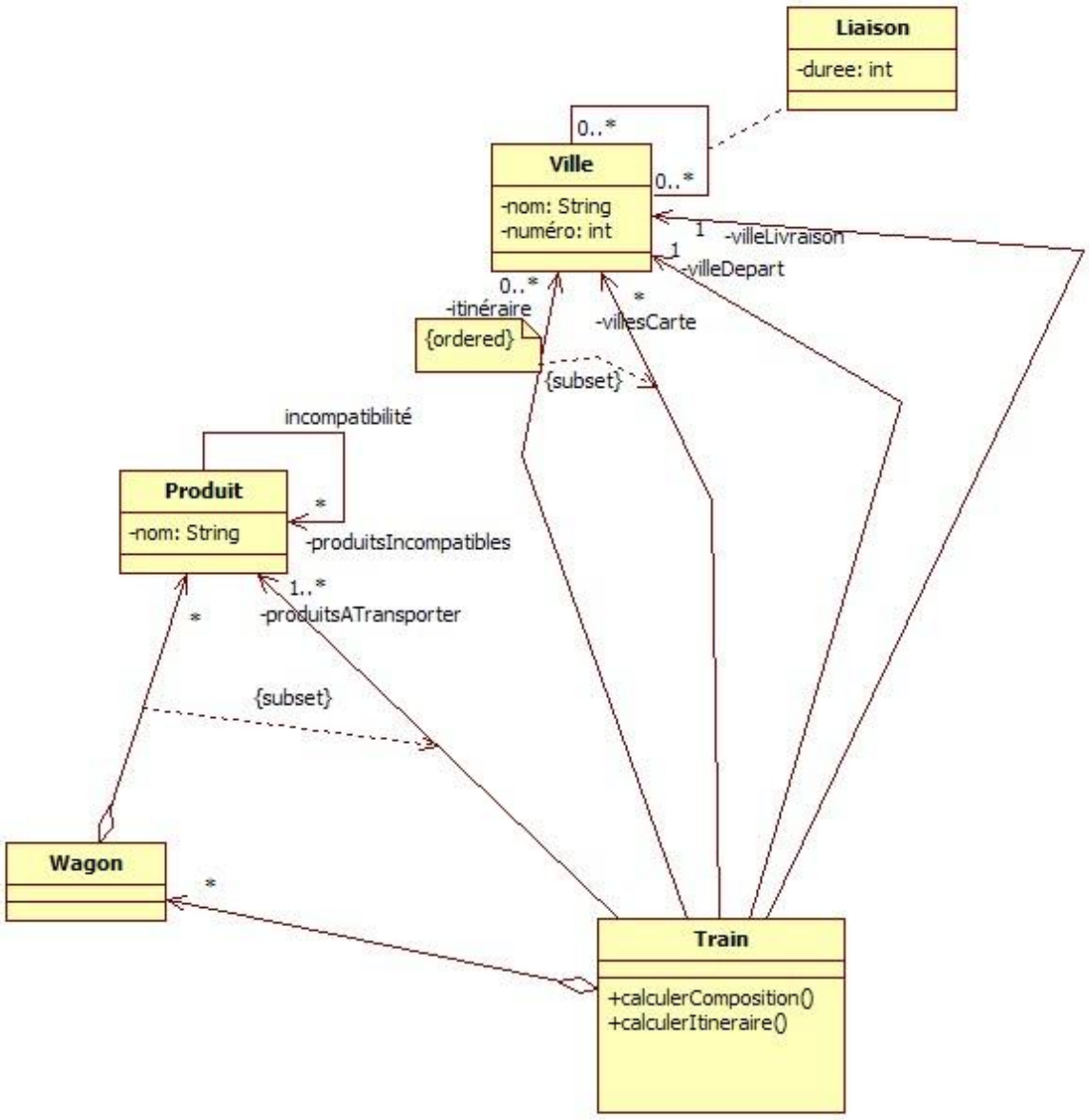
Projet Transport par Voie Ferrée

Eléments de corrigé UML du livrable 1

Il n'y a pas de solution type, surtout à ce stade de l'analyse. Il faut prendre en compte les différents éléments du cahier des charges et arriver à une solution cohérente. Ce premier diagramme de classe pourra être complété lors de l'analyse dynamique : ajout d'attributs, méthodes ; précision de navigabilité.

