Université de Montréal

Département Informatique et Recherche Opérationnelle

IFT 3913 - Genie Logiciel : Qualités métriques

Travail Pratique 2 - Parseur de fichier UML

Rapport

présenté par

Mohamed SARR
Kevin Palmang KOMBATE

Table de matières

A. INTRODUCTION

Le deuxième incrément du travail pratique Parseur de fichier UML consiste à ajouter une fonctionnalité permettant le calcul de certaines métriques sur les classes parsées par le parseur. Notre parseur nous permet par la même occasion de produire à la demande un fichier CSV (séparation par virgules) contenant pour un schéma, une matrice ayant pour lignes les classes et les colonnes les métriques.

B. CONCEPTION PROJET

1. CODE

Pour la deuxième partie de partie du travail pratique, il consistait à intégrer une nouvelle classe métrique. La modularité de la première version de mon code fourni dans le cadre de la remise du travail pratique 1, m'a permise de rajouter des fonctions sans toucher aux anciennes composantes. Grâce à cette modularité, j'ai pu étendre mon projet sans pour autant avoir peur de causer une brisure. Afin d'initialiser le calcul des métriques, il faut créer un nouvel objet Métrique et lui passer comme arguments le Model déjà parsé ainsi que le nom de la classe dont on veut calculer les métriques. Tous les calculs des différentes métriques sont définis par les fonctions portant leurs acronymes respectifs. Ces calculs sont exécutés à chaque fois que l'on exécutés à chaque fois que l'on sélectionne une classe.

Vous trouverez que la majorité des méthodes calculant les métriques sont définies deux fois, une fois sans paramètre et l'autre avec un paramètre (la classe sélectionnée). La première méthode sans argument, sert d'interface publique pour les autres classes du programme tandis que la seconde méthode elle permet le calcul des métriques qui invoque le principe d'heritage. Ce choix permet un calcul récursif et facilite la complexité des calculs comparé à un code de calcul itératif contenant plusieurs boucles.

