

***ANNEE UNIVERSITAIRE :2015/2016***

**DEPARTEMENT INFORMATIQUE**

***Rapport du projet DevOO***

***Développement d'une application suivant une méthodologie Orientée Objet***

th?id=OIP

***ENCADRE PAR :***

* ***Mme. Christine SOLNON***
* ***M. Elöd EGYED-ZSIGMOND***

***REALISE PAR L'HEXANOME H4402 :***

* ***Nicolas NATIVEL (Chef de Projet)***
* ***Romain MATHONAT***
* ***Mathieu GAILLARD***
* ***Mohammed EL ARASS***
* ***Guillaume KHENG***
* ***Thomas FAVROT***
* ***Killian OLLIVIER***

Table des matières

[1. Introduction 3](#_Toc434950906)

[2. Livrables de capture et analyse des besoins 3](#_Toc434950907)

[2.1 Planning prévisionnel du projet 3](#_Toc434950908)

[2.2 Modèle du domaine 5](#_Toc434950909)

[2.3 Glossaire 5](#_Toc434950910)

[2.4 Diagramme de cas d'utilisation 7](#_Toc434950911)

[2.5 Description textuelle structurée des cas d'utilisation 7](#_Toc434950912)

[3. Livrables de conception: 11](#_Toc434950913)

[3.1 Liste des événements utilisateur et diagramme Etats-transitions 12](#_Toc434950914)

[3.2 Diagrammes de packages et de classe 15](#_Toc434950915)

[3.3 Diagrammes de packages et de classes retro-générés à partir du code 16](#_Toc434950916)

[2.4 Choix architecturaux et design patterns utilisés 17](#_Toc434950917)

[3.5 Diagramme de séquence du calcul de la tournée à partir d'une demande de livraison 18](#_Toc434950918)

[4. Implémentation et tests 23](#_Toc434950919)

[4.1 Code du prototype et des tests unitaires 23](#_Toc434950920)

[4.2 Documentation JavaDoc du code 23](#_Toc434950921)

[4.3 Diagrammes de packages et de classes rétro-générés à partir du code 23](#_Toc434950922)

[5. Bilan 25](#_Toc434950923)

[5.1 Planning effectif du projet 25](#_Toc434950924)

[5.2 Bilan humain et technique 26](#_Toc434950925)

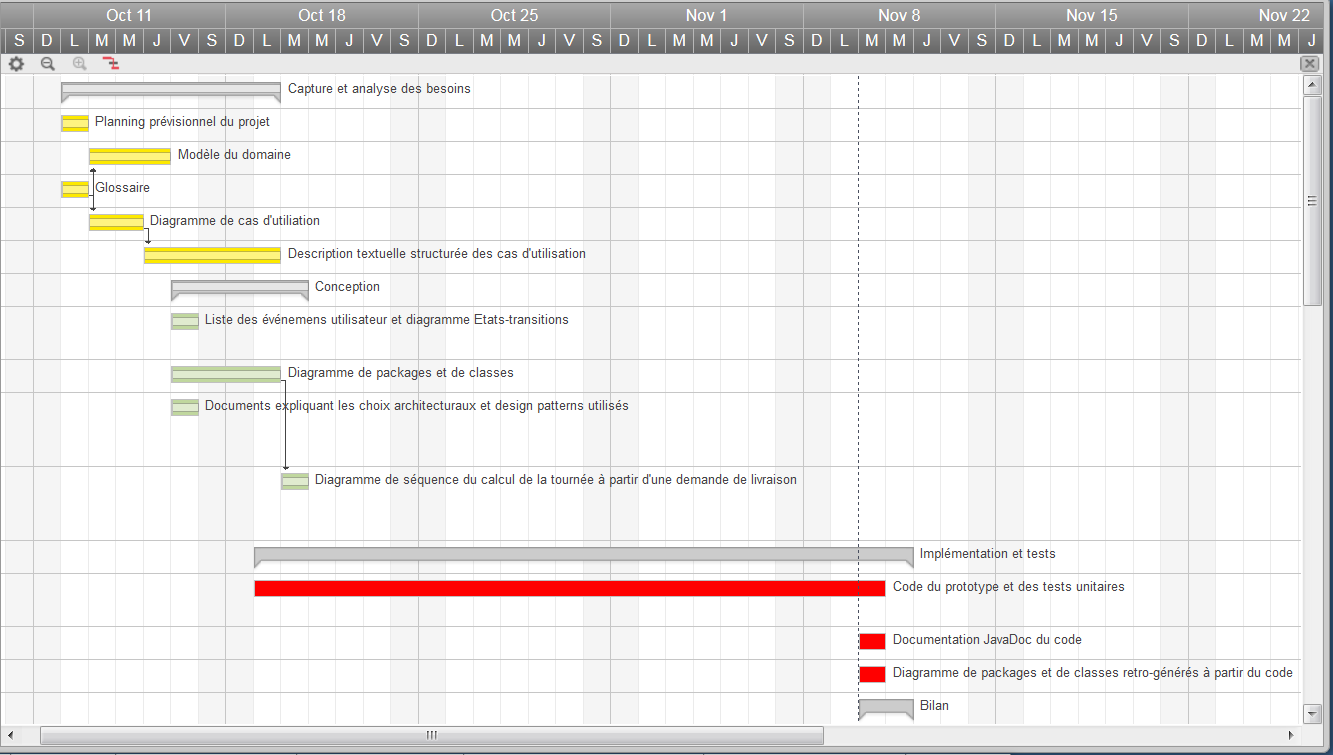
# 1. Introduction

L'application développée est inspirée d'un projet réel, piloté par le Grand Lyon entre 2012 et 2015, qui vise à optimiser la mobilité durable en ville (voir www.optimodlyon.fr). Nous nous focaliserons ici sur la partie du projet concernant le fret urbain et l'optimisation des tournées de livraisons en ville.

