Compte-rendu : Devéloppement Orienté Objet

Hexanome: 4105

Alexis Andra
Jolan Cornevin
Mohamed Haidara
Alexis Papin
Robin Royer
Maximilian Schiedermeier
David Wobrock

3 décembre 2015

Table des matières

Chapitre 1

Capture et analyse des besoins

1.1 Planning prévisionnel du projet

Le planning prévisionnel du projet s'est déroulé en trois étapes. Tout d'abord, nous avons identifié les stages nécessaires pour réaliser le projet. Nous avons transformé les missions en tâches réalisables qui sont exprimées par la liste suivante :

- Planning previsionnel du projet (3.0 h)
- Analyse du modèle (3.0h)
- Conception du modèle (2.5 h)
- Analyse des *CU*s, conception des *diagrammes de séquences* (3.5 h)
- Description textuelle des *CU*s (4.0 h)
- Implémentation des classes représentant les données dans les fichiers *XMLs* (1.5 h) [1]
- Conception d'IHM (prototype) (8.0 h)
- Conception d'IHM (précise) (8.0 h)
- Spécification d'*événements utilisateur* (4.0 h)
- Conception du diagramme *états-transitions* (6.0h)
- Implémentation du sérialiseur / désérialiseur (8.0 h)
- Spécification des interfaces du package contrôleur (5.0 h)
- Spécification des interfaces du package vue (3.0 h)
- Spécification des interfaces du package modèle (5.0 h)
- Conception de diagramme de classes du package contrôleur (6.0 h)
- Conception de diagramme de classes du package vue (3.0 h)
- Conception de diagramme de classes du package modèle (6.0 h)
- Implémentation des observateurs qui notifient la vue (4.0 h)
- Réalisation de l'IHM (package vue) (20 h)
- Mock implémentation des classes du package contrôleur (3.5 h)
- Conception et réalisation des tests unitaires (20 h)
- Conception et réalisation des tests fonctionnels (20 h)

1. La planification d'une heure et demi peut apparaître insuffisante pour implémenter un modèle. Au moment de la conception de l'architecture, le modèle était considéré comme un ensemble de *JavaBeans*. La réalisation des algorithmes était prévue dans le package contrôleur.

— Implémentation du package contrôleur (30 h)
— Implémentation du package vue (20 h)
— Implémentation du package modèle (10 h)
— Rétro-génération des diagrammes de classes à partir du code (3.0 h)
— Redaction du glossaire (2.0 h)

Deuxièmement, nous avons essayé de trouver toutes les dépendances entre les missions prévues. Le but de cette démarche était de pouvoir paralléliser au plus les développements. Le graphique suivant montre le déroulement prévisionnel du projet et également qui était le responsable prévu de chaque étape.

