

**PlanIFTicateur**

**Travail pratique 4**

GLO-2004 – Génie Logiciel Orienté Objet – Hiver 2015

Travail présenté à

M. Jonathan Gaudreault

Chayer, Philippe [Philippe.chayer.1@ulaval.ca](mailto:Philippe.chayer.1@ulaval.ca) IFT PHCHA47

Khouma, Abdou [abdou.khouma.1@ulaval.ca](mailto:abdou.khouma.1@ulaval.ca) GIF ABKHO9

Gadoury, Gabriel [Gabriel.gadoury.1@ulaval.ca](mailto:Gabriel.gadoury.1@ulaval.ca) IFT GAGAD1

Yéo, Clotioloman [Clotioloman.yeo.1@ulaval.ca](mailto:Clotioloman.yeo.1@ulaval.ca) GLO CLYEO1

Table des matières

[Introduction 3](#__RefHeading__278_560640856)

[Principales interfaces utilisateurs 4](#__RefHeading__280_560640856)

[Fenêtre principale 4](#__RefHeading__282_560640856)

[La fenêtre principale est essentiellement composée des parties suivantes : 5](#__RefHeading__284_560640856)

[Une barre de menu au haut de l’écran, permettant l’accès à des fonctionnalités triviales 5](#__RefHeading__286_560640856)

[Fenêtre statistiques 5](#__RefHeading__288_560640856)

[Modèle du domaine 6](#__RefHeading__290_560640856)

[Modèle des cas d’utilisation 8](#__RefHeading__292_560640856)

[Architecture logique 9](#__RefHeading__294_560640856)

[Diagrammes de classes de conception 10](#__RefHeading__296_560640856)

[Diagrammes de séquences 12](#__RefHeading__298_560640856)

[Explications du design 13](#__RefHeading__300_560640856)

[Gestion de projet 14](#__RefHeading__302_560640856)

[Conclusion 15](#__RefHeading__304_560640856)

# Introduction

Ce rapport consiste en une analyse de la faisabilité pour la conception d’un logiciel permettant la gestion et la création d’horaire. Cette analyse sera basée sur les informations fournies par le client. Le but du logiciel est de faciliter la gestion des horaires de session effectuée par la direction de programme.

- ***PlanIFTicateur*** est un programme apte à construire un horaire de session de façon interactive et peut, si souhaité, faire la génération de votre horaire de façon optimale (rapide, efficace et flexible). Le but primaire est de rendre agréable la construction de l’horaire tout en réduisant le temps consacré à ce dernier.

- Une planification automatique optimale grâce à un algorithme de recherche efficace se basant sur les restrictions et les statistiques de ce qu’est un bon horaire.

- Une interface simple et intuitive visant à aider l’utilisateur dans son travail.

- Un suivi simple et efficace grâce à une fonctionnalité permettant de prendre des notes en lien avec la grille horaire que vous construisez.

- Les horaires créés seront en tous points valides grâce à des fonctionnalités de validations optimales. Ces fonctions rendront facile et stimulante la création d’un horaire grâce à son aspect visuel et dynamique.

- Une importation des données et restrictions des cours instantanée facilitant le travail et limitant les erreurs ou les oublis.

- Des statistiques pertinentes disponibles pour la grille horaire en construction visant à l’optimisation de votre horaire.

- Une exportation de fichier simple et facile à des fins d’utilisation hors programme.

- Un historique est disponible pour un suivi efficace de toutes les modifications réalisées sur votre planification.

# Principales interfaces utilisateurs

Tout d’abord, une interface utilisateur doit miser sur l’aspect ergonomique. Tout en respectant les critères du client, nous avons réussi à rendre simple et intuitif l’interface graphique de notre programme. Les informations généralement recherchées par les utilisateurs sont concentrées au haut de la fenêtre, tandis que les autres sont positionnées de manière à respecter les contraintes du client.

