Document d'Architecture Logicielle
Projet UML Reverse

25 mai 2016

Version :	0.1
Date :	25 mai 2016
Rédigé par :	Yohann Henry Nabil Belkhous Stephen Cauchois Anthony Godin Florian Inchingolo Saad Mrabet Nicolas Meniel

Mises à jour

Version	**Date**	**Modifications réalisées**
0.3	20/05/2016	
0.2	22/01/2016	Architecture refaite dans l'intégralité
0.1	14/01/2016	Première version

Table des matières

1 Objectif

Ce document représente la structure générale du logiciel et les modèles de conception mis en oeuvre pour le réaliser. Il est destiné aux membres de l'équipe de développment, notamment aux concepteurs, ainsi qu'aux superviseurs du projet.

2 Les technologies utilisées

Nous allons utiliser différentes technologies pour la construction du projet.
Le projet est développé en Java 1.8. Le projet utilise :

- *JUnit* : Framework pour valider chaque classe par le biais de tests unitaires.
- *Maven* : Un outil pour la gestion des dépendances de l'application.
- *Antlr* : Un parser dans lequel nous pourrons définir les grammaires pour l'extraction des données d'un fichier java ou plantUml.
- *OpenJFX* : Une bibliothèque graphique libre parfaite pour ce projet. La bibliothèque permet d'associer à une entité du CSS, ce qui simplifie fortement notre travail.
- *SceneBuilder* : Logiciel permettant de créer des vues de façon ergonomique en drag and drop. Les vues sont créées en FXML.

3 Organisation des paquetages

```
src
├── model
│   ├── diagram ..... Modèle des diagrammes
│   │   ├── clazz
│   │   │   └── visitor ..... Permet le parcours du diagramme de classe
│   │   ├── common
│   │   ├── usecase
│   │   │   └── visitor ..... Permet le parcours du diagramme de cas
│   │   ├── util
│   │   └── visitor .... Visiteurs globaux. Choisit d'appeler un visiteur en particulier selon le type de
│   │                    diagramme
│   ├── io.parser ..... Parseur de plantUML et Java en diagramme
│   ├── project ..... Modèle d'un projet qui contient des diagrammes
│   └── util
├── ui
│   ├── component ..... Contient les composants utilisés dans la vue
│   │   ├── clazz
│   │   ├── common
│   │   └── usecase
│   └── view ..... Contient les vues principales de l'application avec leurs contrôleurs
│       ├── clazz
│       ├── common
│       ├── usecase
│       └── util
├── util
└── MainApp.java
```

4 Paquetage model

Le modèle est représenté par une classe principale : **Project**. Cette classe va servir à réunir tous les diagrammes et les éléments communs aux différents diagrammes. La sauvegarde et le chargement sont effectués par le biais d'une sérialisation/désérialisation de cette classe et de son contenu. La plupart

des fonctionnalitées ajoutées s'effectuent par le biais de visiteur qui appelleront à leur tour les visiteurs spécifiques à chaque diagramme.

4.1 Eléments liés

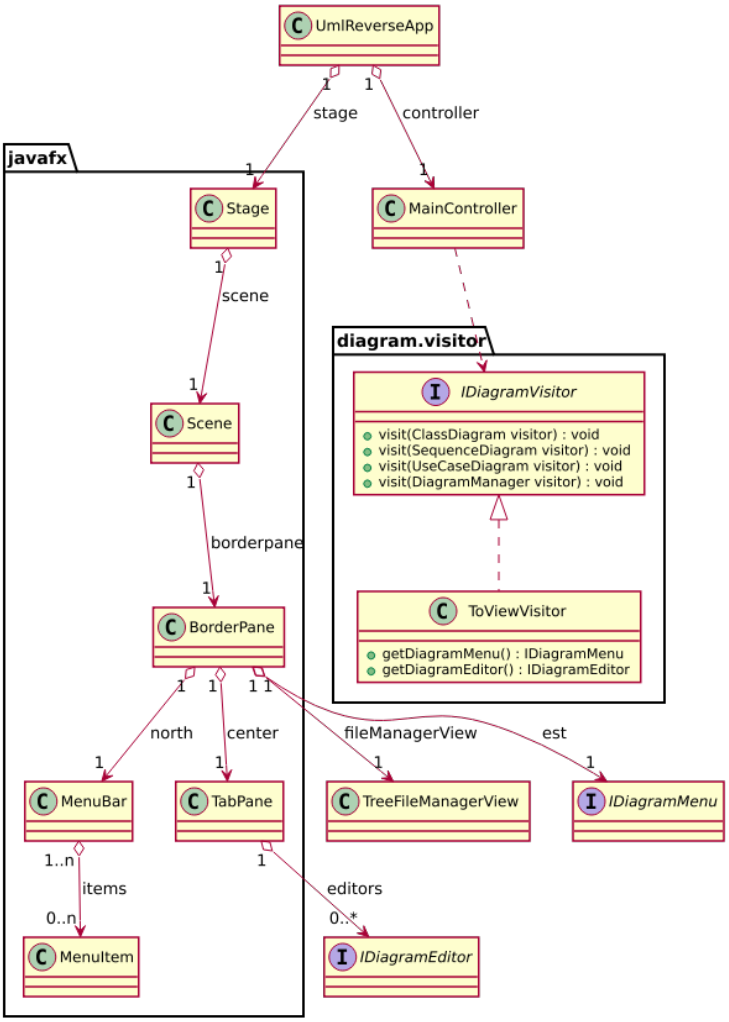
Le projet contient les éléments pouvant être liés entre différents diagrammes. Actuellement, cela représente les **IObjectEntity** et les **IRelation**. En les stockant dans **Project**, chaque diagramme peut facilement récupérer les informations de ces éléments, voir les lier fortement pour synchroniser les contenus. C'est ainsi que fonctionne le diagramme de classe. Par contre, le diagramme de cas n'utilise en aucune façon la liaison d'élément.

