# Rapport Final ISA-DevOps

## Drone Delivery : Livrair

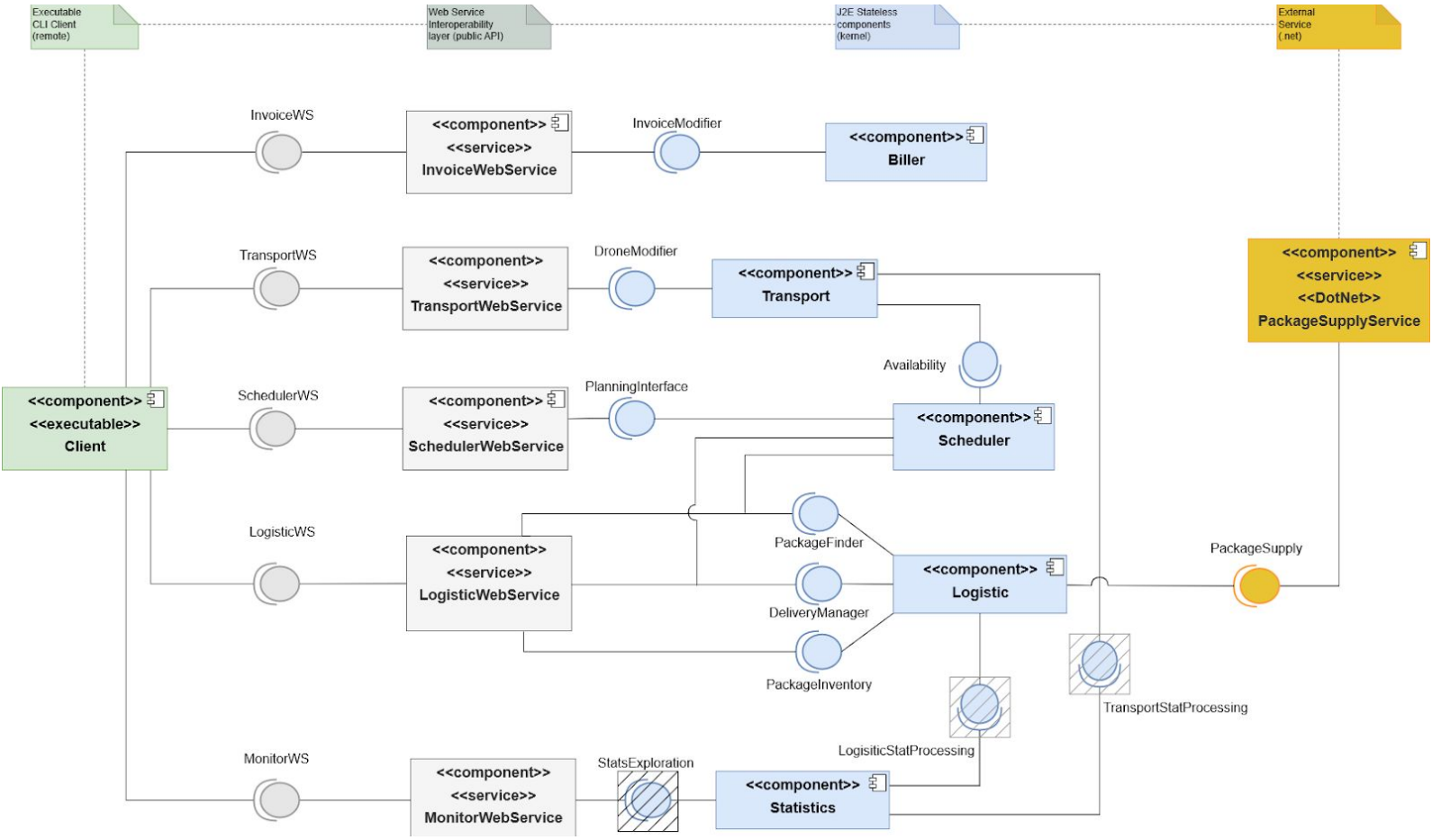
Université Nice Sophia Antipolis | POLYTECH NICE-SOPHIA

Membre de UNIVERSITÉ CÔTE D'AZUR

Pierre-Antoine Boulat - Sylvain Masia - Richard Peres - Maxime Leras - Anthony Choquard

# 1. Architecture

## 1.1 Diagramme de composants




Si l'on compare ce diagramme avec celui que l'on avait pour le MVP (*Celui-ci étant différent du diagramme dans architecture.pdf, le diagramme MVP est visible à l'annexe 1*), on s'aperçoit que l'architecture du projet a changé.