Une fenêtre pour la grille horaire encadrée par une couleur vivante et représentative de l’état du travail indiquera à l’utilisateur s’il y a un problème ou si tout va bien, un champ au bas communiquera avec précision les différentes erreurs au besoin ou encore des messages de confirmation et une barre de menu semblable à toutes les barres représentées dans les différents logiciels connus de tous. Voyez maintenant les différents croquis :

## Fenêtre principale

## La fenêtre principale est essentiellement composée des parties suivantes :

## Une barre de menu au haut de l’écran, permettant l’accès à des fonctionnalités triviales

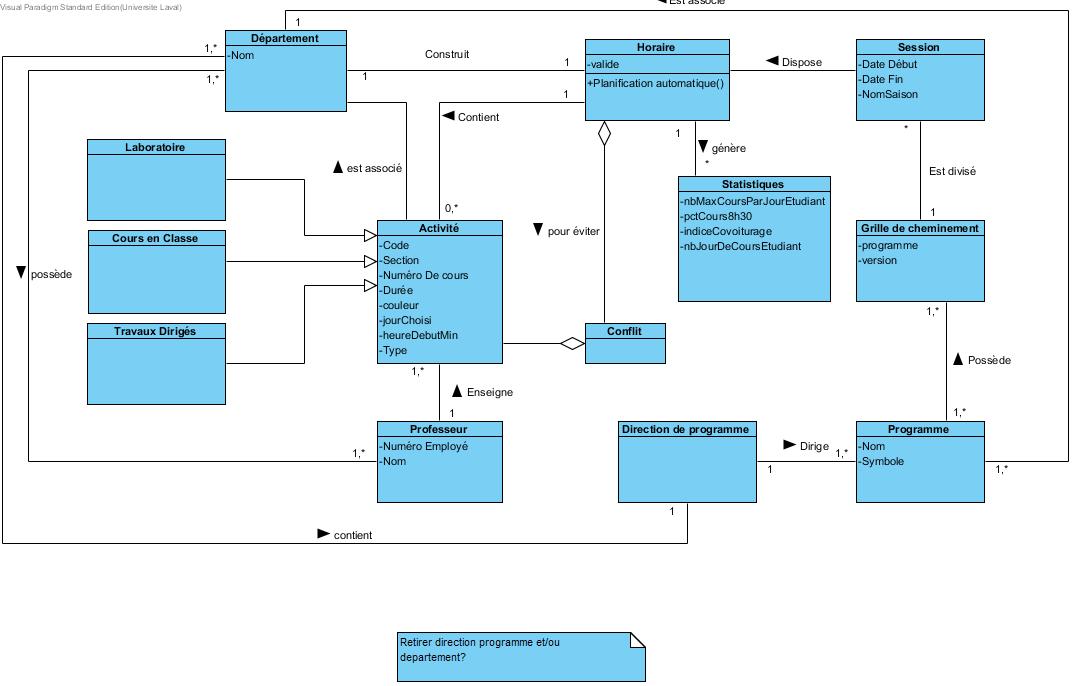
* Une grille horaire qui suivra la même logique, c'est-à-dire simple et intuitive.
* Une liste des activités non attribuée (située à droite)
* Un champ pour l’indication d’erreurs
* Un bouton planification automatique permettant de faire l’essai de la fonctionnalité de planification automatique de l’application
* Un bouton note permettant à l’utilisateur de prendre des notes
* Un bouton statistique dont le contenu est détaillé ici-bas.

## Fenêtre statistiques

La liste des statistiques est automatiquement affichée pour tous les jours de la semaine.

# Modèle du domaine

Cette section consiste à vous présenter une perspective générale du fonctionnement de *PlanIFTicateur*. Comme tout bon modèle d’affaires, le schéma ci-dessous contient l’ensemble des informations nécessaires à la compréhension de notre projet, voire notre logiciel. Suite au visionnement du dît schéma, nous pourrons constater l’ampleur du travail accomplit.



