INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

DEPARTEMENT INFORMATIQUE

Rapport du projet DevOO

Développement d'une application suivant une méthodologie Orientée Objet

OPTIMOD'LYON
OPTIMISER LA MOBILITÉ DURABLE EN VILLE

REALISE PAR L'HEXANOME H4402 :

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

ENCADRE PAR :

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

ANNEE UNIVERSITAIRE : 2015/2016

Table des matières

1. Introduction 3
2. Livrables de capture et analyse des besoins 3
2.1 Planning prévisionnel du projet 3
2.2 Modèle du domaine 4
2.3 Glossaire 4
2.4 Diagramme de cas d'utilisation 6
2.5 Description textuelle structurée des cas d'utilisation 6
3. Livrables de conception: 10
3.1 Diagramme Etats-transitions 10
3.2 Diagrammes de packages et de classe 12
Diagramme de classe du modèle: 12
Diagramme de classe du contrôleur: 13
Diagramme de classe de la vue : 14
3.3 Diagrammes de packages et de classes retro-générés à partir du code 15
Diagramme de classe rétro-généré du modèle: 15
Diagramme de classe rétro-généré du contrôleur : 17
Diagramme de classe rétro-généré de la vue: 19
2.4 Choix architecturaux et design patterns utilisés 20
3.5 Diagramme de séquence du calcul de la tournée à partir d'une demande de livraison 21
5. Bilan 26
5.1 Planning effectif du projet 26
5.2 Décompte des heures par personnes 26
5.3 Bilan humain et technique 27
Bilan de Guillaume: 27
Bilan de Mathieu: 27
Bilan de Mohammed: 27
Bilan de Kilian: 28
Bilan de Romain: 29
Bilan de Thomas: 29
Bilan de Nicolas, chef de projet: 30

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.

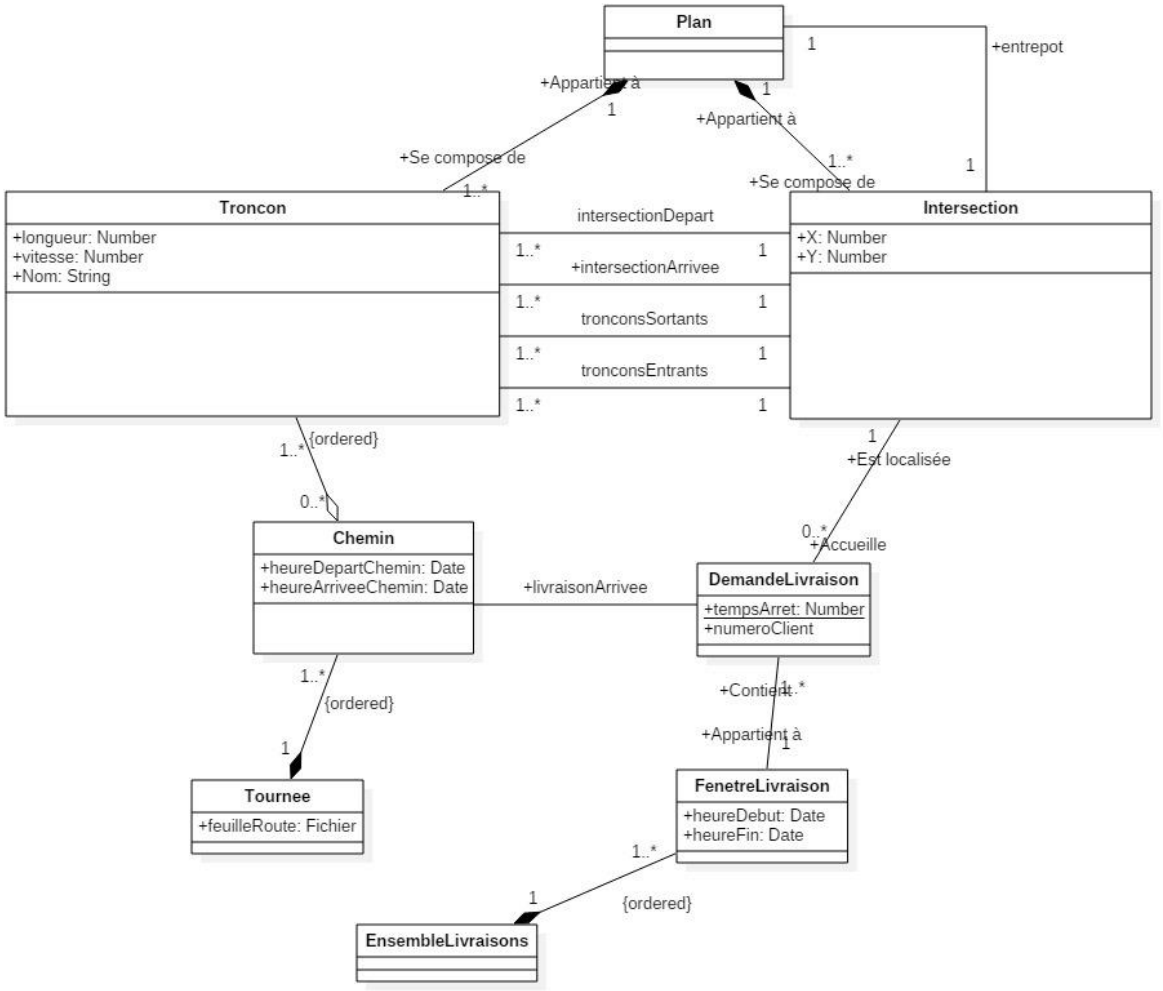
Pour ce faire, nous avons suivi une méthodologie de développement orienté objet. Nous avons commencé par une phase de capture et d'analyse des besoins, pour ensuite enchaîner sur une phase de conception détaillée et enfin implémenter notre application. Nous avons consacré 4 semaines de travail à ce projet, avec une charge de 310 heures au total. Nous avons utilisé le langage objet Java pour développer notre application, couplé au framework SWING pour la réalisation de l'IHM.

2. Livrables de capture et analyse des besoins

2.1 Planning prévisionnel du projet

	Romain	Mathieu	Thomas	Mohammed	Killian	Guillaume	Nicolas	Abordé en groupe
Séance 1 : Capture et analyse des besoins	Etude et essais des possibilités IHM de Java	Use case, diagrammes et description détaillée	Modèle de domaine et Glossaire	Use case, diagrammes et description détaillée	Modèle de domaine et Glossaire	Use case, diagrammes et description détaillée	Mise en place du planning prévisionnel et gestion des tâches	Modèle de domaine et Glossaire
Séance 2 : Conception	Diagrammes états transition	Diagramme de classe et de package	Diagramme de classe et de package	Diagramme de séquence du calcul de la tournée.	Diagramme de séquence du calcul de la tournée.	Diagramme de séquence du calcul de la tournée.	Diagramme de classe et gestion des tâches	Architecture globale de l'application
Séance 3 : Prise en main et développement	Développement et tests							
Séance 4 : Développement	Développement et tests							
Séance 5 : Fin développement, documentation	Développement et documentation							