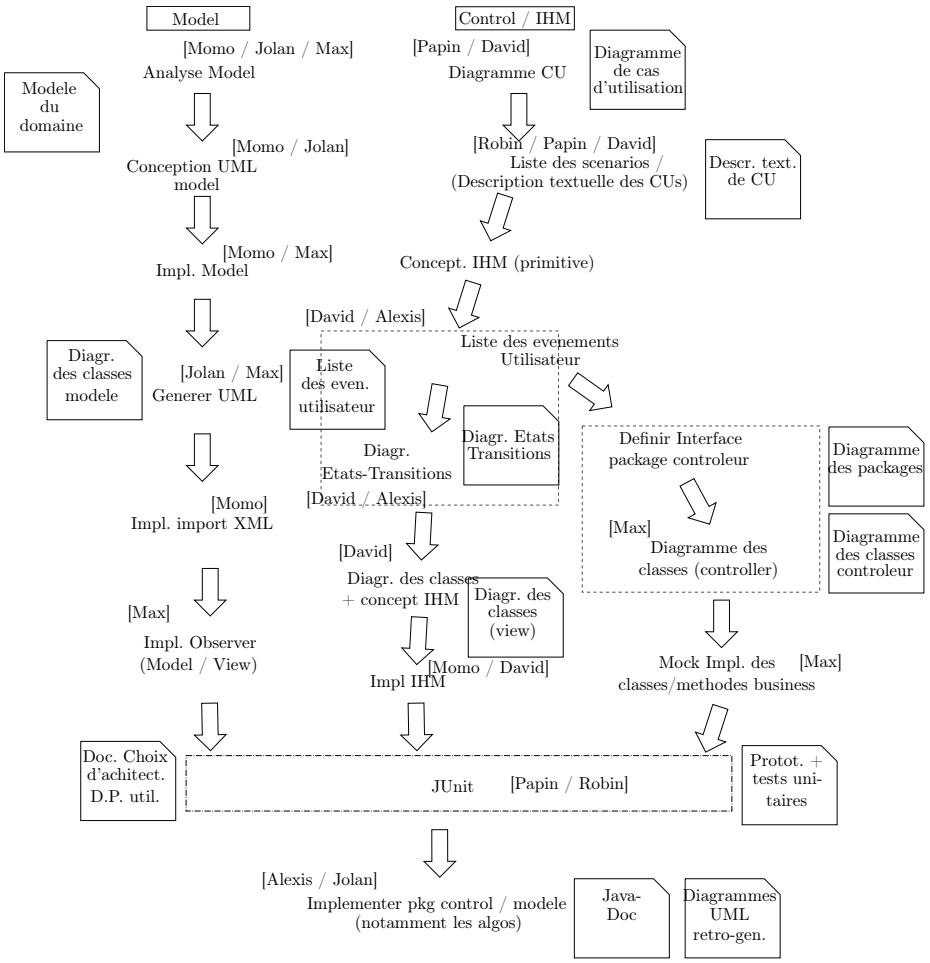
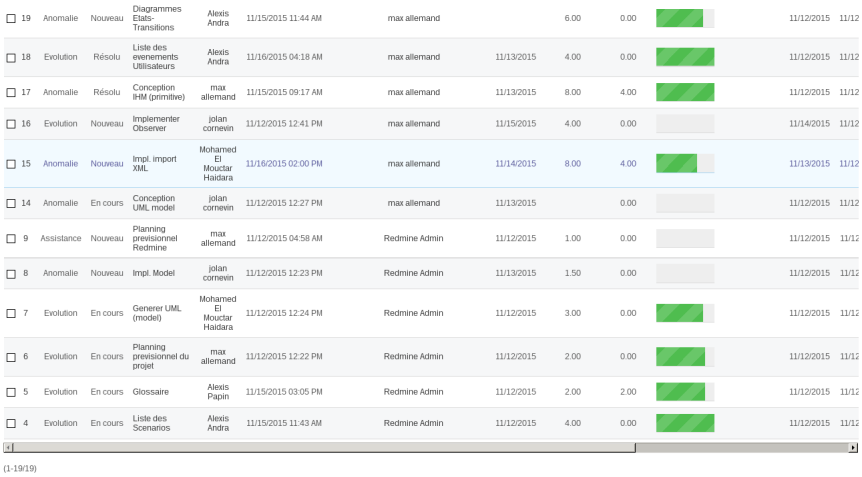


FIGURE 1.1 – Dépendances et déroulement prévisionnel du projet

Dans la troisième et dernière étape, nous avons estimé le temps nécessaire pour réaliser ces tâches en utilisant redmine.



