POLYTECH NICE-SOPHIA

Université Nice Sophia Antipolis

Membre de UNIVERSITÉ CÔTE D'AZUR

Rapport d'Architecture

LivrAir

Team L

Bertin Loïc - Hoareau Grégory - Ladorme Guillaume -
Passin-Cauneau Barthélemy - Viale Stéphane

Sommaire

I - Cas d’utilisations

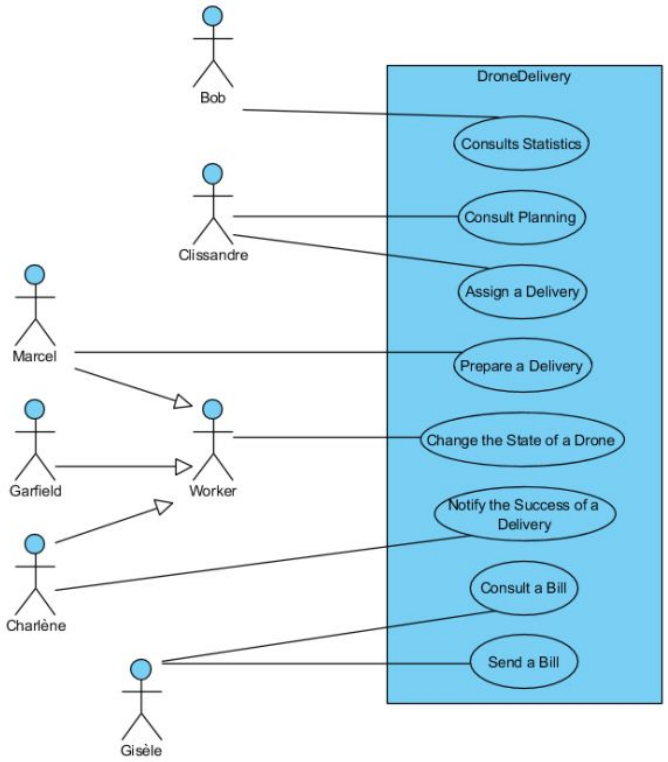


Image 1 : Diagramme des cas d’utilisation

Ce diagramme des cas d’utilisations permet de visualiser les interactions entre les employés de LivrAir et notre système informatique.

Bob a besoin de consulter en temps réel différentes statistiques afin de surveiller l’évolution de son entreprise et son rendement.

Clissandre doit pouvoir interagir avec le planning afin de consulter les rendez-vous et assigner un colis à une heure de livraison. En interne le programme doit ensuite voir si la livraison est réalisable et renvoyer une information de validation ou non à Clissandre.

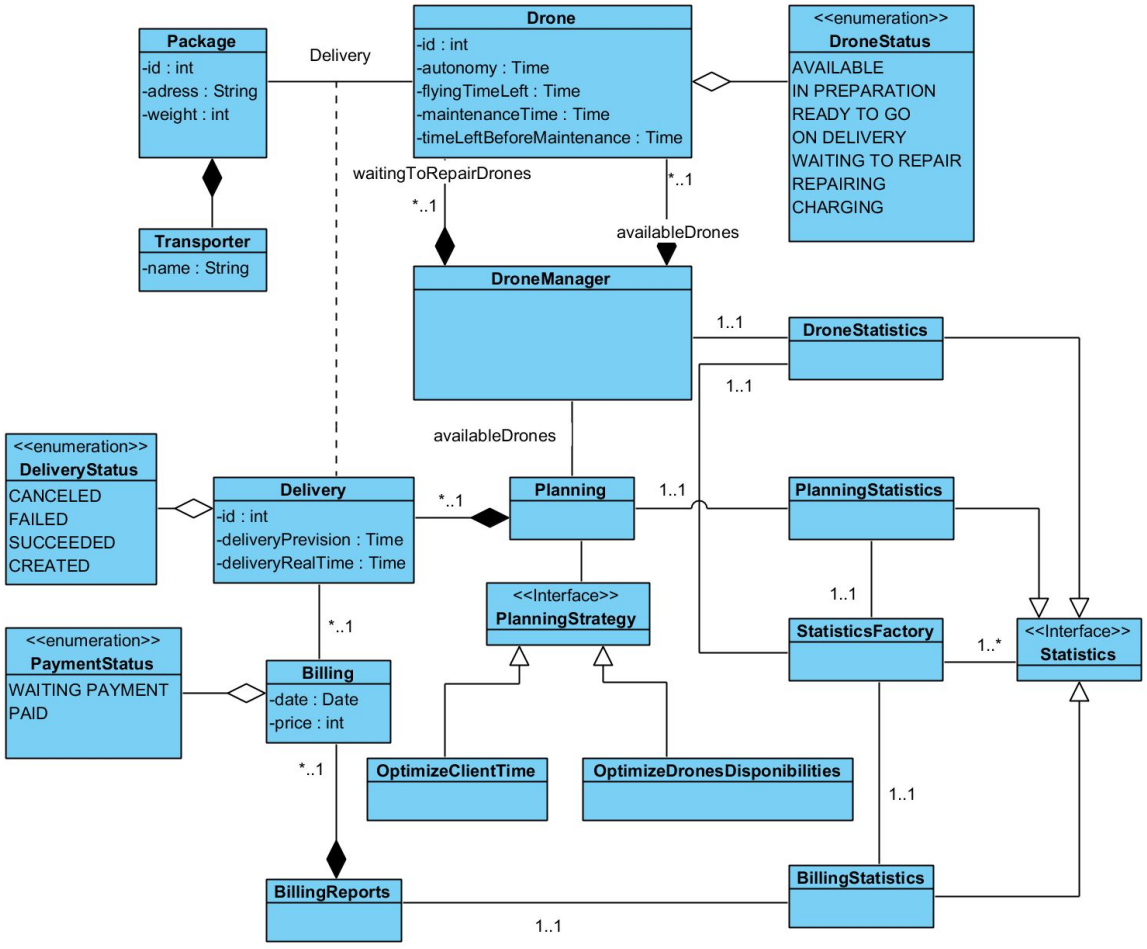
Marcel a besoin de savoir quel colis accrocher à quel drone et à quel moment le faire. Le système informatique doit donc lui remonter cette information au bon moment. Marcel doit ensuite pouvoir cliquer sur l’écran pour indiquer que le drone est prêt à décoller et ainsi changer l’état du drone.