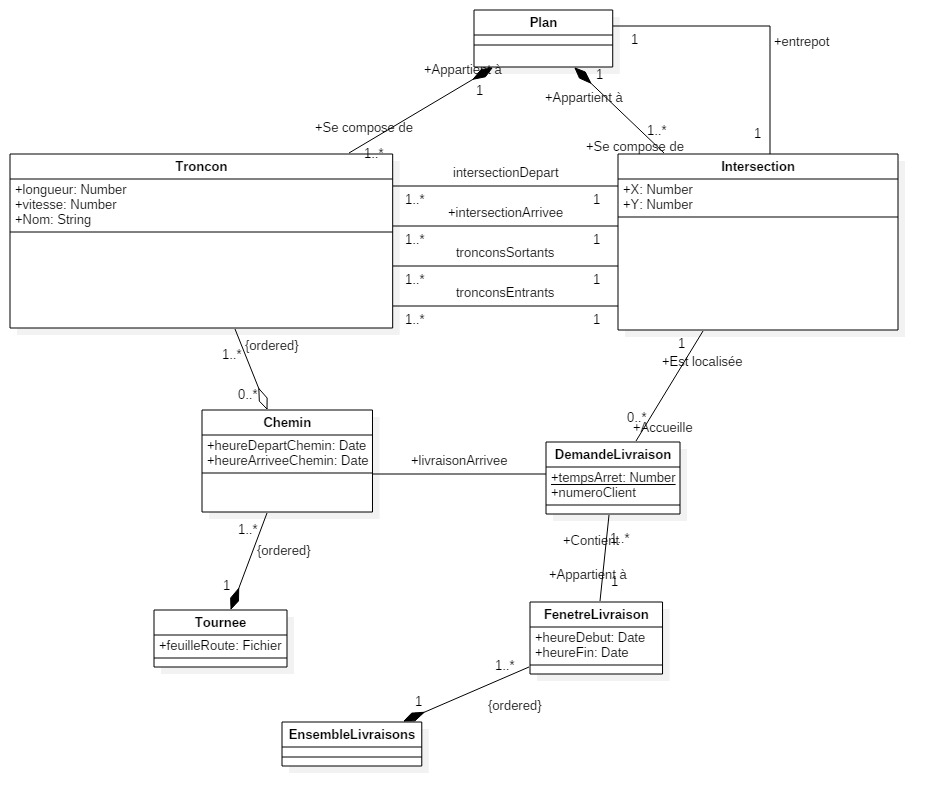
Pour ce faire, nous avons adopté une méthodologie de développement orienté objet. Nous avons commencé par une capture et une analyse des besoins, ensuite une conception détaillée pour en finir avec une implémentation et des tests de l'application en question. Pour satisfaire le cahier des charges avec les fonctionnalités demandées, nous avons consacré 4 semaines de travail avec une charge de xxx heures. Le groupe s'est mis d'accord sur le langage orienté objet java vu sa performance et sa réputation mondiale et le framework SWING pour la partie IHM.

