

**PlanIFTicateur**

**Travail pratique 2**

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

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

Table des matières

[Introduction 3](#_Toc413142405)

[Interfaces utilisateurs 4](#_Toc413142406)

[Fenêtre principale 4](#_Toc413142407)

[Fenêtre statistiques 5](#_Toc413142408)

[Diagramme des classes de conception 6](#_Toc413142409)

[Diagramme de package 7](#_Toc413142410)

[Diagrammes de séquence 9](#_Toc413142411)

[Indicateur du nombre de cours par jour 9](#_Toc413142412)

[Affichage de la grille horaire 11](#_Toc413142413)

[Annexe 12](#_Toc413142414)

[Modèle du domaine 12](#_Toc413142415)

[Modèle des cas d’utilisation 13](#_Toc413142416)

[Glossaire 14](#_Toc413142417)

[Gestion de projet 17](#_Toc413142418)

# Introduction

La construction des horaires de session est une tâche difficile pour les directions de programme. À l’aide du logiciel dont nous avons le mandat, cette tâche deviendra visuelle et interactive, donc plus facile à réaliser. Ainsi, la direction économisera beaucoup de temps et, par le fait même, diminuera significativement le coût associé à la production d’un horaire optimale.

Ce rapport consiste à présenter le modèle de conception et l’architecture logicielle de planIFTicateur. Cette analyse sera basée sur les éléments préalablement réalisés.

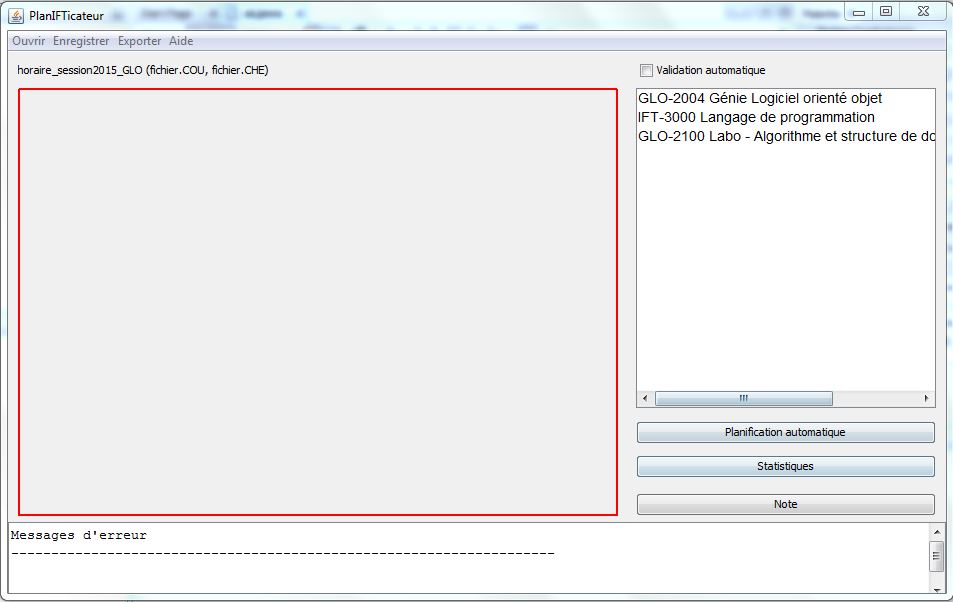
La section suivante illustre les principales interfaces utilisateurs, qui permettront d’interagir avec le système. Les diagrammes de classe de conception sont par la suite présentés. Il s’agit de l’architecture qui sera utilisée pour implanter notre solution en langage Java. Ensuite, une autre section est consacrée à expliciter la communication entre les différents packages du système.

Dans une quatrième partie nous établirons les diagrammes de séquence permettant d’accroitre la compréhension de notre système. Finalement, une dernière section vient mettre à jour la gestion du projet ainsi que les éléments du rapport précédent.

# Interfaces utilisateurs

Les interfaces utilisateurs doivent être le plus ergonomique possible. L’utilisation du principe du K.I.S.S. (Keep It Simple and Stupid) nous a permis d’en arriver au prototype suivant. 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.