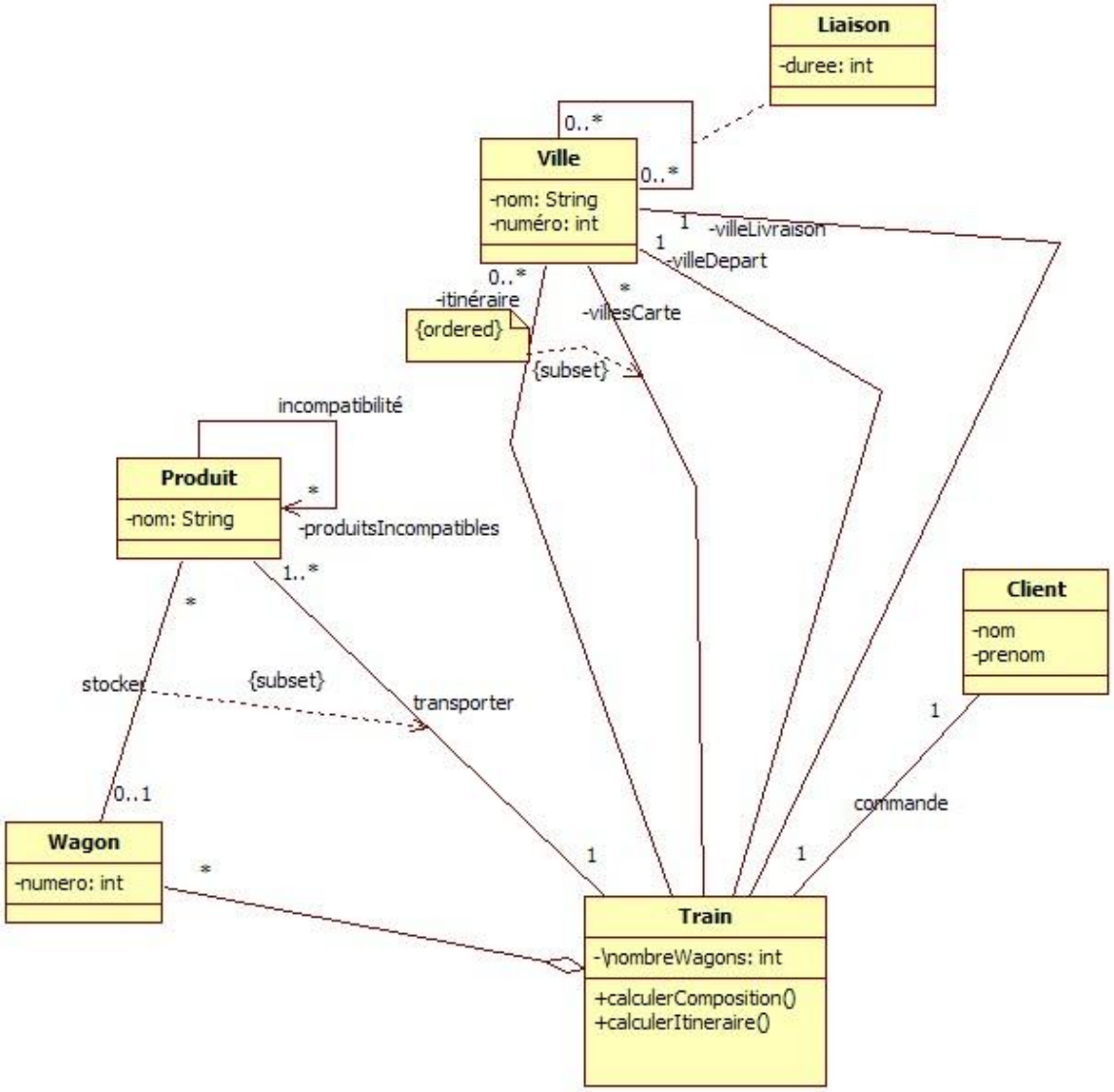
Une solution

Une première solution avec le strict minimum. La commande d'un client est constituée d'un train avec une liste de produits à transporter (avec leurs incompatibilités) et une ville livraison. Le logiciel du transporteur pourra calculer la composition de ce train en répartissant les produits à transporter dans ses wagons puis calculer un itinéraire depuis la ville de départ (ville du transporteur) vers la ville de destination parmi les villes de la carte.