# 2. Livrables de capture et analyse des besoins

## 2.1 Planning prévisionnel du projet



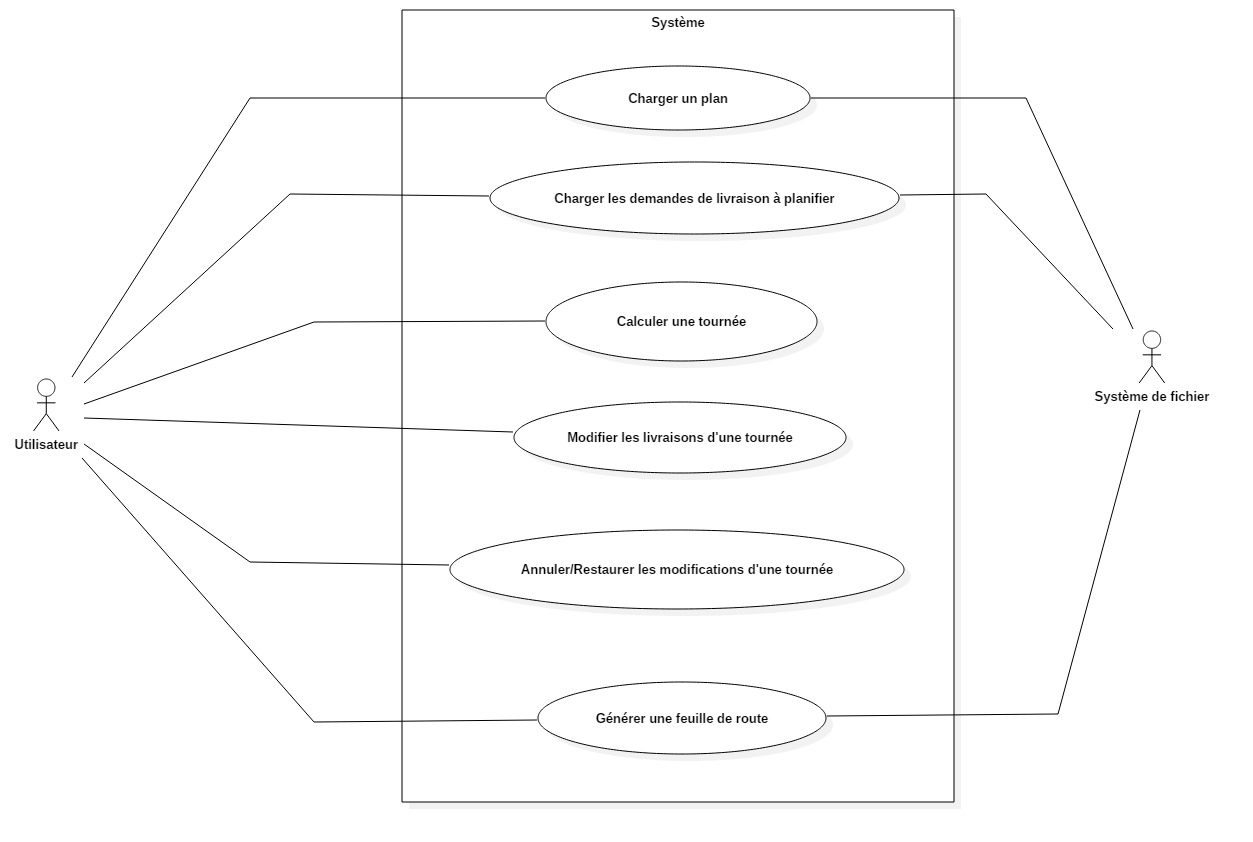
## 2.2 Modèle du domaine



## 2.3 Glossaire

* Chemin : Chemin entre deux livraisons.
* Heure de départ (d’un Chemin) : Heure à laquelle le livreur quitte la livraison précédente (donc temps de livraison non compris) ou l’entrepôt.
* Heure d’arrivée (d’un Chemin): Heure à laquelle le livreur arrive à la livraison suivante, nommée livraison arrivée (temps de livraison non compris).
* livraisonArrivee : La dernière livraison d'un chemin.
* Demande de livraison : Une demande de livraison.
* Temps d’arrêt (d’une demande de livraison) : Temps d'arrêt pour une livraison, ici fixé à 10 minutes pour chacune d'entre elles.
* Fenêtre de livraison : Regroupement des demandes de livraisons partageant la même fenêtre temporelle de livraison.
* Heure de début (d’une fenêtre de livraison) : Heure à partir de laquelle les demandes de livraison d’une fenêtre de livraison sont attendues.
* Heure de fin (d’une fenêtre de livraison) : Heure jusqu’à laquelle les demandes de livraison d’une fenêtre de livraison sont attendues.
* Ensemble de livraisons : Regroupement, ordonné temporellement, des fenêtres de livraison, qui contiennent-elles même les demandes de livraison.
* Intersection : Intersection de deux ou plusieurs tronçons. Une intersection peut correspondre à un point de livraison.
* x (d’une intersection) : Coordonnée X d’une l'intersection dans le plan.
* Y (d’une intersection): Coordonnée Y d’une l'intersection dans le plan.
* Tronçons sortants (d’une intersection) : Ensemble des tronçons quittant une intersection.
* Tronçons entrants (d’une intersection) : Ensemble des tronçons arrivant sur une intersection.
* Tronçon : Route empruntable entre deux intersections.
* Longueur (d’un tronçon) : longueur physique d'un tronçon.
* Vitesse (d’un tronçon) : vitesse moyenne du livreur sur le tronçon.
* Intersection de départ (d’un tronçon) : intersection à partir de laquelle commence un tronçon
* Intersection d’arrivée (d’un tronçon) : intersection sur laquelle arrive un tronçon.
* Plan : Cartographie les intersections et les tronçons.
* Tournée : Ensemble ordonné de chemins. Détermine la totalité du trajet du livreur depuis l'entrepôt jusqu'à l'entrepôt, en passant par les points de livraison.
* Feuille de route : fichier au format texte généré donnant la liste des livraisons à faire, dans l'ordre, avec l'adresse, horaires de livraison (arrivée et départ) et l'itinéraire entre chaque livraison.
* Entrepôt : Intersection de départ et d'arrivée d'une tournée.

## 2.4 Diagramme de cas d'utilisation



## 2.5 Description textuelle structurée des cas d'utilisation

**Cas d'utilisation:**

Charger un plan

**Description abrégée:**

L’utilisateur demande au système de charger un plan. L'utilisateur choisit un fichier XML décrivant le plan à charger. Le système affiche le plan.

**Précondition:**

L'application est lancée

**Scénario principal:**

1) Le système demande à l’utilisateur de choisir un fichier XML décrivant le plan à charger.

*2)* L'utilisateur choisit le fichier xml valide.

*3)* Le système affiche le plan et rend la fonctionnalité "Charger demandes de livraison" active.

**Alternatives:**

*2a)* Le fichier choisit par l’utilisateur n’est pas valide (se référer à la liste des erreurs pour la définition des cas de non validité du fichier). Le système indique que le fichier n'est pas valide et retourne à l'étape 1.

*1-3a)* L'utilisateur indique au système qu'il souhaite annuler le chargement du fichier. Le système annule la demande de saisie.

Liste d’erreurs : - Le fichier xml n’est pas syntaxiquement valide.

- L’un des tronçons décrit dans le fichier xml référence une intersection inconnue.