2. DIAGRAMME DE CLASSE

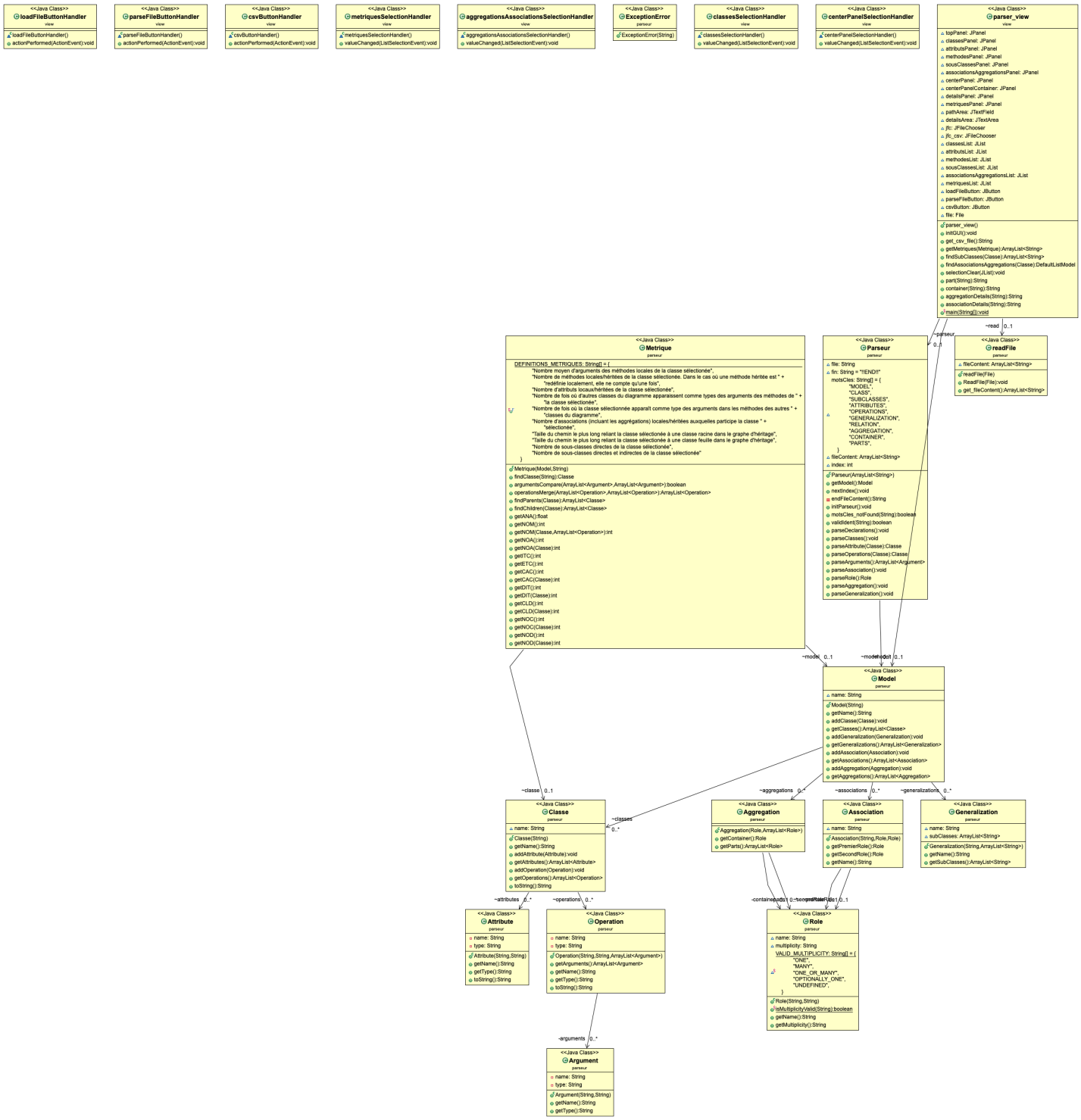


DIAGRAMME DE CLASSES

Notre travail pratique possède 14 classes réparties en deux packages “parseur” et “view”. Les différentes classes identifiées sont :

- Package “view”
  - parser_view : Classe
- Package “parseur”
  - Aggregation : Classe
  - Argument : Classe
  - Association : Classe
  - Attribute : Classe
  - Classe : Classe
  - ExceptionError : Classe
  - Generalization : Classe
  - Metrique : Classe
  - Model : Classe
  - Operation : Classe
  - Parseur : Classe
  - readFile : Classe
  - Role : Classe

