Document d'Architecture Logicielle
Projet UML Reverse

Version :	1.0
Date :	10/01/2017
Rédigé par :	Amine Bouaziz François Kopka Guillaume Naimi Etienne Royet Jérémy Tevenin Romain Vialatte

Mises à jour

Version	Date	Modifications réalisées
0.1	04/11/2016	Début de la rédaction du DAL
0.2	11/11/2016	Ajout de tous les diagrammes de séquence
0.3	23/11/2016	Modification du DAL en fonction de la nouvelle STB
0.4	09/01/2017	Ajout des dernières modifications par rapport à la STB
1.0	10/01/2017	Relecture avant présentation du document
1.1	17/02/2017	Modification du diagramme de séquence "Editer un diagramme de séquence"

Table des matières

1 Objectif

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

2 Documents applicables de référence

2.1 Expression du besoin du projet UML Reverse

L'utilisation des diagrammes UML fait partie des outils de tout développeur informatique. Les applications (ou plugins) de bon niveau permettant d'utiliser efficacement ces diagrammes sont limitées dans le domaine du libre, et ne disposent pas de toutes les fonctionnalités utiles.
Remarque : Les logiciels disponibles en libre ne sont pas pérennes. En effet, souvent rachetés par des entreprises, les allers-retours entre versions libres et propriétaires sont très fréquents. Quant aux versions restées libres, elles ont pour défaut leur manque de fonctionnalités, l'ergonomie très discutable et le manque de documentation fiable.

Objectifs

Développer une application ergonomique permettant d'effectuer rapidement un développement informatique intégrant des diagrammes UML. La première phase du projet a été développée l'année dernière, et doit être complétée en exploitant les éléments déjà disponibles. Cette contrainte est fréquemment rencontrée en entreprise où il est demandé de faire évoluer une application existante.

Objectifs imposés

Afin de limiter le périmètre de l'application et d'en faire un outil exploitable, les contraintes imposées sont les suivantes :

- Améliorer le code existant (Correction de bugs).
- Exporter/Importer au format XMI.
- Générer des diagrammes de séquence, d'état et de package. Une interface graphique doit permettre de définir les paramètres graphiques :
  - positionnement des éléments.
  - sélection des éléments affichés/cachés.
- L'application doit permettre le *reverse engineering*, c'est-à-dire l'extraction du diagramme de classe à partir du code source en Java d'une application. Le résultat sera stocké au format XMI.
- POC : Réaliser un reverse engineering en exploitant l'introspection JAVA.