2.2 Modèle du domaine



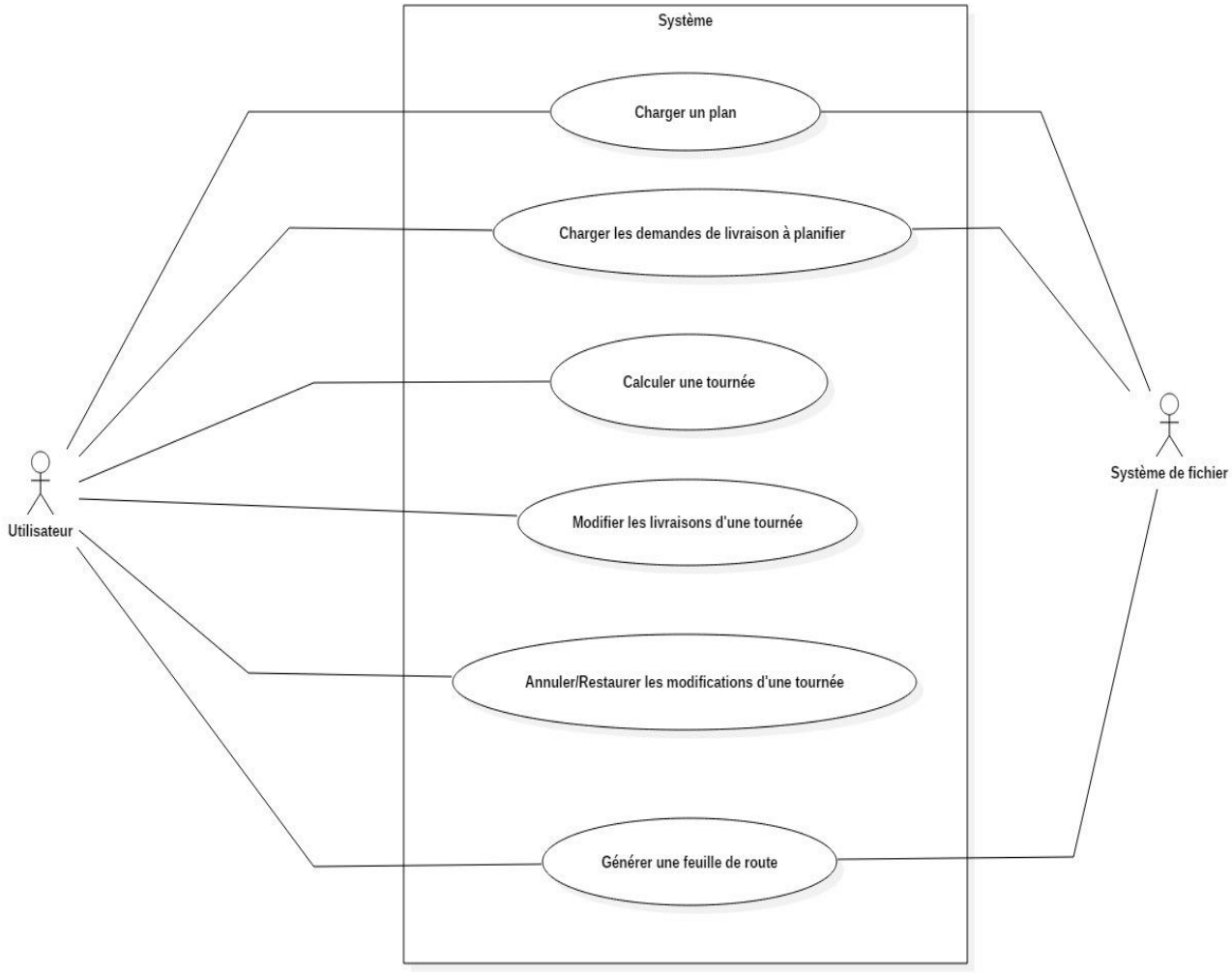
2.3 Glossaire

- **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.

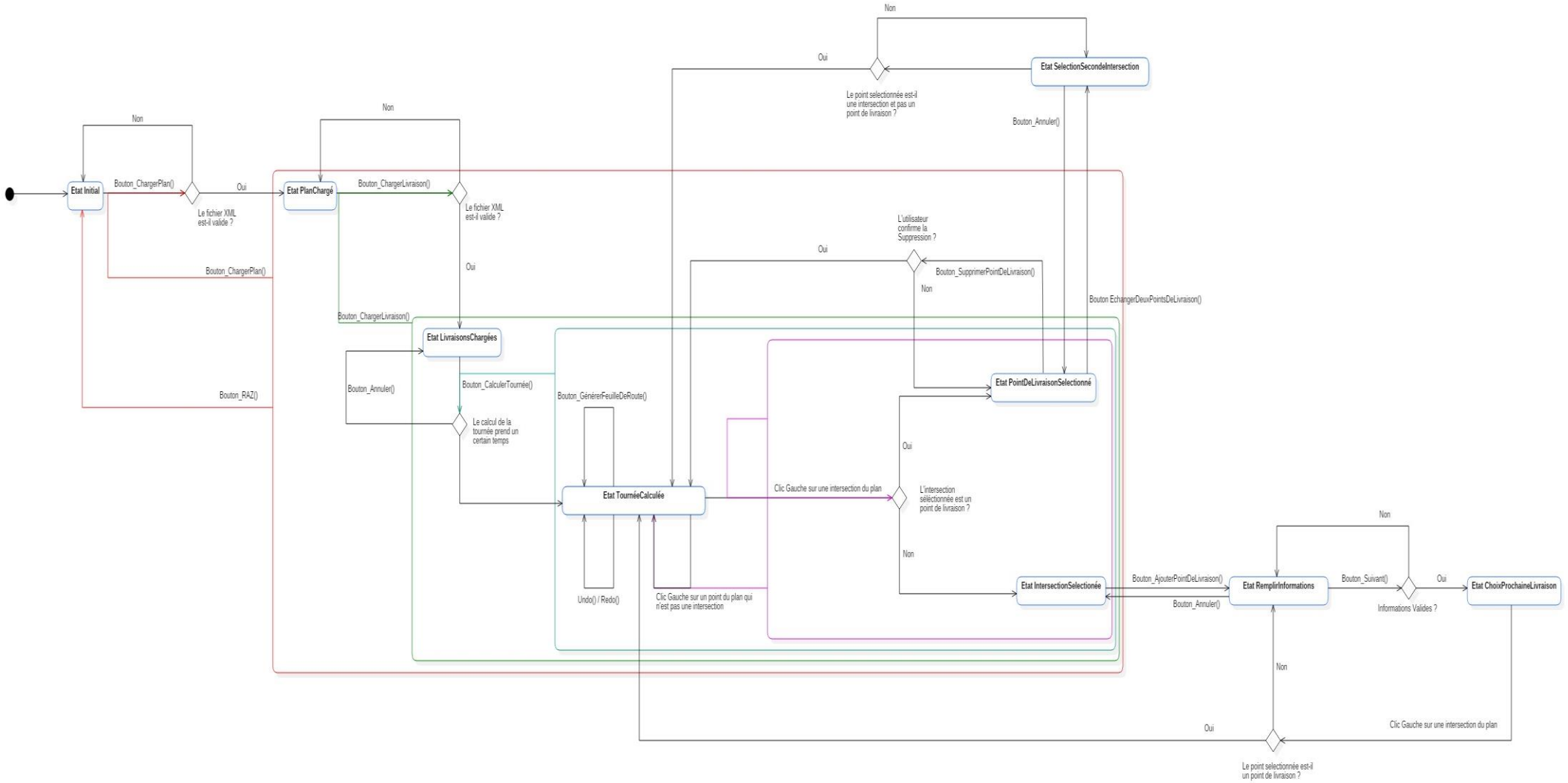
Alternatives:

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 Diagramme Etats-transitions

Suite à une erreur de notre part, notre diagramme état-transition a été réalisé avec le formalisme d'un diagramme d'activité. Sur les conseils de Mme Solnon, nous n'avons pas refait le diagramme car cette erreur de formalisme ne gêne pas la compréhension du diagramme. De plus, réaliser un diagramme sous format numérique se révèle être particulièrement long.


Etat Initial
Bouton_ChargerPlan()
Non
Oui
Le fichier XML est-il valide ?
Etat PlanChargé
Bouton_ChargerLivraison()
Non
Oui
Le fichier XML est-il valide ?
Etat LivraisonsChargées
Bouton_CalculerTournée()
Le calcul de la tournée prend un certain temps
Bouton_Annuler()
Bouton_RAZ()
Bouton_ChargerPlan()
Bouton_ChargerLivraison()
Etat TournéeCalculée
Bouton_GénérerFeuilleDeRoute()
Undo() / Redo()
Clic Gauche sur un point du plan qui n'est pas une intersection
Clic Gauche sur une intersection du plan
L'intersection séléctionnée est un point de livraison ?
Oui
Non
Etat PointDeLivraisonSelectionné
L'utilisateur confirme la Suppression ?
Oui
Non
Bouton_SupprimerPointDeLivraison()
Bouton EchangerDeuxPointsDeLivraison()
Etat SelectionSecondeIntersection
Le point selectionnée est-il une intersection et pas un point de livraison ?
Oui
Non
Bouton_Annuler()
Etat IntersectionSelectionée
Bouton_AjouterPointDeLivraison()
Bouton_Annuler()
Etat RemplirInformations
Bouton_Suivant()
Informations Valides ?
Non
Oui
Etat ChoixProchaineLivraison
Non
Clic Gauche sur une intersection du plan
Oui
Le point selectionnée est-il un point de livraison ?

3.2 Diagrammes de packages et de classe

Voici nos diagrammes de conception, réalisés pendant la phase de conception du projet. Ces diagrammes diffèrent plus ou moins du code actuel, car à l'implémentation nous avons ajouté/supprimé/modifié ce que nous avions conçu au départ. Dans la partie du rapport contenant les diagrammes rétro-générés vous trouverez des explications sur les différences avec les diagrammes de conception.

Diagramme de classe du modèle:

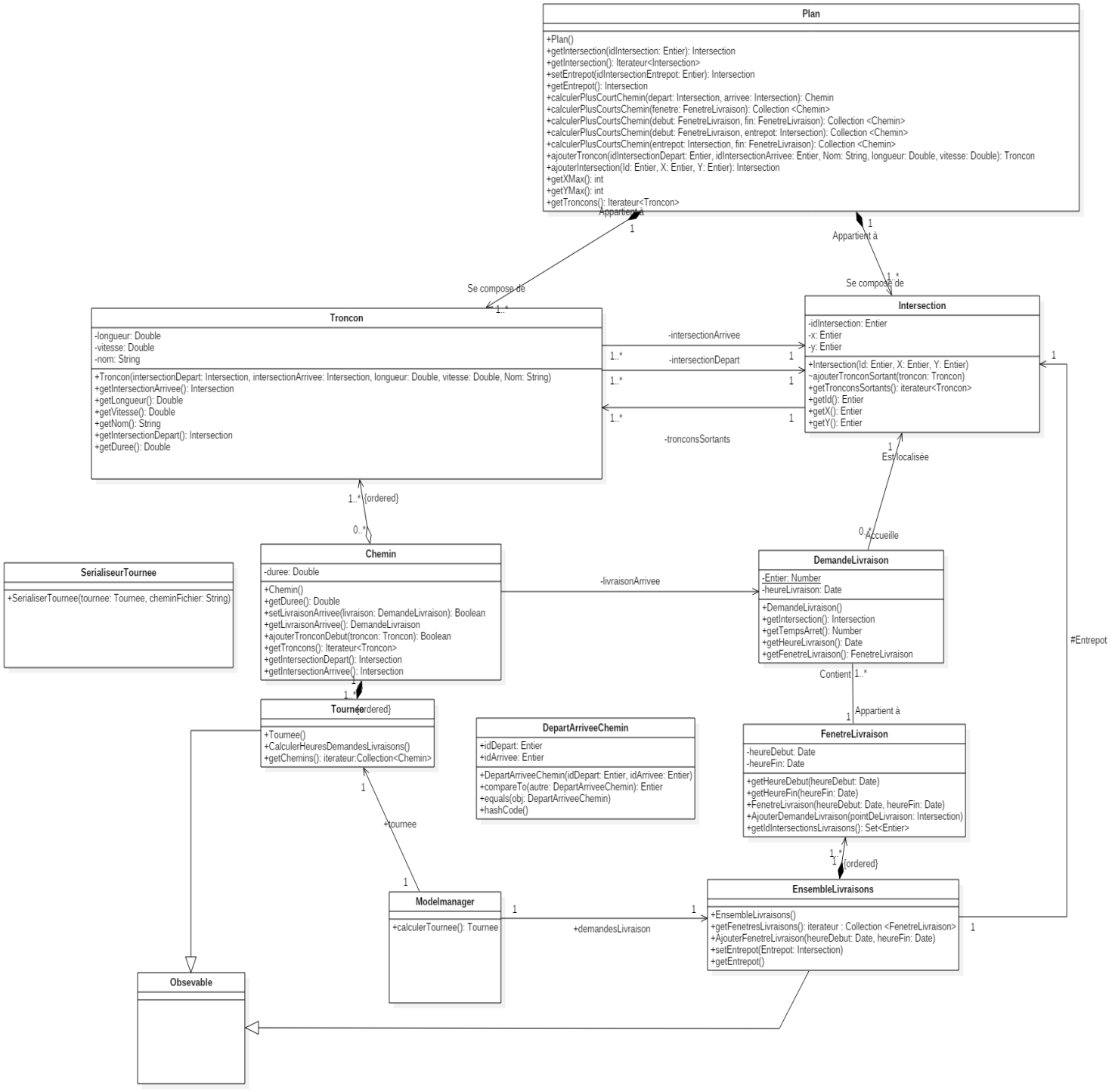


Diagramme de classe du contrôleur:

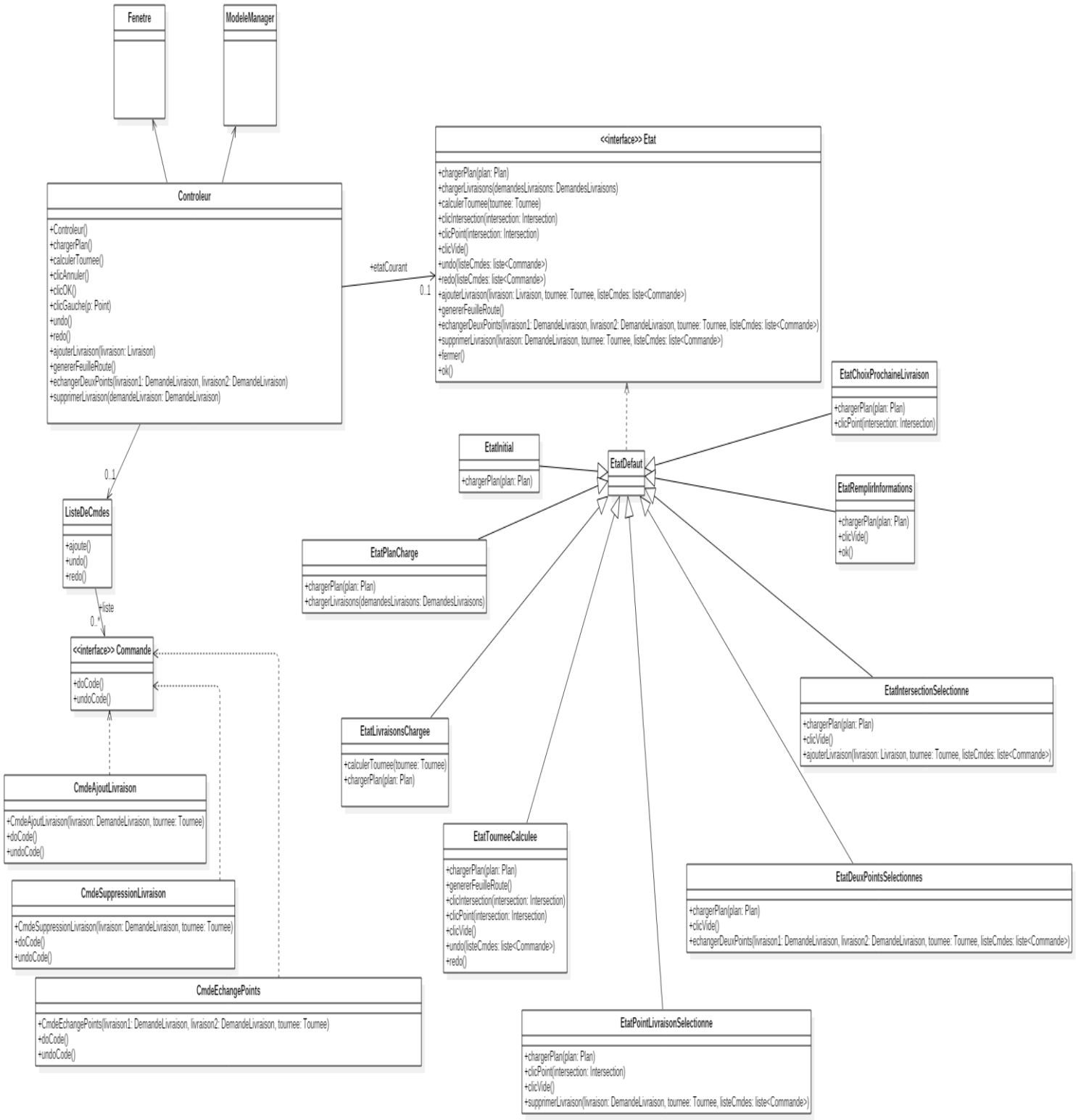
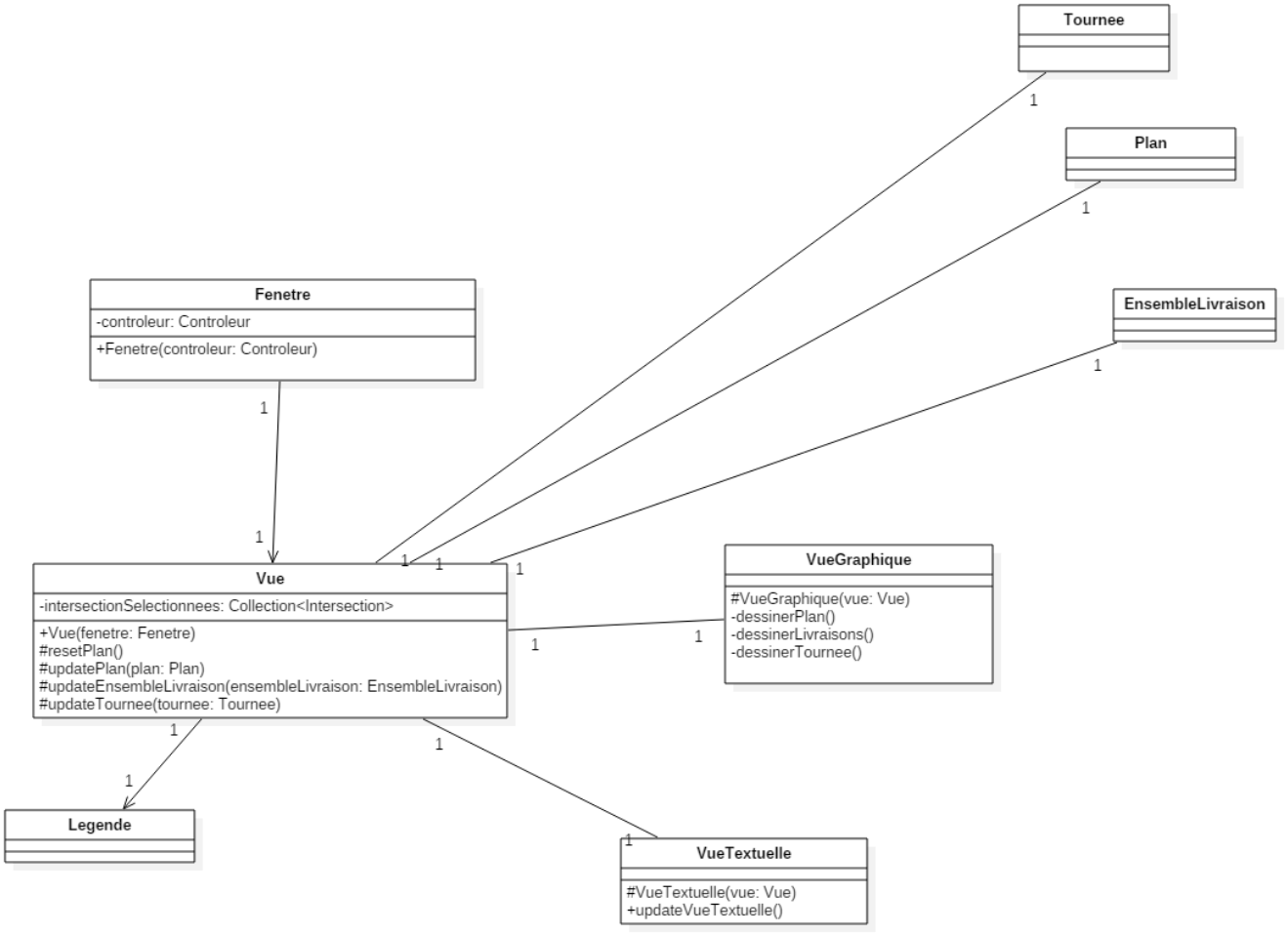
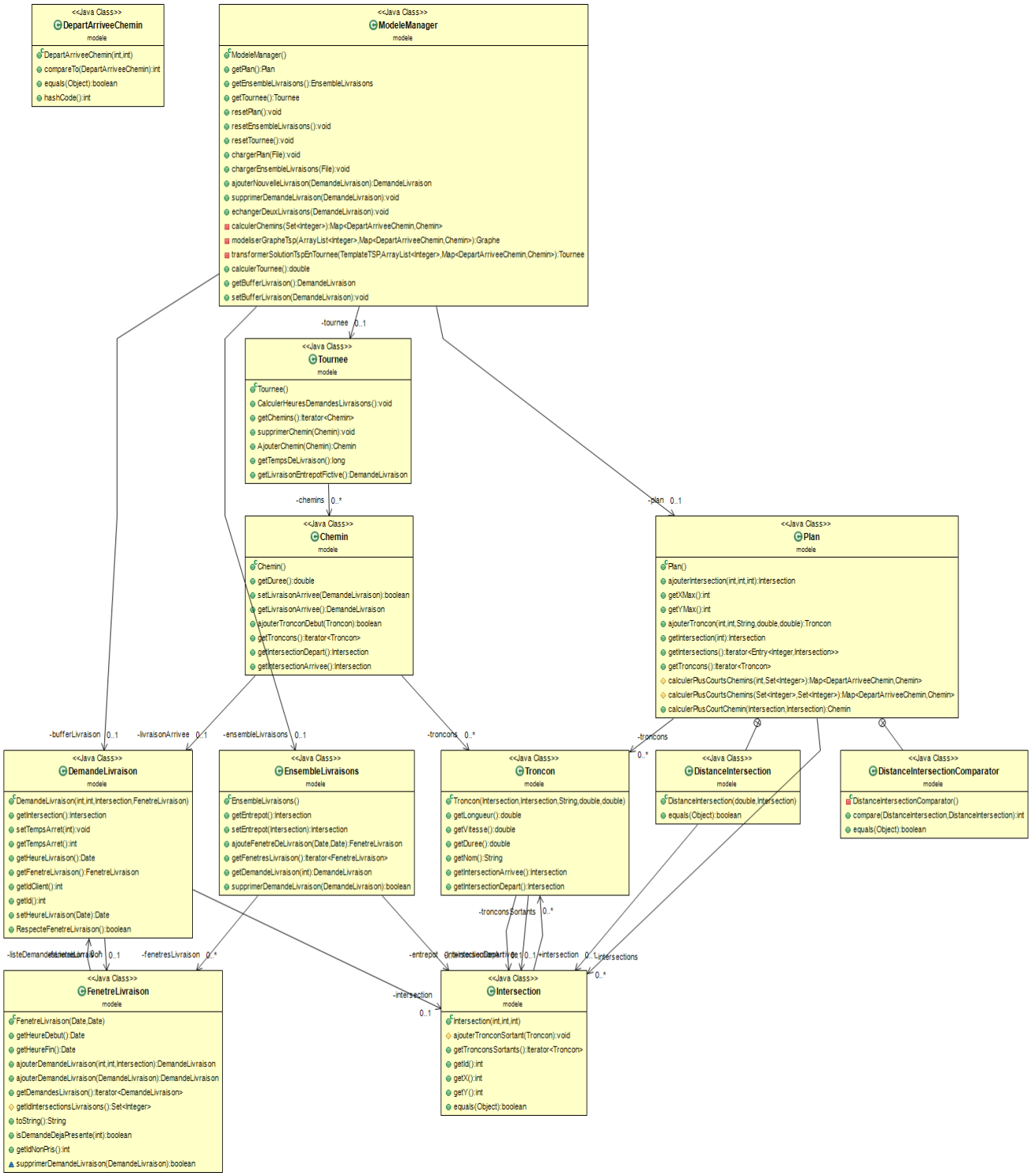