## Fenêtre principale

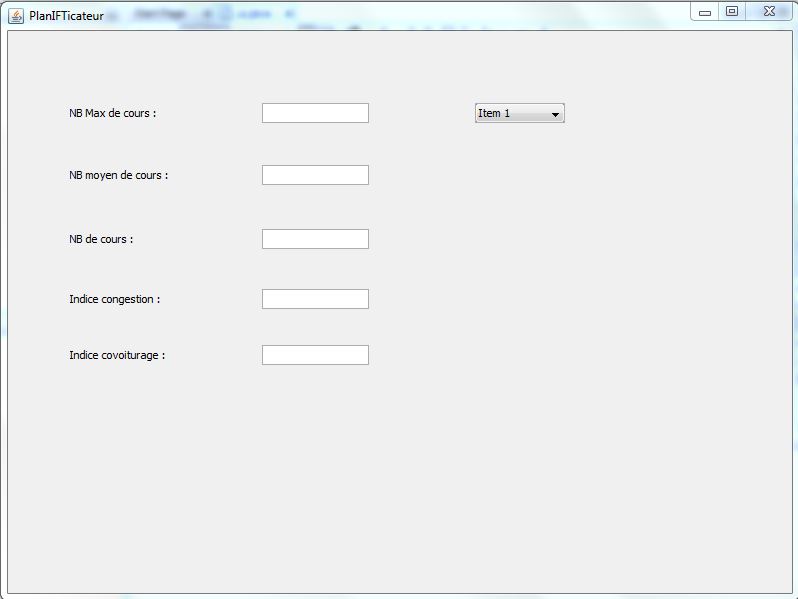


La fenêtre principale est composée des parties suivantes :

* Une barre en haut de l’écran, avec des boutons à fonctionnalités triviales.
* La grille des activités à l’horaire
* La liste des activités non attribuées
* Une liste d’erreur en cas d’horaire invalide
* Un bouton planification automatique pour activer la planification automatique
* Un bouton note pour que l’usager puisse entrer des notes
* Un bouton statistique qui conduit au prototype qui suit.

## Fenêtre statistiques

La liste des statistiques est automatiquement affichée, et l’utilisateur peut changer le jour désiré à l’aide d’un menu déroulant.

