Drone Delivery

Rapport d’architecture

ABDENNADHER Younes

HAENLIN Jason

LONGORDO Alexandre

LEFEBVRE Alexis

MARCELLIN Betsara

Table des matières

I) Introduction

Drone Delivery est un système permettant de gérer une expédition de colis par drone sur les derniers kilomètres de la livraison. Les acteurs principaux sont les employés qui s'occupent soit de la gestion de l'entreprise, soit de l'envoi des drones. L'objectif est d'avoir un système qui permet de planifier et de gérer l'ensemble des drones et leurs livraisons.

II) Cas d'utilisations

A) Diagramme de cas d'utilisations

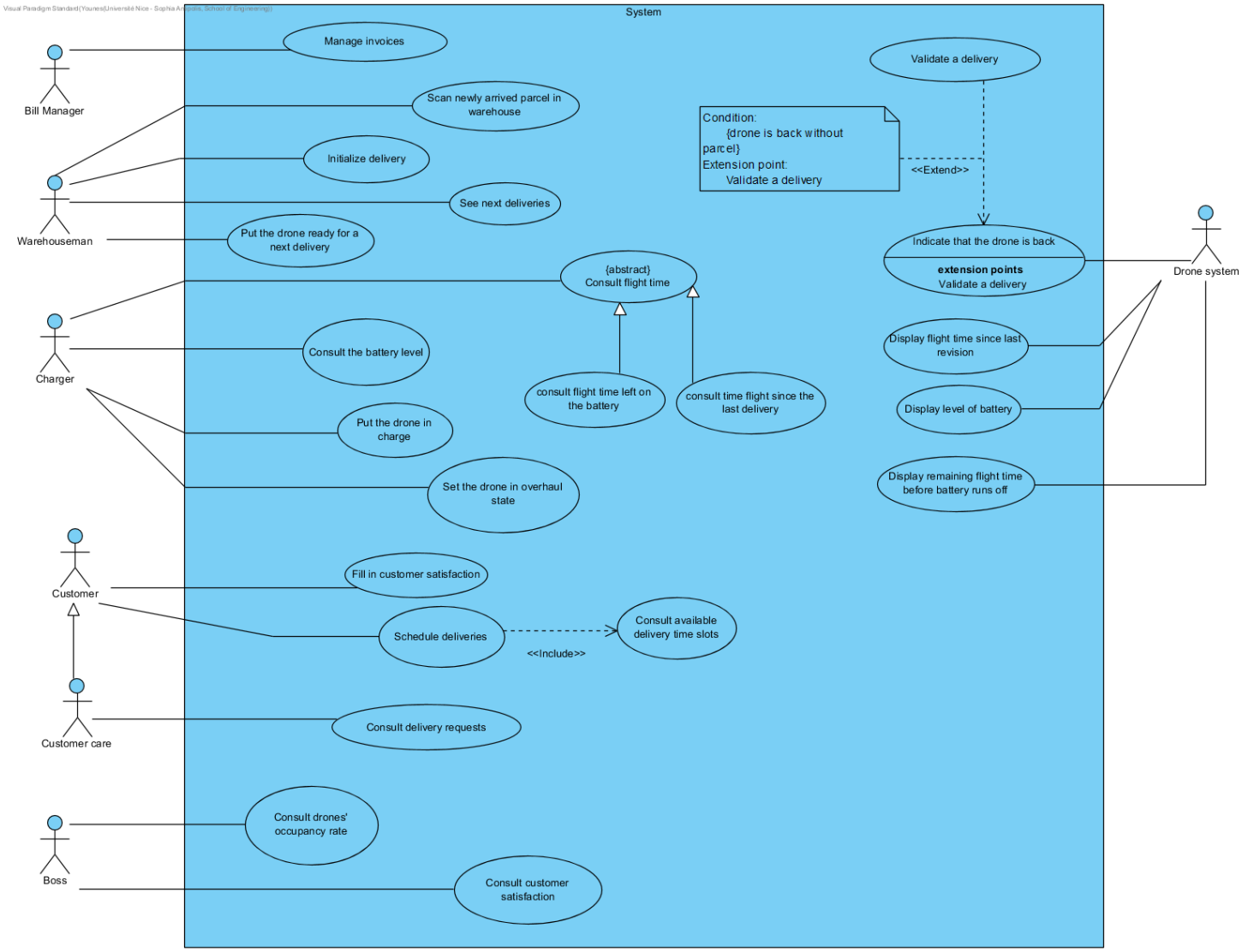

Figure 1 : Diagramme de cas d'utilisations

Précisons certains points du diagramme de cas d'utilisations (fig. 1).

