Document d'aide pour la reprise du projet
Projet UML Reverse

25 mai 2016

Version :	1.0
Date :	25 mai 2016
Rédigé par :	Anthony Godin Yohann Henry Florian Inchingolo
Relu par :	Nicolas Meniel

Table des matières

1 Introduction

2 Modèle

2.1 Projet

2.1.1 Présentation

Un objet **Project** est un conteneur qui stocke les différents diagrammes ainsi que les informations pouvant être utilisées dans plusieurs diagrammes à la fois. Actuellement, les seuls liens existants se font uniquement entre les différents diagrammes de classes. Un **ObjectEntity** représente une classe et tous les éléments qu'elle contient. Il stocke aussi les relations du diagramme de classes. La plupart des fonctionnalités se font par le biais de **DiagramVisitor** qui servent principalement à appeler les visiteurs spécifiques à chaque diagramme.

2.1.2 Les relations

À terme, les relations pourraient être aussi liées entre différents diagrammes. En effet, ces informations auraient pu avoir une utilité. Par exemple, connaître les relations d'héritage et d'implémentation aurait pu permettre des fonctionnalités supplémentaires comme faire apparaître une méthode d'une interface dans les classes l'implémentant. Une partie du code a donc été créé au niveau du modèle pour prévoir cet enregistrement des relations liables dans le projet, néanmoins la vue ne permet pas de réutiliser ces relations dans un autre diagramme ce qui rend cette disposition de projet inutile.

2.1.3 La sérialisation/désérialisation

Actuellement, la sauvegarde du projet se fait avec une sérialisation assez sommaire mais tout du moins efficace. Cependant, cette méthode présente la difficulté que nous avons dû adapter les méthodes **equals** et **hashcode** pour les simplifier au maximum. Il serait sage de modifier cette sérialisation pour la transformer en sérialisation XML. La sérialisation XML demanderait un travail supplémentaire au développement, néanmoins dans les faits la sérialisation serait lisible et moins contraignante pour les modifications du modèle.

2.1.4 Liaison avec les diagrammes

Pour relier les diagrammes avec les entités, nous avons utilisé une collection au sein des **ObjectEntity**. Cette méthode n'est pas particulièrement optimale mais nous l'avons réalisé tardivement. Une technique beaucoup plus efficace aurait été de laisser les éléments liés écouter les éléments par le biais de **PropertyChangeListener**.

2.1.5 Bilan

Cette partie du projet a été conçue de manière à pouvoir y ajouter aisément de nouveaux types de diagrammes sans modification si aucun élément ne devait être lié. De plus, la plupart des fonctionnalités étant effectuées par le biais de visiteurs, la principale modification pour ajouter un diagramme est de compléter les implémentations de **DiagramVisitor**.

2.2 Diagramme de classe

2.2.1 Présentation

Un **ClassDiagram** est un objet contenant la totalité des informations du diagramme de classe. Il contient principalement des vues sur les **ObjectEntity** et **Relation** contenus dans le projet. Ces deux éléments utilisent donc une référence vers un élément du projet. Les packages et les notes ne sont pas liés avec le projet.

2.2.2 Liaison avec le projet

Symétriquement avec le projet, il aurait été intéressant d'écouter les **ObjectEntity** avec des événements plutôt que de forcer les modifications par le biais de l'objet **Project**.

2.2.3 Initialisation

À la création d'un **ViewEntity** ou d'une **ViewRelation**, les constructeurs demandent de lier le **ClassDiagram** et ce dernier au **Project**. Le problème de cette solution pour garder une synchronisation avec **Project** est que si une entité venait à être supprimée du diagramme de classe, la référence vers cette entité deviendraient complètement inutilisable.

2.2.4 Bilan

Cette partie est particulièrement complexe pour gérer la synchronisation avec **Project**. Il existe plusieurs façons d'améliorer cette partie. Le plus simple pour y intégrer les packages liés et simplifier le diagramme de classe serait de la reprendre complètement.

2.3 Diagramme commun

Nous avons développé toute une série de classe réutilisable dans le paquetage fr.univrouen.umlreverse.model.diagram.common. Ces classes n'ont pas été utilisées au maximum par le diagramme de classe. Par contre le diagramme de cas d'utilisation les utilisent au maximum ce qui a réduit son développement. Ces classes peuvent être réutilisées par de nouveau diagramme.

2.4 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation a un fonctionnement très simple. Il utilise au maximum les classes du paquetage fr.univrouen.umlreverse.model.diagram.common. On lui associe un projet qui le stock.