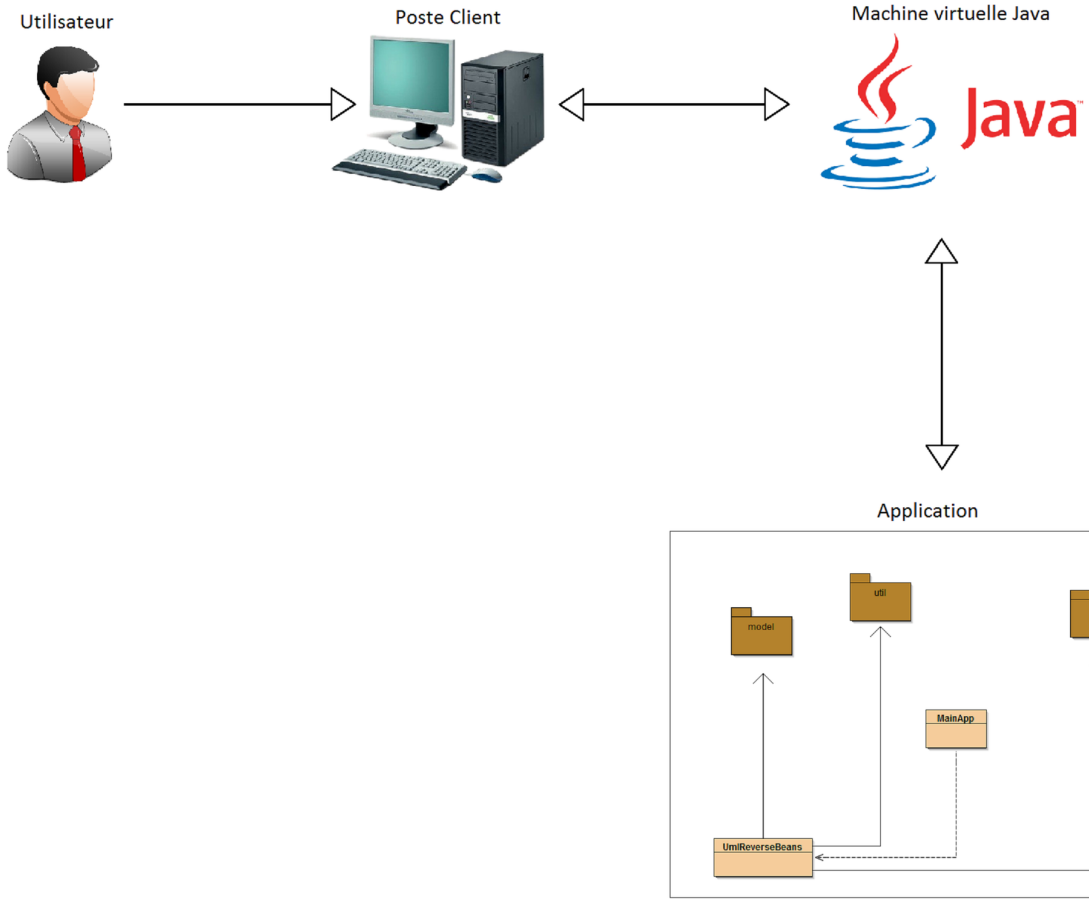
3 Lexique

Se référer au document Terminologie.

4 Configuration requise

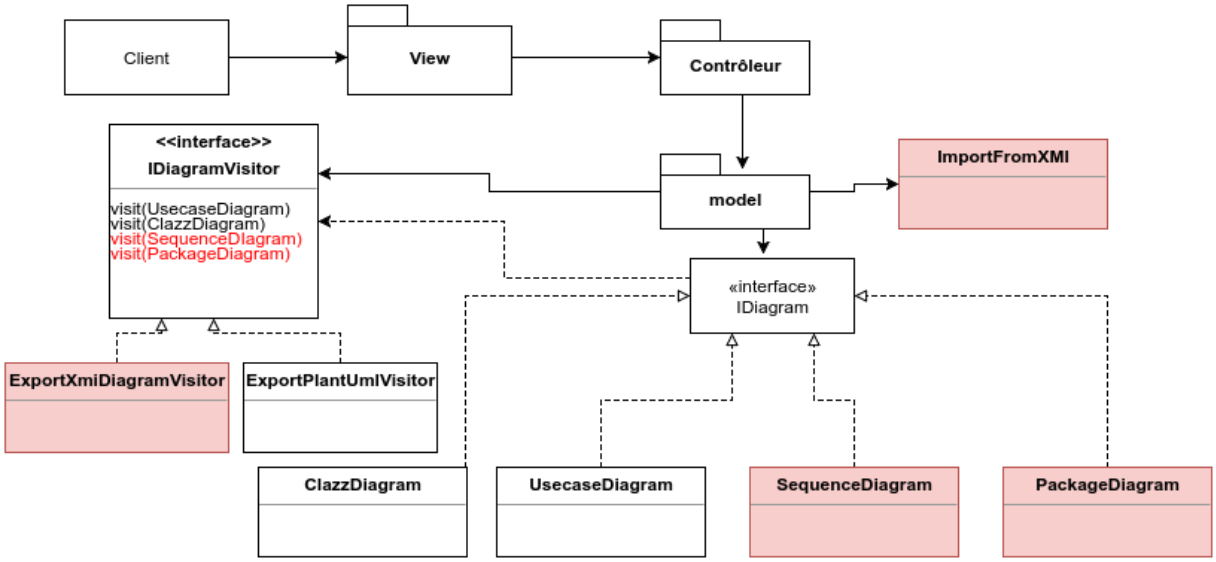
Il faut que l'application fonctionne sur n'importe quel ordinateur supportant la machine Java 7 et plus spécifiquement sur les ordinateurs de l'université.

5 Architecture statique



L'utilisateur doit posséder une machine sur laquelle doit être installée une version java 7 minimum doit être installé. En ayant ceci, l'utilisateur pourra exécuter le logiciel directement en cliquant sur l'icône.

5.1 Vue succinte de l'architecture et des modifications qui seront réalisées



En rouge on peut voir les classes ou méthodes qu'on va ajouter.
On doit pouvoir importer/exporter en XMI les diagrammes :
— de cas d'utilisation,
— de classe,
— de séquence,
— de paquetage.

5.2 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 fonctionnalités ajoutées s'effectuent par le biais de visiteurs qui appelleront à leur tour les visiteurs spécifiques à chaque diagramme.