Diagramme de classe de la vue :



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

Diagramme de classe rétro-généré du 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.

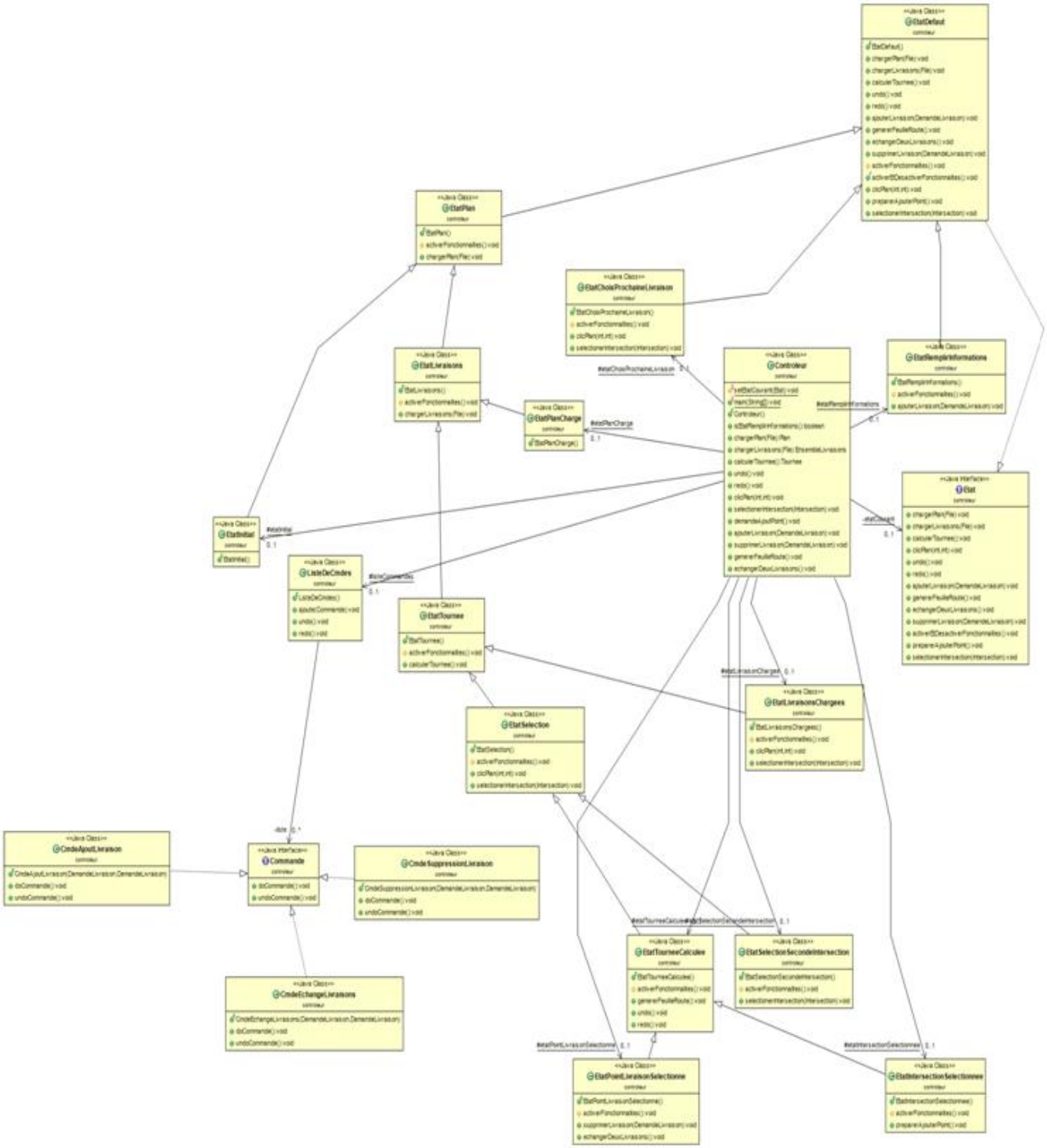
Certaines méthodes prenaient en charge trop de fonctionnalités, il a fallu les scinder en plusieurs parties. Une partie au sein de la même classe en visibilité privée, et une partie dans d'autres classes concernées, en visibilité package ou bien publique.

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 tronçon à la fin du chemin. Cette dernière est devenue ajouterTronconDebut, pour ajouter le tronçon 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 tronçon 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.