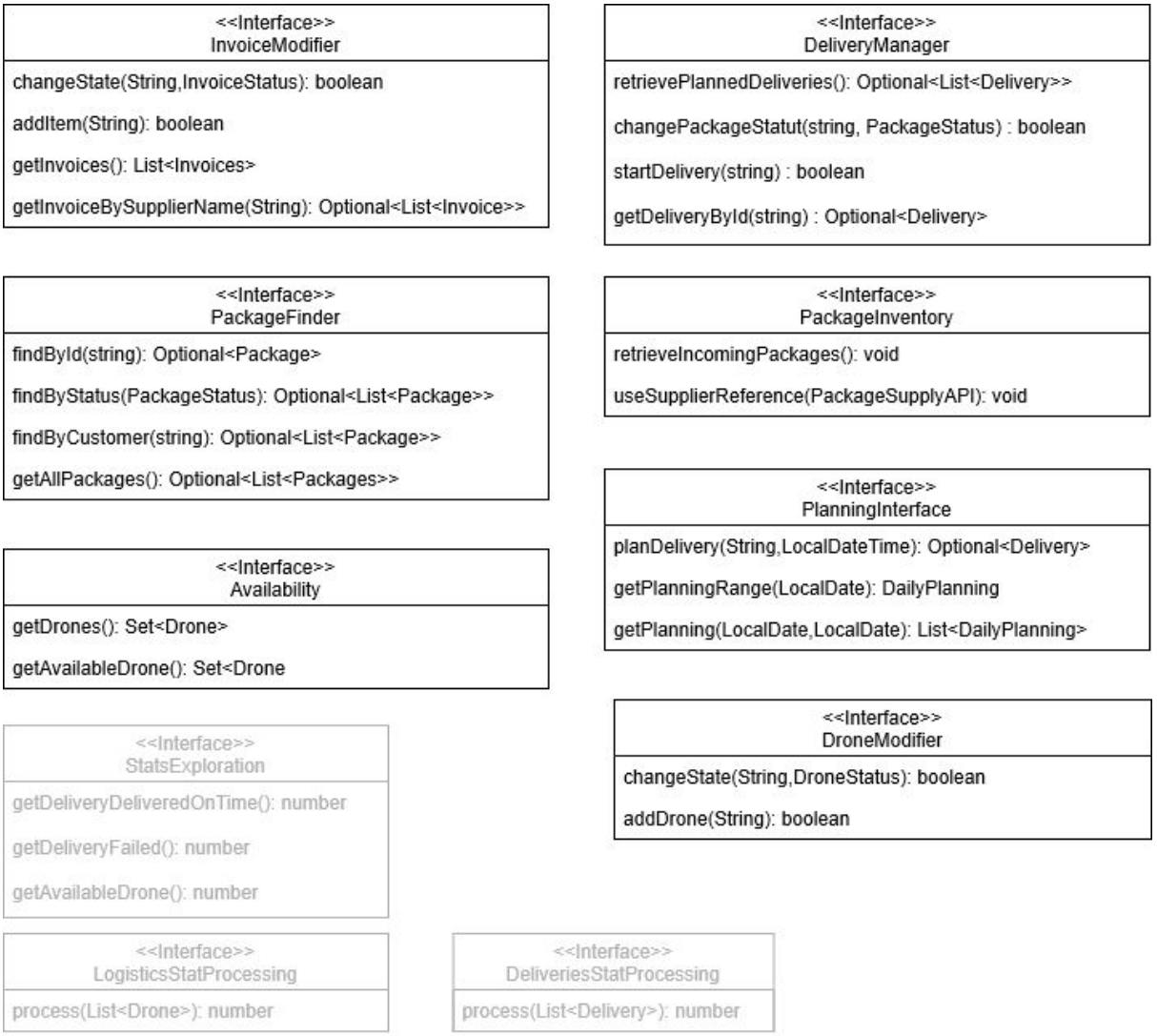
Tout d'abord, le composant Logistic qui gérait les drones a été renommé Transport et le composant Delivery qui gérait les packages et les livraisons a été renommé Logistic. *(Dans un souci de compréhension, les changements évoqués dans la suite de ce rapport prennent en compte ce changement, donc le composant Logistic correspond bien à celui qui s'occupe des packages/livraisons.)*