5.2.1 Eléments liés

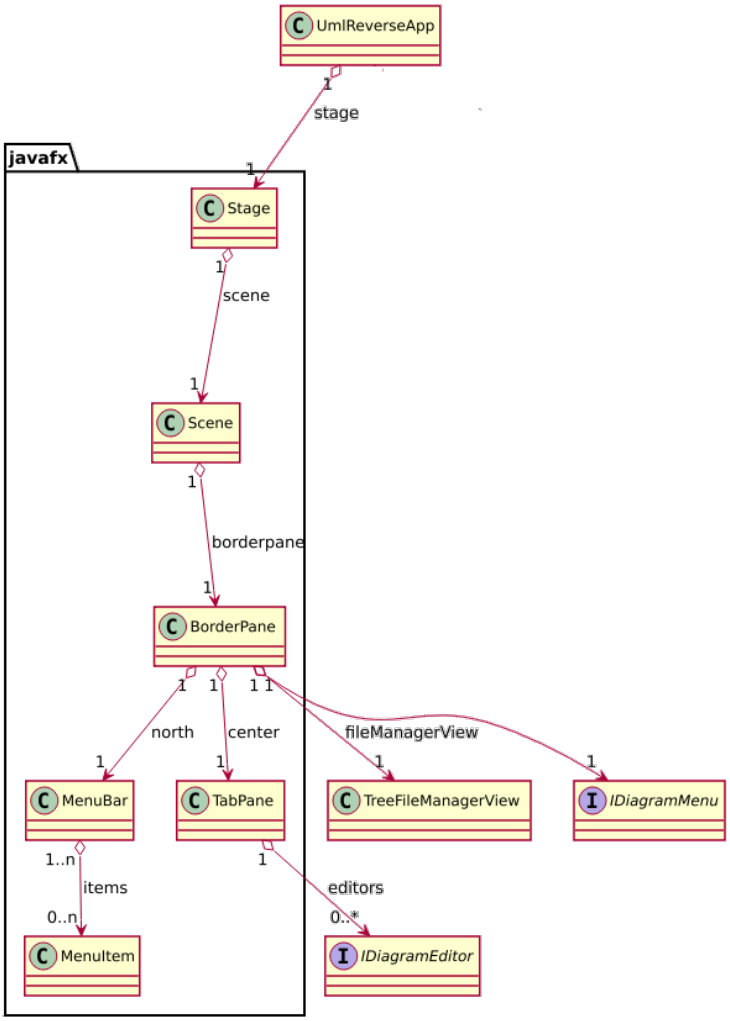
Le projet contient les éléments pouvant être liés entre différents diagrammes. Actuellement, cela représente les **IObjectEntity** et les **IRelation**. En les stockant dans **Project**, chaque diagramme peut facilement récupérer les informations de ces éléments, voir les lier fortement pour synchroniser 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.

5.2.2 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.

5.3 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 amène 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.

