INSA de LYON

Département Informatique

Projet Développement Orienté Objet

Compte Rendu

- Rapport de réalisation du projet -

30 octobre 2015

Auteurs
Estelle Lepeigneux
Lisa Courant
Hugues Verlin
Pierre Jarsaillon
Paul Dautry

Chef de projet
Antoine Chabert

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

TABLE DES MATIÈRES

TODO LIST

Première partie

Introduction

I.1 PRÉSENTATION DU CONTENU DU RAPPORT

Ce document constitue la synthèse de l'ensemble des livrables devant être remis à la fin du projet Développement Orienté Objet effectué en 4IF. Il a été convenu avec les professeurs encadrant le projet que le code pouvait être réalisé en langue anglaise mais que ceci impliquait également le rendu de certains livrables en Anglais. Les livrables devant être rendu en Anglais ont été identifiés avec les professeurs lors de la première séance de TP. Il sera donc indiqué dans le titre de chaque partie si le contenu de la partie est rédigé en Anglais *(EN)* ou en Français *(FR)*.

DEUXIÈME PARTIE

CAPTURE ET ANALYSE DES BESOINS

II.2 Planning prévisionnel du projet *(FR)*

Ajouter le planning effectif (Gantt prévisionnel)

II.3 Domain model *(EN)*

We should add a comment here

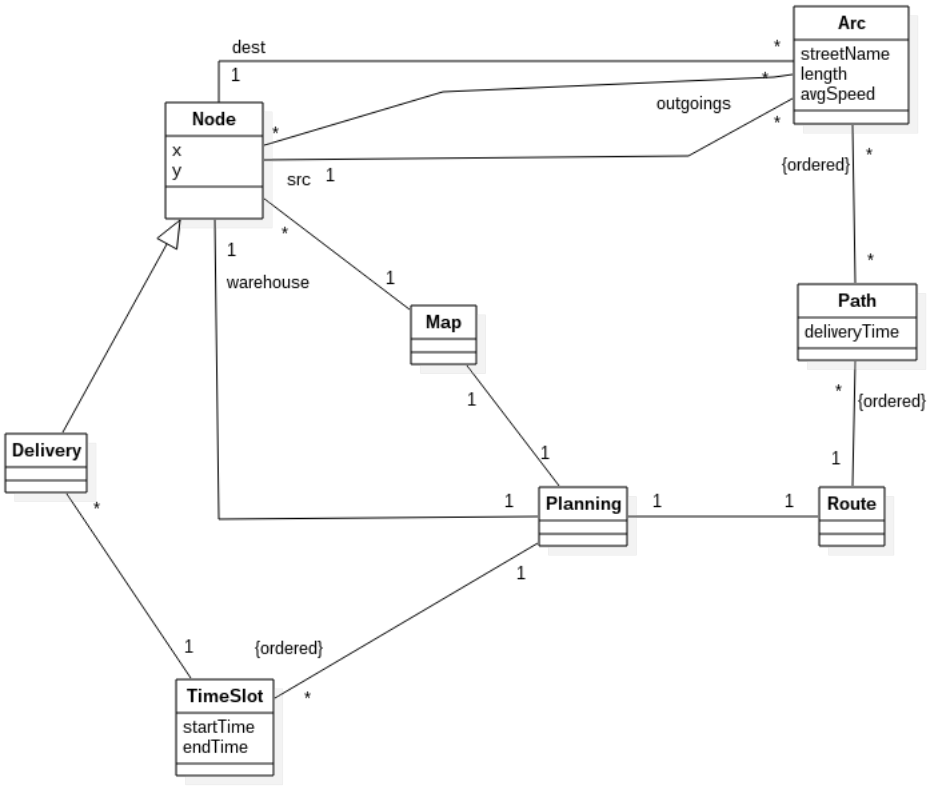

Figure II.3.1 – Domain Model

II.4 Glossary *(EN)*

Syntax used :

- **Object** *(french equivalent)* : definition

Glossary :

- **Arc** *(tronçon)* : section of road between two intersections described by a street name, an average speed, a length and the id of the destination intersection.
- **Node** *(intersection)* : Intersection between two or more arcs (cf. Arc) described by its coordinates and an id.
- **Path** *(chemin)* : Ordered list of arcs between two delivery nodes, representing the shortest travel duration.
- **Map** *(plan)* : Set of arcs (cf. Arc) and intersections (cf. Node)
- **Delivery** *(livraison)* : Special node which must be contained by the final route.
- **Planning** : Set of intersections (cf. Node) to be processed by the route computation algorithm (cf. Route).
- **Route** *(tournée)* : Ordered list of paths to travel along (cf. Planning).
- **TimeSlot** *(fenêtre de livraison)* : time interval bounded by a start time and an end time containing a set of deliveries to be honored.
- **Warehouse** *(entrepot)* : Special intersection representing both the start and the end point of the route (cf. Route).
- **Average Speed** *(vitesse moyenne)* : average speed of vehicles in meters per second on an arc (cf. Arc).
- **Planning File** *(fichier de planning)* : XML formated input file, describing a planning (cf. Planning).
- **Map File** *(fichier de plan)* : XML formated input file, describing a Map.
- **Route File** *(fichier d'itinéraire)* : text formated output file, describing paths to travel along.