Pour faire évoluer notre mvp, notre Scheduler, qui à l'origine n'était qu'un composant interne dont l'accès se faisait exclusivement par le composant Logistic (à l'époque responsable de

la création des livraisons), a vu sa logique métier s'alourdir ce qui a impliqué qu'il devait dépendre du composant Logistic, alors même que le composant Logistic dépendait de lui (Scheduler). Pour éviter cette double dépendance, nous avons donc décidé d'exposer directement notre composant Scheduler et de lui donner la responsabilité de création des livraisons, de ce fait, le composant Logistic n'était plus dépendant du composant Scheduler.

Les interfaces rayées sur le diagramme correspondent à celles du composant Statistics qui est un composant qui n'a pas été implémenté.

## 1.2 Interfaces

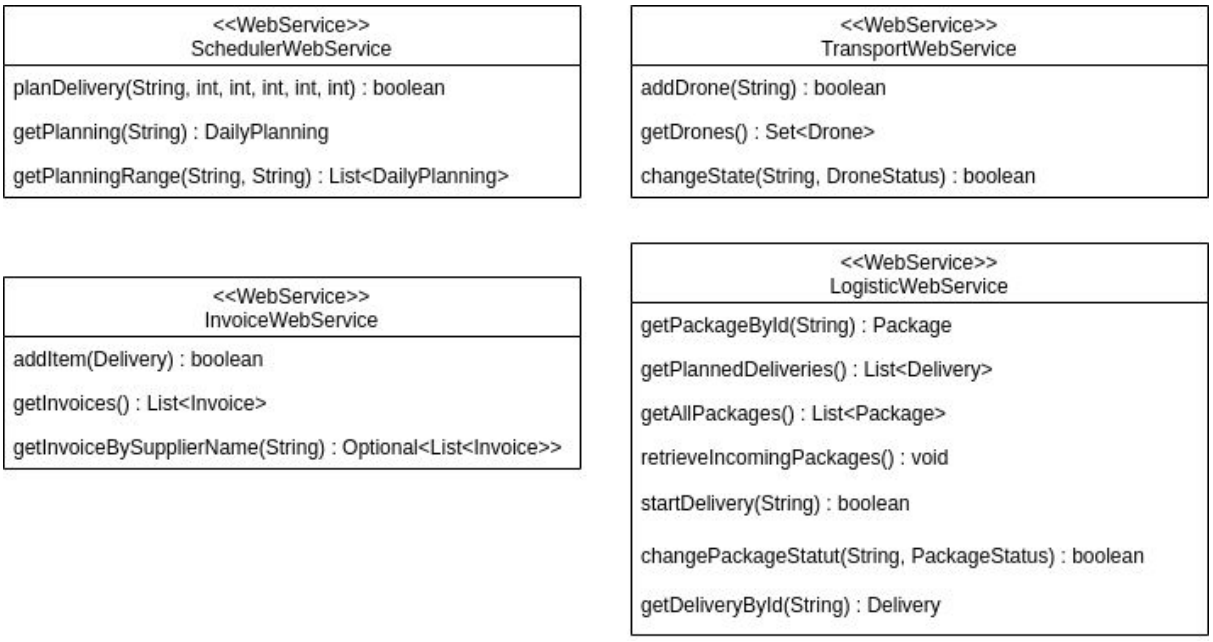



Pour ce qui est des interfaces qui connectent nos composants entre eux, Celles-ci ont beaucoup évolué du fait que nous n'avions pas une vision précise de leur fonctionnement lors du premier rapport. Une interface PlanningInterface a été créée pour notre composant Scheduler et l'interface DeliveryTracker qui était dans le composant Biller a