19	Anomalie	Nouveau	Diagrammes Etats-Transitions	Alexis Andra	11/15/2015 11:44 AM	max allemand		6.00	0.00		11/12/2015	11/12
18	Evolution	Résolu	Liste des evenements Utilisateurs	Alexis Andra	11/16/2015 04:18 AM	max allemand	11/13/2015	4.00	0.00		11/12/2015	11/12
17	Anomalie	Résolu	Conception IHM (primitive)	max allemand	11/15/2015 09:17 AM	max allemand	11/13/2015	8.00	4.00		11/12/2015	11/12
16	Evolution	Nouveau	Implementer Observer	jolan cornevin	11/12/2015 12:41 PM	max allemand	11/15/2015	4.00	0.00		11/14/2015	11/12
15	Anomalie	Nouveau	Impl. import XML	Mohamed El Mouctar Haidara	11/16/2015 02:00 PM	max allemand	11/14/2015	8.00	4.00		11/13/2015	11/12
14	Anomalie	En cours	Conception UML model	jolan cornevin	11/12/2015 12:27 PM	max allemand	11/13/2015		0.00		11/12/2015	11/12
9	Assistance	Nouveau	Planning previsionnel Redmine	max allemand	11/12/2015 04:58 AM	Redmine Admin	11/12/2015	1.00	0.00		11/12/2015	11/12
8	Anomalie	Nouveau	Impl. Model	jolan cornevin	11/12/2015 12:23 PM	Redmine Admin	11/13/2015	1.50	0.00		11/12/2015	11/12
7	Evolution	En cours	Generer UML (model)	Mohamed El Mouctar Haidara	11/12/2015 12:24 PM	Redmine Admin	11/12/2015	3.00	0.00		11/12/2015	11/12
6	Evolution	En cours	Planning previsionnel du projet	max allemand	11/12/2015 12:22 PM	Redmine Admin	11/12/2015	2.00	0.00		11/12/2015	11/12
5	Evolution	En cours	Glossaire	Alexis Papin	11/15/2015 03:05 PM	Redmine Admin	11/12/2015	2.00	2.00		11/12/2015	11/12
4	Evolution	En cours	Liste des Scenarios	Alexis Andra	11/15/2015 11:43 AM	Redmine Admin	11/12/2015	4.00	0.00		11/12/2015	11/12

(1-19/19)

FIGURE 1.2 – Capture d'écran de la gestion du projet avec redmine

1.2 Modèle du domaine

1.3 Glossaire

1.4 Diagramme de cas d'utilisation

1.5 Description textuelle structurée des cas d'utilisation

Chapitre 2

Conception

2.1 Liste des événements utilisateur et diagramme états-transitions

2.1.1 Liste des événements utilisateur

2.1.2 Diagramme états-transitions

2.2 Diagrammes de packages et de classes

Précisions sur les diagrammes Il s'agit des diagrammes de conception. Ils ne correspondent pas forcement à l'implémentation finalement réalisé.