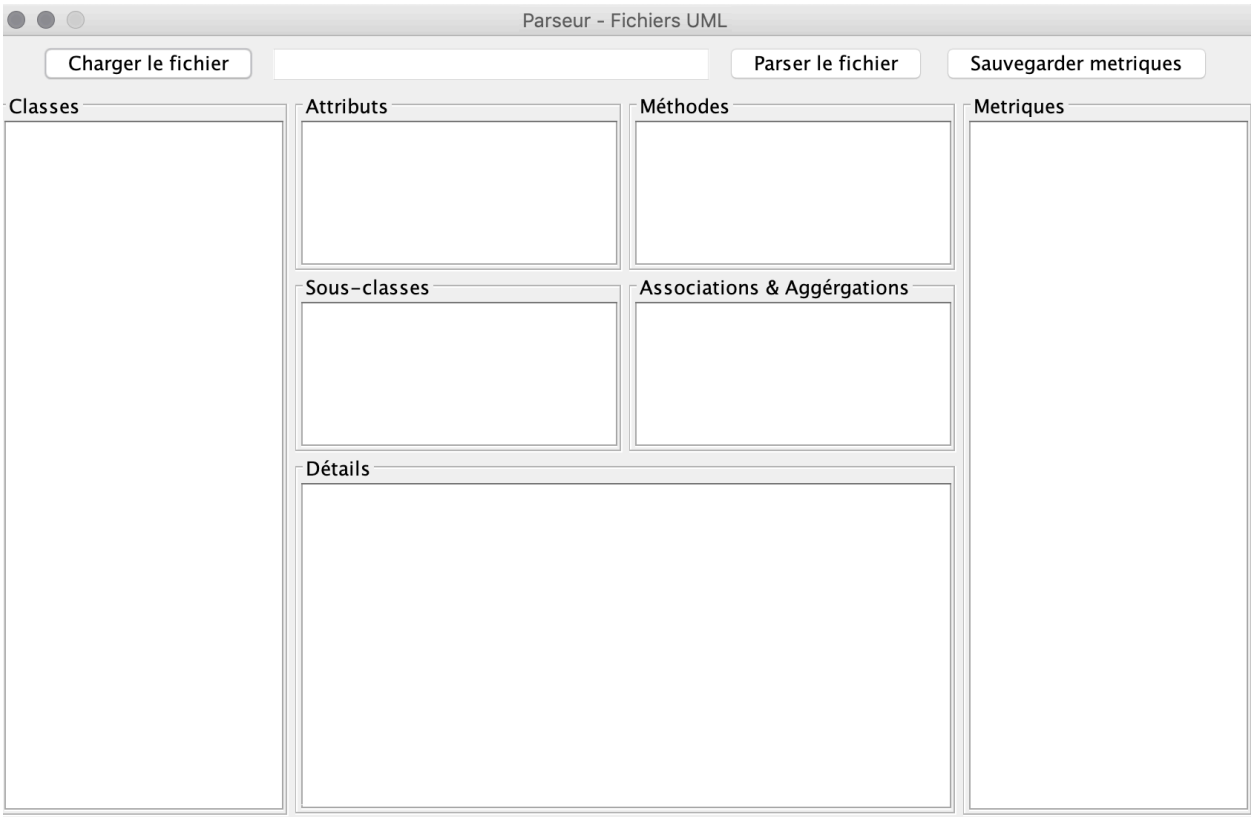
3. INTERFACE GRAPHIQUE

La définition de l’interface graphique s’est entièrement fait dans le package view et implémenté dans la classe parser_view. Chaque composante de l’interface graphique est initialisée en tant que variable globale (attributs) à la classe.

Tous les événements tirés lors de l’interaction avec l’interface graphique sont traitées par les différentes sous-classes de parser_view.java au lieu d’une fermeture pour chaque objet. Cette décision rend le code plus lisible et permet la réutilisation de fonctions s’il y a des éléments semblables. Comme parser_view::centerPanelSelectionHandler qui contrôle la selection des listes Attributs, Méthodes et Sous-classes.

Au lancement du logiciel, le programme appelle la fonction initGUI() qui, tout comme son nom l’indique, initialise la création de l’interface graphique et attache des contrôles aux objets créés. Ensuite, initParseur() attend que l’usager interagit, ce n’est que lorsque celui-ci appuie sur « Parser le fichier » que le programme continue.

Aussitôt le parsing complétée et sans erreur, le logiciel rempli la JList de Classe avec l'information parsée. Après, lors de la sélection d'une classe, on affiche les propriétés de celle-ci dans les listes appropriées (telles que les attributs, les sous-classes, les méthodes, les associations et aggrégations). Le champ « Détails » est utilisé seulement lorsque l'on choisit une aggrégation ou une association, car attributs, méthodes et sous-classes sont assez représentatifs.