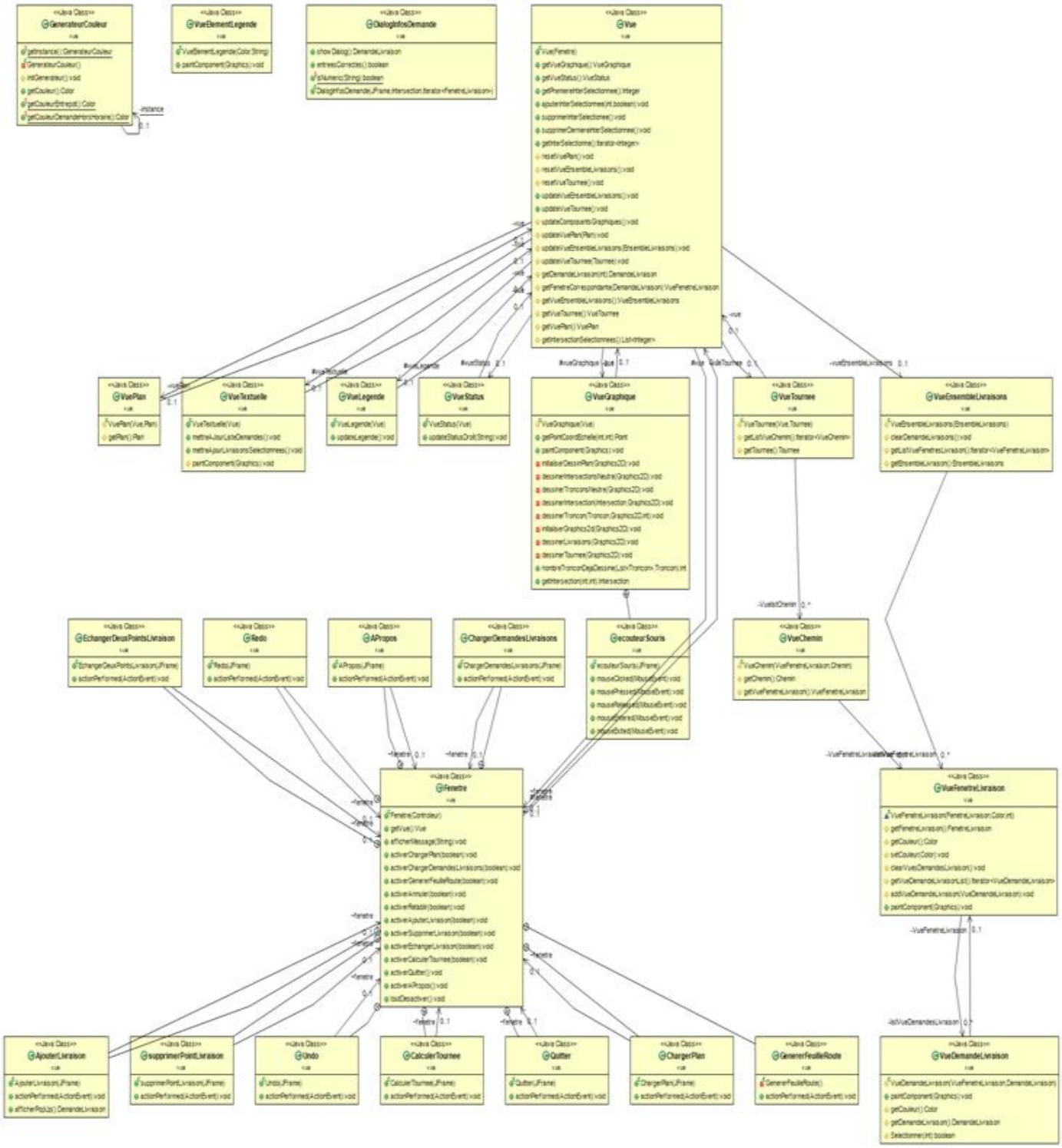
Diagramme de classe rétro-généré du contrôleur :



Explication de la différence entre le diagramme de classe du contrôleur de conception et celui rétro-généré à partir du code:

La différence principale concerne la structuration des états. Notre diagramme état transition est composé de « boites » d'état (par exemple, l'ensemble des états à partir desquels on peut charger un plan sont dans une certaine « boite » du diagramme). A la conception nous n'avions pas pensé à utiliser ces boites, mais à l'implémentation, lorsque nous avons dû réfléchir à un système d'activation/désactivation de fonctionnalités, il nous a alors fallu modifier notre structure des états. Ainsi, pour reprendre l'exemple précèdent, l'état EtatPlan représente l'ensemble des états à partir desquels on peut charger un plan, et tous les états de la « boite » hériteront de cet état, et pourront appeler la méthode d'activation de fonctionnalités de leur ancêtre.

Diagramme de classe rétro-généré de la vue:



Explication de la différence entre le diagramme de classe de la vue de conception et celui rétro-généré à partir du code:

Le diagramme de conception de la vue conceptuel est très différent du diagramme retro-généré. Cela est principalement dû à notre manque d'expérience dans la conception d'interface graphique, notamment en utilisant SWING.

On peut d'abord observer, qu'à la conception, nous n'avions pas pensé au système d'activation/désactivation de fonctionnalités. Ensuite, point principal de différence, nous n'avions pas pensé à faire une classe de vue pour chaque « objet métier ». Cela a été nécessaire pour pouvoir donner une couleur à chaque fenêtre par exemple, mais aussi pour pouvoir relier les demandes de livraison aux fenêtres à partir de la tournée.

Enfin le système de dessin du plan, ensemble de livraison et tournée est au final plus complexe que nous ne l'avions pensé à la base. Cependant, il aurait été difficile (et sûrement peu rentable) de chercher à définir précisément la méthode et le workflow de dessin au départ du projet.

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), car cette architecture était imposée, et car cela correspondait aussi 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 de fichier dans deux packages séparés : Un package io, qui concerne uniquement la gestion globale de fichier (ouverture de fichier

xml, sauvegarde de fichiers au format txt, sélection de fichier), et un package xmlModele, qui contient toutes les méthodes de parsing et de remplissage des objets métier. Ainsi, le package io 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 mêmes 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 doivent pas pouvoir être appelées (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 du modèle 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 changements à faire dans le code.

Nous avons aussi réalisé 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

Le diagramme de séquence de calcul de la tournée se révèle être très volumineux. C'est pourquoi nous l'avons séparé en quatre diagrammes.

Diagramme de séquence de calcul de la tournée:

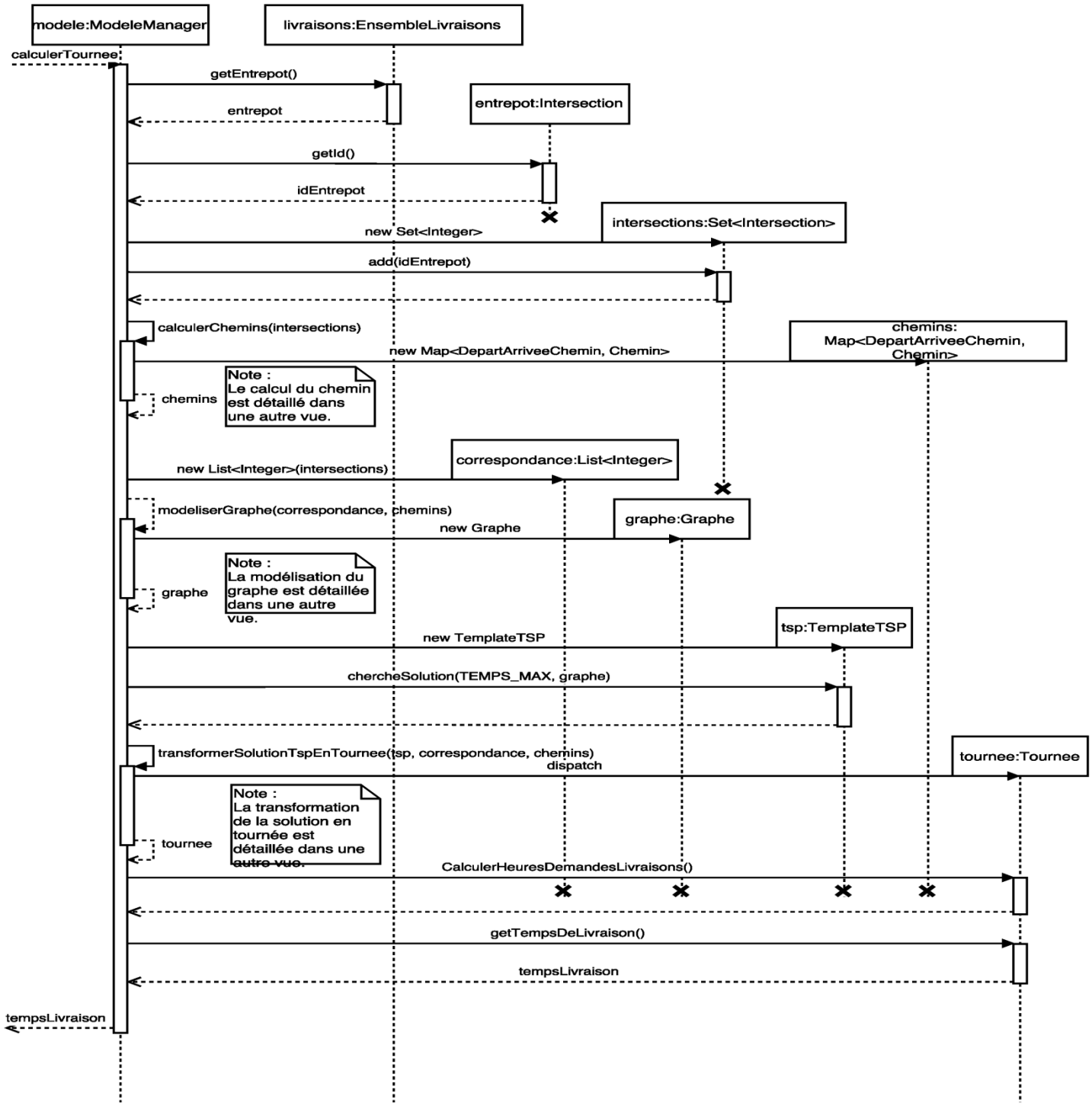


Diagramme de séquence de calcul de chemin :