2.2.1 Packages

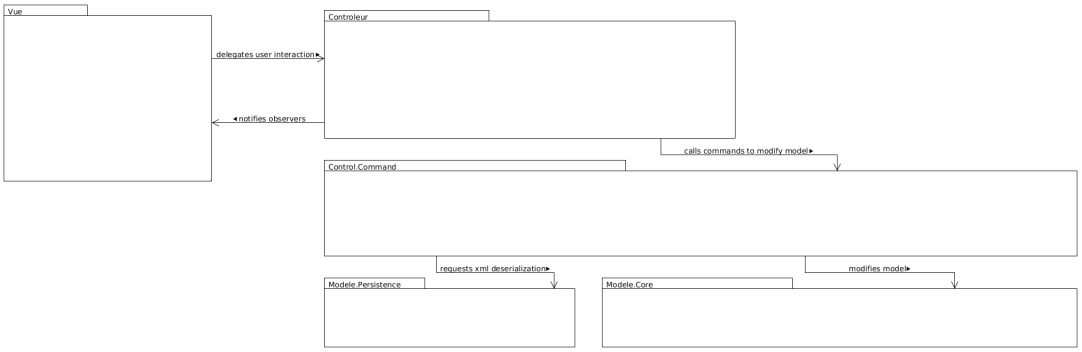


FIGURE 2.1 – Packages

2.2.2 Controleur

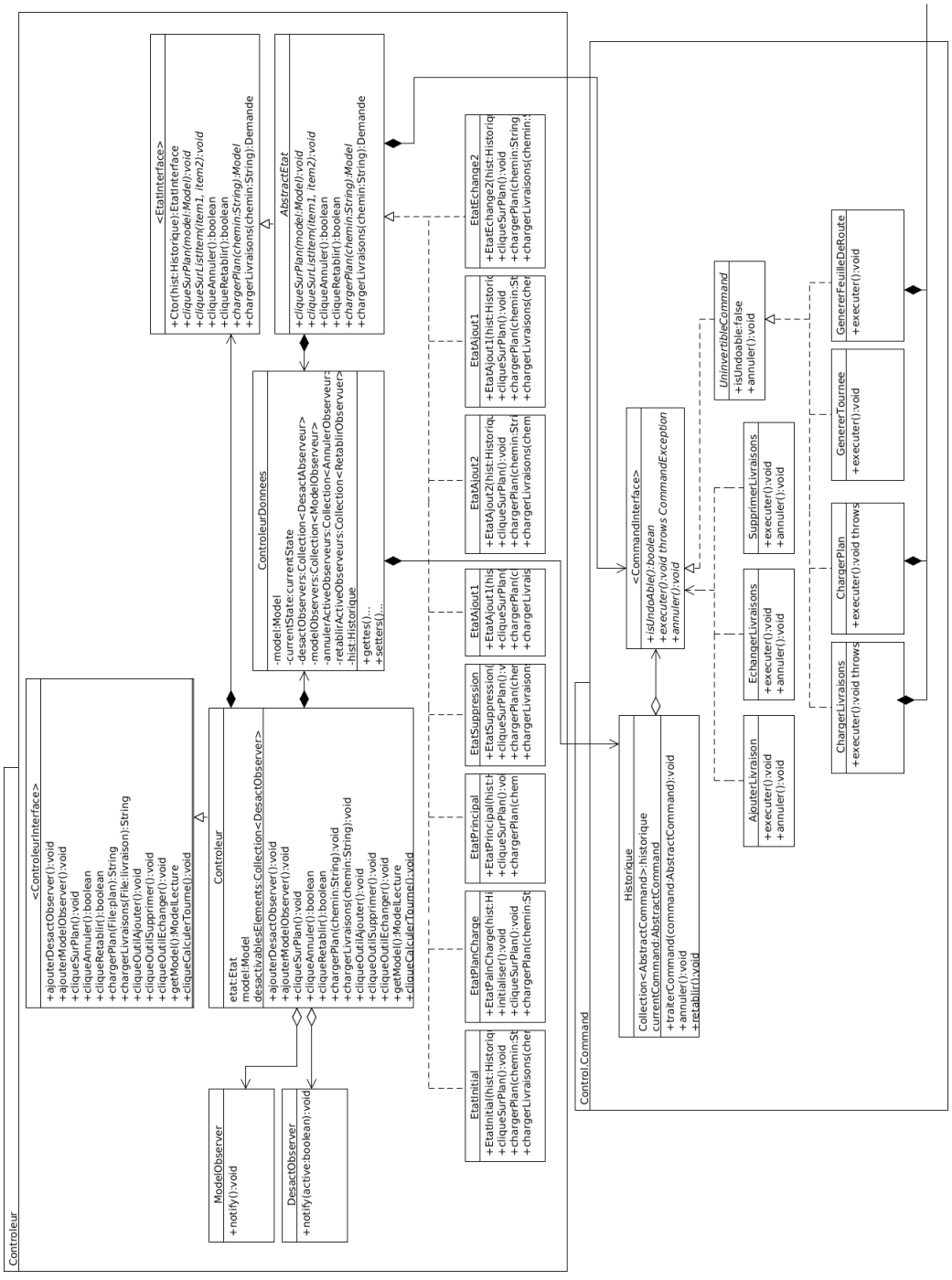


FIGURE 2.2 – Diagramme des classes du *controleur*

2.2.3 Modele

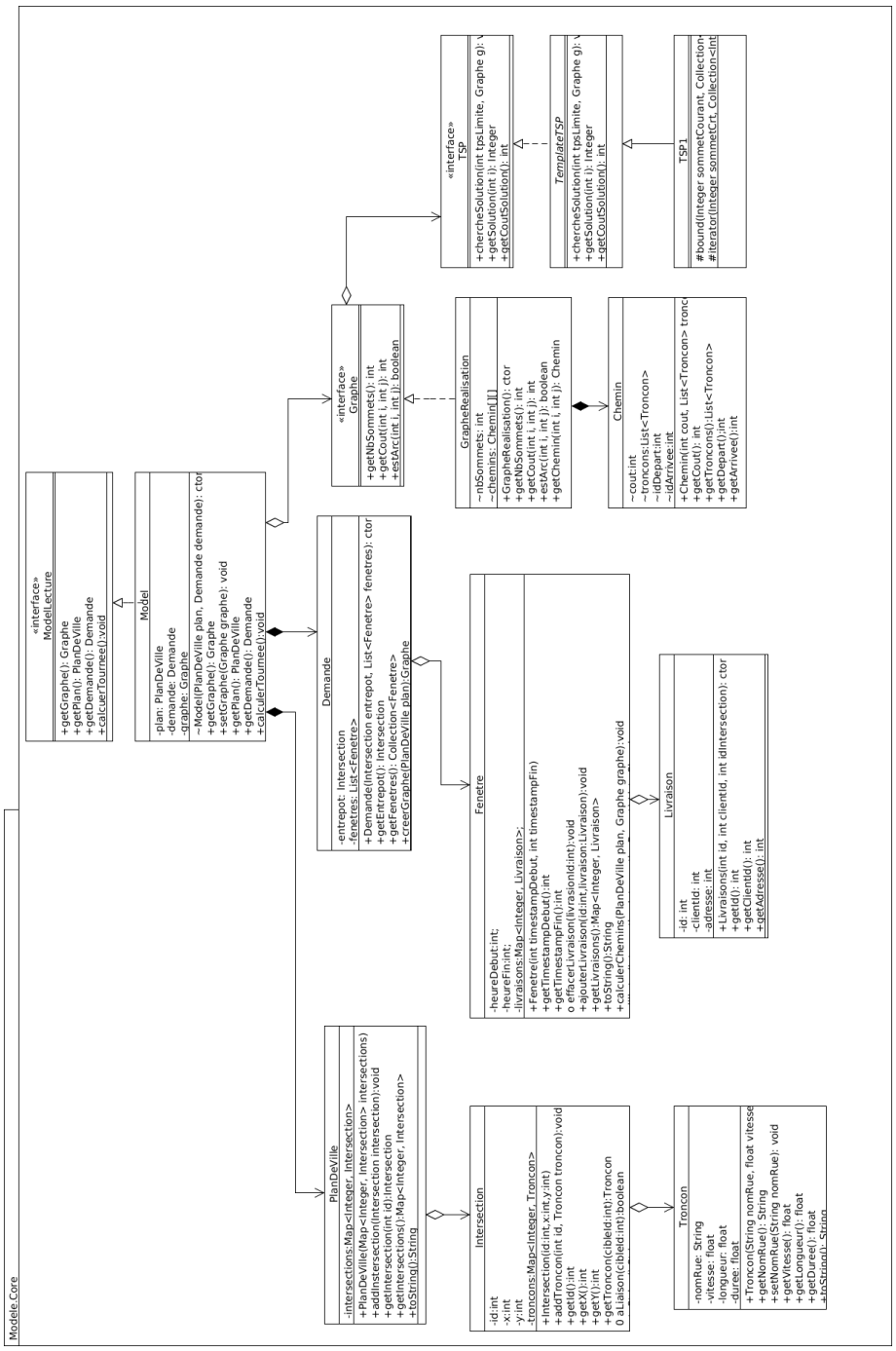


FIGURE 2.3 – Diagramme des classes du *modele*

2.2.4 Vue

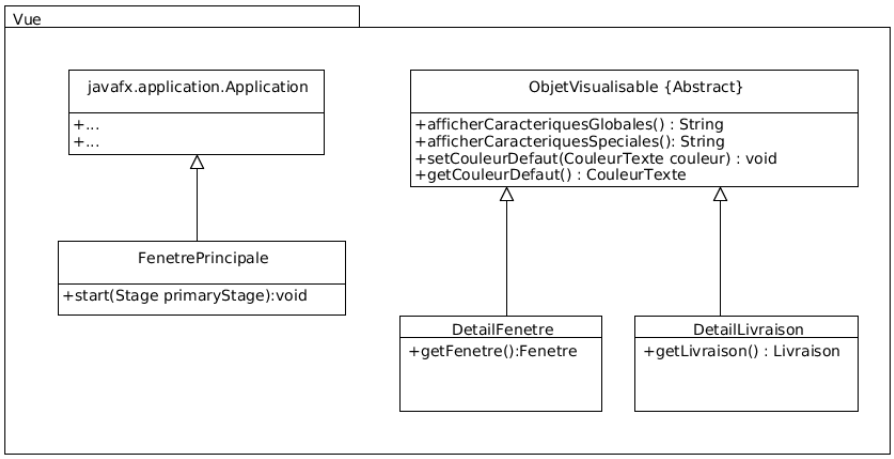


FIGURE 2.4 – Diagramme des classes du *vue*

2.3 Document expliquant les choix architecturaux et design patterns utilisés

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

Chapitre 3

Implémentation et tests

3.1 Code du prototype et des tests unitaires

3.2 Documentation JavaDoc du code

3.3 Diagramme de packages et de classes rétro-générés à partir du code