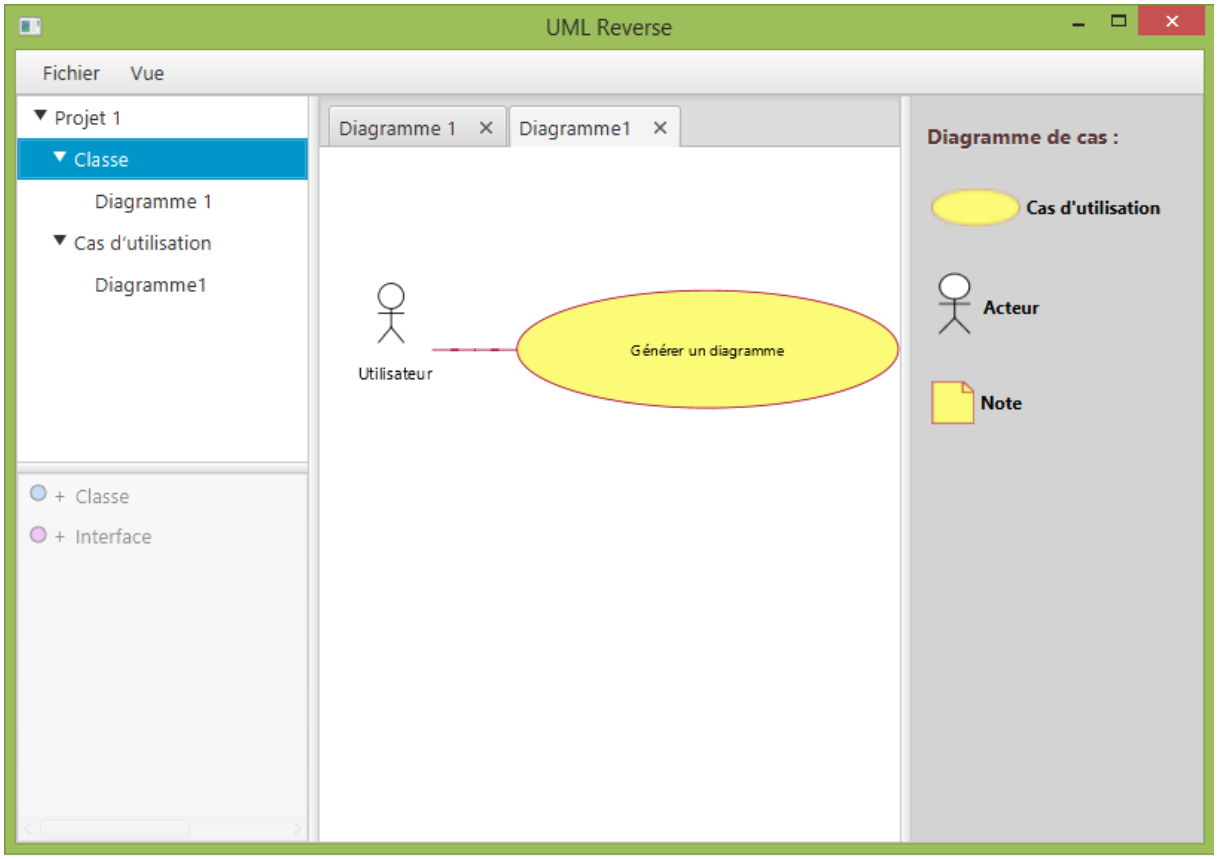


Ce paquetage contient toutes les classes utilisées pour gérer la vue. Ces classes sont toujours associées à un contrôleur. Les vues sont codées en FXML grâce à un logiciel de construction de FXML (SceneBuilder). Chaque contrôleur aura le même nom que la vue associée avec le mot « Controller » ajouté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

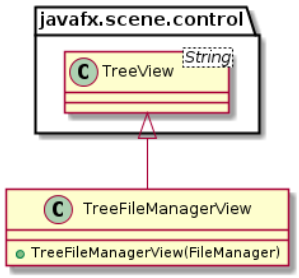
— component : Contient toutes les classes utilisées pour construire les vues. Ce sont leurs composants.

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



5.3.1 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.



5.3.2 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.

5.3.3 Éditeur de diagramme

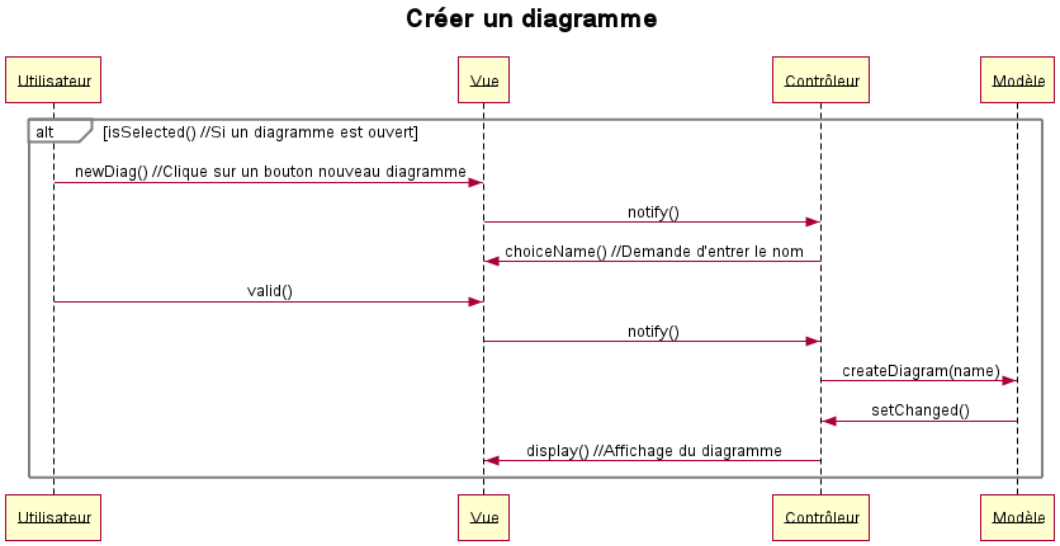
La partie MVC a été volontairement omise pour éviter de surcharger le diagramme de classe. Elle sera en revanche implémentée dans les parties prévues.
Dans les diagrammes ci-contre, les contrôleurs contrôleront les actions de **IDiagramEditor**. Des pop-ups d'édition seront disponibles, elles seront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

5.4 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.