quant à elle été supprimée. De plus plusieurs méthodes ont également été ajoutées et celles déjà existantes ont vu leur type de retour et leurs arguments changés.
Nous avons également ajouté une interface PackageInventory, que l'on a exposé à travers le WS de Logistic, et qui permettrait dans une architecture complète de pouvoir mettre un cron job journalier sur la récupération des packages depuis le service externe.

Les interfaces grisées (correspondant au composant Statistics) n'ont pas été implémentées mais restent prévues par exemple pour l'ajout des statistiques. Leur pseudo implémentation n'a cependant pas été revue et est susceptible de changer.

D'un point de vue métier, les méthodes de nos interfaces sont trop similaires aux fonctions qui les exposent dans nos web services au niveau de leur prototype. Cette erreur de notre part implique que la correspondance "value <-> objet" se fait dans le Bean au lieu d'être fait dans le web service.



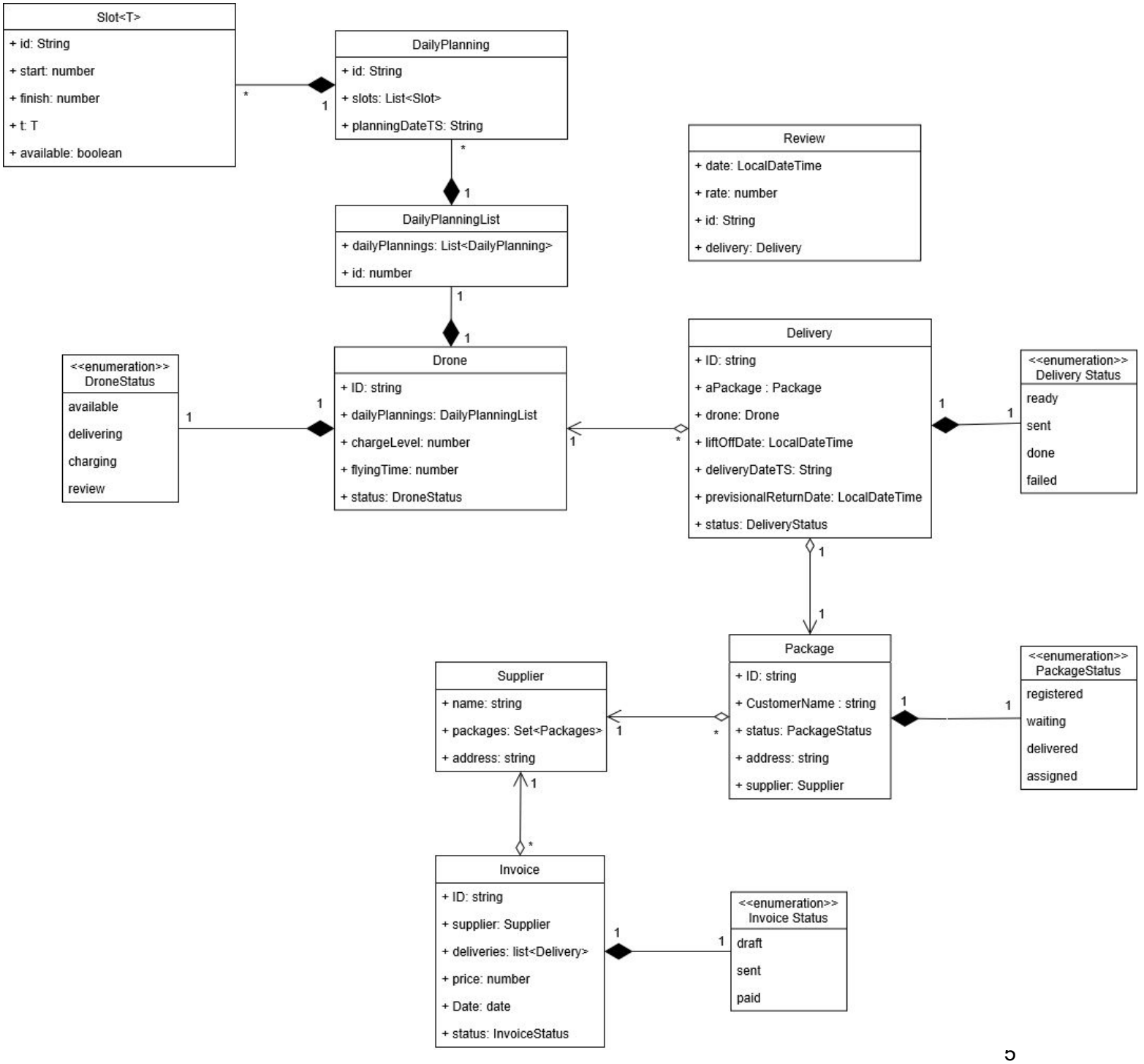

Les méthodes des WS qui vont avoir un impact sur un simple changement de statut prennent simplement l'id / la référence de l'objet en question. En effet, dans de nombreux cas, l'utilisateur n'aurait qu'à remplir/scanner la référence de l'objet qu'ils auraient devant lui. Par exemple lors de l'arrivée des colis, le manutentionnaire scanne l'id du package (qui serait actuellement en status Registered en BDD) et pourrait simplement de cette manière changer son statut à Waiting (en attente d'assignation à une livraison).

La méthode "AddDrone", permet d'ajouter un drone à la flotte, on part du principe que lorsqu'on ajoute un nouveau drone, son temps de vol est à 0 et qu'il est complètement chargé, donc il suffit de renseigner son id/ référence (qui est inscrite sur le drone).