Tout d'abord, chaque acteur du sujet correspond à un acteur du système mis à part le garagiste qui n'interagit jamais avec notre système. Les deux actions système relatives à son métier (c'est-à-dire le

fait d'indiquer que le drone entre en phase de révision et le fait d'indiquer que le drone n'est plus en révision), incombent à deux autres acteurs :

- le chargeur, qui, une fois la vérification du niveau de charge et du temps de vol faits, change lui-même le statut du drone en *révisé* avant de le remettre au garagiste ;
- et le manutentionnaire qui reprend le drone une fois que le garagiste a fini la révision et va le poser à un emplacement prévu à cet effet. Nous considérons donc que le fait de ranger le drone à son emplacement pour la mission suivante enclenche automatiquement le changement de son statut en *disponible* (il est aussi possible de considérer qu'une fois le drone remis au manutentionnaire, ce dernier interagit avec le système d'une autre manière pour changer son statut en *disponible*, dans les deux cas, le garagiste n'interagit pas avec le système).

D'autre part, on peut remarquer que le client (celui qui reçoit le colis) est un acteur du système. Il pourra en effet, à terme, effectuer lui-même les prises de rendez-vous et donner son avis sur la livraison sans avoir à passer par le service client, mais via une interface dédiée.

Le système embarqué dans le drone est également l'un des acteurs. Celui-ci nous fournira une API permettant de lui envoyer des signaux de contrôle (comme *démarrer*) et de récupérer des informations fournies par celui-ci (notamment son niveau de batterie et le son temps de vol).

Le gestionnaire peut gérer les factures. Cela comprend :

- la création d'une facture,
- l'édition et la génération d'une facture,
- la suppression d'une facture qui est avortée par exemple,
- la validation d'une facture (le client a bien réglé la facture).

B) Scénario étendu

Afin de cerner précisément le périmètre du projet, étendons le scénario décrit dans le sujet en prenant en compte et en précisant tous les cas d'utilisations couverts par notre diagramme.

- **Marcel reçoit les nouveaux colis dans l'entrepôt et les scanne pour les ajouter avec toutes leurs informations dans le système puis les range dans l'entrepôt ;**
- le client appelle Clissandre pour prévoir un horaire de livraison et Clissandre prévoit le créneau dans le planning **ou le client alors se connecte au site Web pour entrer lui-même un horaire de livraison dans un créneau libre pour sa livraison** ;
- **Marcel regarde son écran pour avoir les références des prochains colis à charger ainsi que le numéro du drone associé,** attache le colis à ce drone, **et appuie sur le bouton *Initialiser* pour que le système envoie le signal de lancement au drone au bon moment** ;
- au retour du drone, Charlène le récupère, et soit le met en charge, soit l'apporte à Garfield pour révision **et change son état depuis son interface pour avertir qu'il est en révision**, soit l'apporte à Marcel pour la mission suivante. **En charge, un écran indique à Charlène le temps de charge restant.** Le retour du drone vide (donc colis livré) déclenche une écriture comptable qui indique à Gisèle que la livraison a bien eu lieu ;
- **lorsque Marcel reçoit le drone de la part de Garfield ou de Charlène, il le pose sur son emplacement afin que le système passe son état en prêt pour la prochaine livraison** ;
- tous les soirs, Gisèle émet les factures ;
- **lorsqu'un client reçoit une livraison, il peut donner son avis soit par téléphone à Charlène soit en passant par le système** ;
- en continu, Bob suit les indicateurs métier.

III) Objets métiers sous forme de diagramme de classes

Les objets métiers sont représentés sur la figure 2.

