Thomas Neyraut
1812273

LOG6302 : Ré-ingénierie du logiciel

Rapport d'avancement N°2

INTRODUCTION

Ce document présente l'avancement qui a été effectué au cours de la troisième et quatrième semaine de projet. Durant ces deux semaines, l'objectif était de générer à l'aide de javacc et GraphViz le diagramme de classes d'Eclipse JDT core. Nous avons aussi profité de ces deux semaines pour améliorer la première étape du projet qui consistait à obtenir un ensemble de métriques du code source d'Eclipse JDT core. Dans une première partie, nous présentons l'ensemble de la méthodologie. Dans une seconde partie, nous présentons les avancées et les résultats que nous avons réalisés au cours des deux dernières semaines, ainsi que les difficultés que nous avons pu rencontrer.

I. METHODOLOGIE

Dans un premier temps, nous avons implémenté un nouveau visiteur (UmlVisitor.java) afin de récupérer l'ensemble des données nécessaires du code source d'Eclipse JDT core pour générer son diagramme de classes. Pour des soucis de lisibilité et de maintenabilité, nous avons décidé d'implémenter de plusieurs classes permettant la représentation de certains concepts.

En effet, nous avons implémenté la classe Classe nous permettant de représenter une classe avec l'ensemble des données la caractérisant dans un diagramme de classes, c'est-à-dire :

- Son nom,
- Ses attributs de classes (Variable),
- Ses méthodes (Method),
- Son héritage,
- Ses « implements ».

Nous avons aussi implémenté une classe Method pour représenter une méthode est l'ensemble de ses données présentes dans un diagramme de classes, c'est-à-dire :

- Son nom,
- Son accessibilité,
- Son type de retour,
- Ses paramètres (Variable),
- Son package.

Enfin, nous avons implémenté une classe Variable permettant de représenter un attribut de classe ou un paramètre d'une méthode, ainsi que l'ensemble de ses données visibles sur un diagramme de classes, c'est-à-dire :

- Son nom,
- Son accessibilité,
- Son type.

Une fois ces différentes classes implémentées, nous avons implémenté, et test au fur et à mesure notre nouveau visiteur UmlVisitor. L'implémentation de ce visiteur a consisté à « overload » les méthodes intéressantes de la classe AbstractVisitor dans l'optique de récupérer l'ensemble des données énoncées ci-dessus pour l'ensemble des fichiers .java composant Eclipse JDT core.

Par la suite, afin de générer le diagramme de classe nous avons implémenté la classe ClassDiagramGenerator permettant de générer les fichiers .dot exécutable par GraphViz et représentant le diagramme de classes. Le diagramme de classes étant trop imposant et GraphViz étant dans l'incapacité d'ouvrir un fichier .dot de trop grande taille, nous avons dû le diviser en plusieurs fichiers. Nous avions au départ décidé de diviser le diagramme de classes de manière arbitraire. Cependant, après avoir réussi à extraire le nom du package pour chacune des classes, nous avons décidé de diviser le diagramme de classes par package. Malheureusement, il est tout de même possible d'avoir un package contenant de très nombreuses classes. De ce fait, il n'est exclu dans certains cas d'obtenir des fichiers .dot de trop grande taille ne pouvant être ouvert par GraphViz. C'est ce qui est notre cas pour de nombreux packages du logiciel Eclipse JDT core.

Les différents fichiers .dot générés comportent l'intégralité du diagramme de classes d'Eclipse JDT core avec toutes les données nécessaires pour les différentes classes (données des classes, données des attributs de classe et données des méthodes). De plus, les fichiers .dot représentent aussi les liens d'héritage des classes, mais aussi leurs « liens implements », et aussi leurs liens de composition interne aux classes du logiciel.

Dans un second temps, nous avons modifier l'implémentation de la première étape du projet. En effet, nous avons implémenté la classe MetricFileGeneretor permettant de générer un fichier .json ou un fichier .csv comportant l'ensemble des métriques calculées par le visiteur ExampleVisitor. Le format csv a l'avantage de proposer une représentation des métriques bien plus lisible pour un utilisateur, et permet notamment d'effectuer bien plus facilement des traitements sur les métriques obtenues.

Enfin, nous avons modifié la commande permettant de lancer notre parseur. A présent, pour lancer notre programme il faut utiliser les commandes suivantes :

- java Javaparser1_7 path_name_file_java_or_list_file 0 json : permet de récupérer les métriques dans un fichier json,
- java Javaparser1_7 path_name_file_java_or_list_file 0 csv : permet de récupérer les métriques dans un fichier csv,
- java Javaparser1_7 path_name_file_java_or_list_file 1 : permet de générer le diagramme de classes via plusieurs fichiers .dot.

II. Resultats et discussion

Comme énoncé ci-dessus, le principal élément portant à discussion est le mode de séparation du diagramme de classes à mettre en place. En effet, la taille d'un fichier .dot trop importante empêche GraphViz d'ouvrir le fichier. De ce fait, nous avons dû réfléchir à comment diviser un diagramme de classes en plusieurs fichiers. En premier lieu, nous avons décidé de séparer le diagramme de manière arbitraire en incluant que 20 fichiers .java par fichier .dot. Par la suite, nous avons réalisé une séparation par package. Ce deuxième mode de séparation est nettement plus intéressant et pertinent. Malheureusement dans certains cas, il est probable d'avoir des packages contenant énormément de classes. De ce fait, il est tout à fait possible d'obtenir des fichiers .dot impossible à ouvrir dans cette configuration. Dans le cas d'Eclipse JDT core, en utilisant la séparation par package, on obtient de nombreux fichiers impossible à ouvrir. Ainsi, l'un de nos futurs objectifs sera de déterminer un mode de séparation approprié.

De plus, si le temps nous le permet, nous chercherons dans les semaines à venir à compléter le diagramme de classes que nous générons, en y ajoutant d'autres liens entre les classes. Il faudra cependant vérifier que cela n'encombre pas trop le diagramme de classes, pour que celui-ci reste lisible.

Durant ces deux dernières semaines, nous avons pu constater l'intérêt de proposer à l'utilisateur la possibilité de générer les métriques dans un fichier .csv. En effet, ce format propose une représentation parfaitement lisible pour un utilisateur, et permet de réaliser facilement des traitements sur les métriques obtenues. Ainsi, notre programme permet de générer les métriques sous format csv pour les utilisateurs, et au format json pour des logiciels.

Conclusion

Pour conclure, nous avons réussi à implémenter dans les temps un programme fonctionnant parfaitement, et permettant de générer le diagramme de classes d'un logiciel développé en java. Nous avons notamment pu améliorer l'architecture de notre logiciel dans l'optique de le rendre plus lisible et maintenable. Enfin, nous avons pu compléter l'extraction des métriques en proposant de les extraire dans un fichier au format csv. Cependant, nous n'avons pas encore réussi à concevoir un système de séparation du diagramme de classes en plusieurs fichiers, nous permettant de lire les fichiers quel que soit la configuration du logiciel. Ainsi, dans les prochaines semaines, l'une de nos tâches consistera à réfléchir sur cette problématique.