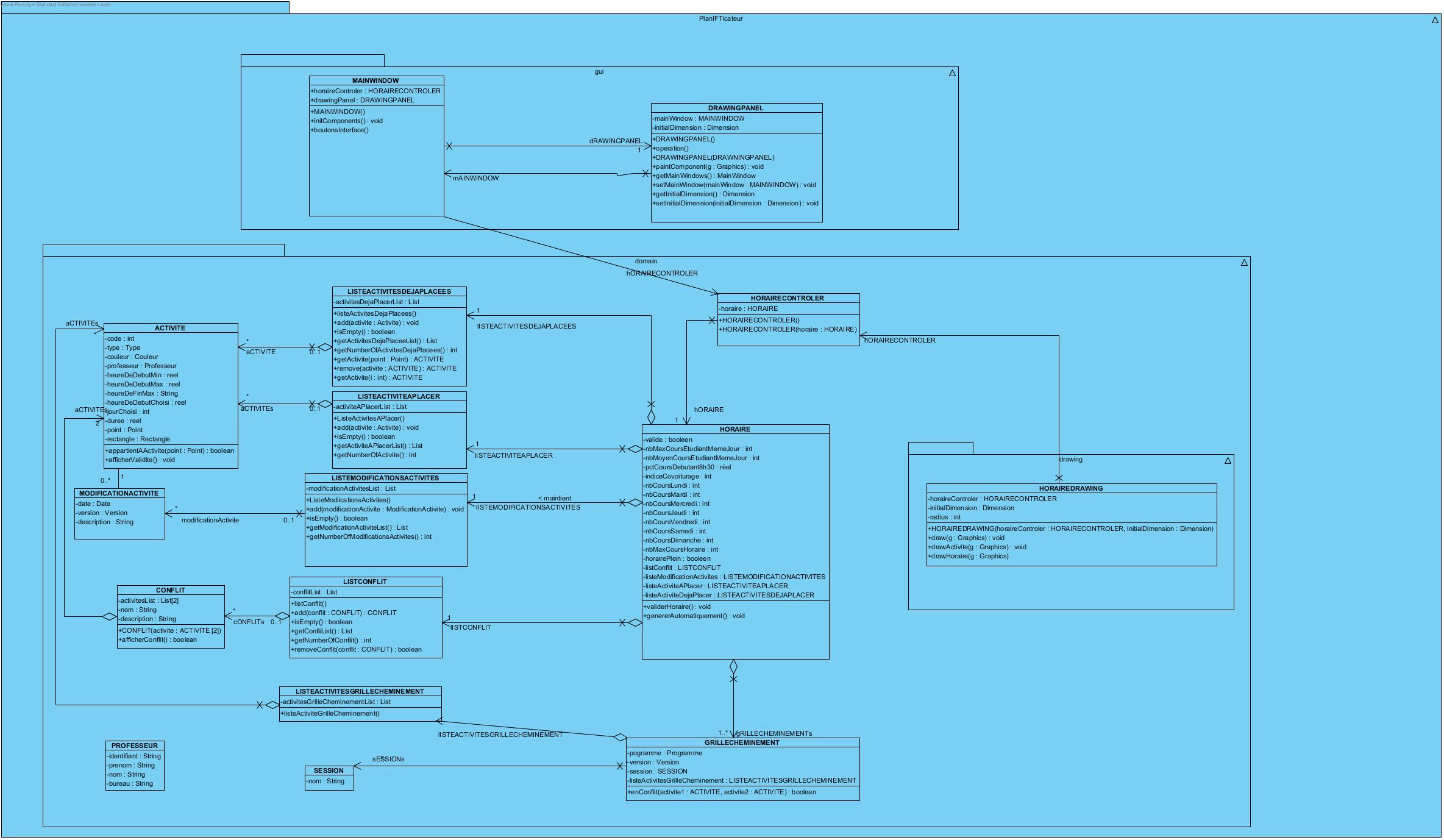


# Diagramme des classes de conception

Les classes de conception sont le cœur du système que nous développerons dans les prochaines semaines. Notre équipe propose une solution axée sur la clarté et la compréhension. Notre architecture facilitera ainsi la maintenance du logiciel.

Notre solution est présentée à travers le diagramme UML suivant :

# Diagramme de package



Le communication entre l'interface graphique et la logique métier (le domaine) se fait par le billet du contrôleur de Larman. Ce dernier reçoit de l'interface graphique des objets primitifs de type Int. Il sert d'articulation ente l'interface et le domaine et il fait les bons appels dans le domaine.

L'utilisateur appelle le **paintComponent** du **drawingPanel** . Cette méthode est chargée d'initialiser les dimensions de la zone totale utilisée par notre application. Cette classe appartient au **package** de l'interface graphique utilisateur (**GUI**).

Puis le **paintComponent** instancie un objet de type **HoraireDrawing** qui possède un objet de l**'HoraireControler**.

Par le billet de ce contrôleur qui a un objet de type Horaire alors le **HoraireDrawing** dispose de toute l'information nécessaire pour dessiner un Horaire.

Listes des activités à placer, listes des activités déjà placées, listes des conflits à afficher, listes des modifications ayant eu lieu dans l'horaire, les panneaux de chaque jour,...

Cette classe appartient à la couche logique et est dans le package de **drawing**. Un objet graphique passe de l'objet **drawingCadrePanel** vers le **HoraireDrawing**.

Cet objet graphique permet de dessiner les activités qui sont des rectangles. La longueur de de ce rectangle correspond à la durée de l'activité et sa largeur est une constantes pour toutes les activités. Les points d'intersection de la longueur et de la largeur (une droite) correspondent au début de l'activité.

La hauteur des panneaux de chaque jour de la semaine est constante et permet de connaitre à quel jour se déroule une activé. À condition que le début de l'activité soit dans cette hauteur.

La largeur du panneau de chaque jour nous donne les heures de l'horaire.