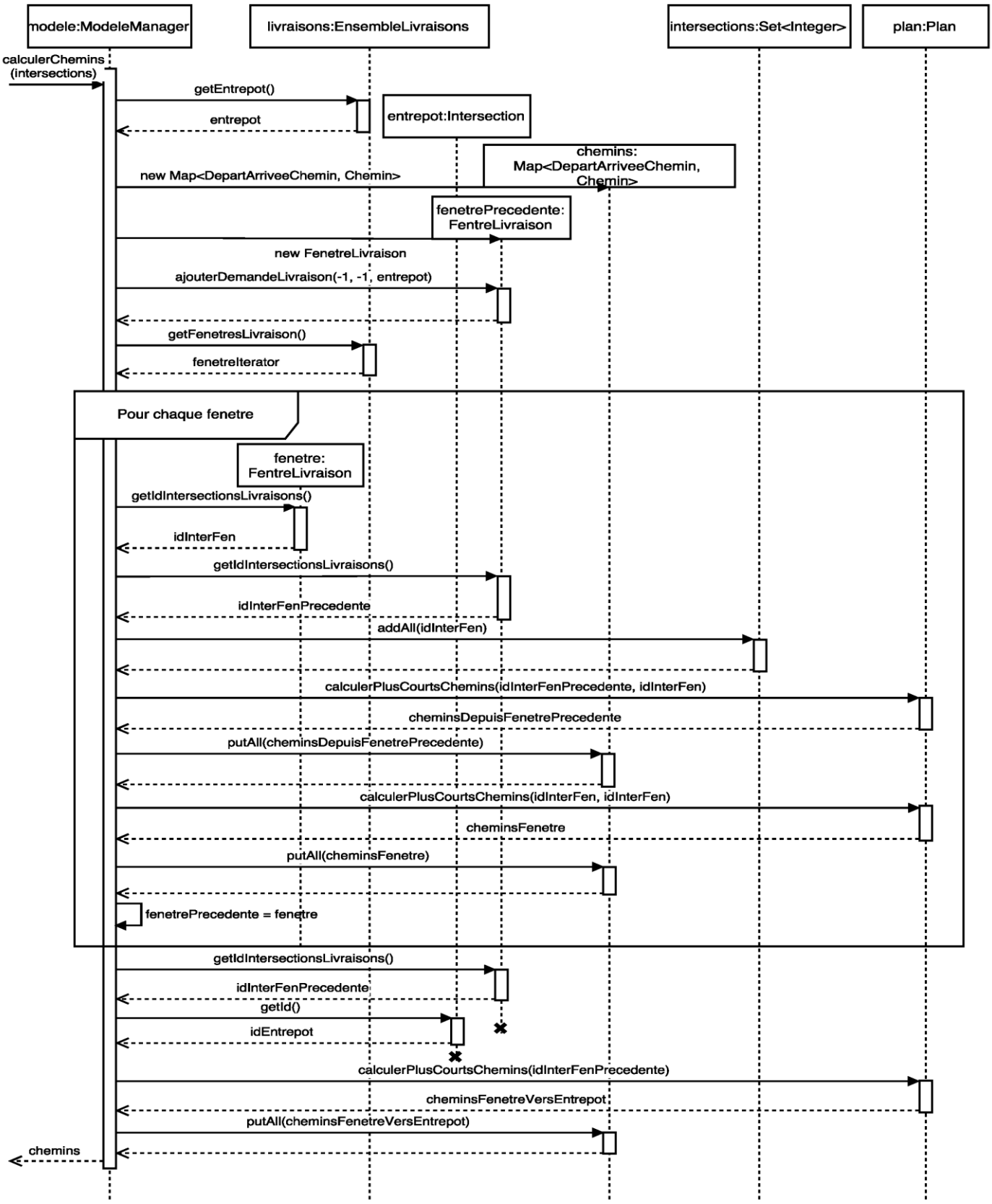


Diagramme de séquence de modélisation du Graphe TSP:

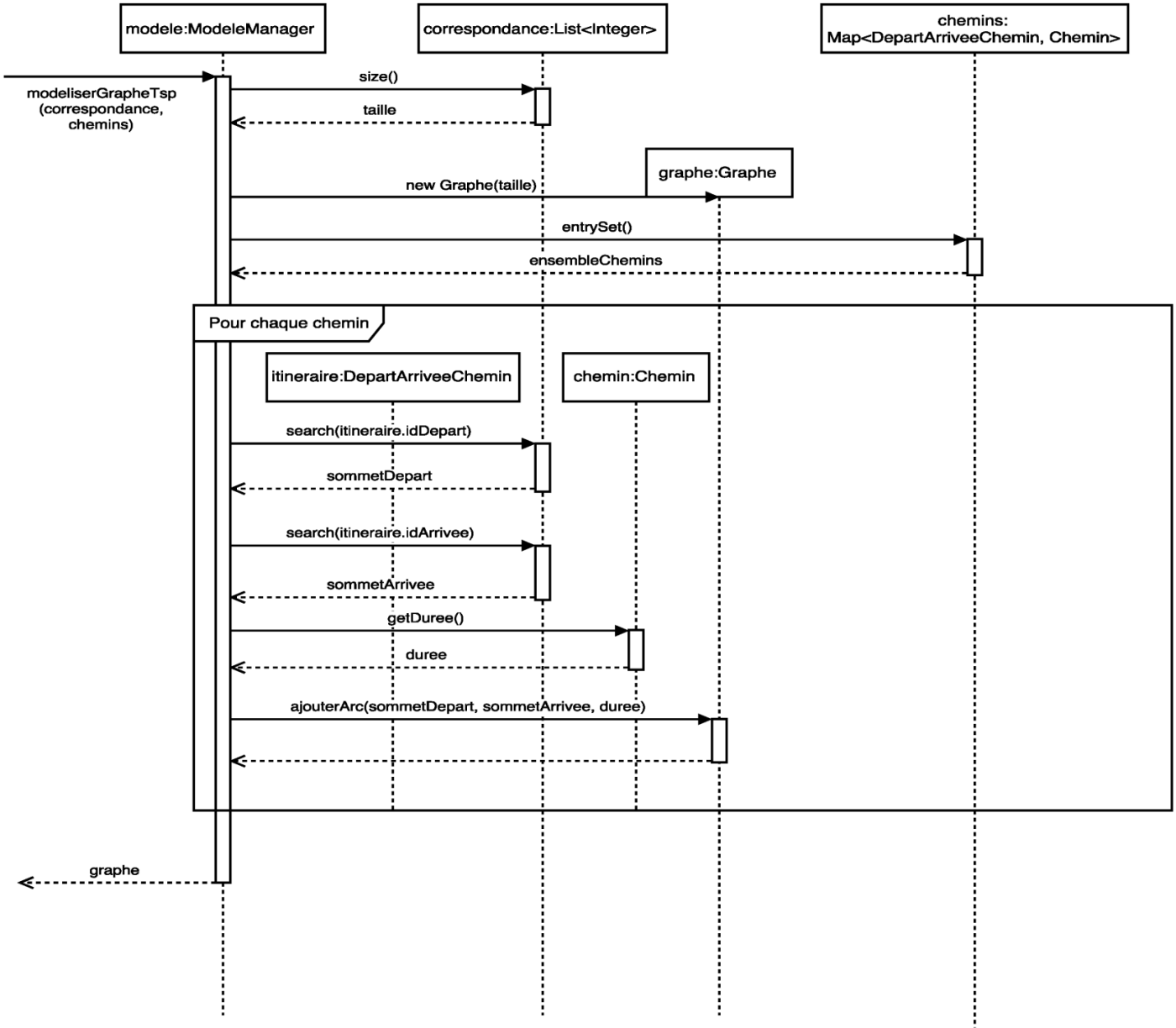
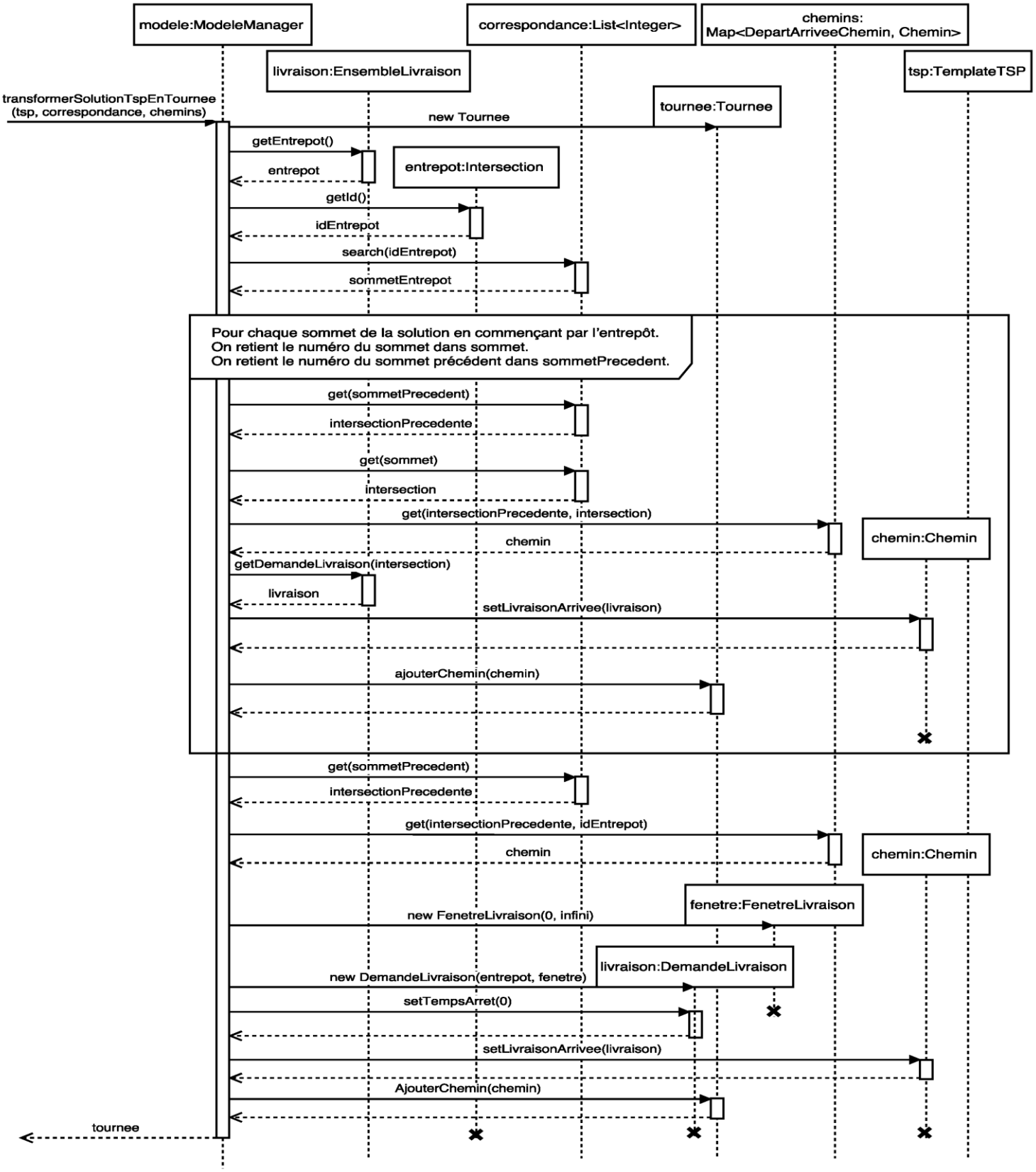