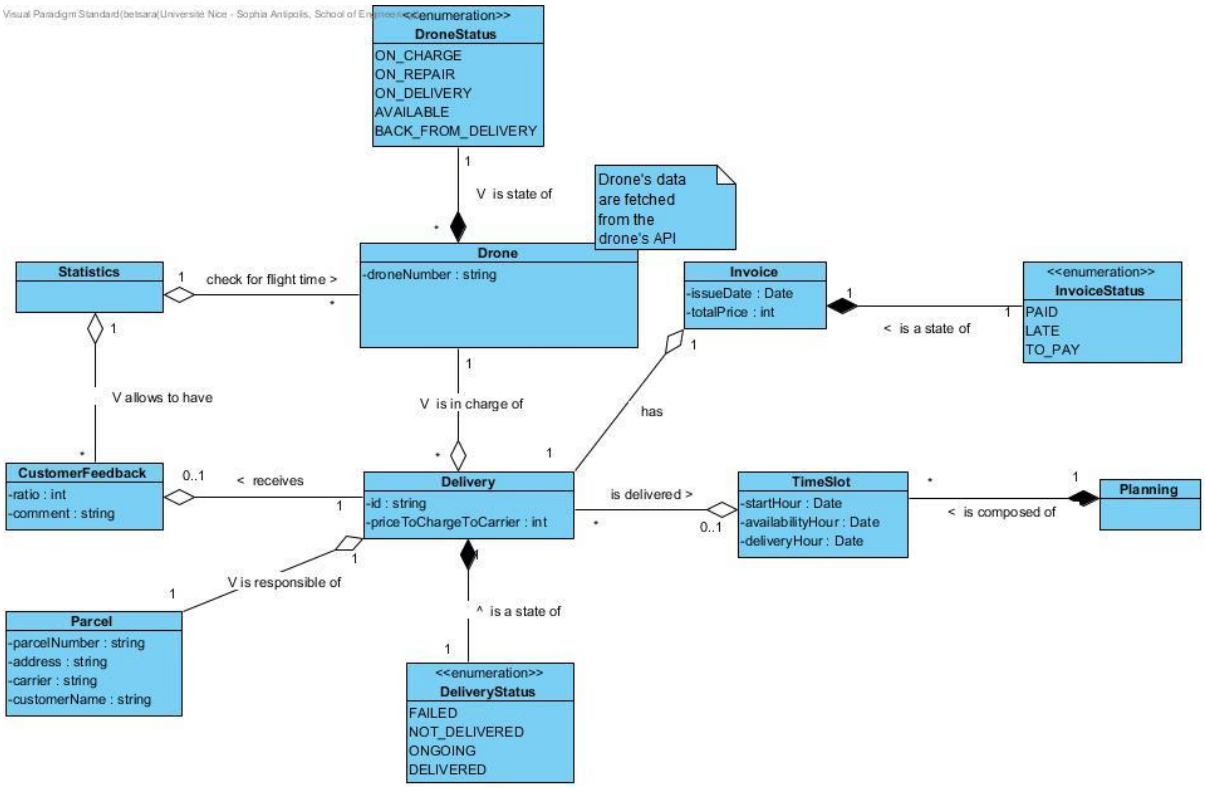

Figure 2 : Objets métiers sous forme de diagramme de classes

IV) Diagramme de composant

A) Vue d'ensemble

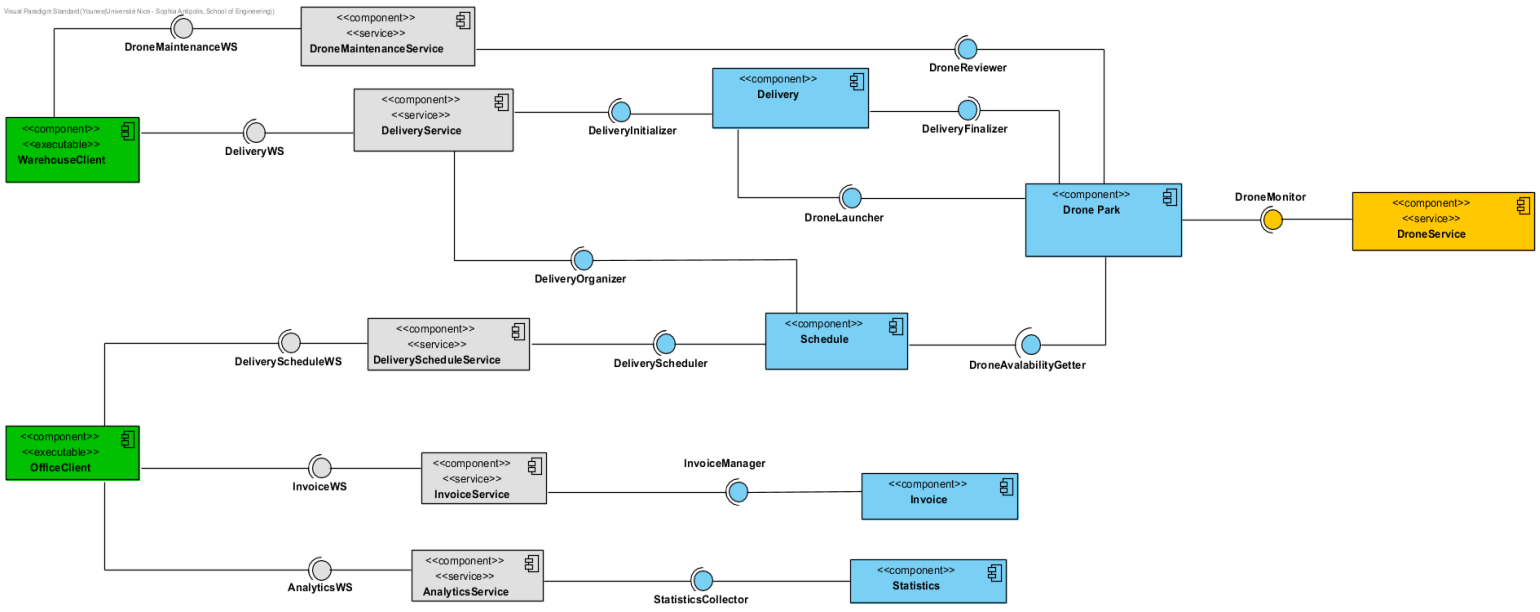

Figure 3 : Diagramme de composants

B) Précisions

Comme nous pouvons le voir dans notre diagramme de composant (fig. 3), l'architecture est découpée en 4 parties.

- Le client en **vert** correspond aux CLI qui vont interagir avec notre système.
- Les web services en **gris** sont les points d'entrées de notre système.
- Le système en **bleu** correspond aux composants de notre système.
- Le système externe en **orange** correspond à l'API du drone qui permet de le gérer depuis notre système.

Nous avons deux CLI différents, nous faisons ce choix afin de différencier la manutention de la bureautique. En effet, le CLI manutention sera dédié à la gestion des colis, des drones et de la mise en œuvre des livraisons. Le CLI bureautique sera dédié à la gestion des factures, des plannings de livraison, des avis clients et des statistiques.

Chaque CLI fera appel aux différents points d'entrée que sont les web services. Prenons pour exemple les composant *DeliveryScheduleService et DeliveryService*. Le composant *DeliveryScheduleService* permet de planifier les livraisons (récupérer les disponibilités, insérer une date de livraison) il sera utilisé par le CLI *OfficeClient*. Le composant *DeliveryService* s'occupe de gérer les livraisons (créer une livraison, démarrer une livraison) il est utilisé par le CLI *Warehouse*.

Nous avons plusieurs composants qui contiennent la logique métier. Ils sont appelés par les web services :