**Cas d'utilisation:**

Charger les demandes de livraisons à planifier

**Description abrégée:**

L’utilisateur demande au système de charger des demandes de livraison à planifier. L'utilisateur choisit un fichier XML décrivant les demandes de livraisons à planifier. Le système affiche la position de chaque demande de livraison sur le plan, ainsi que, pour chaque demande de livraison, sa plage horaire.

**Précondition:**

Un plan est chargé.

**Scénario principal:**

1) Le système demande à l’utilisateur de choisir un fichier XML décrivant les demandes de livraisons à planifier.

2) L'utilisateur choisit le fichier contenant les demandes de livraisons à planifier.

3) Le système affiche le plan où chaque demande de livraison est mise en évidence.

**Alternatives:**

*2a)* Le fichier choisit par l’utilisateur n’est pas valide (se référer à la liste des erreurs pour la définition des cas de non validité du fichier). Le système indique que le fichier n'est pas valide et retourne à l'étape 1.

*1-3a)* L'utilisateur indique au système qu'il souhaite annuler le chargement du fichier. Le système annule la demande de saisie.

Liste des erreurs :

*-* Le fichier n'existe pas.

*-* Le fichier décrit des fenêtres de livraisons qui se chevauchent.

*-* Le fichier contient une adresse qui n'existe pas dans le plan.

*-* Le fichier décrit une fenêtre de livraison dont la date de fin est antérieure à la date de début. *-* Le fichier n’est pas syntaxiquement valide.

**Cas d'utilisation:**

Calculer une tournée.

**Description abrégée:**

L'utilisateur demande au système de calculer la tournée de livraison. Le système calcule la tournée à partir des données qu'il possède sur les points de livraison puis affiche l'itinéraire à emprunter sur le plan. Le système affiche aussi, dans une autre partie de l'écran, les livraisons dans l'ordre dans lesquelles elles vont être réalisées.

**Préconditions:**

Un plan est chargé.

Un ensemble de points de livraison est chargé.

**Scénario principal:**

1) Le système calcule l'itinéraire à partir des données qu'il possède sur les points de livraison (fenêtre temporelle et localisation).

2) Le système affiche sur le plan, l'itinéraire à emprunter.

3) Le système affiche la liste des livraisons dans un autre cadre de l'écran. Cette liste respecte le même ordre que celui de la tournée.

**Alternatives:**

*1a)* Le temps de calcul de la tournée par le système dépasse le temps limite de calcul fixé. On passe à l’étape 2, mais l’itinéraire affiché par le système n’est pas l’itinéraire optimum, c’est le meilleur itinéraire que le système ait pu calculer dans le temps limite.

*3a)* Certaines livraisons ne respectent pas la fenêtre temporelle qui leur imposée.

Ces livraisons sont signalées à l'utilisateur par un indicateur visuel.

**Cas d’utilisation :**

Modifier les livraisons d’une tournée

Ce cas d’utilisation est détaillé ici en plusieurs scenarios possibles de modification

***Cas d’utilisation 1:***

Supprimer des livraisons

**Description abrégée:**

L'utilisateur sélectionne un point de livraison d'une tournée qu’il souhaite supprimer de la tournée et demande au système de supprimer cette livraison. Le système effectue la suppression et affiche la nouvelle tournée.

**Précondition:**

Le plan est chargé, la demande est chargée et une tournée est calculée.

**Scénario principal:**

*1*) L'utilisateur choisit un point de livraison qui appartient à la tournée calculée.

*2)* L'utilisateur demande au système de supprimer le point de livraisons choisi.

3) Le système enlève le point de livraison choisi de la tournée.

*4)* Le système affiche la nouvelle tournée modifiée.

**Alternatives:**

*2a)* Le point de livraison choisi n'appartient pas à la tournée calculée. Le système indique que le point n'est pas valide et retourne à l'étape 1.

*2b)* Le point sélectionné est l'entrepôt. Le système indique qu'il n'est pas possible de supprimer l'entrepôt.

***Cas d’utilisation 2:***

Ajouter des livraisons

**Description abrégée:**

L'utilisateur choisit un point de livraison sur le plan et demande au système d’ajouter ce point de livraison. Le système affiche la nouvelle tournée contenant le point de livraison sélectionné.

**Précondition:**

Le plan est chargé, la demande est chargée et une tournée est calculée.

**Scénario principal:**

*1)* L'utilisateur choisit un point de livraison à ajouter à la tournée.

*2)* L'utilisateur demande au système d’ajouter le point de livraison choisi.

3) Le système demande à l’utilisateur de choisir un point de livraison de la tournée avant lequel le point de livraison choisi précédemment doit être ajouté.

4) L’utilisateur choisit le point demandé.

5) Le système ajoute à la tournée le point de livraison choisi.

6) Le système affiche la nouvelle tournée.

**Alternatives:**

*2a)* Le point sélectionné appartient à la tournée calculée. Le système indique que le point n'est pas valide et retourne à l'étape 1.

*4a)* Le point sélectionné n’appartient pas à la tournée calculée. Le système indique que le point n'est pas valide et retourne à l'étape 3.

***Cas d’utilisation 3:***

Echanger deux livraisons

**Description abrégée**:

L'utilisateur choisit deux points de livraison de la tournée calculée et demande au système d’échanger les points de livraison. Le système effectue le changement et affiche la nouvelle tournée avec les deux points de livraison échangés.

**Préconditions:**

Le plan est chargé, les points de livraison sont chargés et une tournée est calculée.