Une deuxième solution

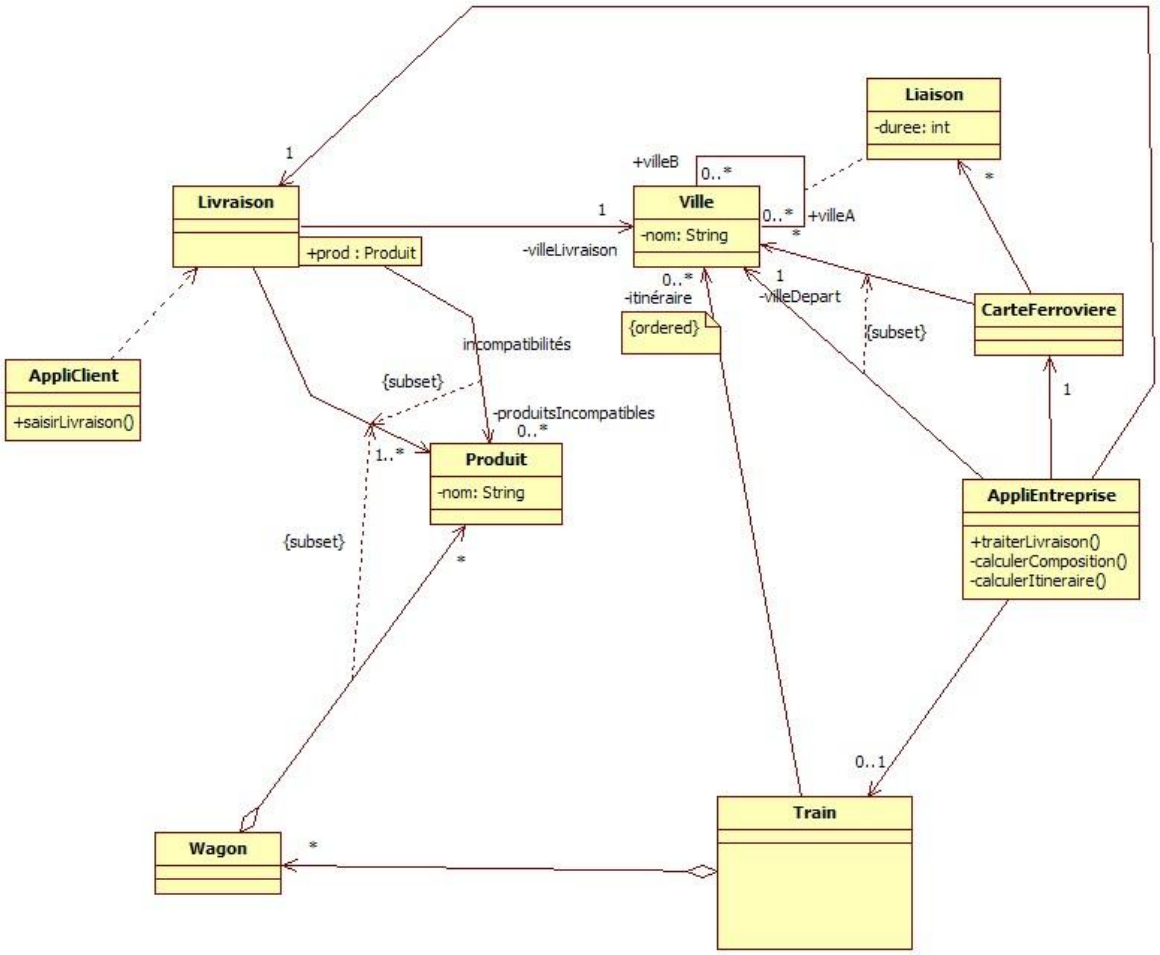
Une autre solution avec des associations nommées, sans navigabilité précisée. On ajoute quelques informations (n° de wagon et de produit). On peut accéder au nombre de wagons d'un train (qui pourra se calculer à partir du nombre de wagons associés à un train). Le principal ajout est le client qui passe la commande de transport. Cette information peut-être intéressante même toute la gestion client (facturation, historique, …) n'est pas à faire dans ce projet.