Garfield a peu d’interaction avec le système informatique, il doit uniquement informer lorsqu’il a fini de réviser un drone et donc changer son état.

Charlène doit pouvoir notifier du succès ou non d'une livraison afin d'avoir un retour sur le taux de réussite des livraisons et sur l'édition de la facture pour les transporteurs. Elle doit également pouvoir changer l'état d'un drone afin de définir si ce dernier est en charge, si elle l'amène à Garfield ou si elle le rend à Marcel.

Gisèle doit pouvoir consulter les factures puis les envoyer aux différents transporteurs afin d'enclencher le paiement. Nous avions un choix à faire : soit le système envoyait tout seul la facture tous les jours à 23h59 soit Gisèle le faisait. Afin d'avoir des valeurs d'entreprise et de maximiser l'embauche, nous avons choisi de conserver Gisèle ainsi que son poste : c'est donc à elle d'envoyer les factures.

II - Diagramme de classes des objets métiers



Nous avons décidé pour notre architecture d'utiliser un pattern *Strategy* pour la réalisation du planning car la société LivrAir pourrait souhaiter appliquer différentes solutions pour choisir le drone qui livrera un colis. Ainsi elle pourra optimiser l'utilisation de ses drones pour gérer leur usure globale et pouvoir réduire les coûts matériels. On peut aussi imaginer qu'elle souhaite augmenter la satisfaction de ses clients et ainsi se tenir le plus possible au premier horaire donné par ces derniers, quitte à augmenter leurs coûts de réparation. Pour le MVP nous ferons un algorithme simple pour gérer le planning, mais nous laissons ainsi la

possibilité de faire évoluer le modèle simplement vers d'autres idées pour optimiser la gestion des livraisons.

Pour la gestion des statistiques nous les avons séparées selon différentes catégories :
- le planning : nombre de demande journalière de livraison, moment dans la journée où les demandes sont les plus fréquentes, …
- les drones : taux d'utilisation de la flotte de drones, nombre de mission par drone, …
- les factures : nombres de factures payées/impayées, raison de ces impayés (par exemple une livraison qui a échoué par la perte du colis ou non récupération par le client), durée moyenne entre la demande de livraison et le paiement, …