**Scénario principal:**

*1)* L'utilisateur choisit un point de livraison.

*2)* L'utilisateur demande au système d’échanger ce point de livraison avec un autre point de livraison de la tournée.

3) Le système demande à l’utilisateur de choisir le point de livraison avec lequel il veut échanger le premier point de livraison sélectionné.

4) L’utilisateur choisit le point demandé.

5) Le système échange les deux points de livraison choisis.

*6)* Le système affiche la nouvelle tournée.

**Alternatives:**

*4a)* Le point sélectionné n'appartient pas à la tournée calculée. Le système indique que le point n'est pas valide et retourne à l'étape 1.

*4b)* Un point sélectionné est l'entrepôt. Le système indique qu'il n'est pas possible d'échanger un point de livraison avec l'entrepôt.

**Cas d'utilisation:**

Annuler/Restaurer les modifications d'une tournée

Ce cas d’utilisation est détaillé ici en plusieurs scenarios possibles

***Cas d’utilisation 1:***

Annuler la dernière modification de la tournée.

**Description abrégée:**

L'utilisateur demande au logiciel d'annuler la dernière modification. La tournée revient dans l'état antérieur à la dernière modification.

**Précondition:**

Une modification a été apportée à la tournée proposée par le logiciel.

**Scénario principal:**

*1)* L'utilisateur demande au logiciel d'annuler la dernière modification.

*2)* Le système revient dans l'état où il était avant la dernière modification.

**Alternatives:**

*2a)* Aucune modification à annuler. Le système ne fait rien et l’étape 2) est ignorée

***Cas d’utilisation 2:***

Restaurer la dernière modification annulée de la tournée.

**Description abrégée:**

L'utilisateur demande au logiciel de restaurer la dernière modification annulée.

La tournée revient dans l'état antérieur à la dernière annulation.

**Précondition:**

Une modification apportée à la tournée proposée par le logiciel a été annulée.

**Scénario principal:**

*1)* L'utilisateur demande au logiciel de restaurer la dernière modification annulée.

*2)* Le système revient dans l'état où il était avant la dernière annulation.

**Alternatives:**

*2a)* Aucune modification à restaurer. Le système ne fait rien et l’étape 2) est ignorée

**Cas d'utilisation:**

Générer la feuille de route.

**Description abrégée:**

L'utilisateur valide la tournée et demande au système de générer la feuille de route correspondante. Le système génère la feuille de route à partir de la tournée validée. Le système enregistre la feuille de route calculée dans un fichier texte.

**Préconditions:**

Un plan est chargé.

Un ensemble de points de livraison est chargé.

La tournée a été calculée.

**Scénario principal:**

*1)* L'utilisateur valide la tournée et demande au système de calculer la feuille de route.

*2)* Le système demande à l'utilisateur de choisir un emplacement et un nom de fichier pour la sauvegarde de la feuille de route.

*3)* L'utilisateur indique au système un nom et un emplacement de fichier.

*4)* Le système génère la feuille de route.

*5)* Le système sauvegarde la feuille de route dans le fichier renseigné à l'étape 3.

**Alternatives:**

*2a)* L'utilisateur choisit d'annuler la génération de la feuille de route. Le système termine la procédure de génération de la feuille de route.

*3a)* Il n'est pas possible de créer ou d'écrire à l'emplacement renseigné. Le système indique la nature du problème et retourne à l'étape 2.

*3b)* Il existe déjà un fichier à l'emplacement renseigné par l'utilisateur.

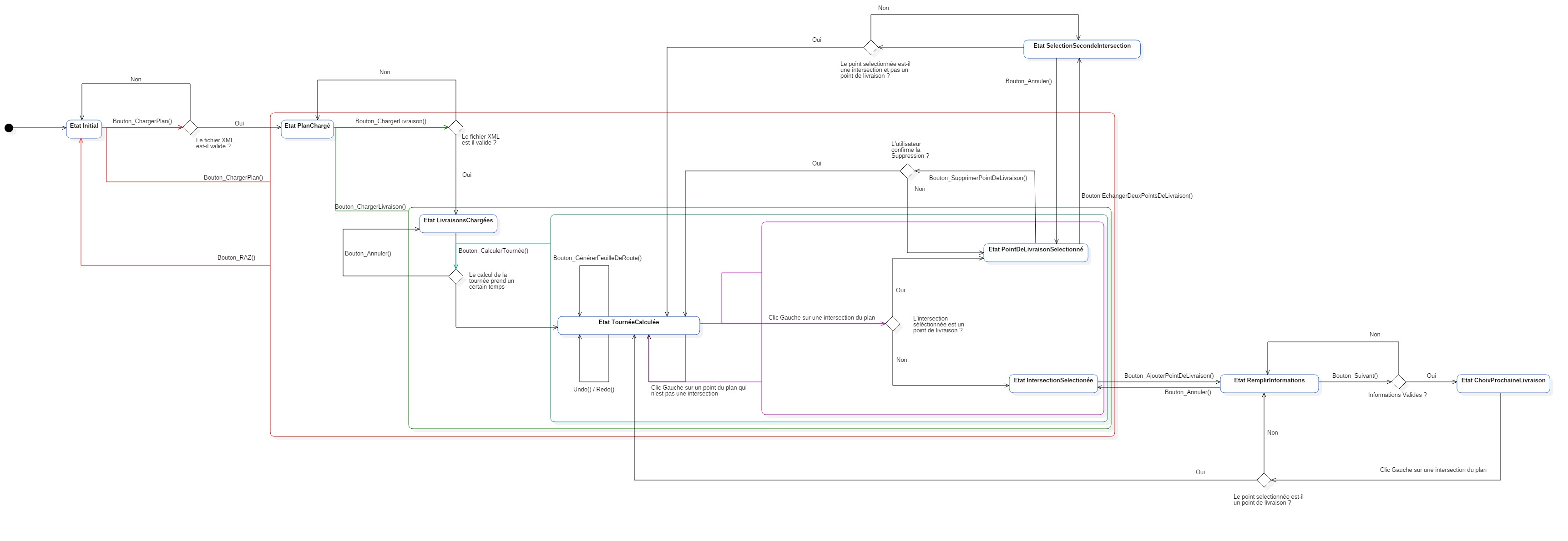
*3b-1)* Le système demande à l'utilisateur s'il doit écraser l'instance déjà existante du fichier.