- Le composant *Schedule* est un composant permettant de calculer les disponibilités des drones, de mettre une livraison dans le planning, et d'afficher les prochaines livraisons. Pour calculer les disponibilités des drones, il se connecte au composant Park de drone.
- Le composant *DronePark* gère l'ensemble des drones, il permet de récupérer les données des différents drones (autonomie, temps de vol…), il permet aussi de lancer un drone et gère les disponibilités des drones. Pour cela il utilise l'API fournie par le service externe du drone.
- Le composant *Delivery* s'occupe de la gestion des livraisons (créer une livraison, lancer une livraison et les différents états de la livraison). Pour cela il fait appel à *DronePark* pour lancer le drone lorsqu'une livraison doit être démarrée.
- Le composant *Invoice* gère les factures (créer, modifier, consulter et supprimer les factures)
- Le composant *Statistics* gère les entrées de nouvelles données utiles aux statistiques et l'affichage des statistiques.

Enfin nous avons un composant symbolisant l'API externe fournie par le drone pour avoir ses informations.

Le drone pourra fournir depuis son système le niveau de charge de la batterie, la valeur du compteur du nombre d'heures de vol total et celle d'un deuxième compteur qui pourra être remis à 0 autant de fois que l'on souhaite. Nous utiliserons ce dernier compteur pour compter le nombre d'heures de vol depuis la dernière livraison.

Le composant *DroneService* permet aussi d'ajouter une adresse de destination et de lancer le drone vers l'adresse.