# Diagrammes de séquence

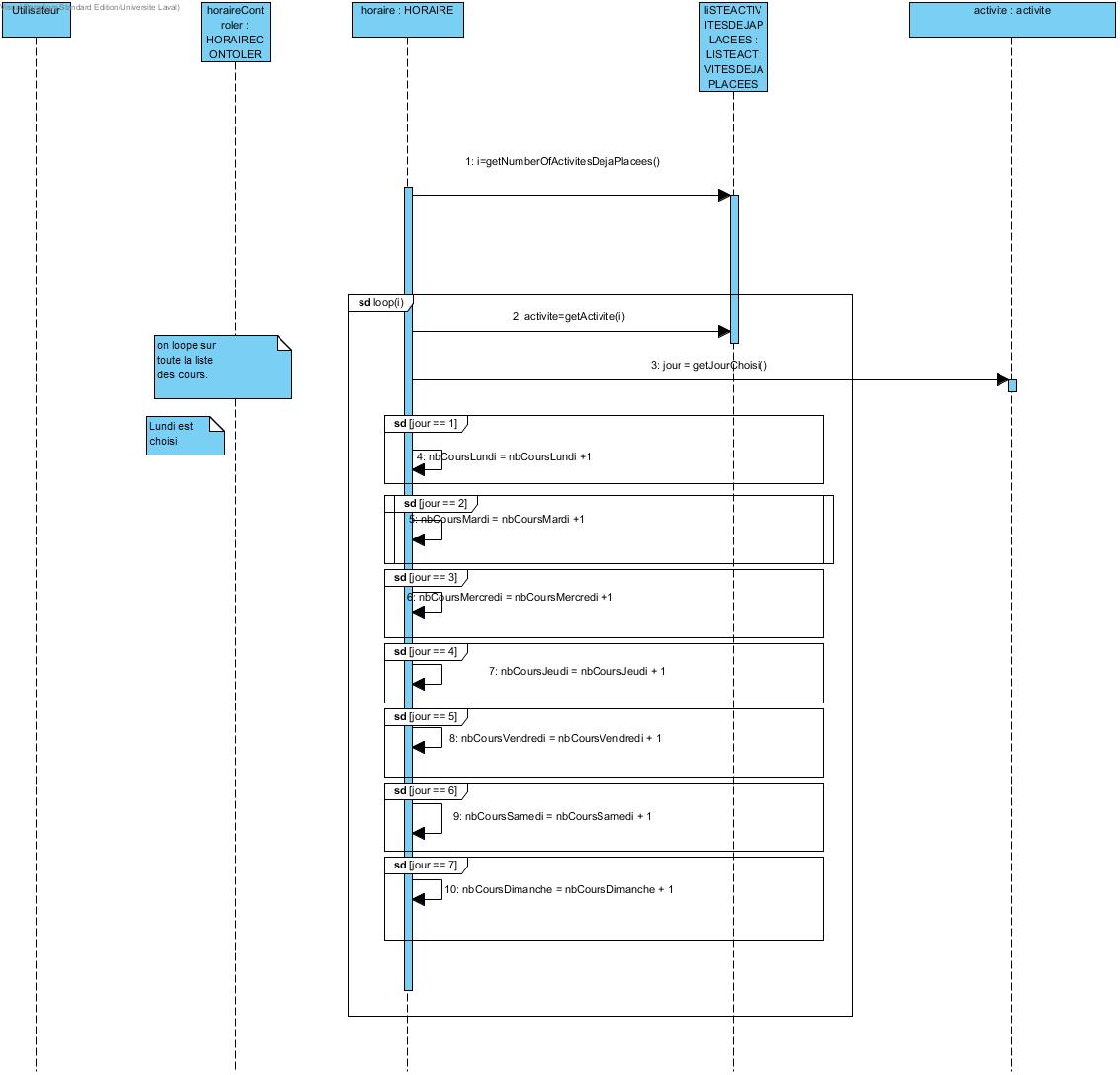
Dans cette section, il est question des diagrammes de séquence qui vous permettront de comprendre de quelle façon les classes interagissent entre elles.

## Indicateur du nombre de cours par jour

Afin d’obtenir cet indicateur, la classe **Horaire** fait la demande à la classe **ListeActivitesDejaPlacees** du nombre de cours déjà placé à l’horaire et l’enregistre dans un tableau **i**. Une fois cette opération faite, Une boucle permet d’incrémenter la variable **nbCoursJournee** appropriée (voir ici **nbCoursLundi, nbCoursMardi**, etc.)