II.5 USE CASE DIAGRAM *(EN)*

We should add a comment here

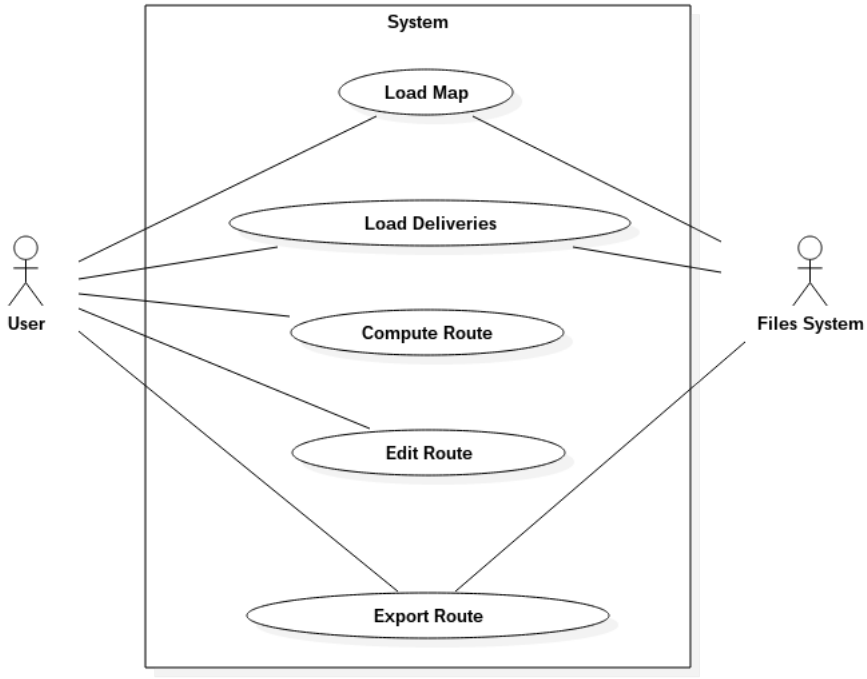

FIGURE II.5.1 – Use case diagram

TROISIÈME PARTIE

CONCEPTION

III.6 User-level events list and State-Transition Diagram *(EN)*

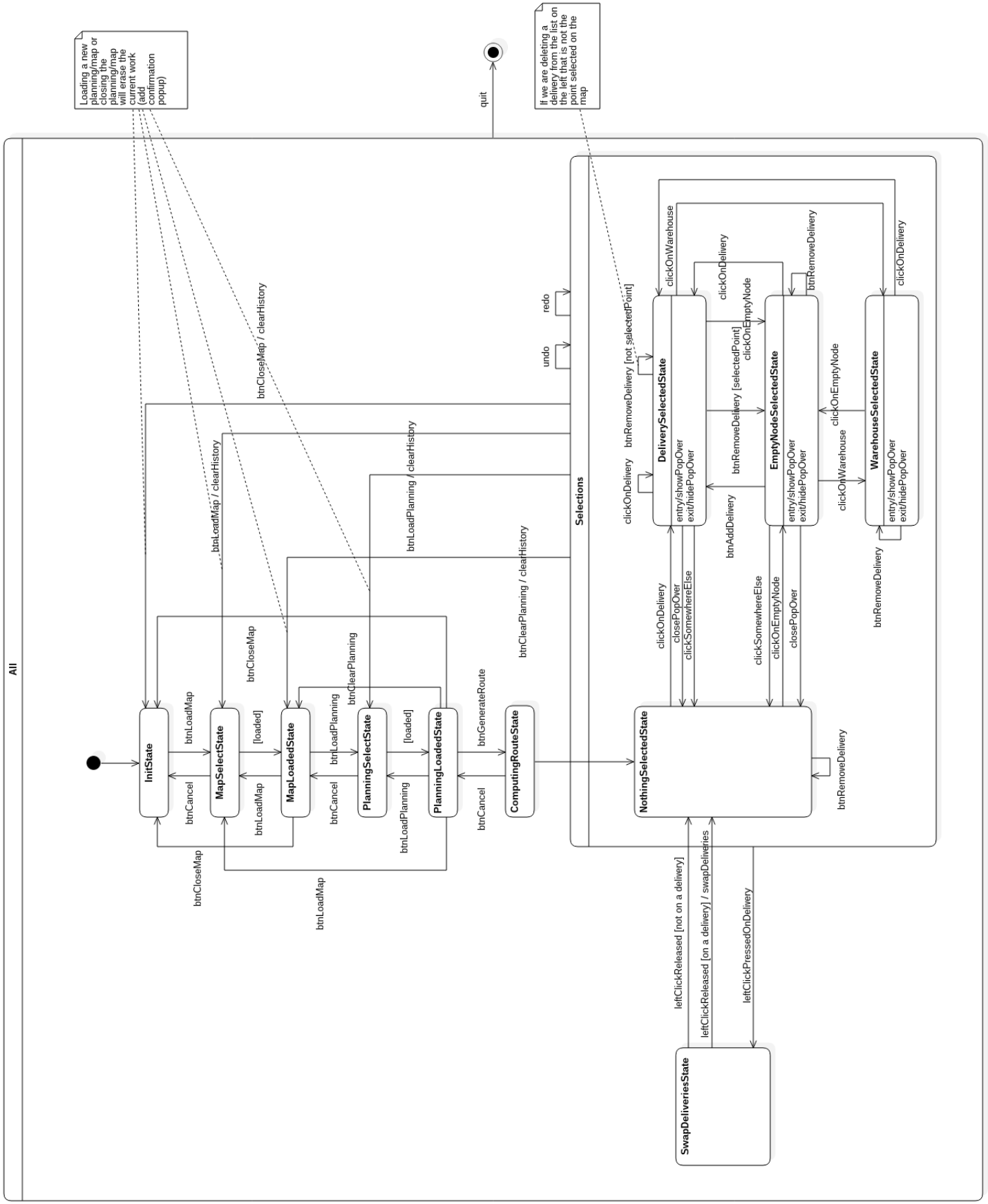

FIGURE III.6.1 – State-Transition Diagram

Add user-level events list

We should add a comment here

III.7 Packages and Classes Diagrams *(EN)*

III.7.1 Model related diagrams

We should add a comment here

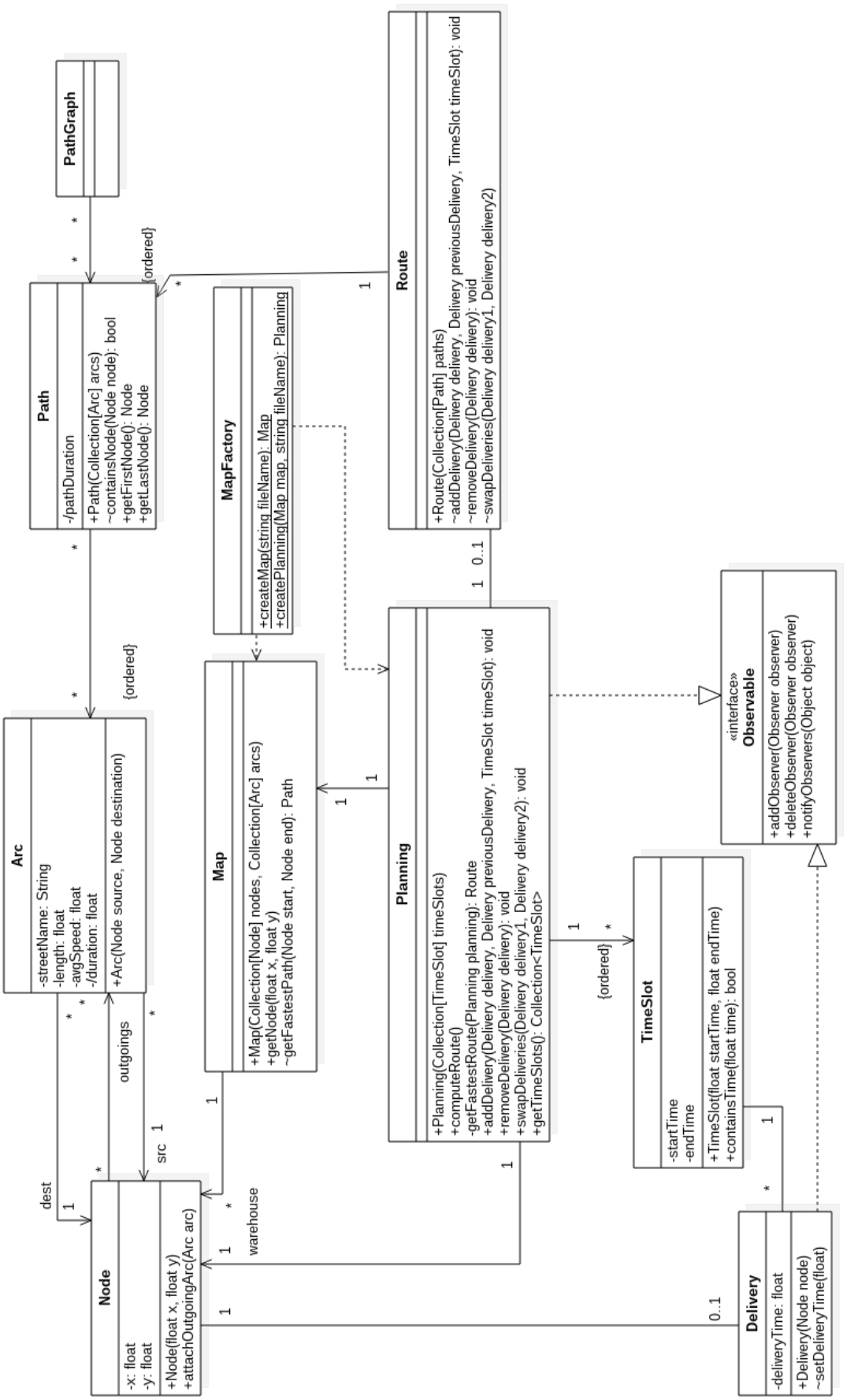