4.2 Diagramme de classe

Le diagramme de classe (**IClassDiagram**) est relativement simple. Pour la majorité de son travail, il ne s'agit que d'une vue des **IObjectEntity** et des **IRelation**. Ces vues sont des éléments se contentant d'associer une boite de style (**StyleBox**) à chaque élément pour ajouter des fonctionnalités esthétiques. Le diagramme peut aussi contenir des notes.

5 Paquetage ui

L'application s'appuie sur la structure JavaFX avec la classe applicative UmlReverseApp qui est composée d'un stage qui en suivant la hiérarchie JavaFX nous ammène au BorderPane. Le BorderPane nous servira de base pour les différents éléments de notre application, tel qu'une TreeFileManagerView, une MenuBar, un IDiagramEditor et un IDiagramMenu.

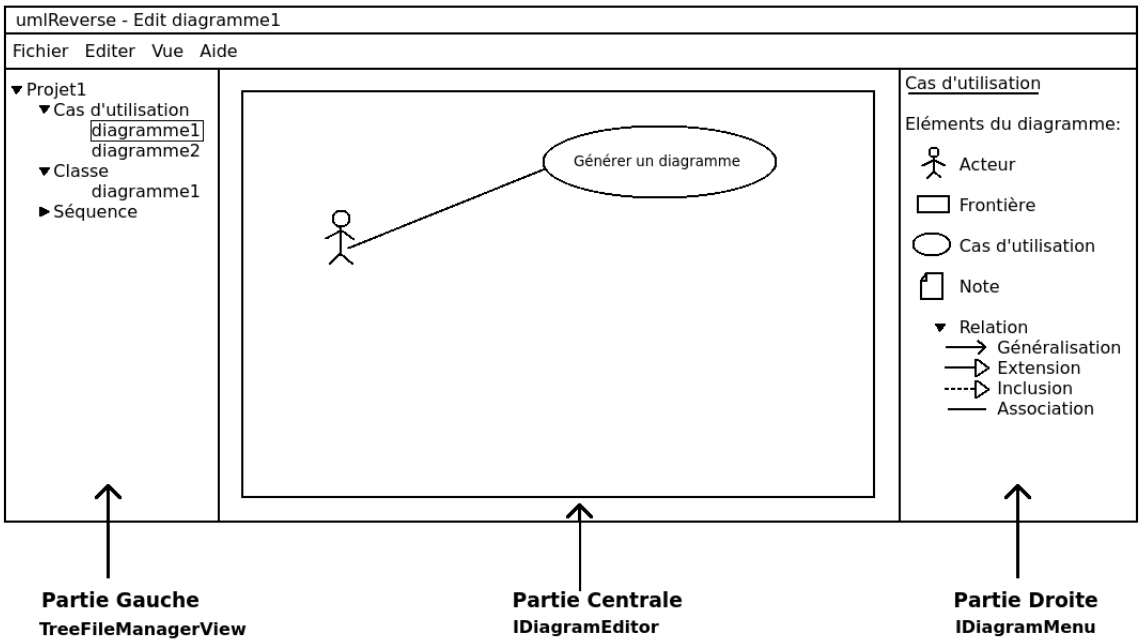


Ce paquetage contient toutes les classes utilisées pour gérer la vue. Ces classes sont toujours associées à un contrôleur. Les vues sont codées en FXML grâce à un logiciel de construction de FXML (SceneBuilder). Chaque contrôleur aura le même nom que la vue associée avec le mot « Controller » ajouté à la fin. ui contient 2 paquetages :

— view : Contient les différentes vues intégrées dans l'application. Le paquetage contient également tous les contrôleurs associés à leur vue.