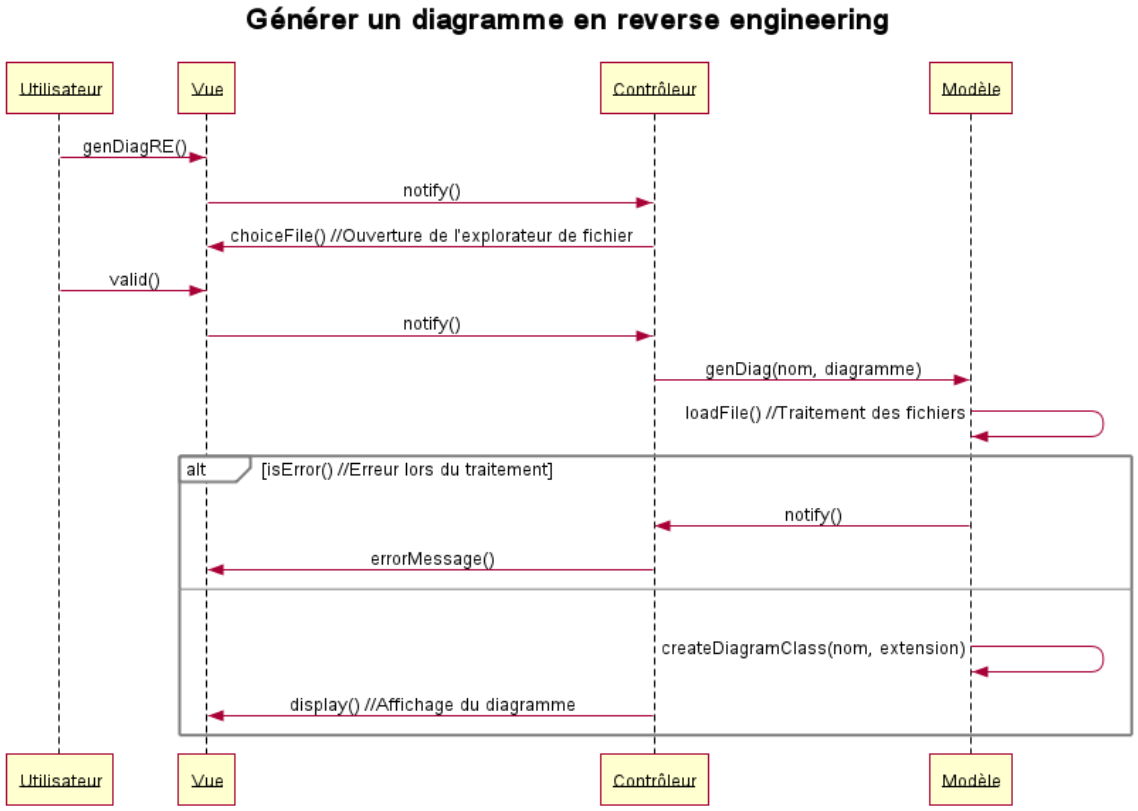
6 Fonctionnement dynamique

6.1 Créer un diagramme (DIA_10, DIA_20, DIA_30)

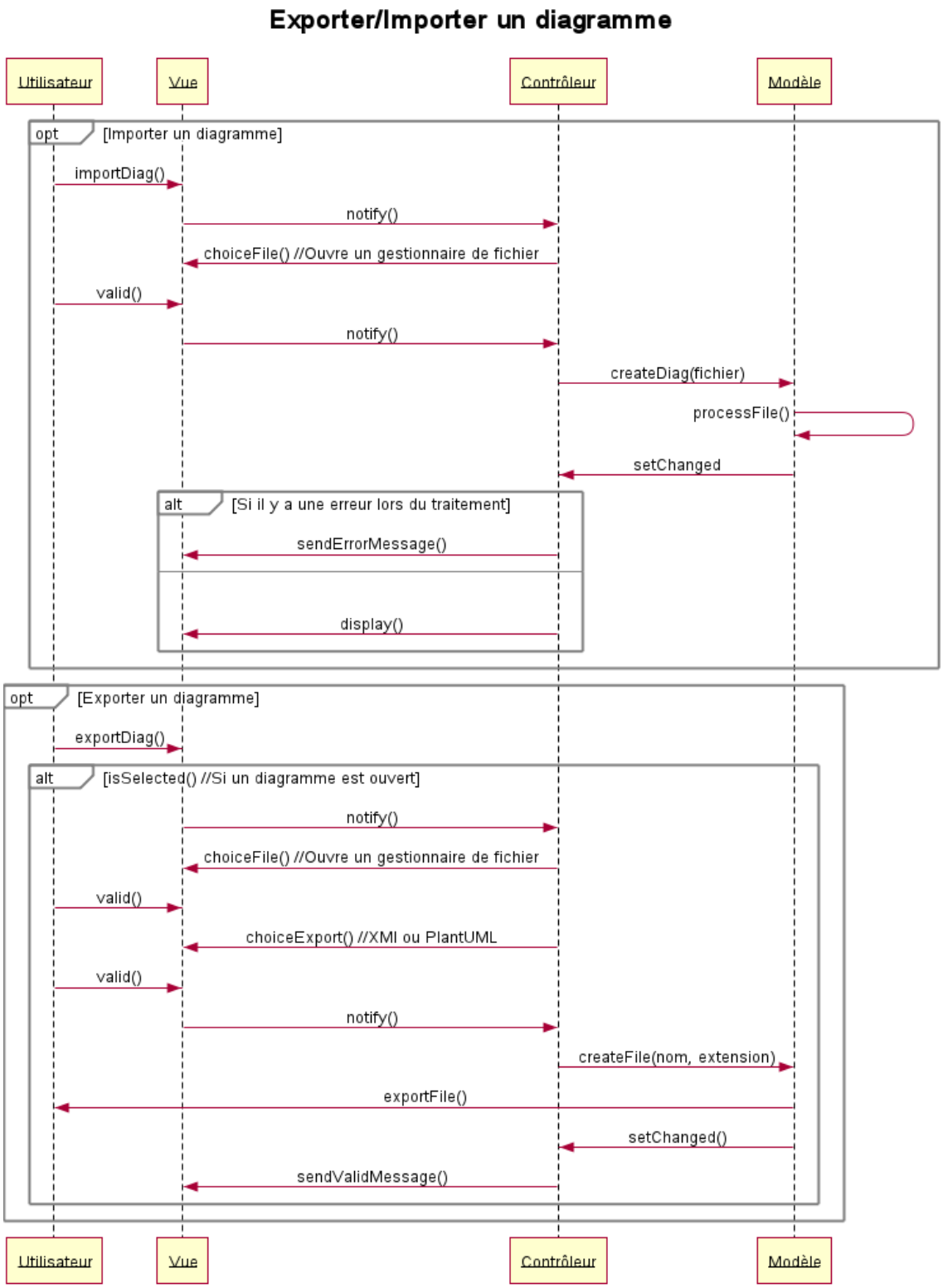


Avec ce fonctionnement, on peut créer un diagramme de séquence, d'état et de paquetage.