La méthode “planDelivery” qui permet de planifier une livraison, prend en premier argument l’id du Package qui correspond à cette livraison, pour autant nous aurions pu prendre directement un objet package en argument étant donné que lors de la création d’une livraison, nous devrions déjà avoir accès à la liste des packages présents.

Notre client étant en Vue.js, il n’était pas possible de générer les stubs à partir du wsdl, il était donc plus simple d’utiliser les types primitifs pour requêter sur le backend.

## 1.3 Diagramme de classes



Si l'on compare notre diagramme de business object à celui que l'on avait pour la MVP les objets initialement définis restent les mêmes, seuls ont été ajoutés les *Slots*, les *DailyPlanning* ainsi que la *DailyPlanningList*.

Ces objets sont ceux propre à notre Scheduler, et permettent la gestion du planning. Cette partie métier est le coeur de l'application.

Pour implémenter cette logique de planning, nous avons créé objet Slot qui est une classe Template qui a une heure de début, une heure de fin, et un objet quelconque. Cet objet quelconque nous permet à la fois de pouvoir attribuer des deliveries, ou bien des tâches (recharge/maintenance).

Nous avons hésité entre deux approches:

- 1) Soit nous n'avions qu'un objet DailyPlanning dont l'id était la date du jour qui était global à tous les drones et qui avaient des slots propres à chaque drone (donc ajout d'un attribut drone au slot) et propres à chaque horaire.

- 2) Soit nous avions un objet DailyPlanningList, qui est propre à chaque Drone, et qui contient une liste de DailyPlanning propres à chaque jour et qui ont simplement des slots qui correspondent à chaque horaire.

L'avantage de la première étant que la création de livraison est beaucoup plus simple à gérer, cependant il est beaucoup plus dur de gérer la maintenance, la recharge et en général la disponibilité des drones à un instant précis. De plus l'ajout d'un drone impliquerait de devoir rajouter les slots correspondant à ce nouveau drone à tous les DailyPlanning à venir.