FIGURE III.7.1 – Model package including classes

III.7.2 VIEW RELATED DIAGRAMS

We should add a comment here

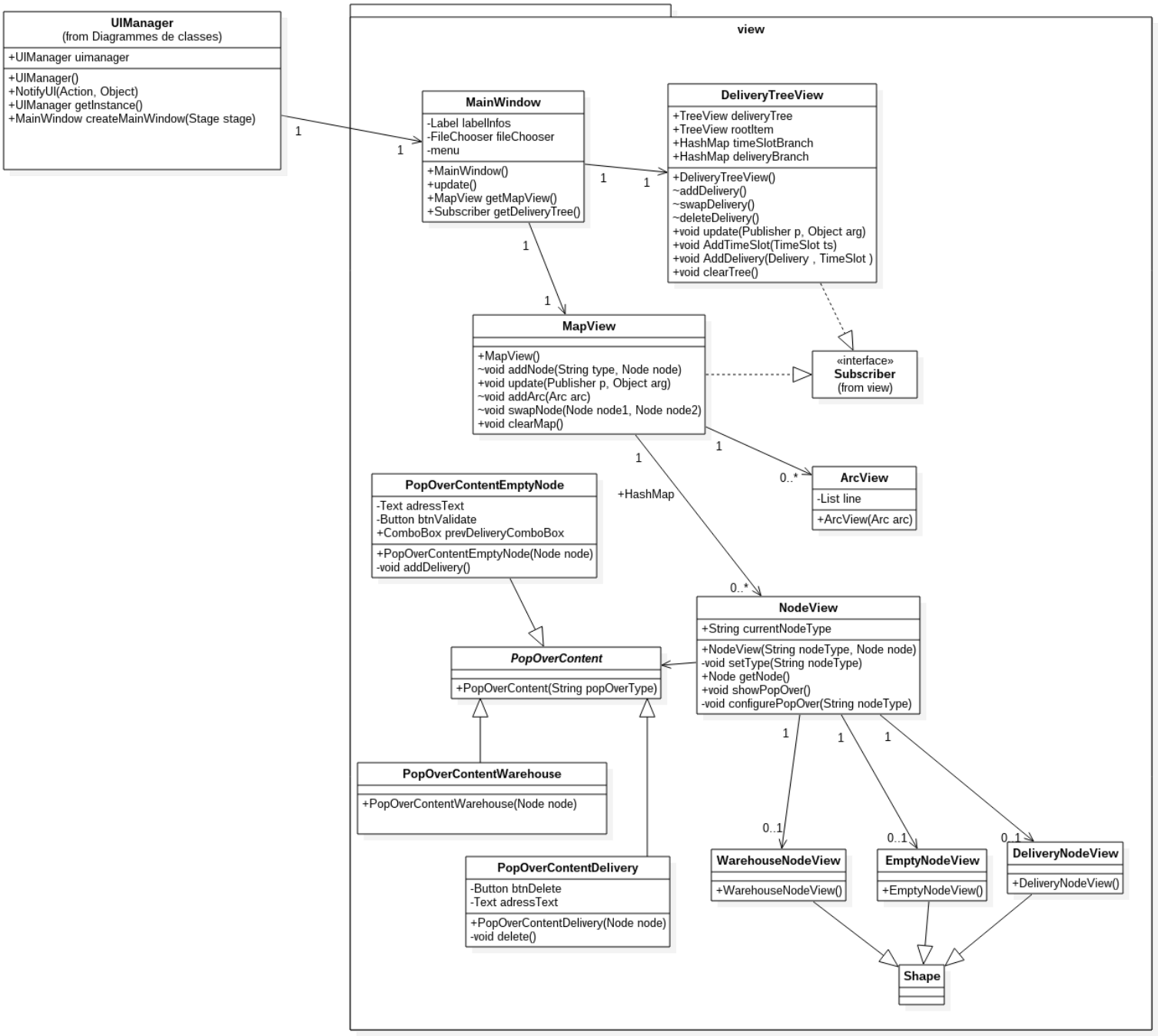

FIGURE III.7.2 – View package including classes

III.7.3 CONTROLLER RELATED DIAGRAMS

We should add a comment here

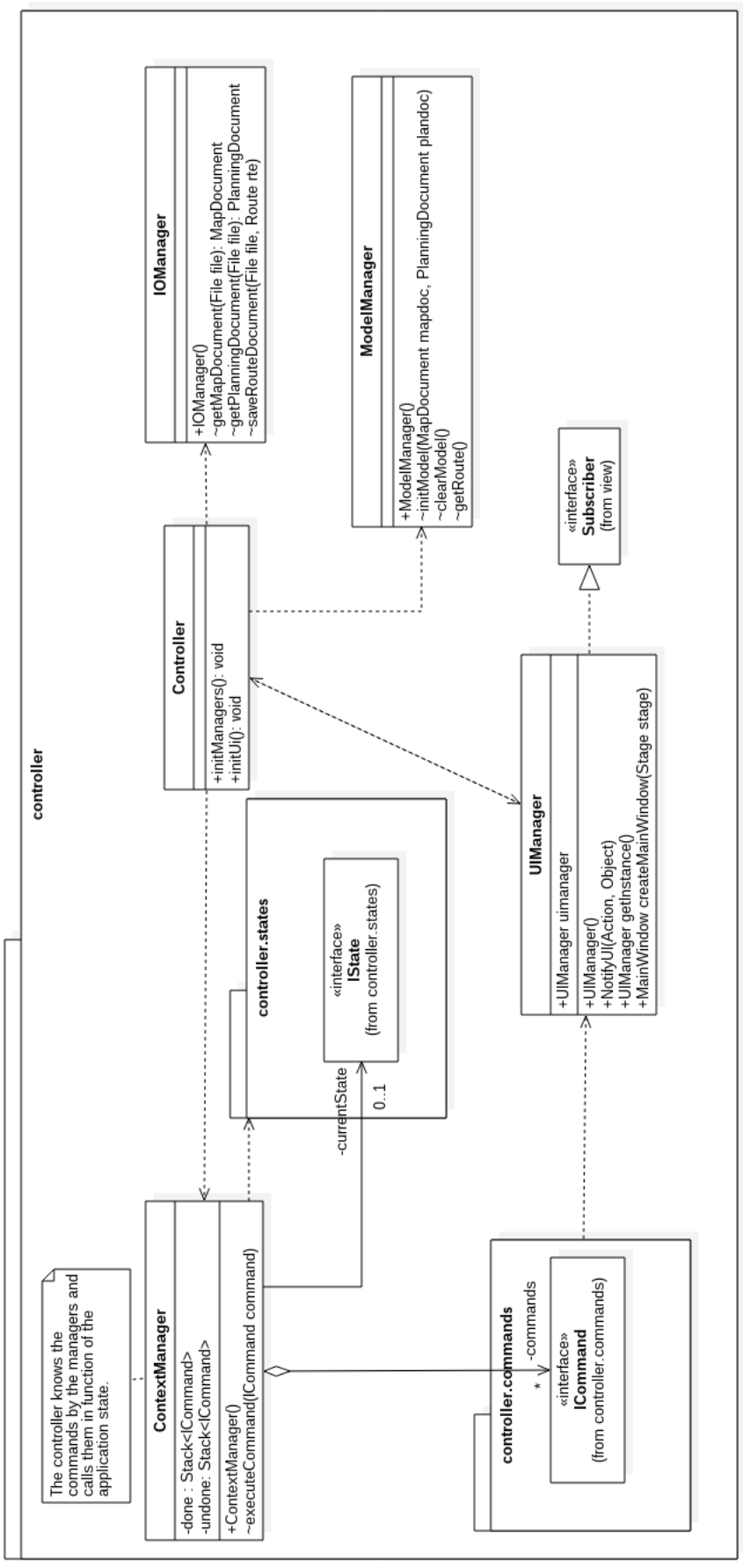

FIGURE III.7.3 – Controller package including classes

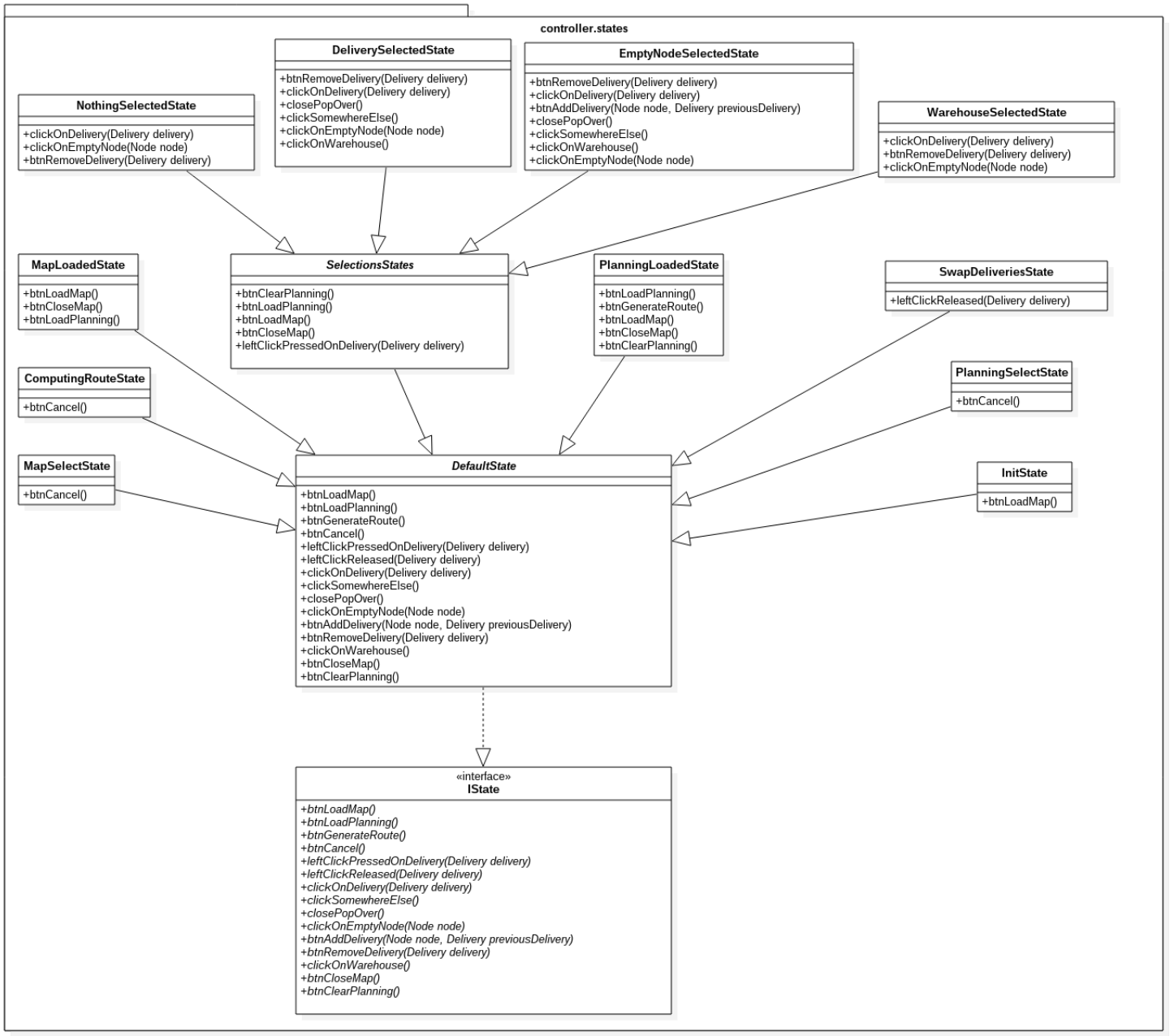

FIGURE III.7.4 – Controller States package including classes

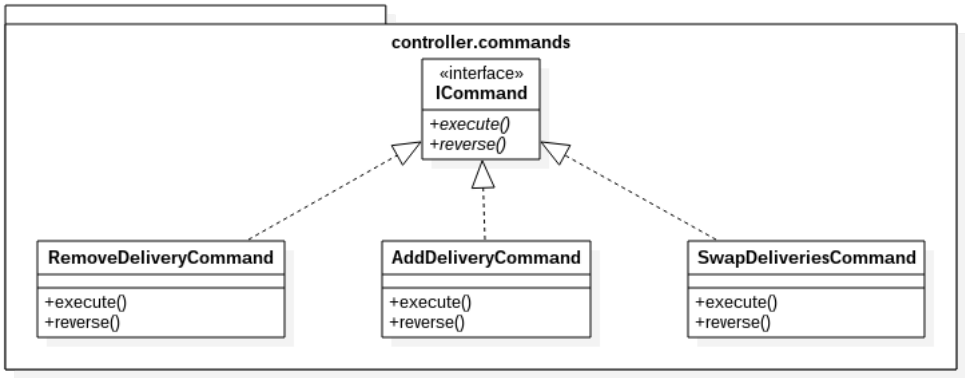