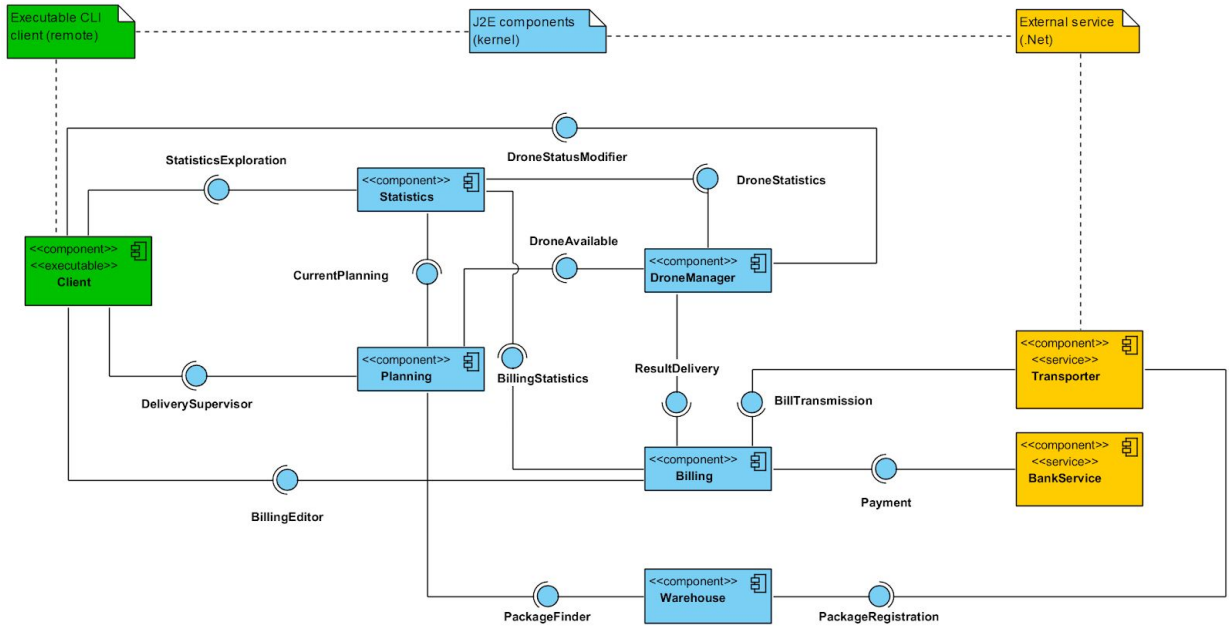
Pour gérer toutes ces statistiques et n'afficher que celles qui intéressent Bob (le boss), nous avons choisi d'utiliser un pattern *Factory* qui nous permettra de construire les statistiques en fonction de ce qu'il souhaite consulter. De plus, cela nous permettra facilement de rajouter de nouveaux types de statistiques.

Pour la gestion des drones, nous avons créé une nomenclature avec les états *DroneStatus* qui permettent de trier les drones en fonction de leur mission actuelle. Cela nous permet de savoir les drones qui sont disponibles pour une mission, ceux qui doivent être chargés et enfin ceux qui doivent être maintenus. Comme Garfield ne peut réparer qu'un drone à la fois, les autres sont mis dans une file d'attente pour pouvoir les réparer dans l'ordre, puis leur état sera remis à *Available*. Nous avons d'autres états qui nous permettent de mieux visualiser ce que font nos différents drones pour mieux gérer nos statistiques.

Pour la gestion des factures, l'ensemble des factures sera contenu dans *BillingReports* quel que soit leur statut ("*PAID*" ou "*WAITING PAYMENT*"), une facture sous le status "*WAITING PAYMENT*" est créée quand une livraison a été effectuée avec succès par un drone. Et passe sous le status "*PAID*" quand le transporteur a réglé le montant de la facture.

Pour la gestion des livraisons, chaqu'une d'entre elle est représentée par un identifiant, un horaire de livraison défini par le client (*deliveryPrevision*) et l'horaire de livraison réel (*deliveryRealTime*) du colis. De plus, ces deux horaires seront utilisés pour définir la satisfaction du client. Une livraison peut avoir différents statuts : "*CANCELED*" si la livraison est annulée par le client, "*FAILED*" si la livraison n'a pas été achevée (colis non récupéré par exemple), "*SUCCEEDED*" si la livraison a bien été effectuée et "*CREATED*" si la livraison n'a pas encore été traitée.

III - Diagramme de composants



Nous avons décidé d'intégrer un service externe *Transporter* qui aura différentes utilités. Tout d'abord, il permettra via *PackageRegistration* d'ajouter les colis livrés à notre système. On pourrait par exemple avoir un système de scanner à code-barre qui, en l'utilisant, ajoute le colis à notre système. Ensuite, il permettra l'envoi des factures aux différents transporteurs pour qu'ils puissent avoir un retour de ce qu'ils ont payé.

Nous avons également décidé d'utiliser un service externe *BankService* qui a pour seul but de gérer le paiement des factures qui auront au préalable été envoyées aux différents transporteurs via l'interface *Payment*. Cette interface permet d'envoyer une demande de paiement aux entreprises ayant fait appel au service de LivrAir.

Le composant *Billing* est le point d'interaction majeur du système avec les composants externes. Il doit gérer les factures en prenant en compte les résultats des livraisons effectuées dans la journée, mais aussi d'interagir avec les services externes (donc d'envoyer les factures aux transporteurs ainsi que de gérer leurs paiements).

Le composant *Planning* sert à organiser le planning des livraisons. Lorsqu'un client effectue une demande de livraison, c'est ce composant qui va confirmer (ou infirmer) la date en prenant en compte la stratégie d'utilisation des drones définie par LivrAir.