Diagramme de séquence de transformation de la solution TSP en tournée

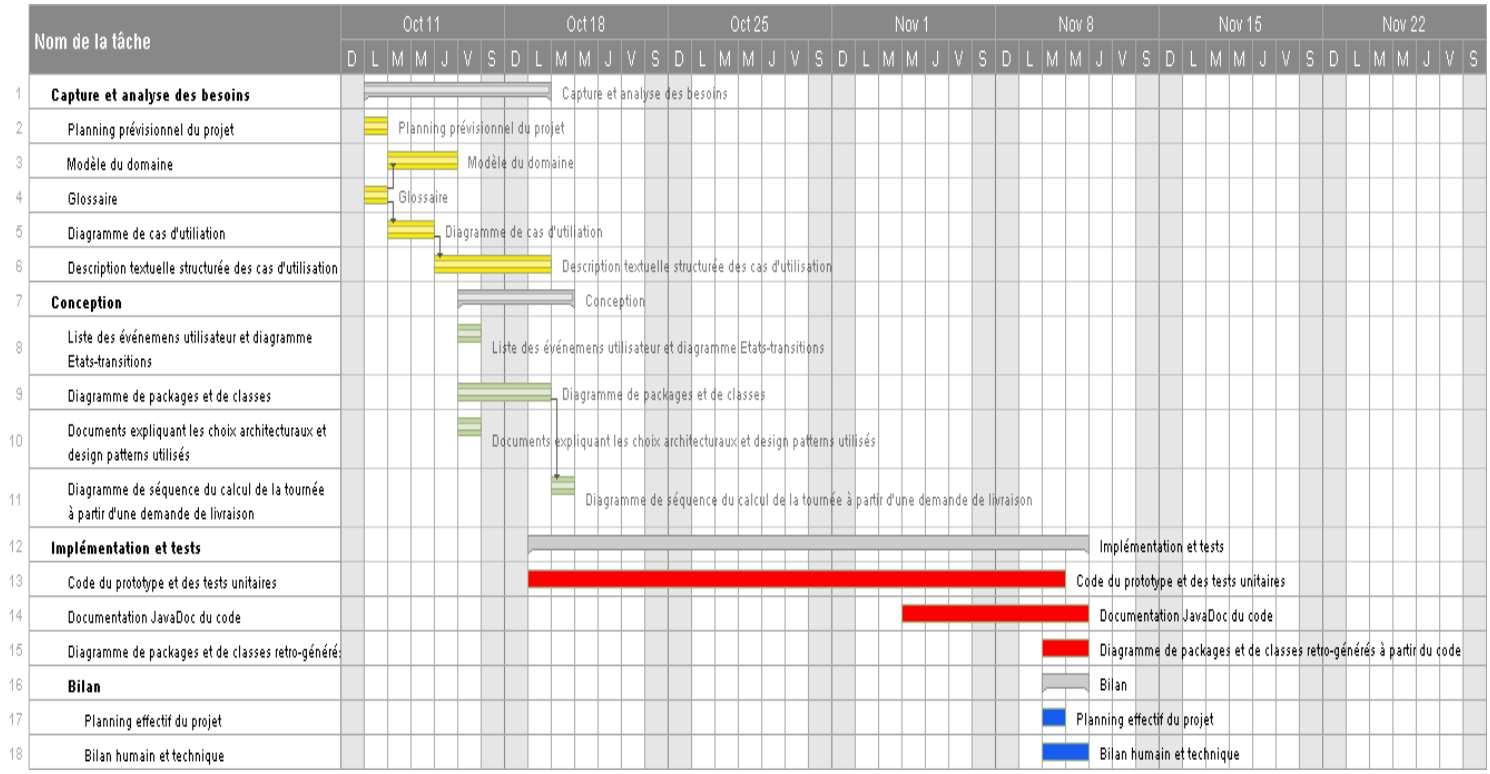


5. Bilan

5.1 Planning effectif du projet

Nous avons tenu à présenter le planning effectif du projet sous forme de diagramme de Gantt.

On peut voir que pour l'ensemble des tâches nous avons respecté les délais théoriques.



5.2 Décompte des heures par personnes

	Capture et analyse des besoins	Conception détaillée	Impléméntation du code du prototype	Tests	Rapport	Gestion de projet	Total
Nicolas NATIVEL	6	4	40	1	10	4	65
Romain MATHONAT	4	5	45	1	0	0	55
Mathieu GAILLARD	5	10	18	5	1	1	40
Mohammed EL ARASS	6	10	10	3	9	2	40
Guillaume KHENG	5	9	20	12	2	0,5	48,5
Thomas FAVROT	4	4	16	5	1	0	30
Killian OLLIVIER	4	7	20	0	1	0	32
Total	34	49	169	27	24	7,5	310,5

5.3 Bilan humain et technique

Nous avons trouvé intéressant, que dans ce bilan, l'on puisse retrouver le point de vue de chaque membre du groupe. En effet nous ne tenons pas, dans un premier temps, à ce que la voix de chacun soit possiblement atténuée dans un but de synthèse. Un bilan général et plus synthétique pourra être retrouvé dans le bilan du chef de projet.

Bilan de Guillaume:

C'est un projet où je me suis vraiment investi. Il y a eu beaucoup de travail à réaliser mais j'apprécie programmer donc ça ne m'a pas dérangé. Cependant la fatigue s'est bien fait sentir sur la fin du projet.

Concernant le développement en groupe durant ce projet, j'ai particulièrement apprécié l'aspect modulaire du développement (une partie du groupe s'est chargée du modèle, une autre de l'interface, ...). J'ai apprécié la cohésion du groupe et le soutien mutuel au cours des différentes phases de développement.

Concernant les différentes étapes du projet, J'ai trouvé les durées des phases d'analyse et de conception adaptées au projet. En revanche, la durée théorique de la phase de développement était un peu sous-estimée.

Je regrette aussi de ne pas avoir eu davantage l'occasion de mettre en pratique la méthodologie de développement agile. Si je devais donner une idée d'amélioration du projet à l'avenir, ce serait d'organiser une phase de rendu intermédiaire avec un prototype non final, mais contenant le cœur des fonctionnalités.

Bilan de Mathieu:

Pédagogiquement, le projet de développement orienté objet a été une réussite de mon point de vue. Nous avons pu concevoir et réaliser, en heptanôme, un logiciel complet en partant de zéro. Outre l'aspect technique et la conception orientée objet, nous avons pu apprendre énormément dans d'autres domaines. Par exemple la gestion de projet et le génie logiciel. Nous avons mis en place tout un cadre de travail propice à la réussite. Slack pour communiquer, Git pour gérer nos sources code ainsi que JUnit pour assurer la non régression. De mon point de vue c'est, pour le moment, le projet qui se rapproche le plus de notre futur métier.