FIGURE III.7.5 – Controller Commands package including classes

III.7.4 Other diagrams

We should add a comment here

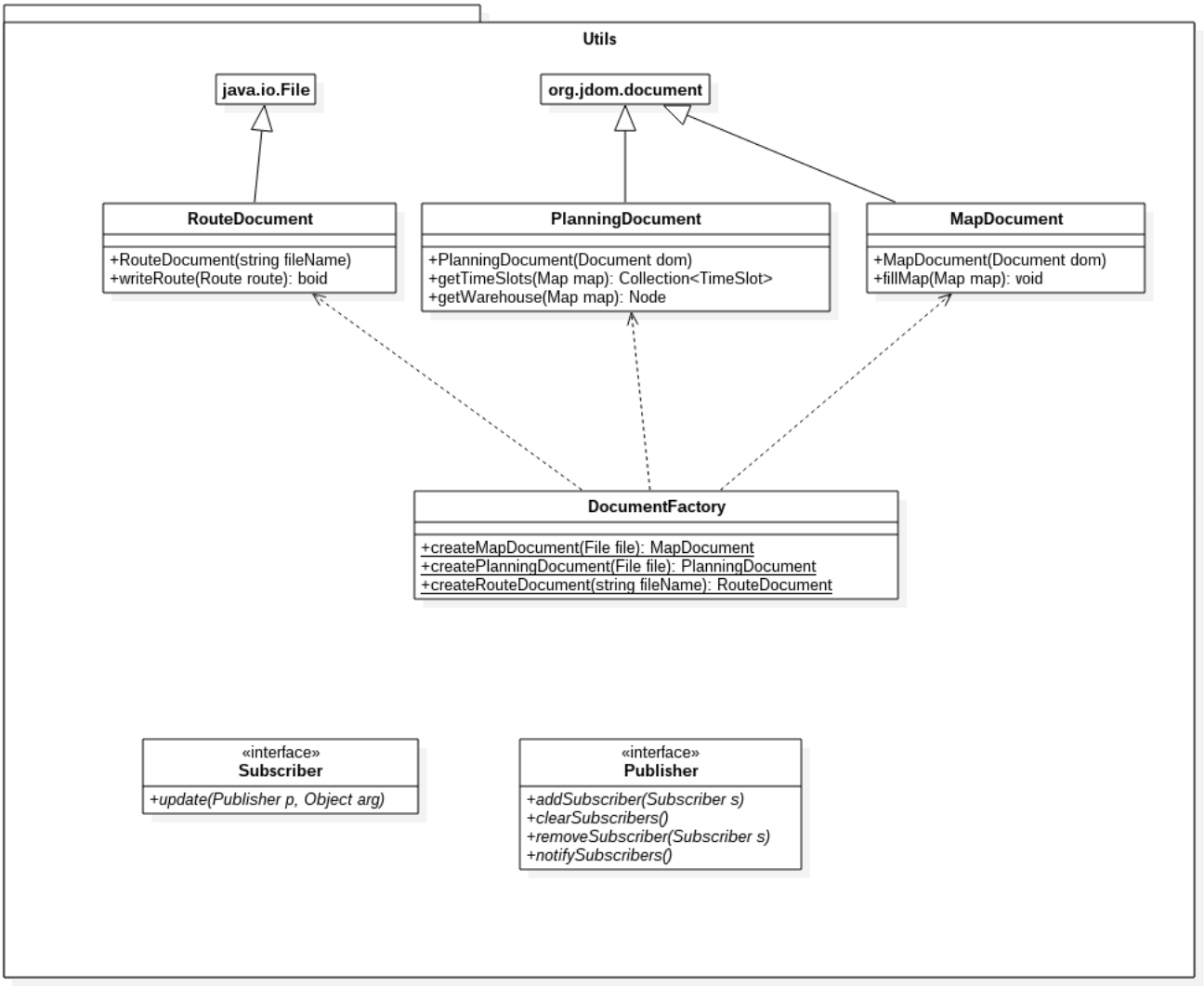

Figure III.7.6 – Utils package and its classes

III.8 Choix architecturaux et patrons de conception utilisés *(FR)*

Les classes cités dans ce document pouvant présenter des ambiguïtés sont référencés dans le glossaire.

Nous choisirons tout d'abord de respecter l'architecture globale de type **MVC** afin de garantir une simplicité de maintenance corrective