***Détails du diagramme* :**

Pour le diagramme de modèle de domaine, plusieurs choix ont été faits afin d’obtenir le diagramme ci-dessus. Tout d'abord, le département est présent puisque c'est ce dernier qui utilisera l'application (ou quelqu'un de mandaté par la direction du département). Ensuite, la direction de programme est directement liée aux programmes puisque c'est la direction qui dirige ces programmes et qui construit les grilles de cheminement.

Ces grilles sont alors divisées en sessions qui ont tous un horaire à respecter selon certains critères, c'est-à-dire d'éviter les conflits d'horaires entre les cours d’une même session (facteurs fournis par les grilles de cheminement). Dans notre diagramme, l’horaire peut être vu de deux façons :

-Soit comme étant un horaire d'étudiant qui contient normalement cinq cours avec, possiblement des laboratoires;

-Soit comme étant un horaire de tous les cours qui sont donnés dans un département. D’ailleurs, c’est de cette façon que la direction planifie les horaires des sessions qui permettra ensuite d’obtenir celle des étudiants.

Par la suite, il y a les activités, qui sont réunies dans une superclasse pour ce diagramme et qui se divise en trois sous-classes : les laboratoires, les cours en classe et les travaux dirigés. Ces sous-classes correspondent aux différents types d'activités qui seront mises à l'horaire. Finalement, il y a les étudiants qui font partie du modèle puisque l'horaire construit concerne directement les étudiants. D'ailleurs, les statistiques générées par l'application sont basés sur ces étudiants.

# Modèle des cas d’utilisation

Maintenant que nous avons une bonne vue d’ensemble, il sera plus simple de comprendre la section ci-présente. Le modèle qui suivra représente le système, *PlanIFTicateur*, abritant tous ses cas d’utilisation. Il y aura également une représentation sommaire des acteurs du système. Suite à ce modèle, suivra l’architecture logique du logiciel.