INTERFACE GRAPHIQUE - PARSEUR

C. TEST

Pour la réalisation de nos test, nous avons utilisé la librairie JUnit 4. Nous avons tester dans la classe TestMetrique (dans le package Test) toutes les métriques dont nous avons implémenté le calcul dans la classe Metrique. Ils ont été réalisé dans de différentes conditions comme peut l'indiquer le noms des méthodes de la classe TestMetrique (exemple: public void **testNOM_avec_heritage_et_redefinition_operation ()**). Des commentaires sont disponibles au début de chaque méthodes indiquant le test qu'on réalise.

Toutefois, d'autres fonctionnalités du logiciel ont été aussi testé telles que le ReadFile qui lit un fichier, le découpe en morceaux et le mets dans un ArrayList et les erreurs syntaxiques avec les messages affichés.

D. MANUEL D'UTILISATION

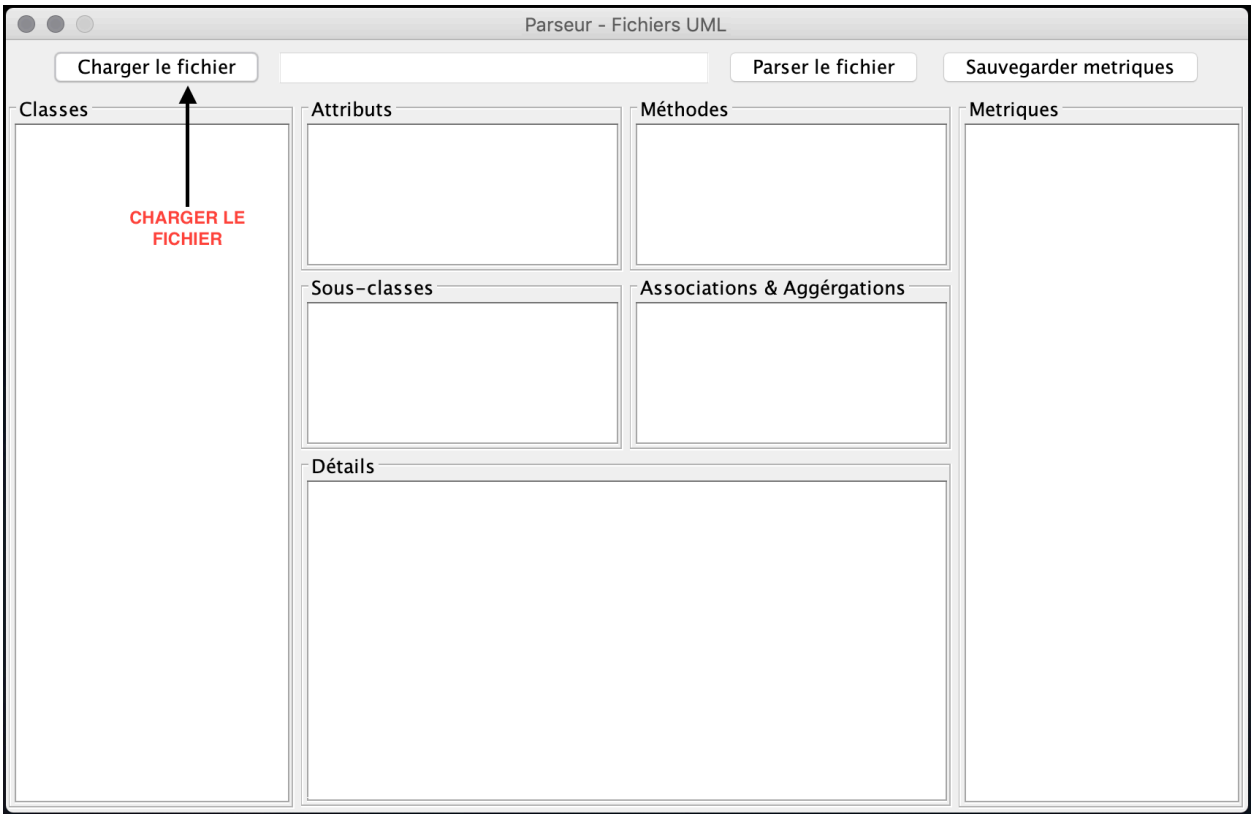
1. EXÉCUTION

Pour exécuter notre projet (Parseur de fichiers UML), il faut pour cela :