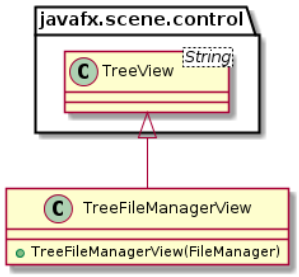
— component : Contient toutes les classes utilisées pour construire les vues. Ce sont leurs composants.

L'application graphique est l'association de plusieurs vues dans un BorderPane.



5.1 La partie gauche

La partie gauche est séparée en deux verticalement. La partie supérieure présente le projet ouvert et la liste des diagrammes qu'il contient, et permet de naviguer entre eux dans une vue hiérarchique construite à partir du modèle. La partie inférieure affiche une liste des entités (classes, interfaces, classes abstraites et énumérations) présentes dans le projet, et permet d'insérer dans le diagramme courant toute entité qui n'y est pas déjà.



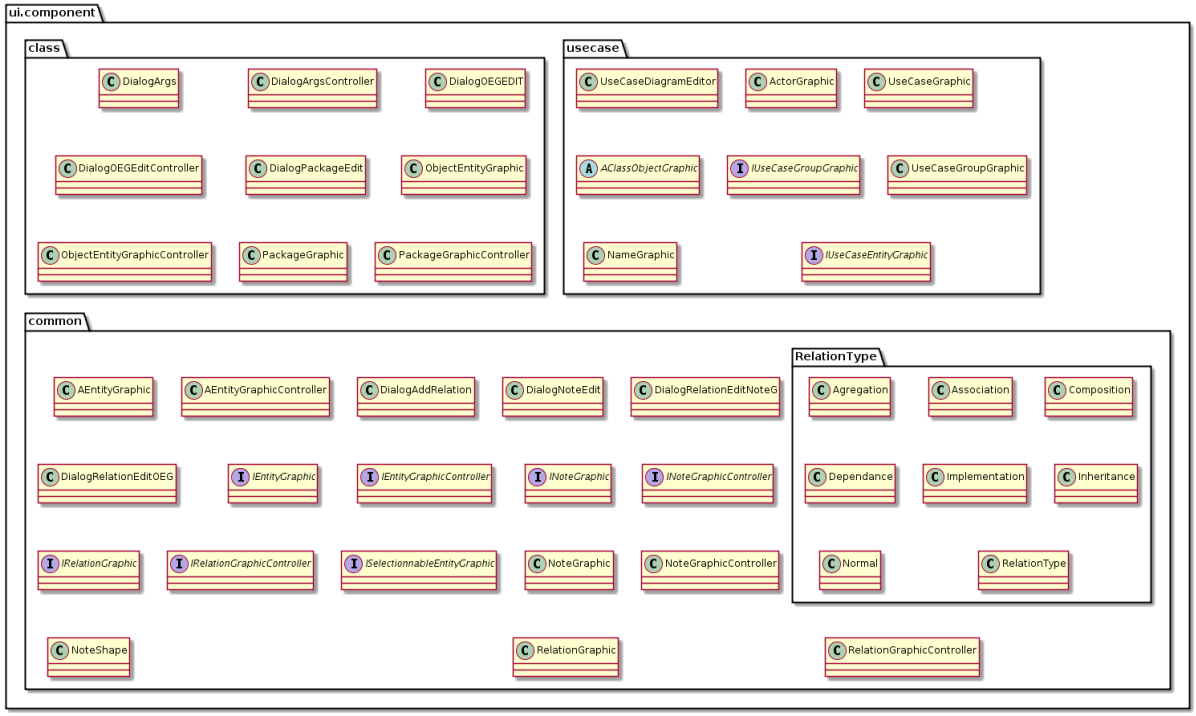
5.2 La partie centrale

La partie centrale sert à éditer graphiquement un diagramme.
Explication : Toutes les classes qui finissent par **Graphic** sont des représentations graphiques des élements du modèle. Elles contiennent des écouteurs de souris sur elles-mêmes pour pouvoir permettre aux utilisateurs de les modifier, ce qui modifiera le modèle directement. La modification du modèle modifie obligatoirement la vue du diagramme (les éléments graphiques donc) grâce à des écouteurs sur le modèle.