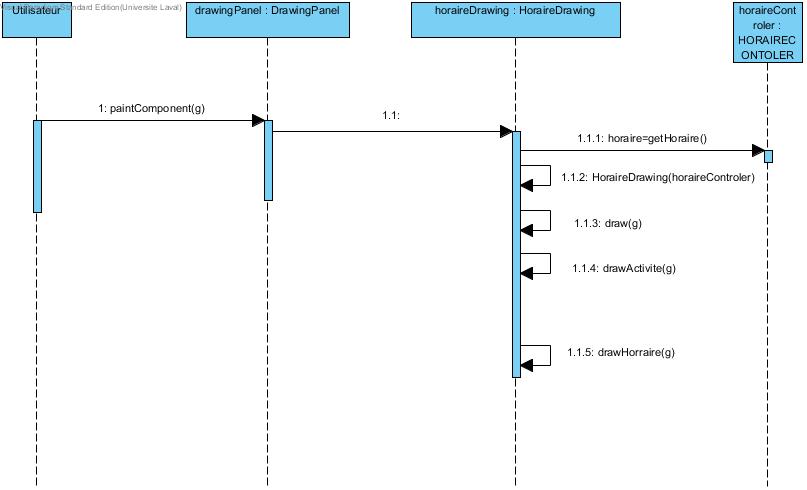
En effet, Horaire demande à la classe **ListeActivitesDejaPlacees** l’identification du premier cours déjà placé, fait une demande à la classe **Activite** pour connaître la journée attitrée à ce cours (sous forme de nombre entier) et l’enregistre sous la variable **jour**. Ensuite, comme mentionné ci-haut, **Horaire** incrémente la variable appropriée selon la variable **jour**. Une fois cette opération complétée, on incrémente l’index du tableau **i** et on répète la boucle jusqu’à ce que l’on ait traversé le tableau dans son entièreté.

Par la suite, la variable désirée pourra être cueillie avec une simple commande demandant la valeur de **nbCoursJournee** appropriée afin de l’intégrer aux statistiques ou autres.