*3b-2a)* L'utilisateur confirme l'écrasement du fichier déjà existant. Le système écrasera l'instance existante du fichier à l'étape 5. Le système passe à l'étape 4.

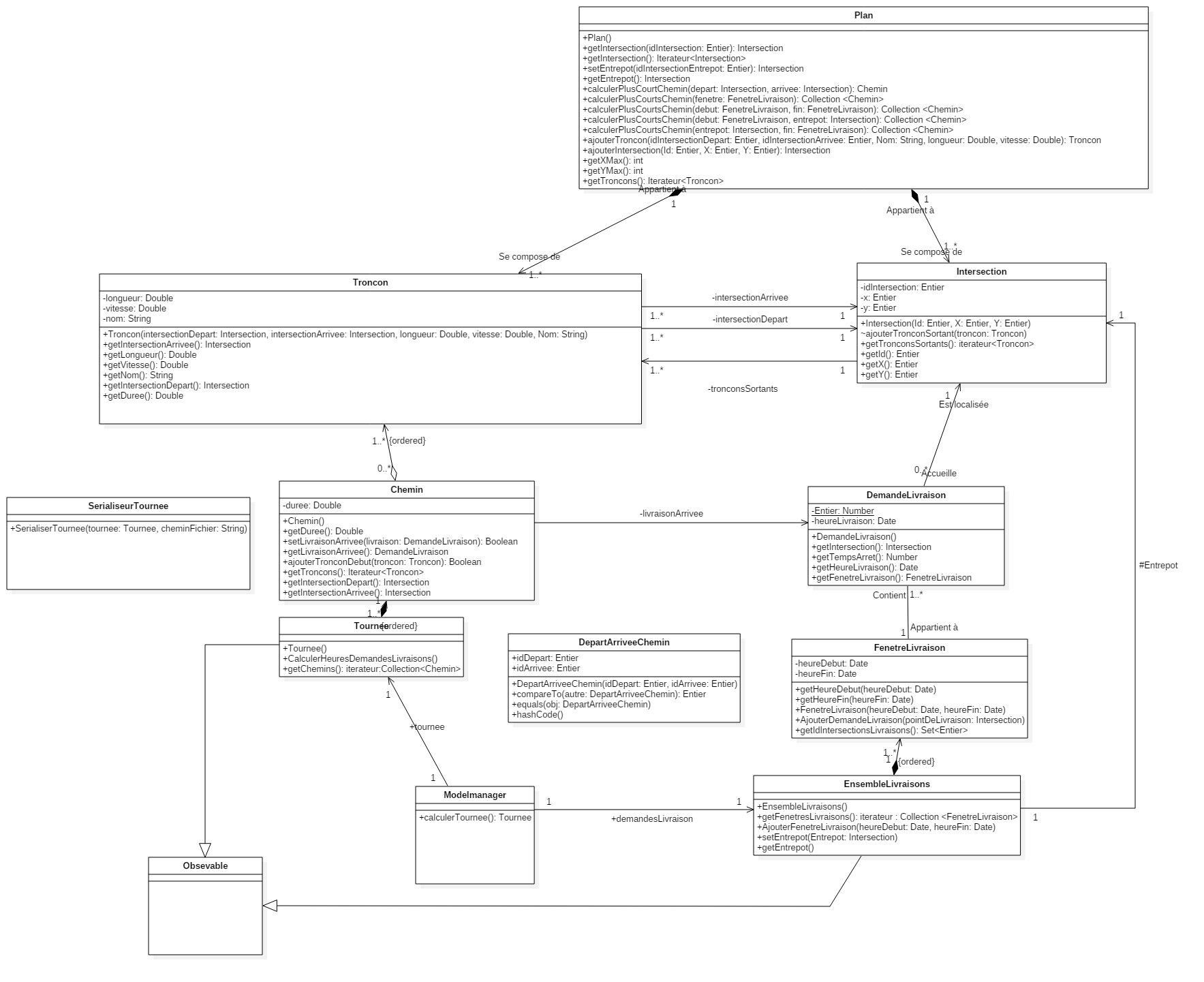
*3b-2b)* L'utilisateur refuse l'écrasement du fichier déjà existant. Le système retourne à l'étape 2.

# 3. Livrables de conception:

## 3.1 Liste des événements utilisateur et diagramme Etats-transitions



## 3.2 Diagrammes de packages et de classe



## 3.3 Diagrammes de packages et de classes retro-générés à partir du code

## 2.4 Choix architecturaux et design patterns utilisés

Au niveau de l’architecture générale du projet, nous avons construit l’application sur une architecture MVC (Modèle-Vue-Contrôleur). Premièrement car cette architecture était imposée, et ensuite car cela correspondait au type d’application développé ici.