Le composant *Warehouse* sert à organiser le stockage des colis (*packages*). Lorsqu'un transporteur apporte un colis, il est enregistré par ce composant et le planning pourra par la suite le trouver, si un client appelle, pour livrer un de ses colis.

Le composant *Statistics* sert à agréger les données du système pour être ensuite affichées de façon compréhensible au composant *Client*.

Le composant *DroneManager* gère la flotte de drone de l'entreprise. Il doit à tout moment, connaître les drones qui sont disponibles, en charge ou en réparation.

IV - Interfaces pseudo-code

StatisticsExploration :

```
List<Statistics> getStats(statistiques: Statistics)
```

Permet à Bob de renseigner le nom des statistiques qu'il souhaite voir et ainsi d'obtenir un état des lieux de sa société.

DroneStatusModifier :

```
void updateDroneStatus(drone : Drone, droneStatus:
DroneStatus)
```

Permet de mettre à jour l'état d'un drone pour qu'il réalise la mission qu'il a besoin de faire à un instant T. Ce statut peut être changé par un des états suivants: *AVAILABLE*, *IN PREPARATION, READY TO GO, ON DELIVERY, WAITING TO REPAIR, REPAIRING, CHARGING*. Cela permet à Charlène, Garfield et au planning de savoir quoi faire avec les drones en fonction de leur état.

CurrentPlanning :

```
Planning getPlanning(date: Date)

Planning getPlanningForDrone(date: Date, drone: Drone)

int getAmountDelivery(hour: Time)

int getAmountDelivery(date: Date)

List<int> getAmountDeliveryByDrone(date: Date)
```

Permet d'obtenir un visuel sur le planning de vol d'un ou de tous les drones à un jour donné ainsi que de savoir le nombre de livraisons à une heure précise (pour voir les pics d'affluence), le nombre de livraison un jour précis et le nombre de livraison par drone pour

observer s'il y a une sur-utilisation de drones (qui pourrait être dû à un problème dans l'algorithme du planning).

DroneAvailable :

```
List<Drone> getAvailableDrone(hour: Time)
```

Retourne l'ensemble des drones disponibles à une heure fixée.

DroneStatistics :

```
Statistics getDroneStatistic(hour: Time)

Statistics getDroneStatistic()

List<Drone> getDronesByStatus(droneStatus: DroneStatus)
```

Retourne les statistiques des drones sur la journée ou sur une heure précise afin d'avoir une vision globale de la flotte.

DeliverySupervisor :

```
void createNewDelivery(package: Package, date: Time)

Planning getPlanning(date: Date)
```

Permet à Clissandre d'ajouter une nouvelle livraison quand un client la contacte. Envoie au système les informations nécessaires et le système retournera à Clissandre si la livraison pourra avoir lieu à cette heure-ci.

ResultDelivery :

```
void updateDeliveryStatus(delivery: Delivery, status: DeliveryStatus)
```

Permet de mettre à jour l'état d'une livraison parmi les statuts suivants: *CANCELED*, *FAILED*, *SUCCEEDED*, *CREATED*. Ainsi on peut savoir à tout moment l'état d'une livraison et de pouvoir calculer différentes statistiques comme par exemple le nombre d'annulations ou le pourcentage de réussite d'une livraison.

BillingEditor :

```
Bill generateBill(date: Date, transporterName: String)

void changeBillStatus(paymentStatus: PaymentStatus, date: Date, transporterName: String)

void sendBill(date: Date, transporterName: String)
```

Permet à Gisèle de générer les factures pour pouvoir les vérifier et les envoyer aux transporteurs et demander le paiement.

BillingStatistics :

```
Boolean isBillingPaidByThe(date: Date, transporterName: String)

float getPercentBillingPaidBy(transporterName: String)

float getPercentBillingPaidThe(date: Date)

float getPercentBillingPaid()
```

Permet d'obtenir différentes statistiques sur les factures notamment si un transporteur a payé ses livraisons pour un jour donné, son pourcentage de paiement actuel, le pourcentage de paiement des entreprises pour un jour donné et enfin le pourcentage de paiement à jour dans la société LivrAir.

Payment :

```
bool payment(transporter: Transporter, amount: int)
```

Envoi une demande de paiement au service de paiement externe des banques pour que le transporteur paie sa facture.

PackageFinder :

```
Package searchPackageByID(packageID: int)
```

Permet de retrouver un paquet dans le système via son numéro d'identification.

PackageRegistration :

```
void registerPackage(package: Package)
```

Permet au transporteur d'ajouter ses colis à notre système pour que nous puissions les prendre en charge lors de nos réservations de créneaux pour les livraisons.

BillTransmission :

```
Bill sendBill(date: Date, transporterName: String)

BillingReports sendBills(transporterName: String)
```

Permet à Gisèle tous les soirs d'envoyer la facture d'un transporteur à celui-ci.