- Dezipper le fichier "Parseur-IFT3913"
- Ensuite, double cliquer sur le fichier "Qualités-Métriques-TP.jar"

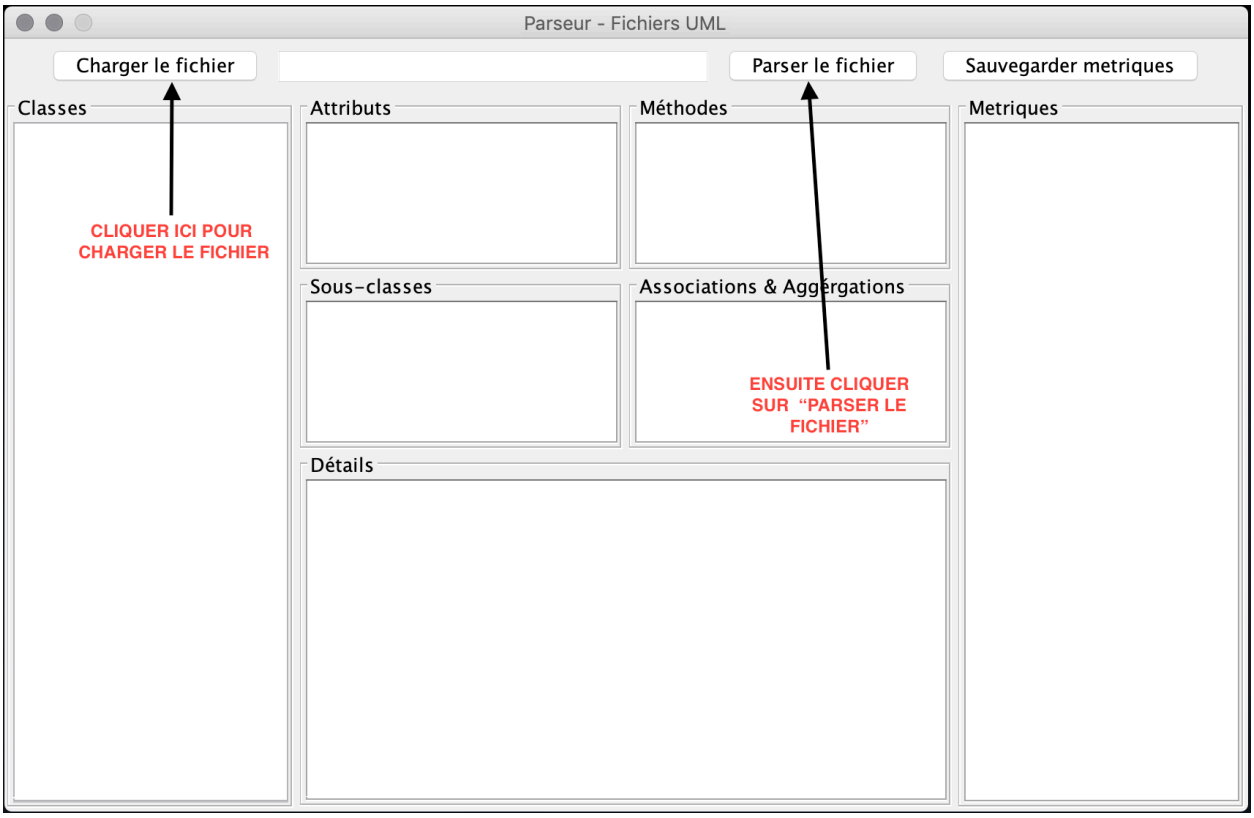
2. UTILISATION