La méthode valid() permet de valider le nom que l'utilisateur à choisit.

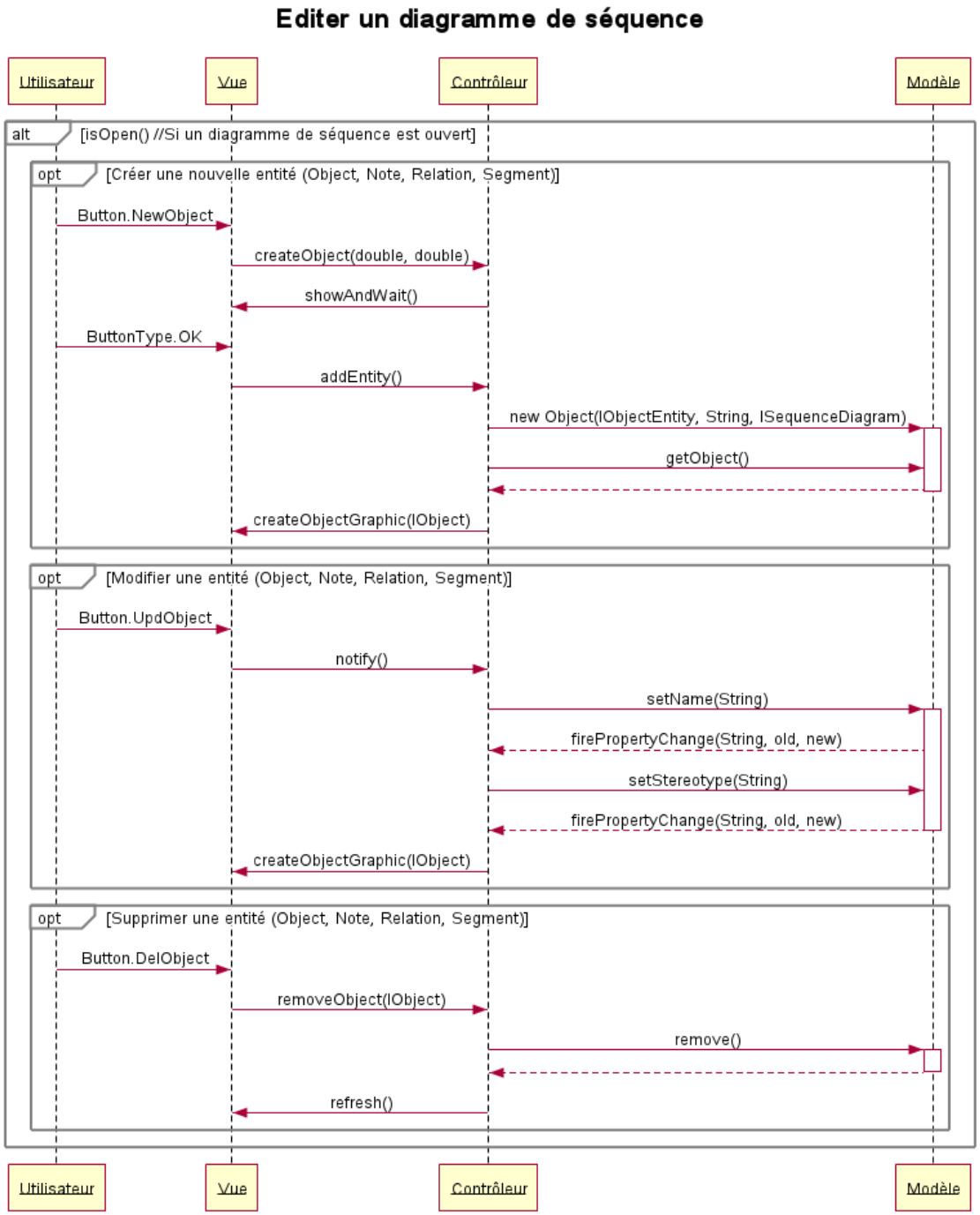
6.2 Générer un diagramme de classe en reverse ingineering par introspection java (DIA_40)