La seconde approche que nous avons conservée, rend la création des livraisons plus complexes (ainsi que leur persistance) mais permet de connaître la disponibilité du drone à un instant précis et également de mieux planifier/anticiper à l'avance les tâches de maintenances et autres.

# 2. Procédés

## 2.1 Persistance

Au niveau de la persistance, nous avons persisté tous nos business objects des composants implémentés, cette persistance a permis de rendre tous nos composants Stateless étant donné qu'ils n'avaient plus à stocker la liste des objets qui leurs sont propres.

Les objets Slots, DailyPlanning, DailyPlanningList, Delivery, Invoice se sont vus attribués des ID générés automatiquement car ceux-ci n'ont pas d'autres attributs qui pourraient être utilisés comme clé primaire, contrairement au Supplier (dont l'id est le nom), au Package (dont l'id est issu de la référence donné par le Supplier), et au Drone (dont on a choisis d'identifier par sa référence).

Les compositions fortes du diagramme de classes ont impliqué un cascading ALL de notre part.

Par défaut, nous avons eu tendance à fetch l'information lorsqu'il était possible de le faire. À l'exception du Slot dont le fetch de l'objet quelconque posait problème au marshaller (certainement une configuration à ajouter).

## 2.2 Composants Stateless

Pour ce projet nous avons choisi de développer des composants en utilisant une implémentation stateless. En effet nous avons souhaité limiter au maximum les données propres aux beans en évitant donc le Stateful. En s'assurant qu'aucun composant ne conserve d'état, cela permet de garantir plus facilement la fiabilité de l'exécution, celle-ci dépendant uniquement des paramètres en entrée sans pouvoir être impacté par l'état interne comme pourrait l'être un composant Stateful. De plus, il n'est à aucun moment question de conserver des informations sur le client ou de dialoguer avec celui-ci sur plusieurs appels consécutifs ce qui aurait pu nécessiter un composant Stateful. L'avantage d'une architecture entièrement orientée Stateless est l'évolutivité pour un potentiel passage à l'échelle par la simple multiplication des beans prêts à répondre aux requêtes.

## 2.3 Intercepteurs potentiels

Concernant les intercepteurs, notre idée était d'intercepter la validation d'une livraison pour implémenter les statistiques de façon très simple. Cela aurait permis de définir de façon unique un point d'entrée des statistiques dans le composant prévu à cet effet sans multiplier les interfaces entre les composants. De plus, les composants auraient eu la responsabilité d'incrémenter les statistiques dans leur code métier en faisant appel aux interfaces du composant statistiques, les rendant en quelque sortes responsables des

statistiques. Cette responsabilité est cédée aux intercepteurs permettant ainsi de maintenir des responsabilités limitées pour les composants.

### 2.4 Orientation message potentielle

L'orientation message est utile dans le cas où aucun retour n'est attendu par l'appelant. C'est le cas de notre composant Biller et de l'information de fin de livraison qui doit entraîner l'écriture d'une ligne de plus dans la facture correspondante. De plus la gestion de l'arrivée des messages sous forme de file d'attente éviterait une écriture concurrente dans une même facture. L'ordre de traitement des messages n'ayant pas d'impact, ce composant se prête très bien à l'implémentation d'un intergiciel à messages.

## 3. Conclusion

Notre implémentation du projet est à améliorer sur certains points, notamment au niveau des objets passés en paramètre des interfaces. Il nous faudrait aussi rajouter la logique pour tout ce qui va concerner l'anticipation des maintenances et des recharges. Pour autant, notre architecture métier devrait pouvoir supporter leurs ajouts.

L'architecture stateless au niveau des composants devrait permettre à l'application de faire un passage à l'échelle sans encombres, il reste également à implémenter les intercepteurs et les composants Orienté Message décrits précédemment.

Notre projet est fonctionnel sur 4 des 6 scénarios des personas à l'exception des statistiques qui ne sont pas implémentés et du scénario correspondant au gestionnaire et à la gestion des factures n'est pas complètement implémenté de bout en bout.

Merci à l'ensemble de l'équipe pédagogique pour leur encadrement et leur disponibilité durant ce projet.