Etape 1: Charger le fichier



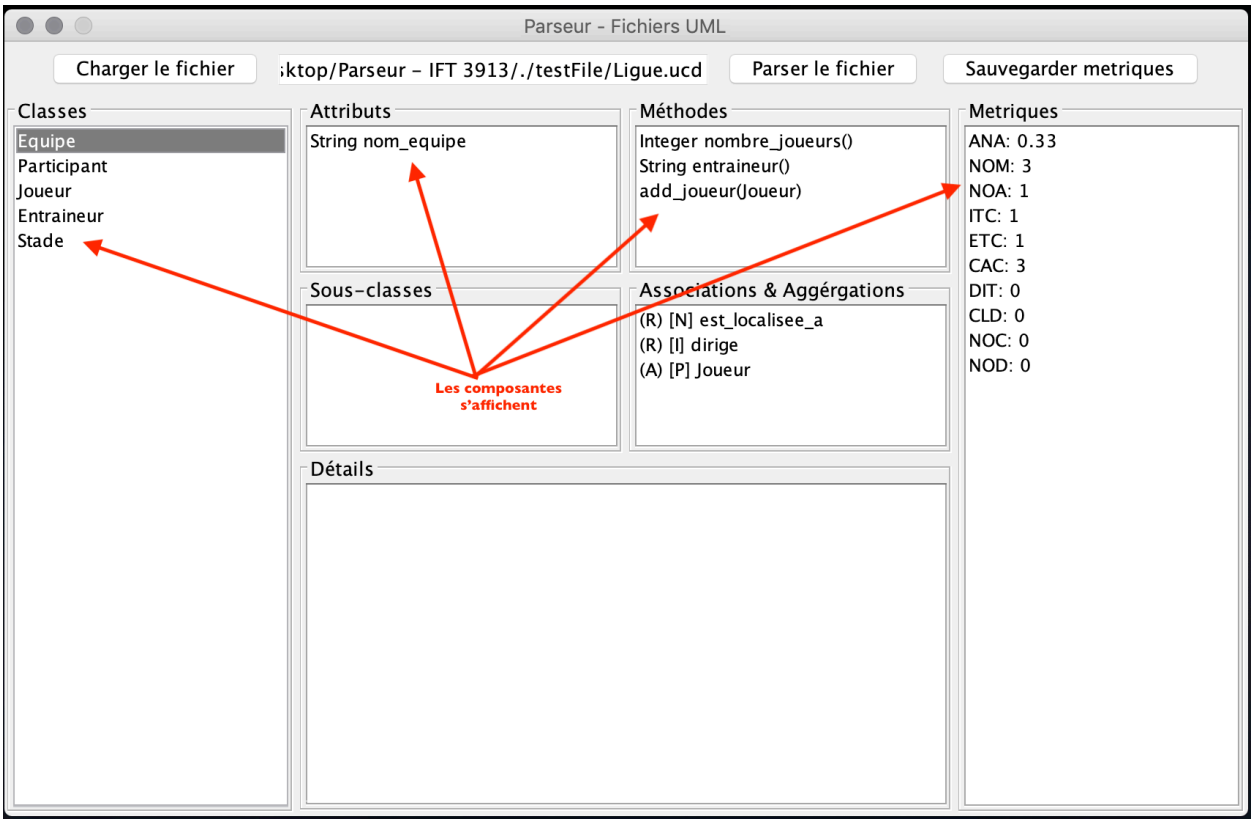
Pour débuter, il faut charger le fichier comme le montre l'image ci-dessus et cliquer sur le bouton "Parser le fichier".