V) Définition des interfaces

1) StatisticsCollector

Cette interface a pour responsabilité de s'occuper des statistiques à la fois pour recueillir les avis des clients mais aussi pour avoir un aperçu du taux d'occupation des drones.

```
void addCustomerFeedback(CustomerFeedback feedbacks);
CustomerFeedback[] getCustomerFeedback();
double getOccupancyRate(Drone drone);
```

2) InvoiceManager

Cette interface regroupe les fonctionnalités pour gérer la partie facturation de notre système. La méthode de génération de la facture ne renvoie rien et va juste générer un document automatiquement par rapport aux livraisons de la journée pour un transporteur.

```
Invoice createInvoice(Delivery delivery);
Invoice[] getInvoices();
bool updateInvoice(Invoice invoice);
bool deleteInvoice(Invoice invoice);
void generateInvoiceDocument(Invoice invoice);
```

3) DroneAvailabilityGetter

L'interface *DroneAvailabilityGetter* s'occupe de vérifier quels drones sont opérationnels lorsque notre client souhaite planifier une nouvelle livraison.

```
Drone[] getAvailableDrones(Date date);
```

4) DroneReviewer

Cette interface permet de gérer la maintenance des drones en vérifiant la charge de la batterie et le temps de vol restant. Elle contient la logique permettant au système de retourner le temps restant avant la fin de la charge par exemple. Si besoin, le statut du drone est changé pour indiquer l'indisponibilité du drone. Et revient à l'état disponible à la fin de la révision lorsqu'il revient dans son socle dédié.

```
Date getFlightTimeSinceLastRevision(Drone drone);
Date getRemainingBatteryFlightTime(Drone drone);
Date computeRemainingTimeBeforeBatteryIsFull(Drone drone);
void setDroneInCharge(Drone drone);
void putDroneInRevision(Drone drone);
bool setDroneAvailable(Drone drone);
```

5) DroneMonitor

Cette interface regroupe les règles d'accès au service dédié aux drones. Elle permet de récupérer les informations qui proviennent directement du drone (niveau de batterie, temps de vol) dans le but de les traiter. Le méthode *waitForDroneToComeBackWithoutParcel* est en fait utilisée pour attendre la réponse du drone depuis le service des drones (de l'api externe) car c'est le drone qui va notifier lui-même de son retour et de l'état du colis.

```
bool waitForDroneToComeBackWithoutParcel(Drone drone);
```

```
Date getFlightTimeSinceLastRevision(Drone drone);
Date getRemainingBatteryFlightTime(Drone drone);
Date getRemainingBattery(Drone drone);
void launchDrone(Drone drone);
```

6) DroneLauncher

Le *DroneLauncher* est uniquement composé d'une méthode pour initialiser le lancement du drone lorsque le manutentionnaire a mis en place le colis. Ensuite c'est le système qui gère lui-même le lancement pour qu'il décolle à l'heure prévue.

```
bool initializeDroneLaunching(Drone drone, double arrivalHour);
```

7) DeliveryFinalizer

Cette interface permet d'obtenir l'information sur le statut d'une livraison effectuée. La livraison est passée en paramètre ce qui permet au composant fournissant le service de modifier l'état de la livraison (*livrée* ou *échouée*).

```
void setDeliveryToDone(Delivery delivery);
void setDeliveryToFailed (Delivery delivery);
```

8) DeliveryScheduler

Cette interface permet d'interagir avec le planning, c'est-à-dire le visualiser et lui ajouter de nouvelles livraisons.

```
TimeSlot[] getPlanning();
bool scheduleDelivery(Date date, Delivery delivery);
```

scheduleDelivery permet d'ajouter une livraison à un créneau horaire et d'associer automatiquement un drone disponible à cet horaire à la livraison.

9) DeliveryOrganizer

DeliveryOrganizer permet de proposer les prochaines livraisons à effectuer selon le planning.

```
Delivery[] getNextDeliveries();
```

10) DeliveryInitializer

Cette interface propose des méthodes permettant la création des commandes, cela débute par le scannage d'un colis puis la création de la commande associée à ce colis. La méthode *initializeDelivery* permet de déclencher le processus de lancement de la livraison (c'est-à-dire le changement de statut de la livraison et le lancement du processus de lancement du drone).

```
Parcel scanParcel(String parcelId);
Delivery createDelivery(Parcel parcel);
bool initializeDelivery(Deliever delivery);
```

Conclusion

Afin d'obtenir cette architecture nous avons commencé par définir les fonctionnalités que notre système propose aux acteurs, nous avons donc obtenu le périmètre du système. Nous avons ensuite

défini le découpage en composant ainsi que les relations entre eux. Enfin nous avons défini les objets métiers représentant les données de notre architecture.