*(Diagramme sur la page suivante)*

# 

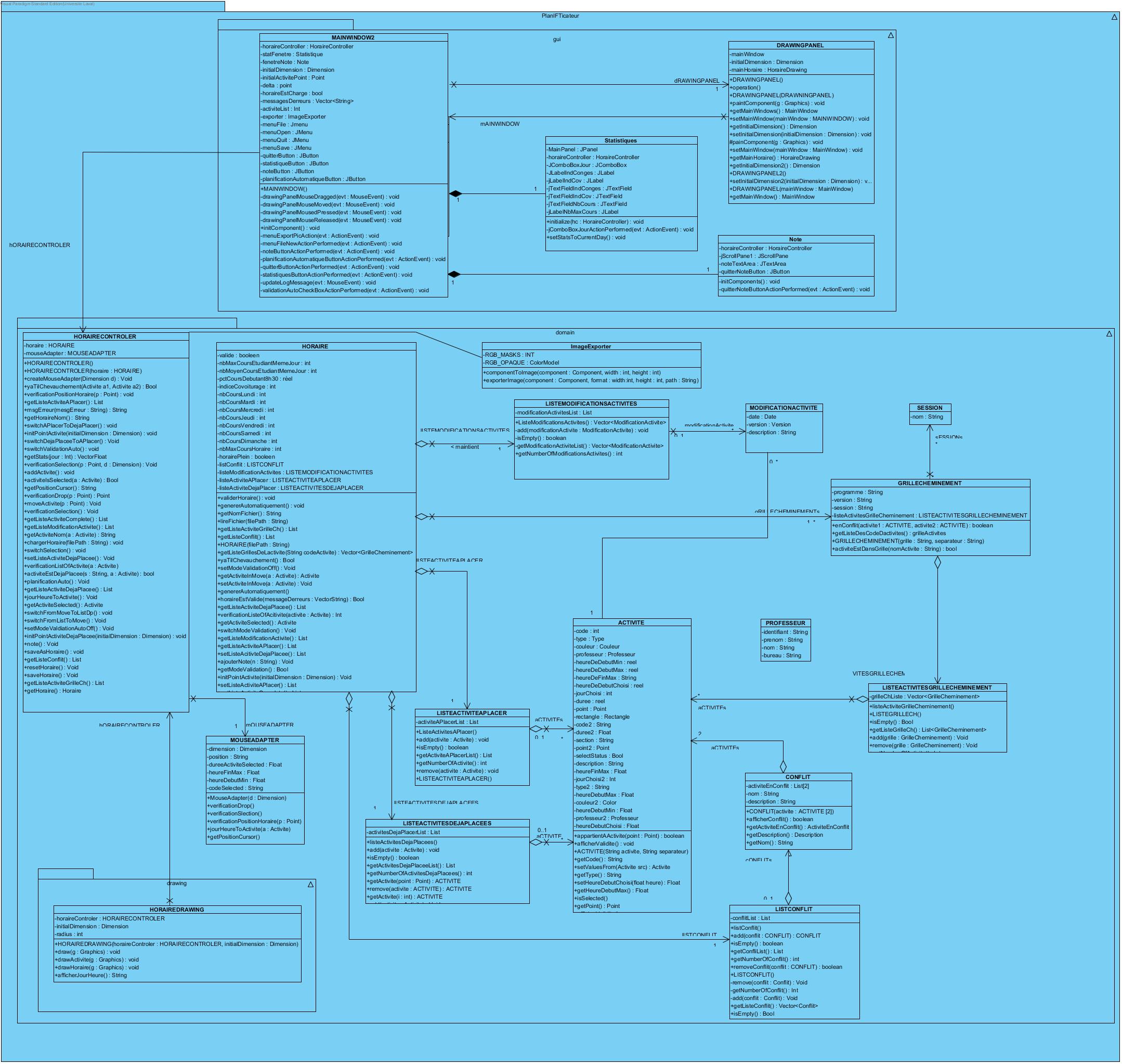
# Architecture logique

# Diagrammes de classes de conception

Le diagramme qui suit représente le corps des classes utilisées par notre application. Par rapport précédent, nous avons ajouté plusieurs méthodes et attributs dans les différentes classes, voir toutes. Certains attributs ont ajoutés suite à plusieurs constations lors de l’implémentation du code.

Nous avons aussi ajouté une classe **ImageExporter** qui est utilisé pourexporter les images de l’horaire dans un format prédéfini. Le contrôleur, nommé ici **HoraireController**, est la seule classe qui échange avec l’interface.

Tout élément de l’interface qui souhaite accéder à une fonction ou un événement doit passer par ce contrôleur qui s’assurera, par la suite, de faire la relation vers le code approprié. Le même phénomène se produit à l’inverse, c’est-à-dire lorsque les éléments du domaine, qui contient le code fonctionnel, désirent interagir avec l’interface utilisateur.