Etape 2: Parser le fichier

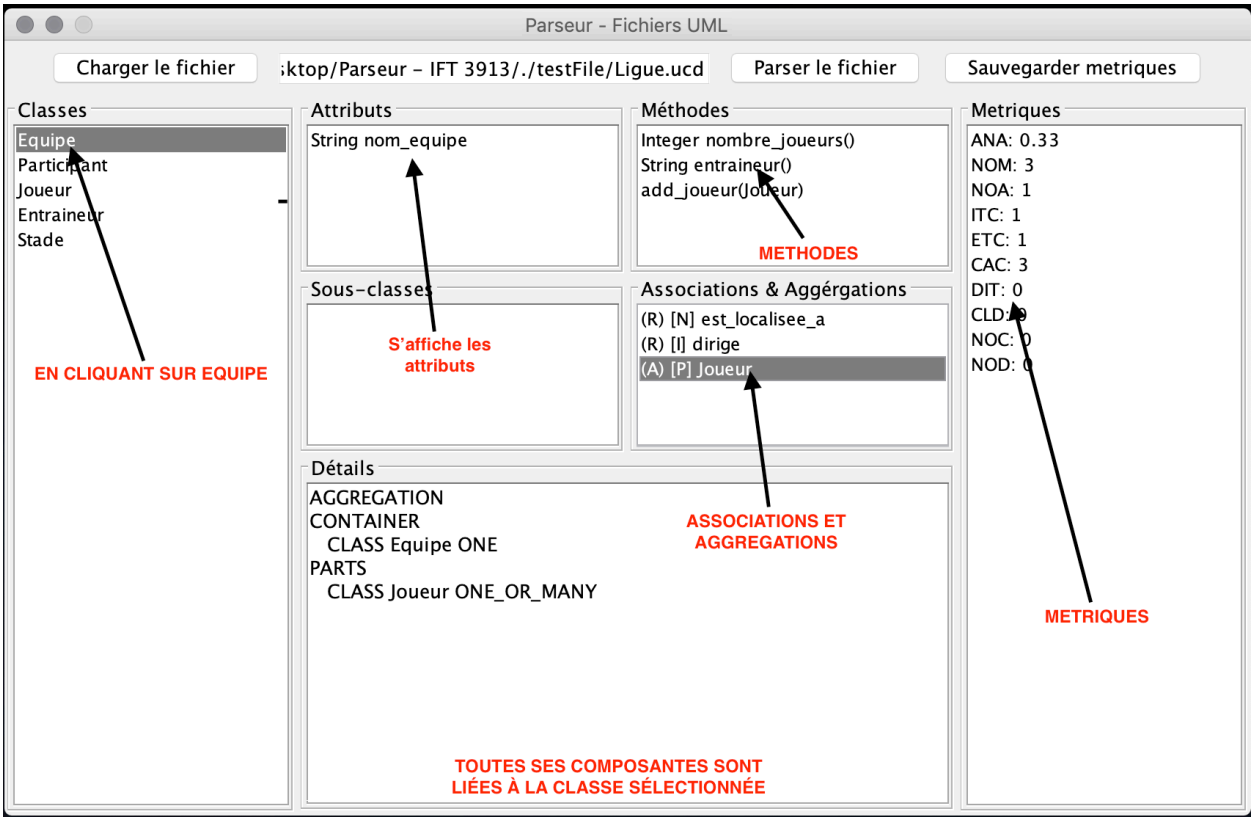


Si le parsing se complète avec succès, es classes seront listées dans la colonne de gauche. Ensuite, lorsque l'on sélectionne chaque classe, les composantes de ces classes et les métriques se calculeront et s'afficheront.