Pattern undo/redo

Dans le cahier des charges, il est spécifié qu’une modification de la tournée doit pouvoir être annulée. Une telle spécification appelle clairement à l’utilisation du pattern Commande, qui permettra de faire aisément des undo/redo sur les modifications.

ModeleManager

Lorsque le contrôleur communique avec le modèle, il est souhaitable que la communication se fasse au moyen d’un unique point d’entrée sur le modèle. Nous avons fait le choix d’une classe ModeleManager, qui contiendrait alors le plan actuellement chargé, les demandes de livraison actuellement chargées, et la tournée actuellement générée. C’est aussi cette classe qui contient la méthode calculerTournee. À noter que le modeleManager aurait pu être implémenté en utilisant le pattern singleton (une seule et unique instance de ModeleManager dans l’application). Cependant, dans un souci de réutilisabilité, nous avons écarté ce choix. En effet, dans le cas où, par exemple, l’application permettrait d’avoir plusieurs onglets contenant plusieurs plans différents en même temps, il serait pratique d’avoir plusieurs instances de ModeleManager.

Package XML

Les descriptions des plans de villes et les descriptions des demandes de livraisons sont réalisées au moyen de fichiers au format xml. Ces fichiers doivent être parsés, et les informations contenues doivent être analysées, puis envoyées dans le modèle. Nous avons décidé de séparer le code relatif au traitement xml dans deux packages séparés : Un package xml, qui concerne uniquement la gestion globale de fichier au format xml (ouverture, sauvegarde, sélection de fichier), et un package xmlModele, qui contient toutes les méthodes de parsing et de remplissage des objets métier. Ainsi, le package xml est générique et pourrait être réutilisé dans une autre application. Le package xmlModele, quant à lui, est intrinsèquement lié au modèle.

Pattern State

L’application que nous développons possède plusieurs états de fonctionnement (Le plan est chargé, puis les demandes de livraisons sont chargées, etc.) Les possibilités d’interaction utilisateur et les effets des interactions ne sont pas les même selon l’état de l’application. Afin que le contrôleur puisse gérer les différents états, nous avons implémenté le pattern state.

**Gestion des fonctionnalités actives en fonction des états**

Selon l’état du contrôleur, toutes les fonctionnalités de l’application ne doive pas pouvoir être appelée (Lorsqu’un plan n’est pas chargé, par exemple, l’utilisateur ne devrait pas pouvoir demander à l’application de charger des demandes de livraison). C’est pourquoi nous avons mis en place un système d’activation/désactivation de fonctionnalité. La fenêtre graphique de l’application implémente une méthode d’activation/désactivation de fonctionnalité pour chaque fonctionnalité (on entend ici les boutons, les items des menus, .etc.). A chaque changement d’état, l’état désactive l’ensemble des fonctionnalités et réactive les fonctionnalités permises dans cet état.

Architecture de la vue

La partie graphique de l’application est constituée d’un objet Fenetre, qui contient un état de l’application dans la vue, modélisé par l’objet Vue. Nous avons essayé d’implémenter notre vue de la manière la plus générique possible. Par exemple, dans le cas où l’application permettrait d’avoir plusieurs onglets contenant plusieurs plans différents en même temps, l’objet Vue déjà existant peut facilement être assimilé à un onglet. Il n’y aurait pas beaucoup de changement à faire dans le code.

Nous avons aussi réalisée une classe de vue dédiée à chacun (ou presque) des objets du modèle, et mis en place des liens entre ces classes qui sont semblables aux liens entre les classes du modèle. Ainsi on peut facilement paramétrer l’apparence graphique de chacun des éléments, tout en conservant la logique métier.