2.5 Parseur

2.5.1 Grammaires

Le parseur utilise Antlr 4 (dont la Javadoc est disponible à ce lien : http ://www.antlr.org/api/Java/) pour obtenir depuis les grammaires des classes Java comme des visiteurs ou des écouteurs, permettant de récupérer facilement des informations depuis un fichier parsé. Ces grammaires sont dans le dossier **src/main/antlr4**. Il y a actuellement deux grammaires utilisées : **Java8.g4** et **PlantUML.g4**. Un exemple de grammaire simple ressemblant à une grammaire JSON est disponible via le fichier **TestGrammar.g4**. Cette grammaire peut être utilisée afin de comprendre la syntaxe d'Antlr et de comprendre comment fonctionne la génération des classes liées à la grammaire. La grammaire Java 8 a été récupérée depuis la liste des grammaires Antlr disponble à l'adresse suivante : https ://github.com/antlr/grammars-v4, et contient toutes les méthodes nécessaires pour récupérer n'importe quelle partie d'une classe Java. Ceci devrait donc permettre l'ajout d'un parseur de diagramme de séquence relativement simple. La deuxième grammaire est la grammaire PlantUML créée pour l'occasion, combinant le diagramme de cas d'utilisation et le diagramme de classes. Le point d'entrée de ces deux grammaires est **entryPoint**. L'ajout du support d'un nouveau diagramme PlantUML nécessitera la compilation de cette grammaire. Ceci peut facilement être fait avec un IDE ou par ligne de commande. Il est même possible de générer ces classes directement dans le bon paquet. Une option est nécessaire pour générer les visiteurs correspondants.

2.5.2 Classes du parseur

UML Reverse utilise actuellement les visiteurs générés par ces grammaires. Ils sont dans le package **fr/univrouen/umlreverse/model/io/parser/[LANGAGE]**. Pour expliquer comment fonctionnent ces classes, nous utiliserons l'exemple du parseur Java 8. La classe **Java8Parser** contient tous les contextes ainsi que de nombreuses classes internes permettant la manipulation de l'arbre sortant du parseur. Pour utiliser ces classes, il suffit d'étendre Java8BaseVisitor, contenant une méthode pour chaque règle de la

grammaire. Le contexte passé en paramètre permet d'accéder aux enfants (via des méthodes), ou aux feuilles du noeud (via des attributs). Par défaut, chaque méthode appelle ses enfants sans rien faire. Il suffit d'étendre ces méthodes afin de choisir le comportement voulu.

2.5.3 Classes créées pour UML Reverse

Dans les dossiers contenant les parseurs, les classes **PlantUMLDiagramVisitor**, **Java8ClassVsitor** et **Java8FieldsVisitor** définissent le comportement des visiteurs afin d'ajouter au modèle de diagrammes du parseur les informations voulues. Ces classes sont utilisées par **PlantUMLLoader** ainsi que **Java8Loader** afin d'importer les diagrammes.

2.5.4 Modèle du parseur

Le parseur remplit un modèle différent du modèle de l'application. Ceci a été fait afin de simplifier l'import des fichiers, et permet une modification après import et avant ajout.

2.6 Ajouts possibles au parseur

Les ajouts possibles sont la gestion de différents langages via des grammaires différentes ou l'ajout d'imports de nouveaux types de diagrammes PlantUML (ce qui sera possible une fois le modèle conçu). Le double modèle permettra de choisir les solutions en cas de conflit ou d'erreurs lors de l'ajout (comme des erreurs de syntaxe à corriger ou refuser la fusion des entités, actuellement automatique). Ceci est possible grâce au modèle du parseur, plus permissif et simple que le modèle du projet. Dans l'avenir, il pourrait être intéressant d'avoir des grammaires différentes pour chaque type de diagramme PlantUML, afin d'améliorer la maintenance. Par contre, cela implique de choisir le type du diagramme avant l'import du fichier. Par ailleurs, l'amélioration des performances et de la taille mémoire lors de l'import Java serait une bonne idée, quoique compliquée (la taille mémoire a l'air d'être liée au fonctionnement d'Antlr), afin de pouvoir importer des projets de grande envergure (actuellement, UML Reverse a besoin de 8 Go de RAM pour être importé).

3 Vue

La vue est codé avec JavaFX. Tous est regroupé dans le paquetage fr.univrouen.umlreverse.ui. On a découpé ce paquetage en deux autres :

- component : Contient tous les éléments graphiques des diagrammes (vue plus controleur). Les éléments ont été découpé en plusieurs classes qui s'héritent entre elles pour être le plus réutilisable possible. Ce qui permet par exemple de coder les classes NotesGraphic en seulement 36 lignes ou les relations graphique en 80.
- view : Contient toutes les classes de la vue sauf les éléments graphique des diagrammes. De même que précédemment, les classes du paquetage common sont réutilisables ce qui évite de devoir tous recoder quand on voudra rajouter un nouveau type de diagramme.

4 Bilan

Nous avons fait en sorte de coder des classes réutilisables pour faciliter et accélérer l'ajout de nouveau diagramme. Libre à vous de les réutiliser.