Nous avons tout de même rencontré certains problèmes principalement inhérents à notre manque d'expérience. Ceux-ci se sont manifestés en particulier lors de la phase d'implémentation. En effet nous n'avions pas assez d'expérience dans le domaine de réalisation d'IHM en Java avant le début du projet. Nous avons en particulier perdu du temps sur ce point. Nous avons aussi remarqués des problèmes de conceptions lors de la phase d'implémentation. Nous avons rencontré le même genre de difficultés que les développeurs chevronnés lors de ce projet.

Finalement, nous avons vraiment gagné en compétences. Je me sens actuellement bien mieux armé pour réaliser un logiciel avec une architecture MVC.

Bilan de Mohammed:

D'un point de vue technique, voilà les différents points que j'ai pu noter concernant ce projet :

- Nous avons dû, plusieurs fois au cours du développement, revenir sur la conception détaillée que nous avions réalisée, notamment le diagramme de classes, le diagramme états-transitions.

- Nous avons voulu réserver un temps important à la phase conception, ou plutôt nous n'avons pas voulu passer hâtivement à la phase de développement, parce que nous estimions qu'une bonne conception faciliterait l'implémentation et l'intégration, et nous aiderait également à répartir les tâches de développement. Dans la phase de développement nous nous sommes rendu compte que ce choix était bon, et que les difficultés que nous avons rencontrées était souvent liée à des aspects de l'application que nous n'avions pas conçu, ou plutôt que nous ne pouvions pas voir durant la phase de conception.
- Le projet nous a d'abord parut très abordable niveau temps, mais au cours de la phase d'implémentation nous avons commencé à rencontrer de plus en plus de difficultés non prévues.

Et du point de vue humain/gestion de projet :

- Le développement en équipe nécessite un ordonnancement des tâches efficace pour permettre à chaque membre d'entamer sa tâche de développement du code au moment prévu. Le diagramme d'état nous a étonnamment bien servi pour réaliser cet ordonnancement.
- La diversité des profils du groupe nous a permis de bien cadrer le projet et de le planifier. L'expert Mathématique et Algorithmique, par exemple, a implémenté plusieurs versions de calcul de la tournée ce qui a ajouté de la valeur à notre application. Notre concepteur d'IHM a dans un premier temps rencontré quelques difficultés avec la librairie SWING mais il s'est rapidement débrouillé pour concevoir une interface fonctionnelle.
- Certaines taches de développement ont été mal estimées au départ. Au contraire, d'autres ont été surestimées. Cela est dû à notre manque d'expérience en la matière.
- La coordination et la communication demeurent des points clés pour la réussite d'un projet tel que celui-ci et permettent d'éviter les conflits au sein du groupe, d'où l'importance du rôle du chef de projet.
- Nous nous sommes confrontés à des soucis avec l'outil de gestion de version Git, mais notre "Expert Git" a rapidement réussi à résoudre ces problèmes.

Finalement, malgré toutes les difficultés rencontrées, nous avons pu rendre le projet en respectant les délais et le cahier des charges, et ce tout en proposant des améliorations non prévues au départ.

Bilan de Kilian:

Les deux points sur lesquels j'ai beaucoup appris lors de ce projet concernent les design patterns et le travail en heptanôme.

Le fait d'avoir non seulement compris, mais codé des patrons de conception m'a permis de me rendre compte des points forts et des limites de chacun. De plus il est évident que je serai maintenant plus efficace lors de leur implémentation ou lorsqu'il s'agira d'en proposer une variante plus adaptée. Le développement nous a forcé à rester rigoureux face à des problématiques très importantes comme la visibilité des objets entre et au sein des packages.

N'ayant jamais développé dans une équipe aussi nombreuse, j'ai appris beaucoup de cette expérience. Nous commençons tous le projet avec des compétences et des sensibilités différentes. Heureusement, chacun a su trouver des tâches qui lui plaisaient et pour lesquelles il était efficace. Cependant malgré l'ampleur du travail à réaliser, nous n'avons jamais pu travailler à sept en parallèle pour plusieurs raisons :

- Les tâches n'étaient pas toutes parallélisables et certaines de ces tâches demandaient un point de synchronisation.
- Travailler parallèlement sur une même fonctionnalité est très difficile, quelque soit l'outil de collaboration.
- Même si chacun a accompli un minimum de travail sur le projet, les degrés d'investissement sont très variés. Cela est surtout dû à la présence non négligeable de deux autres projets au même moment avec une deadline très rapprochée.

Au final, le chef de projet a su nous guider dans notre travail pour terminer le projet à temps. Cependant je pense que dans la réalité de l'industrie, un tel travail de groupe peut se révéler beaucoup plus chaotique.

Bilan de Romain:

Ce projet a été, je trouve, enrichissant et intéressant, malgré quelques difficultés. En effet, se poser les bonnes questions, faire une application au code jugé «propre» ont été assez difficile et éreintant du fait de notre manque d'expérience. Tout au long du développement nous avons souvent fait du refactoring de code, la faute à de mauvais choix de structuration dont nous n'avions pas conscience dans un premier temps. L'avantage est qu'on peut désormais espérer ne plus commettre ces erreurs à l'avenir.

Grâce à ce projet nous avons pu développer nos compétences dans de nombreux domaines, tels que le développement objet, la documentation de code, les tests, le développement sous architecture MVC (et autres patterns), l'algorithmie pour le TSP, le développement d'IHM, la conception, mais aussi l'utilisation de systèmes de versionning (git) et surtout le travail d'équipe !

Discuter, convaincre les autres en cas de bonne idée grâce à des exemples concrets, mais aussi savoir se rétracter lorsque les autres membres de l'équipe donnent (objectivement) de meilleurs arguments sont autant de qualités dont nous avons dû faire preuve.

En conclusion, si j'avais à choisir un mot pour qualifier ce projet, je dirais « complet ».

Bilan de Thomas:

Ce projet fut l'occasion de découvrir de nombreux aspect du génie logiciel. Travailler de la conception jusqu'à l'implémentation ne se fait pas dans tous les projets au département informatique, et c'est un avantage indéniable du projet DevOO.

Nous avons pu comprendre et utiliser les design patterns. Ce qui incontestablement plus formateur que la théorie seule. La maitrise de ces patterns ne peut s'acquérir que par l'implémentation de ceux-ci, et en ce point, le projet était très formateur.

Le travail en heptanôme n'a pas forcément été aisé, ce fut donc l'occasion d'apprendre l'utilisation de git, et d'utiliser une plateforme d'échange comme slack. La gestion de projet était assurée, et le projet s'est bien déroulé.

Dernier point positif, l'application du MVC : comme pour les patterns de design, c'est en codant que l'on apprend réellement. Je suis satisfait d'avoir acquis ce savoir-faire.

Je ferai tout de même remarquer que la quantité de travail dans ce projet est assez colossale, et que la présence d'autres projets en parallèle est éprouvante pour l'ensemble du groupe. Peut-être qu'un décalage dans l'emploi du temps suffirait à améliorer ce point ?

En conclusion, j'ai la sensation d'être plus compétent et plus préparé à ce à quoi nous allons être confrontés plus tard.

Bilan de Nicolas, chef de projet:

En tant que chef de projet, je suis très satisfait du travail réalisé par l'ensemble de l'équipe.

L'un des points difficiles de la gestion de ce projet a été la gestion parallèle des autres projets. En effet chaque membre du groupe avait des responsabilités dans d'autre projets, ce qui pouvait créer des conflits dans l'assignation des tâches. Il est difficile de gérer une équipe dont on ne contrôle pas totalement le travail assigné, et ce principe est je pense, totalement applicable au réel monde de l'entreprise.

Aussi, le rôle de chef de projet est un rôle qui peut rapidement devenir éreintant si ledit chef de projet participe activement au développement, comme un autre membre du groupe. Durant ce projet j'ai fait ce choix de participation active, car, à mon sens, la gestion de projet à elle seule est souvent frustrante (on peut avoir l'impression de ne pas être utile, de ne pas assez faire avancer le développement). La fatigue s'est indéniablement faite sentir sur la fin du projet, alors que nous n'avions pas l'impression d'avoir accumulé de retard important sur le planning théorique du sujet.

Nous avons tous beaucoup appris techniquement, c'est indéniable, mais aussi humainement. Le projet est plutôt complet (développement objet, algorithme, IHM, utilisation d'outil collaboratif, conception) et cela nous a permis de progresser dans plusieurs domaines à la fois.

Concernant le projet en lui-même, c'est l'un des projets les plus formateurs que nous ayons eu au département informatique jusqu'à présent, et aussi le plus proche de notre future activité. J'ai apprécié le fait que le projet soit guidé du point de vue de l'organisation dans la méthode de développement (phase d'analyse, puis de conception, puis de développement).

Cependant, nous avons voulu implémenter l'ensemble des fonctionnalités spécifiées dans le cahier des charges, et les rendre totalement fonctionnelles (tests, correction de bugs), c'est-à-dire que nous avons voulu répondre au sujet. Il s'est avéré que pour pouvoir atteindre cet objectif dans les délais demandés, les membres du groupe ont dû énormément s'investir dans le projet (21 heures de travail par personne en moyenne, heures passées en séance non comprises. ce qui à mon sens est beaucoup). Le projet pourrait être soit diminué (mais dans ce cas, il deviendrait aussi moins intéressant), soit prolongé dans le temps, avec des rendus intermédiaires qui permettraient de lisser la charge de travail.