## 3.5 Diagramme de séquence du calcul de la tournée à partir d'une demande de livraison

**Diagramme de séquence de calcul de la tournée:**

C:\Users\Med\Desktop\z.tif

**Diagramme de séquence de calcul de chemin :**

C:\Users\Med\Desktop\a.tif

**Diagramme de séquence de modélisation du Graphe TSP:**

C:\Users\Med\Desktop\b.tif

**Diagramme de séquence de transformation de la solution TSP en tournée**

C:\Users\Med\Desktop\c.tif

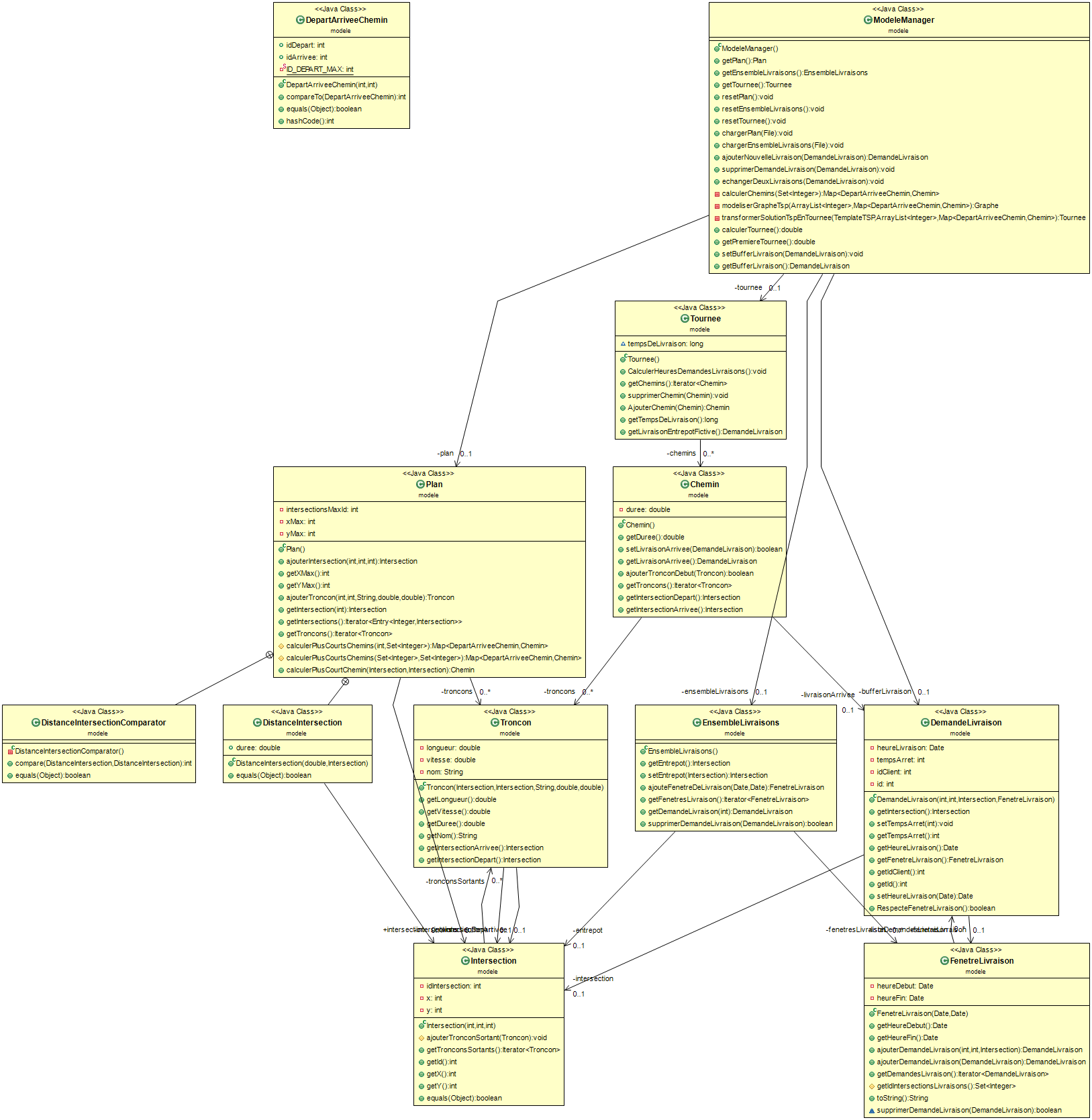
# 4. Implémentation et tests

## 4.1 Code du prototype et des tests unitaires

## 4.2 Documentation JavaDoc du code

## 4.3 Diagrammes de packages et de classes rétro-générés à partir du code

**Le package Modèle:**



**Explication de la différence entre le diagramme de classe du modèle de conception et celui rétro-généré:**

Plusieurs erreurs mineures de conception ont été détectées durant l'implémentation du modèle. Des méthodes manquaient pour modifier des attributs privés.

Certaines méthodes prenaient en charge trop de fonctionnalités, il a fallu les scinder en plusieurs parties. Une partie au sein de la même classe en visibilité privée, et une partie dans d'autres classes concernées, en visibilité package ou bien publique.

Le comportement de certaines méthodes a dû être revu pour faciliter l'implémentation. C'est notamment le cas de la méthode ajouterTroncon de la classe Chemin, qui ajoute un troncon à la fin du chemin. Cette dernière est devenue ajouterTronconDebut, pour ajouter le troncon non pas à la fin, mais au début du chemin. En effet, cette méthode est uniquement utilisée lors de la construction du chemin à l'issue de l'algorithme de Dijkstra. Cette reconstruction se fait à l'envers. On part de l'intersection de d'arrivée, et on remonte dans l'arbre des plus courts chemins, vers l'intersection de départ. Dans ce contexte il parait naturel de modifier l'ajout de troncon comme expliqué ci-dessus.

Nous avons dû ajouter des classes pour l'implémentation du modèle. C'est par exemple le cas des classes : DepartArriveeChemin, DistanceIntersection et DistanceIntersectionComparator. Ces classes permettent d'agréger deux données dans le but de faire un tri. Les concepteurs du langage Java ont pris le parti de ne pas implémenter d'objet représentant une paire. Nous avons donc du implémenter des paires spécifiques à nos besoins. Par exemple DistanceIntersection représente une intersection et sa distance vers un point de départ. Elle est utile pour prendre place dans la file de priorité de l'algorithme de Dijkstra. DistanceIntersectionComparator permet de comparer deux DistanceIntersection. Celle des deux ayant la plus petite distance est remontée en priorité dans la file.

Enfin, certains attributs et méthodes ont été ajoutés pour optimiser les performances ou bien simplifier la vue. C'est par exemple le cas des attributs xMax et yMax de la classe Plan. Ceux-ci sont utiles dans la vue, pour connaitre les dimensions maximales de la zone visible sans parcourir toutes les intersections inutilement.

## 5. Bilan

## 5.1 Planning effectif du projet

## 5.2 Bilan humain et technique