Chapitre 4

Bilan

4.1 Planning effectif du projet

Le planning prévisionnel nous a beaucoup aidé dans une réalisation bien pensée ce projet. La connaissance des dépendances entre les modules nous a permi d'éviter qu'un développeur est bloqué et de réaliser les tâches dans le bon ordre.

Developpeur	**Tache**		**Total**
Alexis A.	Conception CUs	4h	20h
	Diagramme état-transition	4h	
	Diagramme séquence calcul tournée	4h	
	Développement calcul tournée	8h	
Jolan	Implementation JUnit	4h	48h
	Implementation du Dijkstra	25h	
	Développement modèle	10	
	Conception diagramme modèle	2h	
	Conception diagramme séquence	3h	
	Correctinos des bugs	4h	
Mohamed	Conception diagramme modèle	2h	57h
	Développement Vue	29h	
	Développement Contrôleur	16h	
	Documentation	4h	
	Correction des bugs	5h	
	Redaction Design Patterns	1h	
Alexis P.	Conception CUs	4h	20h
	Diagramme état-transition	4h	
	Diagramme séquence calcul tournée	4h	
	Développement calcul tournée	8h	
Robin	Conception IHM	6h	14h
	JUnit modèle	8h	
Max	Planning previsionel	4h	71h
	Conception architecture et interfaces	9h	
	Development Controleur	18h	
	Development Modele	26h	

	Gestion des taches	3h	
	Gestion et réalisation des livrables	8h	
	Development JUnit Modele	3h	
David	Conception CUs	4h	52h
	Diagramme Sequence	3h	
	Développement vue	20h	
	Documentation	8h	
	JUnit Controleur	5h	
	Correction des bugs	10h	
	Redaction rapport	2h	