et évolutive de l'application. Les différents composants appartenant à ces trois grandes catégories (Modèle, Vue et Contrôleur) implémenteront divers patrons de conception.

Concernant l'architecture globale, nous avons choisi de faire en sorte que la hiérarchie en couche soit parfaitement respectée, c'est-à-dire que la vue ne modifie en aucun cas le modèle sans passer par le contrôleur. La vue, cependant, implémente pour certains de ses composants le patron **Observateur** pour détecter les modifications au niveau du modèle et se modifier ou demander au contrôleur d'effectuer des actions en conséquence.

Les objets du modèle, qui constitue l'ensemble des données métier manipulées par l'application représentées par des objets, sont pour la plupart des conteneurs de données améliorés et n'implémentent pas de traitements particuliers. Il y a cependant une exception, l'objet **Map** implémente l'algorithme permettant de calculer la tournée et de créer l'objet **Route** correspondant. Certains objets, tels que **Map** et **Planning**, constituent en réalité des conteneurs d'autres objets du modèle. Ces objets agissent comme des **Fabrique**s pour les objets qu'ils contiennent. Cela permet de garantir la cohérence des objets contenus dans ces conteneurs ainsi que la maîtrise de la durée de vie de ces derniers. Comme

indiqué dans le paragraphe précédent, le modèle est observé par la vue ce qui signifie que certains objets du modèle sont **Observable**s. Ce mécanisme nous permet de notifier la vue de changements au niveau du modèle.

