionner la destination et un fichier *.csv* sera créé contenant le résultat de toutes les métriques pour chaque classe.

Arborescence de l'archive

makefile

Nous avons opté pour un *makefile*, à fin de rendre la compilation et l'exécution la plus facile possible sur les machines du DIRO. Un simple *make* compile l'entierèté du logiciel et l'exécute. Pour voir les autres options, *make help* explique les différentes commandes possibles.

src

Contient tous le code du projet, sans exception.

La fonction *main* de notre programme se trouve dans la classe *App.java*, que nous avons gardé la plus minimale possible.

__build

Histoire de garder notre dossier src assez propre, la compilation des fichiers *.class* se fait vers ce dossier. *make clean* supprime ce dossier.

rapport

Tout simplement toutes les ressources nécessaires à l'impression de ce merveilleux rapport !

tests

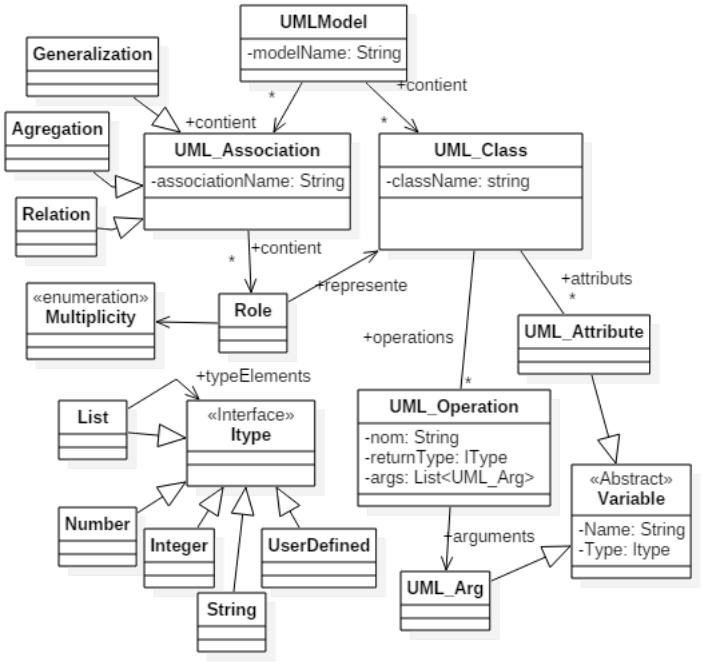
Tout le nécessaire pour rouler notre batterie de test grâce à *JUnit*. Il est primordial de ne rien modifier dans ce dossier pour garantir la validité de nos tests.

Conception

Modifications depuis version 1

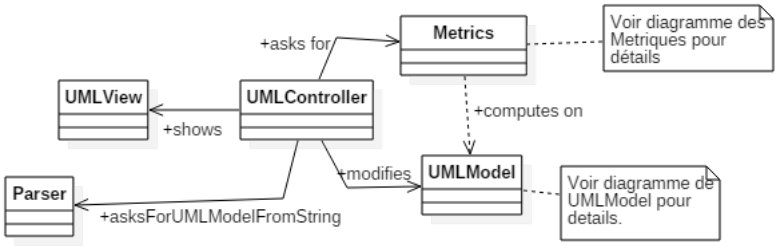
La grande majorité de notre *backend* n'a pas été modifiée, nous avons simplement ajouté les métriques. En effet, le paquet *uml*, et le diagramme de classe de celui-ci, n'ont subi aucun changement depuis le TP1.

Le pattern MVC étant très modulaire, nous avons pû simplement ajouter les éléments nécessaires au contrôleur et à l'interface sans véritable embûche.



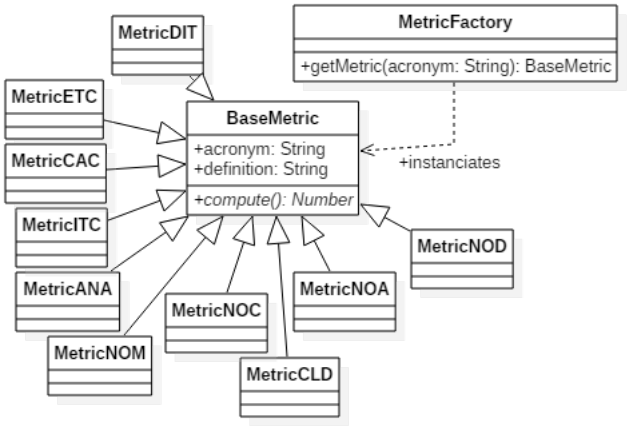
MVC

Ce patron s'est avéré être un excellent choix initial pour notre implantation. Les ajouts apportés au code pour les nouvelles fonctionnalités étaient isolés du code déjà existant. Il ne suffisait que d'ajouter quelques fonctions et *Listeners* au contrôleur pour les implémenter.