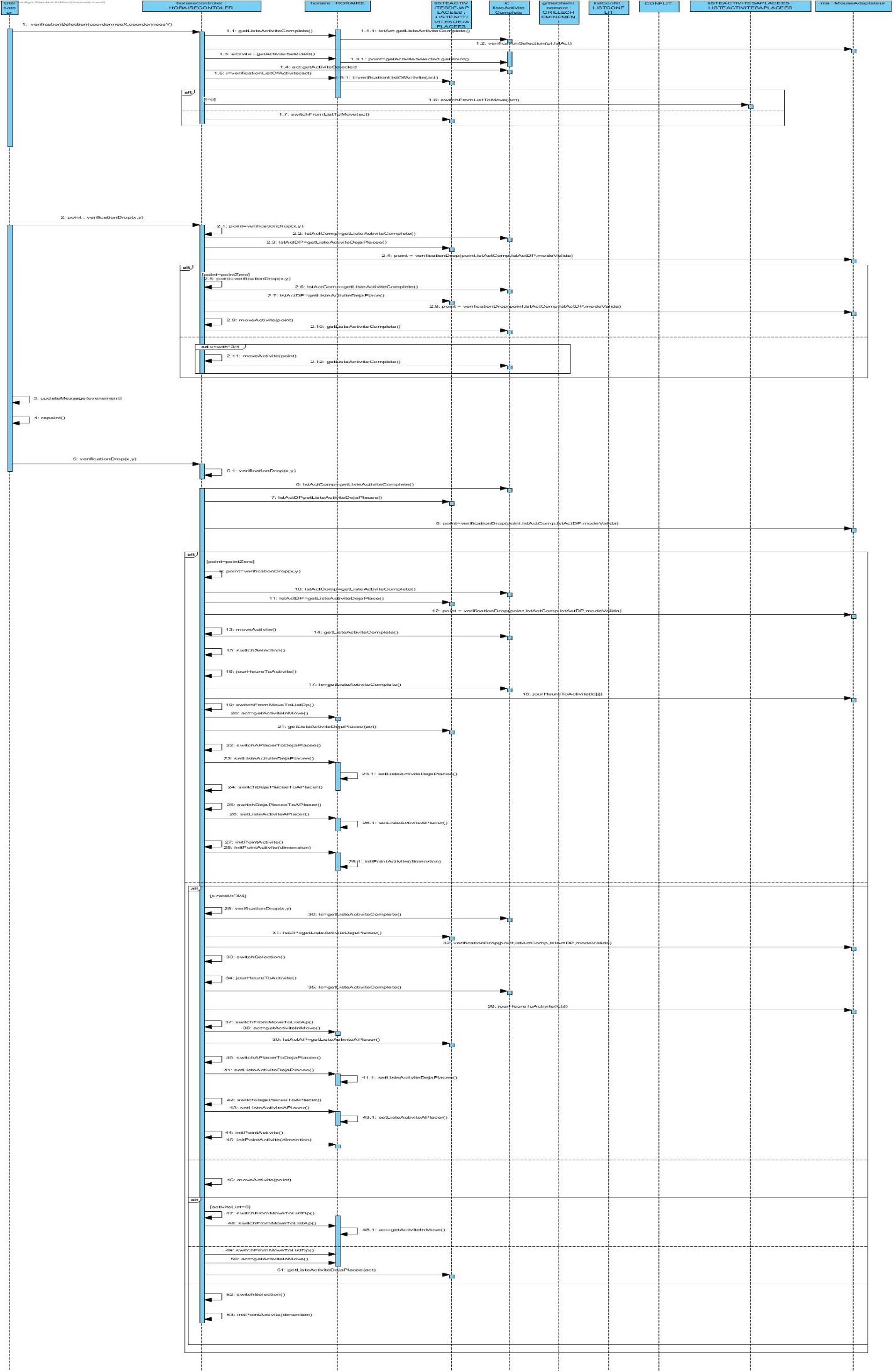
# Diagrammes de séquences

## Déplacement d'un cours

Tout d’abord, lors du déplacement d’un, une méthode du contrôleur **verificationSelection(coodonneeX,coordonneesY)** est lancé. Cette méthode appel **getListeActiviteComplete()** de l’horaire. Par la suite, cette méthode de l’horaire appel une méthode de la classe **listeActiviteComplete** qui retourne la liste des activités complète. Puis, la méthode du contrôleur appel le **mouseAdapter** et la méthode **verificationSelection(pt,lstAct)**. Puis, la même méthode du controleur obtient l’activité qui était sélectionner à l’aide de la méthode **getActiviteSelected()** qui retourne un point avec lequel l’activité est associé. Après avoir obtenu l’activité sélectionnée, il (le contrôleur) fait une vérification sur la liste des activités en appelant la méthode **verificationListOfActivite(act)**. Celle-ci permet de valider le déplacement et ainsi, déplacer l’activité dans la liste des activités placées ou bien la retourner à l’endroit de départ avant le déplacement. Cette première parti, s’assurer que la sélection est valide (par exemple, que nous avions bien sélectionné une activité en cliquant)