6.3 Exporter/Importer un diagramme (IHM_30, IHM_40)

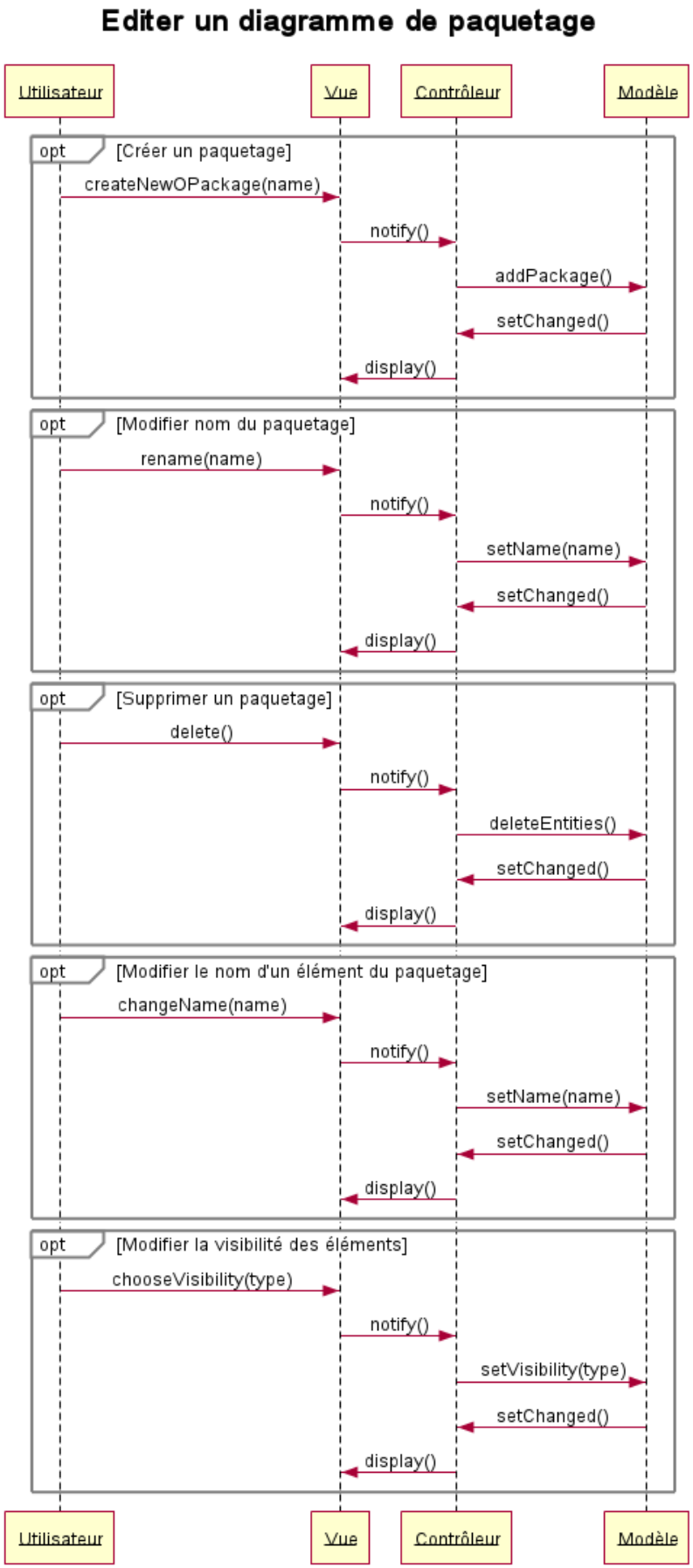


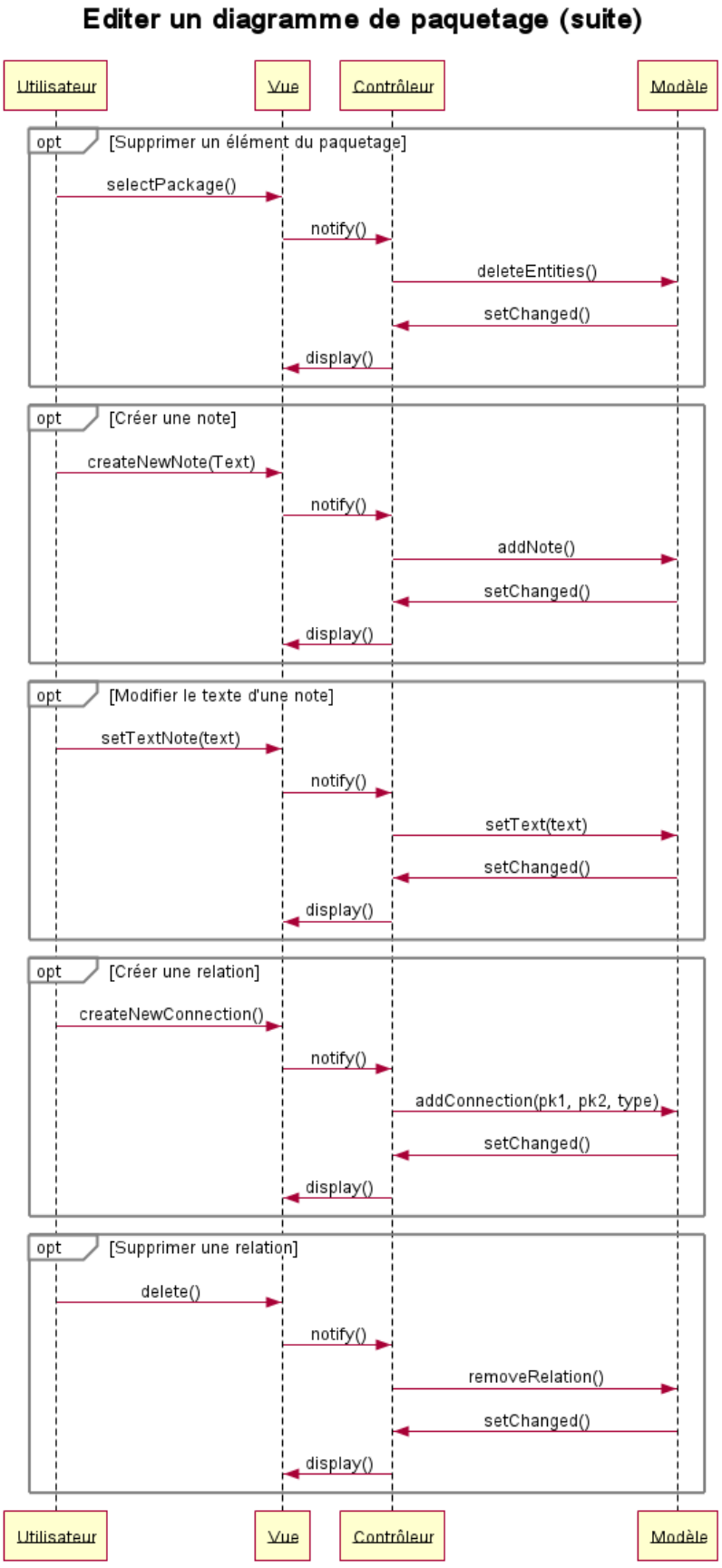
On peut importer un diagramme (qu'il soit en PlntUML ou XMI) et on peut aussi exporter un diagramme en choisissant le type d'export que l'on veut avec la méthode choiceExport().
La méthode choiceFile() permet à l'utilisateur d'ouvrir un gestionnaire de fichiers ou il pourra enregistrer son fichier (pour exporter) ou ouvrir son fichier (pour importer).



On a toutes les fonctionnalités que l'on peut effectuer sur un diagramme de séquence. Ici, on peut voir tout ce qui est lié à un objet.

6.4 Éditer un diagramme de paquetage (PAQ_X)





On a toutes les fonctionnalités que l'on peut effectuer sur un diagramme de paquetage.

7 Traçabilité

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 fonctionnalités 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.