Comme le relevé plus haut le montre, le planning prévisionnel était suffisamment précis en ce qui concerne la planification et conception. Par contre, nous n'avons pas réussi à réaliser les implémentations dans le temps prévu. Il y a plusieurs facteurs qui ont contribué à ce retardement. Parmi eux voici les plus importants :

- On était forcé de profondément changer l'architecture dans un moment où la plupart des classes étaient déjà implémentées.
- Même si pas demandé, nous étions motivés de trouver la meilleure solution possible. Pour ça nous avons parfois implémentés plusieurs alternatives. L'implémentation de l'annulation des commandes est un bon exemple, il sera présenté plus tard dans ce rapport.
- Les interfaces et classes fournies, qui étaient censées nous simplifier le calcul du chemin optimal, nous ont posés quelques problèmes de compréhension. Il nous était pas évident comment il fallait les utiliser pour en profiter dans le cadre du projet.

4.2 Bilan humain et technique

4.2.1 Bilan humain

Coordonner le projet et notamment distribuer les tâches dans l'équipe était un devoir délicat. D'un côté, un but était la parallélisation des fils de développement. D'autre part, il y avait beaucoup de dépendances entre les étapes prévus. C'est pour ça qu'au début du projet, la priorité était d'identifier une procédure raisonnable qui permettait d'identifier les étapes critiques. Comme je n'ais jamais été en charge d'une équipe aussi nombreuse et puissante je dois admettre que j'avais d'abord sous-estimé ce défi.
Le résultat de cet effort nous a bien permis de profiter au maximum des ressources humaines. Néanmoins, il était important de tenir toute l'équipe au courant en ce que regarde les développements effectués en parallèle. En particulier, la convention de bien commenter son code, en combinaison avec une communication fréquente, nous a permis de maîtriser cette mission.
Personnellement, je trouvais éprouvant de trouver une bonne granularité pour la distribution des tâches dans l'équipe. Si une charge était trop petite, on risque de perdre du temps en expliquant le contexte. Aurait-elle pu être réalisée plus rapidement directement par la personne qui s'en est déjà occupée ? Sinon, quand les tâches sont trop complexes, on risque qu'un développeur sera bloqué et exclu

des événements qui se passent à côté. Comme la section précédente l'a relevé, le nombre d'heures passées sur le projet par développeur n'est pas parfaitement équilibré. Cependant on ne doit pas oublier que les chiffres ne correspondent pas toujours précisément au travail réalisé. Notamment le temps passé sur la planification, communication et réflexion n'est pas facile à mesurer. C'est pour cela qu'il me semble plus avéré d'évaluer l'équipe comme une entité indivisible. Parlant de l'équipe, on peut résumer qu'on était enchanté de pouvoir observer la réalisation de nos conceptions. Même s'il y avait parfois des défis qu'on n'avait pas prévu, l'équipe était toujours motivé pour discuter et trouver la meilleure solution possible.

4.2.2 Bilan technique

Pour réaliser ce projet, nous avons profité d'une grande variété des techniques. Dans le but de permettre à chaque développeur de travailler avec son IDE préféré, nous avons décidé de ne pas garder les fichiers de gestion d'IDE dans le répertoire git. Grâce à cela, nous avons réussi à travailler avec les IDEs *NetBeans*, *Eclipse* et *IntelliJ* en même temps. Concernant les fichiers de conception, nous avons également utilisé plusieurs outils :