Ajout des métriques

Puisque nous avions fait attention d'appliquer au meilleur de nos connaissances les *patterns* de conception orienté-objet, notre code était assez facile à étendre avec des nouvelles fonctionalités. En effet, pour représenter les calculs de métriques, nous avons créé une classe abstraite *BaseMetric* contenant tous les attributs et méthodes communes à toutes les métriques, avec la seule méthode abstraite `compute(UMLModel,UMLClass)`. Cette méthode, qui doit être définie par les sous-classes, est évidemment le calcul de cette métrique. Cette hiérarchie permet de facilement ajouter de nouvelles métriques en ne définissant que les parties importantes de celles-ci.



Aussi, comme on peut souvent le voir avec ce genre d'arbre de classes, une classe *MetricFactory* est utilisée pour instancier une métrique en particulier, simplement avec son acronyme.

Ces métriques sont tous simplement utilisées par le contrôleur principal sur le modèle *UML* chargé dans le programme. Il est à noter que le paquet *uml* n'a aucune connaissance des métriques.

Parseur

Nous avons réglé l'erreur de *parsing* causé par un fichier vide en entrée, ceci est maintenant traité comme un fichier valide.

Aussi, on affiche maintenant une petite fenêtre signalant à l'utilisateur une erreur dans le cas où le fichier d'entrée est invalide. De même, les fichiers *.ucd* inexistants affichent maintenant aussi cette fenêtre, avec un message approprié.

Tests

Pour faciliter la création et l'exécution de notre suite de tests, nous avons opté pour l'écriture de tests automatisés grâce à la librairie bien connue *JUnit*. L'immense avantage de cette approche est d'éliminer la lourde tâche d'effectuer des tests manuels à chaque fois que l'on effectue un changement dans le code. Aussi, puisque nous avions déjà les résultats de certaines métriques, un de nous a écrit les tests pendant que l'autre implémentait les dites métriques. On pourrait dire que cette méthode de travail est favorable car celui qui écrit les tests n'est pas biaisé par son propre travail.

Il est à noter que lorsque l'on compile avec la commande `make all`, la compilation ne réussira pas si un ou plusieurs tests échouent.

Parseur

Le fichier *ParserTest.java* teste notre parseur avec tous les fichiers offerts par le démonstrateur, et notre programme passe tous ces tests. Nous avons aussi ajouté quelques tests personnels, soit *BadFile.ucd* qui est simplement un exemple de fichier invalide devant retourner une erreur dans le parseur, ainsi que *LeagueMetriques.ucd* permettant de mieux tester certaines métriques.

Tous ces fichiers passe correctement nos tests.

Métriques

Les fichier *MetricsTestX.java* testent le calcul des métriques sur trois fichiers en entrée, et les Table 1 et 2 donne les sorties prévues de chacun pour chaque métrique.

Problème de bouclage - LeagueMetriques4.ucd

Lors de l'implémentation des métriques DIT,CLD et NOD, nous nous sommes heurtés à un cas extrême. Pour ces 3 métriques, nous devons faire une recherche itérative pour savoir le nombre de sous-causes directes et indirectes(NOD), le

Table 1 – Valeurs espérées des métriques pour les tests

Metrique	League.ucd Equipe	LeagueMetriques1.ucd Equipe	LeagueMetriques1.ucd Joueur
ANA	0.33	0.33	0
NOM	3	4	4
NOA	1	3	3
ITC	1	1	0
ETC	1	1	2
CAC	3	0	0
DIT	0	3	1
CLD	0	0	2
NOC	0	0	1
NOD	0	0	2

Table 2 – Valeurs espérées des métriques pour les tests

Metrique	LeagueMetriques2.ucd Equipe	LeagueMetriques4.ucd Equipe
ANA	0.33	0.33
NOM	3	4
NOA	3	3
ITC	1	1
ETC	2	1
CAC	3	0
DIT	1	-1
CLD	0	-1
NOC	0	1
NOD	0	-1

chemin le plus long entre une classe x et sa racine et le chemin le plus long entre la classe x et sa feuille la plus basse. Dans ces 3 cas, le problème se situe si on a une boucle dans les généralisations. Si une classe "A" a comme enfant une classe "B" et la classe "B" a comme enfant la classe "A", on a une boucle. Cela a causé une boucle infinie pour ces métriques. On peut le voir avec le fichier *League-Metrique4.ucd*. Pour remédier à ce problème, nous avons ajouté une condition de vérification de boucle dans notre arbre de généralisations. Lorsqu'il détecte cette boucle de généralisation, il affectera la valeur de -1 à ces trois métriques, signifiant ainsi qu'il y a un problème dans le graphe d'associations.