L'objet qui représente la **Map**, **MapView**, ainsi que celui qui représente l'arbre des livraisons implémentent le patron **Observateur**. Pour le reste, la vue est isolée du modèle. Celui peut donc fonctionner indépendamment de la vue.

Nous avons ajouté un package supplémentaire n'apparaissant pas dans l'architecture MVC afin d'y placer tous les objets étant catégorisés utilitaires, c'est-à-dire rendant des services aux autres objets de l'application mais dont les traitements auraient pu être intégrés aux objets de l'application si nous n'avions pas fait le choix de segmenter les traitements. Ce package, nommé utils, contient notamment les objets constituant des interfaces plus simples pour la création des objets du modèle possédant une représentation sous forme de fichiers XML. Ce package contient également une classe permettant la conversion de types spéciaux propres à l'application et une **Fabrique** pour les objets d'interface avec les fichiers.

Le choix majeur effectué concernant le contrôleur est de séparer les traitements en créant des sous-composants de ce contrôleur principal. Ce package contient donc un ensemble de contrôleurs spécialisés. Cette organisation au niveau du contrôleur présente de nombreux avantages et notamment la simplification des procédures de gestion des erreurs, leurs causes étant segmentées à l'image des contrôleurs qui peuvent les rencontrer. Nous avons donc choisi de dédier un contrôleur à la gestion du modèle, il constitue en réalité l'interface du contrôleur avec le modèle. Ce contrôleur se nomme **ModelManager** et garantie la cohérence du modèle exploité par l'application, tout en proposant une interface simple pour l'accès au modèle. Il ajoute également un niveau d'encapsulation supplémentaire concernant les données manipulées dans le modèle.

Un second contrôleur, nommé **IOManager**, est chargé des interactions avec le système de fichiers du système d'exploitation exécutant l'application. C'est à son niveau que sont traitées les erreurs de type Entrée/Sortie telles que l'absence de fichier ou encore les privilèges insuffisants pour lire/écrire un fichier. De nouveau, ce contrôleur propose une interface simple pour effectuer les opérations dont il est chargé.

Ensuite, l'**UIManager**, pendant du **ModelManager**, mais pour l'interface graphique cette fois. Il constitue une interface, simple d'utilisation, entre le contrôleur et la vue. Cela permet de centraliser les informations provenant de la vue. C'est aussi le rôle de ce contrôleur d'interpréter les ordres en provenance de la vue pour les convertir en ordres exécutables par le **ContextManager**.

Le **ContextManager** a deux rôles majeurs qui lui valent d'implémenter deux patrons de conception qui sont le patron État et le patron Commande. Il a tout d'abord pour rôle d'entretenir la machine à états globale de l'application, c'est-à-dire de garantir à chaque instant qu'une commande utilisateur déclenchera les bons traitements dans l'application et donc les bonnes réactions de l'interface utilisateur et du modèle. Son second rôle est d'entretenir un historique des modifications effectuées de sorte à ce que l'utilisateur puisse défaire et refaire les dernières actions qu'il a effectué. Ce contrôleur assure à l'utilisateur que l'action qu'il veut effectuer a du sens dans le processus global que déroule l'application. Il est dans cette optique étroitement lié à l'**UIManager** qui lui fait parvenir les commandes de l'utilisateur. Il est, à ce stade, important de préciser que ces différents composants implémentent tous le patron Singleton afin de garantir l'unicité de chaque interface et par conséquent l'unicité du modèle, de l'état de global de l'application et de l'historique des commandes effectuées.

L'objet **Controller** présent dans le package ne constituent en réalité que le point d'entrée de l'application, il est appelé par la procédure main qui l'utilise pour initialiser les différents composants du contrôleur, c'est-à-dire réaliser la première instanciation des composants du contrôleur, puis démarrer l'interface graphique.

On peut également considérer l'**UIManager** et le **ModelManager** comme des **Façade**s respectivement pour la vue et le modèle.

Ajouter des explications concernant les patrons de conception utilisés.

III.9 Sequence Diagram describing Route computation process *(EN)*

Add sequence diagram here

Quatrième partie

Implémentation et tests

IV.10 Notes related to source code, its documentation and unit tests *(EN)*

Add notes here if needed or remove this section

IV.11 Packages and classes diagrams generated from source code *(EN)*

Retro-generated diagrams

CINQUIÈME PARTIE

BILAN

V.12 Planning effectif du projet *(FR)*

Ajouter le planning effectif (Gantt réel)

V.13 Bilan humain et technique *(FR)*

Rédiger le bilan ici