# Annexes

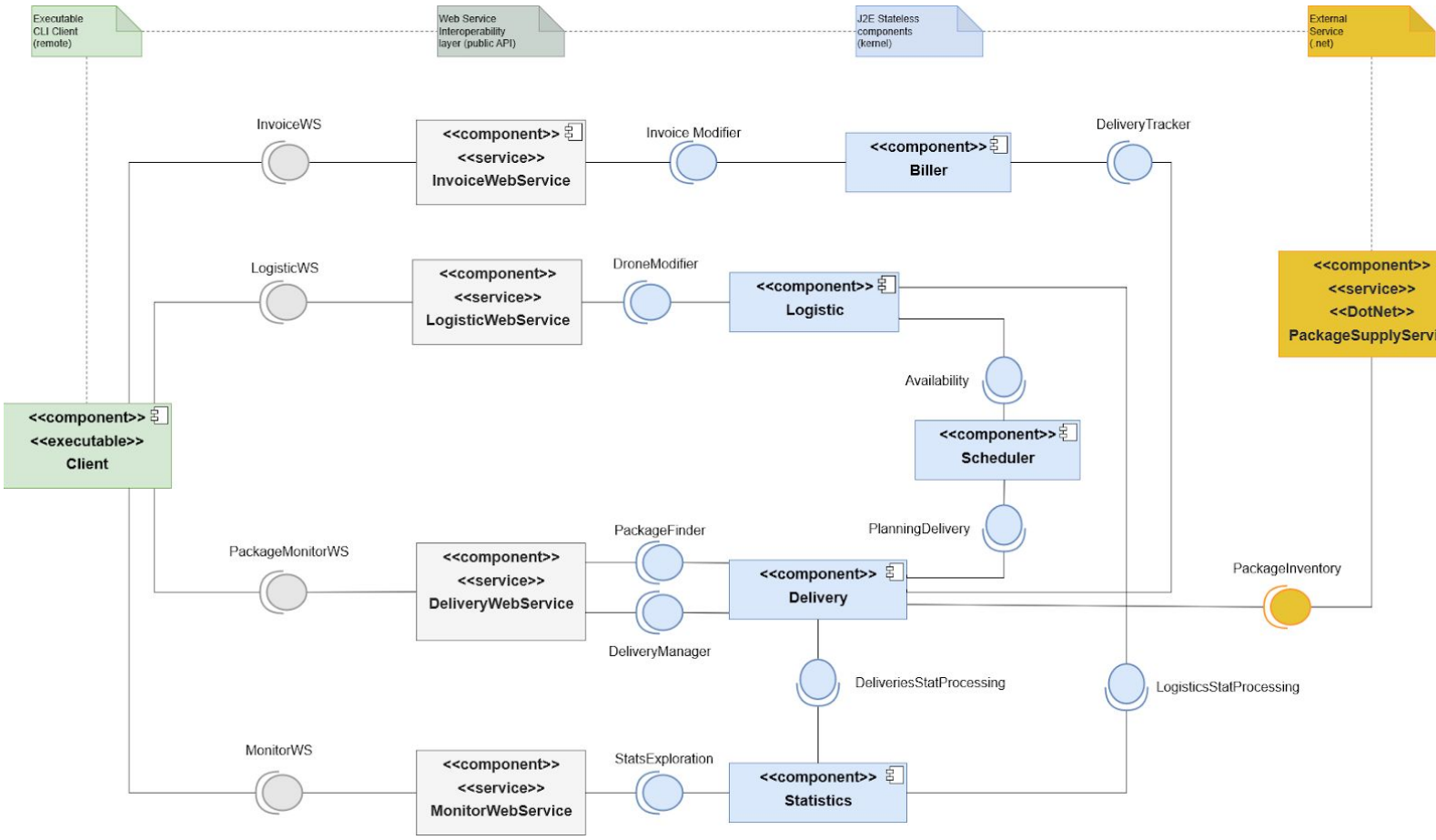



figure 1 : diagramme de component du MVP