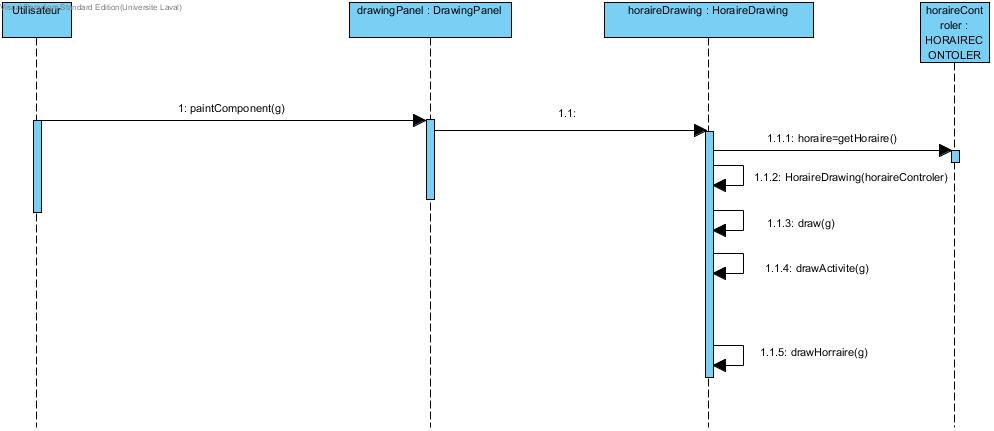
Ensuite, il faut s’assurer que le cours soit accroché à l’horaire. Ceci est éxécuté à l’aide de la méthode **verificationDrop(x,y)**. Cette méthode permet de faire en sorte que le cours s’accroche à la grille horaire lors du déplacement. Ainsi, il ne tombe pas à une position entre deux plages horaire, il est toujours à un emplacement qui est valide pour être relâché (drop).

*(Voir diagramme sur la page suivante)*



## Affichage de la grille horaire

L'utilisateur appelle la méthode **paintComponent(g)** de **DrawingPanel** en passant un objet graphique. **DrawingPanel** instancie un objet **HoraireDrawing** qui invoque à son tour **getHoraire** du contrôleur qui retourne un booléen. Selon la valeur de ce booléen, la méthode **draw(g)** invoque à la fois **drawHoraire(g)** et, soit **drawActivite(g)**, soit **drawActivite(g).**