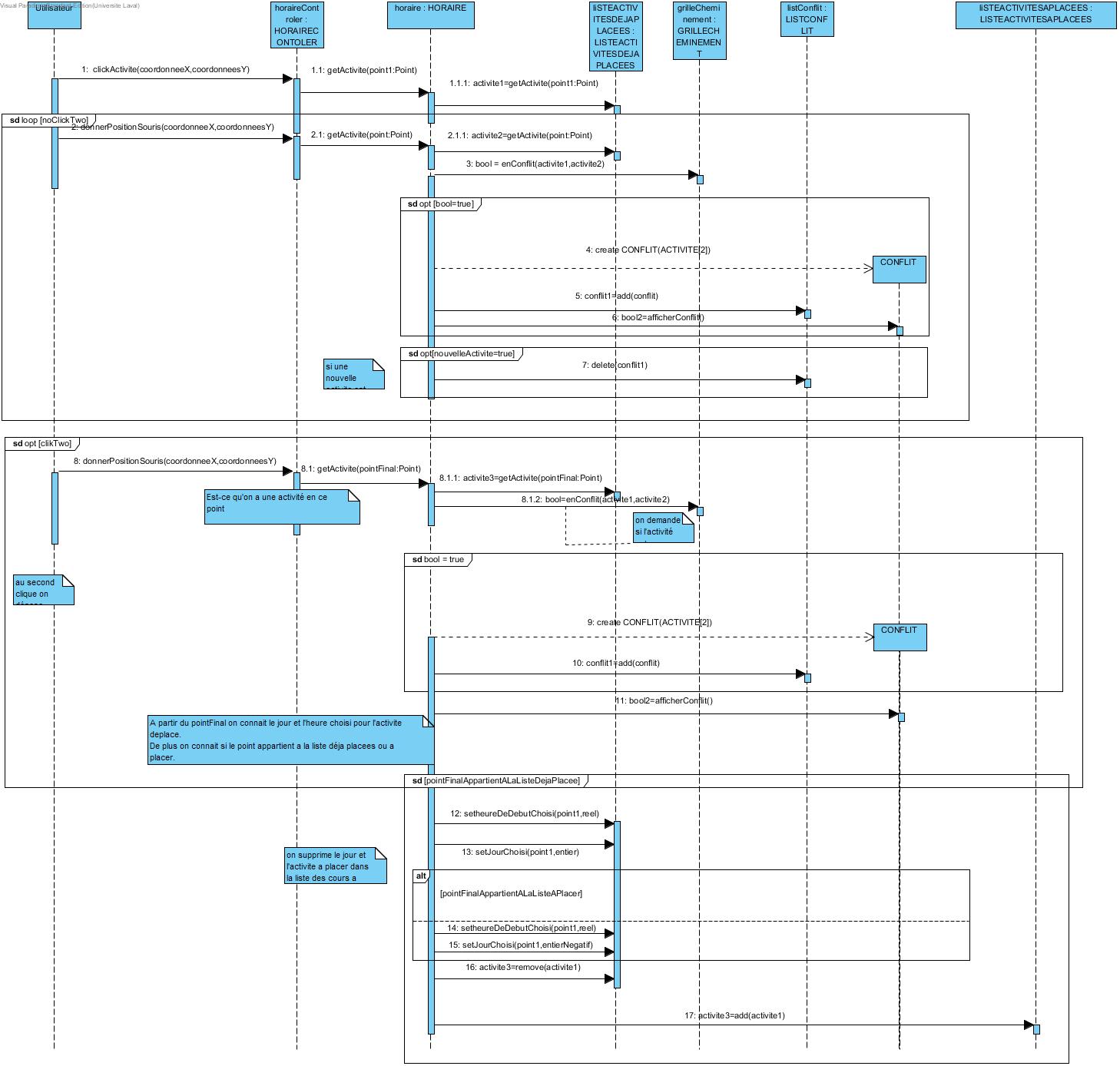
## Affichage de la grille horaire

Lorsque la grille d’affichage est appelée par l’application, une demande est faite à la classe **HoraireDrawing** afin d’ajouter les éléments graphiques nécessaires à la création visuelle de la grille horaire. Cette même classe demande à son tour au contrôleur (**HoraireController**) de lui indiquer l’objet horaire correspondant pour la suite. Notez bien que les éléments graphiques se génèreront à la suite de cette dernière demande. Finalement, la classe **HoraireDrawing** pourra dessiner créer un visuel de l’horaire en cours de création et apposer ces éléments graphiques sur la grille générée.