- La conception des diagrammes de classes était réalisée avec le logiciel *UMLet*[1]. Les fichiers utilisés par ce logiciel étaient également partagés dans le répertoire git.
- Pour tous les autres diagrammes nous avons profité des outils en ligne, particulièrement *Draw.io*, dont une intégration est disponible avec *Google Drive*.
- Pour automatiquement charger les bibliothèques dont nous avons besoin, nous avons intégré *Apache Maven*. On pourrait argumenter que l'intégration de Maven n'était pas forcement nécessaire, car il y avait seulement quatre dépendances. Néanmoins, son intégration était très confortable pour l'équipe parce qu'on n'a jamais perdu du temps en résolvant les dépendances soulevées par un collègue.
- Les tests fonctionnels et unitaires étaient réalises en utilisant le framework *JUnit*. Bien que les tests ont grandement contribué à trouver et résoudre des erreurs, nous aurions encore pu intensifier leur intégration. Nous avons suivi le conseil de ne jamais laisser un développeur tester son propre code. C'est en général une bonne pratique, mais nous avons parfois pas établi assez de communication entre un auteur d'une classe et le testeur. Ça peut causer que les tests ne passent pas, car ils n'utilisent pas les interfaces de la manière prévue.

Précisions sur l'implémentation Il existe quelques problèmes dont nous sommes conscients et quelques détails que nous souhaitons spécifier ici.

Tout d'abord, nous étions très heureux d'avoir pu utiliser JavaFX, le dernier framework de développement de client lourd de Java, bien que les ordinateurs du département n'avaient pas Java 8 d'installé. Nous sommes réjouis de voir que le département fait des efforts pour nous permettre d'utiliser les technologies du moment (JavaFX, Androïd dans le cours d'IHM, ...).

1. http ://www.umlet.com/

Il est possible que lors du redimensionnement de la fenêtre alors que le plan est déjà chargé, il peut problème d'affichage. Ce problème survient essentiellement quand on met la fenêtre en pleine écran directement. De plus, après des agrandissements et des zooms, nous avons remarqué qu'il peut y avoir un décalage entre la position de la souris et la position des intersections sur la partie graphique. Les intersections passent en surbrillance seulement si le curseur est un peu en dessous à droite de la position du rond blanc sur le plan.

Pour le patron de conception Etat (*State*), nous avons préféré recréer un objet de l'état correspond à chaque modification, au lieu d'avoir une instance constante (*final*) et statique de chaque état que nous souhaitons réutiliser. Ce choix a été fait car nous voulions avoir un comportement unique à chaque passage dans un état. Ce comportement se situe dans le constructeur de chaque état.

Notre patron de conception Commande n'est pas totalement respecté. Dû à des contraintes de temps trop importante et la focalisation sur d'autres fonctionnalités, l'annulation et le rétablissement d'une commande se fait par la restitution du modèle à ce moment. Notre patron Commande est en réalité un mélange entre Commande et Mememto, stockant l'état d'un objet à un certain moment. Pour la copie profonde du modèle, nous n'utilisons pas le patron de conception Prototype, déconseillé par la communauté [2]. Nous sommes conscients que cette manière de faire est beaucoup plus gourmande en mémoire et également en temps processeur. Une amélioration sera vraisemblablement proposée lors du prochain sprint. Afin que la mémoire ne croît pas à l'infini pendant la durée de vie de l'application, nous stockons seulement les dix dernières commandes (copie du modèle du coup) exécutées et les dix dernières annulées.

Lors de la création d'une livraison, l'identifiant du client est initialisé à -1.

Nous souhaitons, dans un futur proche, améliorer le rapport généré. Celui-ci ne signale pas les retards, ni les attentes endurés par le chauffeur. Il est pour le moment minimaliste. Nous aurions aimé avoir un rapport en HTML avec un peu de style CSS pour avoir un résultat plus attractif.

Lors de l'échange de deux livraisons successives, une erreur se produit. Nous n'avons pas eu le temps de corriger ce disfonctionnement de l'application.

Finalement, on peut résumer qu'il s'agissait d'un projet complexe et que nous n'avions pas beaucoup de temps pour le réaliser. Nous pensons avoir choisi les bons outils. Ceux-ci nous ont permis d'avancer rapidement et créer un produit fiable.

2. http ://stackoverflow.com/questions/64036/how-do-you-make-a-deep-copy-of-an-object-in-java