Une troisième solution

Une autre solution où les aspects de deux logiciels sont un peu plus séparés. Les deux applications sont représentées. La commande est représentée par la classe Livraison saisie par le client : elle possède une ville de livraison, une liste de produits à transporter et pour chaque produit la liste des produits incompatibles. De l'autre côté, l'application de l'entreprise de transport va gérer une commande pour éventuellement (si tout est possible) constituer un train avec un itinéraire parmi les villes de la carte et ses wagons où sont répartis les produits de la

commande de livraison.



Un peu plus de précision avec OCL

```
-- on ne connait pas la carte à la saisie de la livraison
context Livraison::villeLivraison : Ville
inv : villeA->empty() and villeB->empty()

-- on traite une commande si la ville de livraison est dans la carte
context AppliEntreprise::traiterLivraison()
pre: carteFerroviere.ville->exists(v:Ville | v.nom = livraison.villeLivraison.nom)

-- carte bien construite
-- 1 ville unique
-- 2 villes toujours reliées (TODO)
context CarteFerroviere
inv : ville->forAll(v1,v2|v1.nom=v2.nom implies v1= v2)
inv : ville->forAll(v1,v2| ...???...)

-- incompatibilités symétriques
context Livraison
inv: produits->forAll(p1,p2 : Produit |  produitIncompatibles[p2]->includes(p1)
  implies produitIncompatibles[p1]->includes(p2))

-- cohérence du résultat (composition)
context AppliEntreprise
inv : train->notEmpty() implies (
-- 1. pas de produits incompatibles dans le même wagon
train.wagon->forAll(w|w->produit->forAll(p1,p2 |
    not(livraison.produitsIcompatibles[p1]->includes(p2)))
-- 2. tous les produits dans le train
```

```
and livraison.produit->forAll(p|train.wagon.produit->includes(p))
-- 3. un produit dans un seul wagons
and train.wagon->forAll(w1,w2|
not(w1=w2) implies (w1.produit intersect w2.produit)->isEmpty())
)
--  NB : on peut aussi exprimer cela comme postcondition de calculerComposition()

-- cohérence du résultat (itinéraire)

-- un itinéraire ne passe pas 2 fois par la même ville
context Train
inv : itinéraire->forAll(v1,v2|v1=v2
   implies itinéraire->indexOf(v1)=itinéraire->indexOf(v2))

-- l'itinéraire permet d'aller de la ville de départ à la ville de livraison
context AppliEntreprise
inv : train->notEmpty() implies (
  train.itinéraire->forAll(etape : Ville |
    let i = train.itinéraire->indexOf(etape) in
    if i < train.itinéraire->size() then
          let etapeSuivante = train.itinéraire->at(i+1) in
            carteFerroviere.liaison->exists(l : liaison |
                ( l.villeA = etape and l.villeB = etapeSuivante)
              or ( l.villeA = etapeSuivante and l.villeB = etape))
    end if)
and train.itinéraire->at(0) = villeDépart
and train.itinéraire->at(train.itinéraire->size()-1).nom = livraison.villeLivraison.nom
```