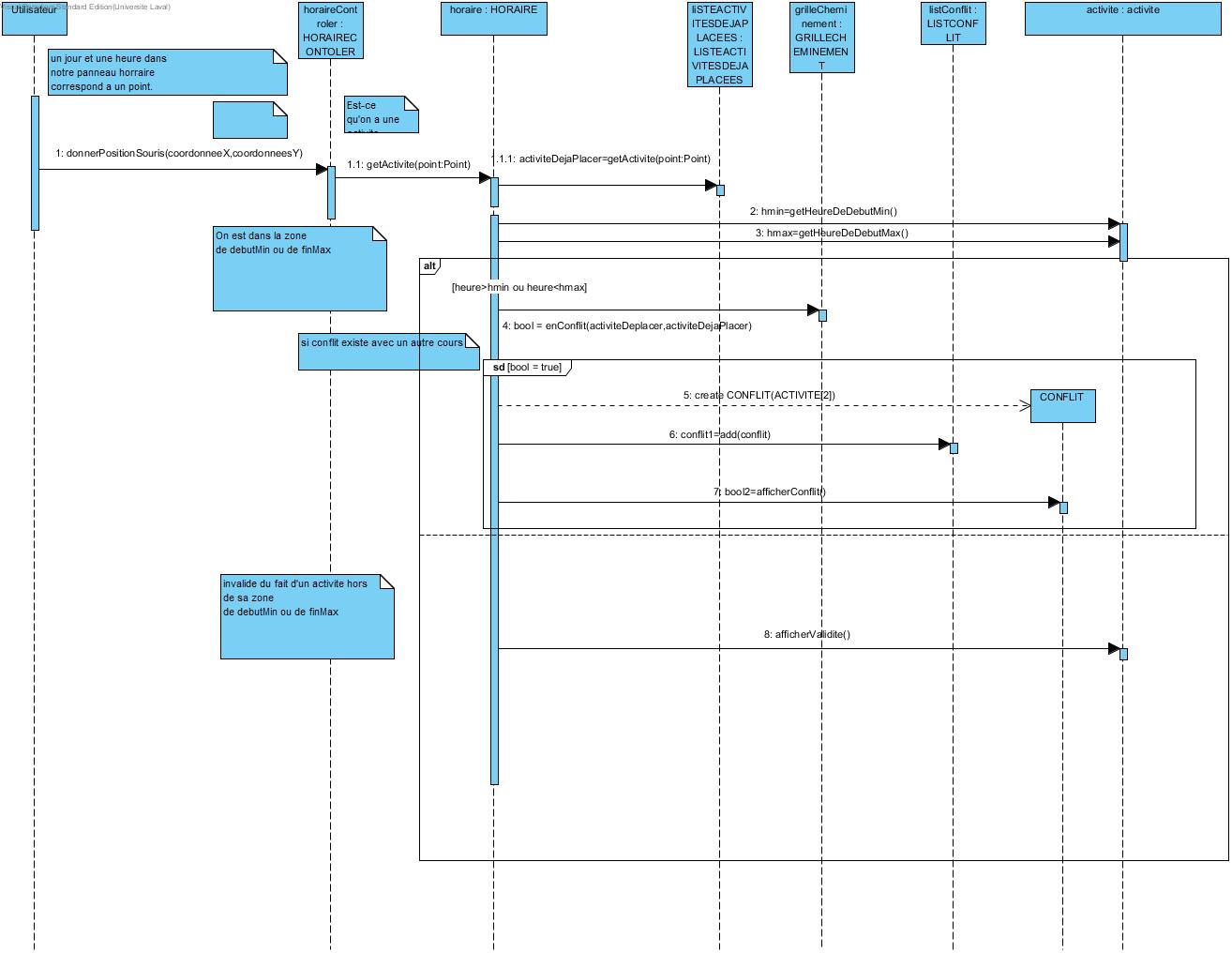
## DiagrammeDesequenceSysteme

L’utilisateur peut, en tout temps, cliquer sur une activité dans l’interface du système via la classe **HoraireController**. Cette dernière utilise la classe **Horaire** pour charger en RAM les informations de l’activité concernée, pour se préparer à un quelconque traitement. L’utilisateur peut ensuite choisir un emplacement pour l’activité dans la grille horaire. Le système, lui, à l’aide des classes auxiliaires, vérifie si il y a un conflit ou contrainte avec l’horaire actuel. Si ce n’est pas le cas l’activité est rajoutée.



## Sélection d’un cours de la grille horaire et déplacement d’un cours.

Lorsque l’utilisateur choisi une position pour une activité dans l’horaire, le système, via la classe **HoraireController**, doit étudier son choix avant de le valider. Pour cela, l’utilisateur fourni la date désirée. Le système vérifie d’abord par la classe **Horaire** si la plage horaire est libre en accédant à **listeActivitesDejaPlacees**. Si elle l’est, il vérifie dans un second temps si le choix respecte la grille de cheminement courante. Lorsque toutes ces contraintes sont respectées, le nouveau positionnement de l’activité est enregistré.