5.2.1 Éditeur de diagramme

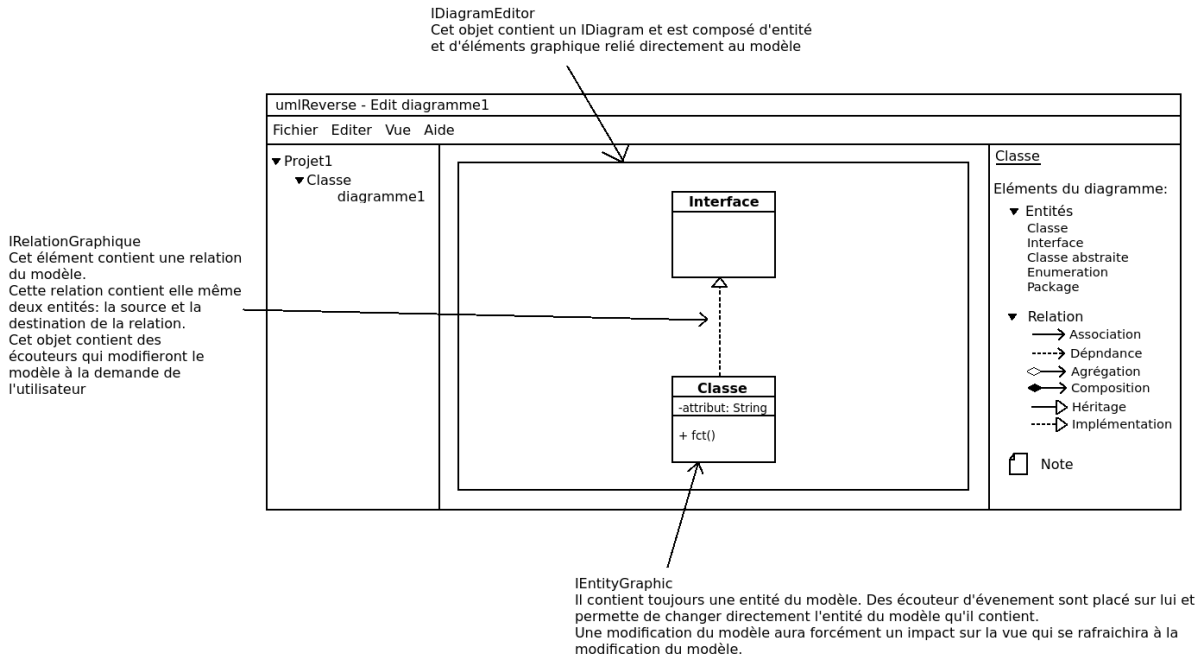
La partie MVC a été volontairement omise pour éviter de surcharger le diagramme de classe. Elle sera en revanche implémentée dans les parties prévues.
Dans les diagrammes ci-contre, les contrôleurs contrôleront les actions de **IDiagramEditor**. Des pop-ups d'édition seront disponibles, elles seront placé dans les paquetages **dialog**.

5.2.2 Paquetages



Voici toutes les entités présentes dans nos packages.

5.2.3 Explications



6 Paquetage main

Contient la classe qui lance l'application. C'est le point d'entrée. Il charge les différentes vues du paquetage ui.view en les rassemblant toutes dans un BorderPane.

7 Extensibilité

L'architecture a été pensée de façon à vérifier l'exigence **EXF_70** (code modulaire, ajout de nouvelles fonctionnalités possible dans le code).

Modèle Le modèle a été pensé de manière à facilement ajouter de nouveaux types de diagrammes. Pour ce faire, le gestionnaire de diagramme ne se soucie pas du type de diagramme. La plupart des actions demandées sur un diagramme de manière extérieure au modèle se font par le biais des visiteurs. Pour ajouter un nouveau type de diagramme, il suffit de créer une nouvelle classe qui implémente **IDiagram** et de mettre à jour les visiteurs globaux.

Partie gauche de l'application Les différentes classes créées peuvent être héritées afin d'étendre leurs fonctionnalitées ainsi que d'ajouter de nouveaux types de diagramme. L'architecture des dossiers d'un projet a été pensée afin que l'ajout de nouveau type de fichier pour la sauvegarde des diagrammes soit simple sans rendre les projets de cette version obsolètes.

IDiagramEditor Tout d'abord, l'ajout de nouveaux types de diagrammes a été pensé de façon à être possible et facile à implémenter. Pour ce faire, il suffit d'ajouter une nouvelle classe **XXXDiagramEditor**. Cette nouvelle classe peut hériter **ADiagramEditor** si le nouveau type de diagramme peut posséder des notes. L'implémentation de celle-ci sera déjà faite. Les classes présentes dans le paquetage view.component.common sont des classes qui peuvent être utilisées dans n'importe quel type de diagramme. Ce sont généralement des classes abstraites qui effectuent une partie du travail, ce qui évite de tout refaire à chaque fois.