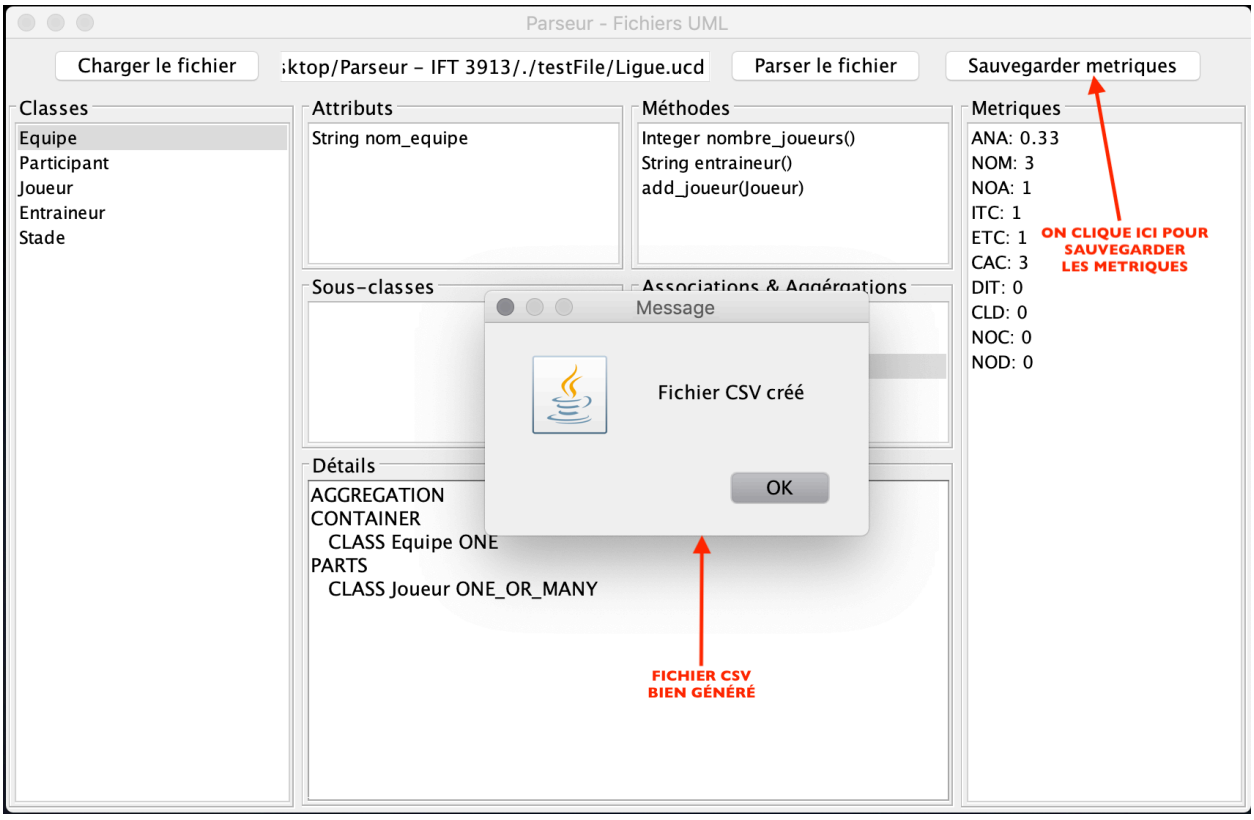
Suite étape 2 :



Etape 3 : Selection des composantes



Etape 4 : Sauvegarder les métriques



La génération du CSV se fait en cliquant sur le bouton “Sauvegarder les métriques”. Il faudra choisir un emplacement et un nom de fichier. Si le fichier ne se termine pas par .csv, le programme imposera cette extension. Un message de complétion s’affichera s’il n’y a pas eut d’erreurs.

E. AMELIORATION POSSIBLE

Pour un diagramme de classe UML, comme celui fourni pour le travail, le calcul de tous les métriques est instantané. Cependant, si le diagramme était plus complexe, il aura besoin de quelques secondes pour chaque classe, cela pourrait rendre une mauvaise expérience utilisateur.

Pour remédier à cette situation, une proposition serait de calculer tous les métriques de chaque classe immédiatement après le parsing du diagramme et de mettre en “cache” les résultats. Le calcul ne se ferait qu’une seule fois, donc un seul délai. L’expérience de l’interface graphique serait beaucoup plus agréable.