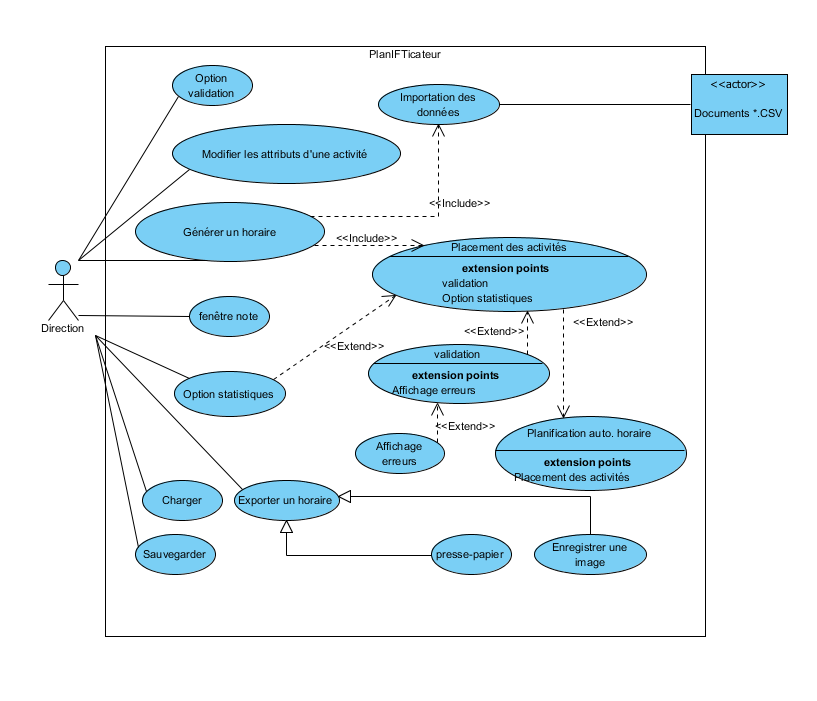


# Annexe

## Modèle du domaine

A confirmer (retrait 2 classes possible?)

## Modèle des cas d’utilisation



## Glossaire

**Algorithme**

Suite de règles permettant de résoudre un problème.

**Application**

Programme ou ensemble de programmes visant à aider un utilisateur d’un ordinateur dans le traitement d’une tâche précise. *(Réf. : Larousse.fr)*

**Cas d’utilisation**

Relation établie entre une fonctionnalité et, soit un acteur ou une autre fonctionnalité.

**Chemin**

Emplacement précis sur un disque dur définit par la suite ordonnée des dossiers pour avoir accès au fichier recherché.

**Diagramme de séquence système (DSS)**

Représentation des interactions chronologique entre un système et ses acteurs. *(réf. : Wikipédia)*

**Domaine d’affaires**

Référent pour l’ensemble des processus d’affaires d’un projet ou d’une organisation. Englobe les entités, les acteurs et autres participants de ces processus.

**« Drag and drop »**

***(Terme anglais)*** En informatique, processus durant lequel un usager d’un ordinateur sélectionne un objet, au moyen d’une souris, en maintenant enfoncé le bouton gauche de cette dernière afin de déplacer le dît objet et de le relâcher à l’endroit voulu en relâchant également le bouton de sa souris.

**Fenêtre**

Zone d’affichage d’informations d’un programme.

**Fonction**

Bloc d’une séquence d’instructions visant un but, une fonctionnalité précise.

**Logiciel**

Ensemble d’instructions et de règles interprétable par un ordinateur.

**Multiplateforme**

Fonctionnant sur plusieurs plateformes, soit plusieurs ordinateurs/systèmes d’exploitation différents.

**Multiutilisateur**

Offrant la possibilité à plusieurs usagers d’interagir simultanément sur le même logiciel.

**PlanIFTicateur**

Nom du logiciel en développement pour le projet contenu dans ce document.

**Plateforme PC**

Liaison entre un ordinateur personnel munit d’un processeur spécifique et du système d’exploitation *Windows*.

**Presse-papier**

Fonction intégrée dans tous les systèmes d’exploitation stockant des données que l’on souhaite déplacer ou copier.

**Programme**

Succession d’instructions qu’un ordinateur peut exécuter afin d’accomplir des opérations.

**Serveur dédié**

Système informatique dont l’ensemble des ressources est dédié à un seul utilisateur.

**Serveur web**

Système informatique qui a pour fonction la publication de sites web à la demande d’un autre système.

**Session**

Période de 15 semaines durant laquelle un étudiant de l’université doit accomplir les objectifs de ses cours.

**GUI**

Signifie : **G**raphical **U**ser **I**nterface, ce qui représente l’interface graphique utilisateur. C’est ce qui est affiché par le programme.

**Diagramme de package**

Permet de représenté l’architecture des différents groupements de classe en différentes couches logique.

**Diagramme de séquence**

Diagramme illustrant ce qui se produit lors de certaines actions. Permet de comprendre la communication entre les classes.

## Gestion de projet

***Voir diagrammeDeGant.gan***