# Fonctionnalité dont nous sommes les plus fiers

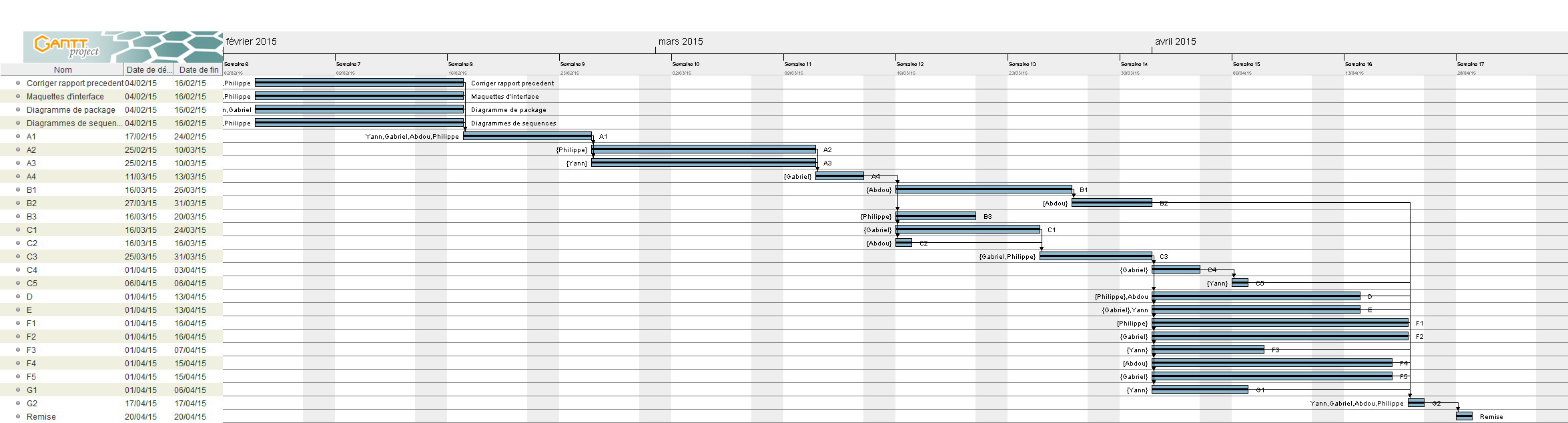
Nous avons choisi la fonctionnalité de *drag-and-drop.* Cette fonction a posé un certains défis à implémenter. Nous voulions avoir quelques choses qui montrais visuellement le processus en action, et non seulement un déplacement de la souris sans objet en dessous. De plus, nous voulions que le cours qui était en train de se faire déplacer par l’utilisateur soit toujours dans une case de l’horaire. Ainsi, lorsque l’utilisateur déplace un cours, dès que le cours est rendu au-dessus de l’horaire, celui-ci est accroché à une case. Lorsque l’utilisateur déplace la souris, le cours reste en place jusqu’au moment où la souris est plus proche de la bordure de la prochaine case.

Cette fonctionnalité nous a aussi permis de faire en sorte que la validation automatique bloque les cases qui n’étaient pas valide. Ainsi, lorsque l’utilisateur sélectionne un cours, les cases invalides (donc, qui ferait en sorte que l’horaire n’est pas valide si le cours était placé à cet endroit) sont inaccessible par l’utilisateur. Même si la souris passe par-dessus les cases grisées par la validation automatique, le cours reste en place. Il se déplacera seulement lorsque la souris sera dans une position qui permettrait de déposer le cours dans une position qui est considéré comme valide.

Cette fonctionnalité est expliqué plus en détails dans la section :Diagramme de séquence dans la sous-section : Déplacement d’un cours.

# Gestion de projet

***Voir diagrammeDeGant.gan .***



# Conclusion

Pour conclure ce rapport, nous croyons que nous avons répondu aux exigences du client. Les fonctions qui ont été demandé ont été implémenter dans notre application. Il est possible d'ouvrir un horaire vide, d'y ajouter des cours en les glissant sur l'horaire à la position désiré (Drag and drop), il y a une indicateur de validité de l'horaire qui est disponible en tout temps(cadre qui entoure la zone de l'horaire qui est vert lorsqu'elle est valide et qui devient rouge lorsqu'il y a un conflit).

Une limite de notre application vient du fait que la zone horaire est composé d'une seule zone de dessin (drawing panel). Donc, lorsqu'il y a un grand nombre de cours, il est possible d'aller voir les cours qui sont dans le bas de la fenêtre, cependant, nous ne pouvons les placer dans l'horaire puisque celle-ci n'est plus visible dans la fenêtre. Par contre, lorsqu'un cours en placé sur l'horaire, tout les cours qui était situer en dessous du cours qui a été placé sont repositionner (il remonte vers le haut). Donc, en construisant l'horaire, les cours du bas qui étaient trop loin et non disponible pour être placé sur l'horaire peuvent être facilement assigner une heure et une journée lorsqu'il remonte. Pour éviter ce problème, nous aurions dû utilisé deux drawing panel pour ainsi avoir une *scroll-bar* ajustable par panel.

De façon général, le programme est facile d'utilisation et il est intuitif pour un utilisateur débutant. Nous avons utilisé des symboles familier pour identifier les boutons de l'application pour permettre à une personne qui n'a jamais utilisé le programme de facilement l'utilisé.