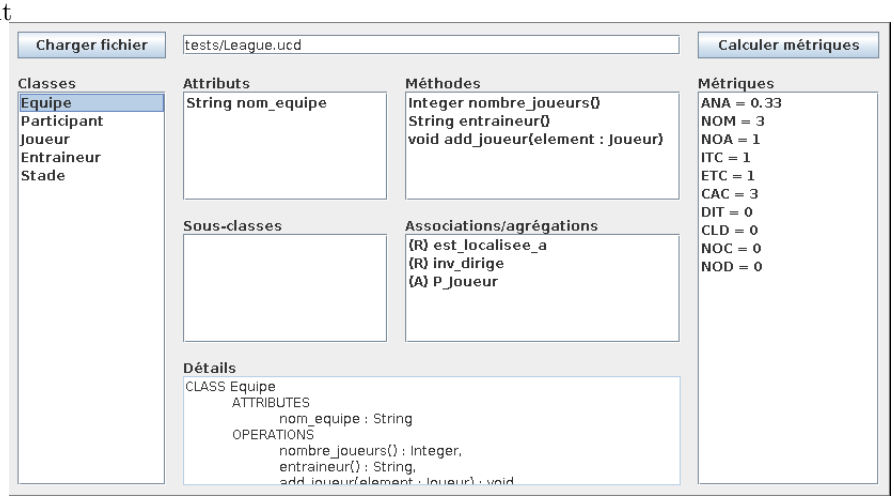
Lorsque l'on détecte un cas problématique d'une boucle à l'infinie pour la généralisation des classes, il serait possible d'afficher un message indiquant qu'il y a un problème avec cette généralisation. Ceci dit, on a préféré seulement montrer ce code d'erreur car le fichier d'entrée n'est pas fautif, mais bien la conception UML elle même.

Interface

Malheureusement, il est très difficile, voire impossible, de tester automatiquement une interface graphique, c'est pourquoi cette section contient quelques tests manuels.

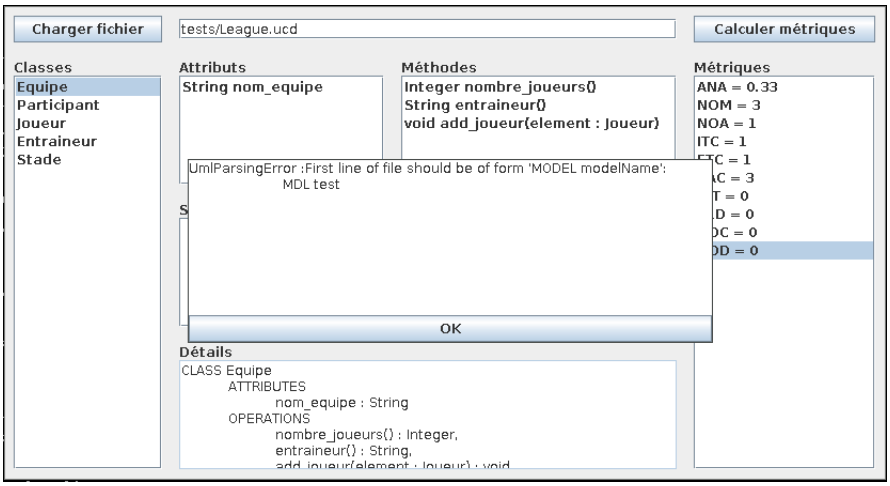
Fichier valide

- Fichier en entrée : tests/League.ucd
- Sortie prévue : Pas d'erreur et affichage des informations valides comme suit



Fichier invalide

- Fichier en entrée : tests/BadFile.ucd
- Sortie prévue : Popup d'erreur affichant un mesage à l'utilisateur, comme suit



Fichier csv

- Fichier en entrée : League.ucd

- Action nécessaire : Cliquer sur le bouton *Calculer métriques*, choisir le fichier de destination et sauvegarder.

- Sortie prévue : /tests/League-result.csv

Développements futurs

Il serait très utile de pouvoir modifier un modèle UML directement à partir de l'interface, et permettre l'exportation du modèle ainsi modifié vers un fichier *.ucd*.