Rédaction des contraintes en OCL:

Nous utiliserons le langage OCL pour modéliser les différentes contraintes que sont les pré et post conditions, ainsi que les invariants du système.

Pour l'historique:

Invariants:

context h:Historique **inv**:
h.vehicules.size() >= 0

context h:Historique **inv**:
h.parametres.size() >= 0

Justifications :

L’historique contient toute une liste de véhicules enregistrés par l’utilisateur, ainsi que les parametres favoris. Leur taille ne peut donc etre négatives, auquel cas cela entrainerait un bug. Sinon, les historiques peuvent etre vides, donc leur taille peut etre nulle. Mais comme on peut rajouter des éléments, leur taille peut donc etre strictement positive. Donc la taille de ces historiques est supérieur ou égale à 0.

Pre et post conditions:

context Historique::enregistrerVehicule(Vehicule v): Boolean

post : **if**(this.vehicules.contains(v)) **implies** result = false
else this.vehicules.add(v) and result = true

context Historique::enregistrerParametres(Parametre p): Boolean

post: **if**(this.parametres.contains(p)) **imples** result = false
else this.parametres.add(p) and result = true

Justifications:

Pour enregistrer un véhicule ou une liste de paramètres, il faut éviter qu’ils soient déjà inclus dans les historiques. Donc on enregistre un nouveau véhicule ou liste de paramètres si la liste ne les contient pas déjà et on renvoie VRAI, pour notifier à l’utilisateur que l’enregistrement a été effectué. Sinon, on renvoie FAUX et on notifie l’utilisateur qu’on ne peut pas enregistrer un véhicule déjà enregistré à nouveau, il en est de meme avec les listes de parametres.

Pour le type Parametres:

Invariants:

context p:parametres **inv**:
p.prixMax >= 0

context p:parametres **inv**:
p.volumeCO2 > 0

context p:parametres **inv**:
(**if**(p.distancePlusCourte = true) **implies** p.rapideDuree = false) **or**
(**if**(p.rapideDuree = true) **implies** p.distancePlusCourte = false)

Justifications:

L'utilisateur peut choisir un trajet gratuit, donc le prix max peut etre égal à 0. Mais il peut également choisir de passer par des routes payantes suivant un tarif strictement positif, donc le prixMax peut etre supérieur ou égal à 0. Il est évident que le prix à payer ne peut pas etre négatif.

Concernant l'émission de CO2 maximale, elle ne peut pas etre négative, car un trajet émettra forcément du CO2, au lieu d'en absorber. Cette valeur ne peut donc pas etre nulle. C'est pour cela que cet attribut a forcément une valeur strictemement supérieure à 0.

L'utilisateur peut choisir ou le chemin le plus rapide ou le plus court en termes de distance, mais il est impossible de satisfaire ces 2 critères en meme temps. C'est pour cela qu'on ne peut avoir rapideDureee et distancePlusCourte vraies en meme temps.

Pour le type route :

Invariants :

context r:Route **inv**:
r.villeDepart <> r.villeArrivee

context r:Route **inv**:
r.vitesse > 0

context r:Route **inv**:
r.type = Type de route::nationale or r.type = Type de route::autoroute

context r: Route **inv**:
r.longueur > 0

context r:Route **inv**:
r.nom.size()>0

Justifications:

Les villes de départ et d'arrivée sont forcément différentes, car une route relie 2 villes différentes et il n'existe aucune route reliant une ville à elle-meme.

La limitation d'une route ne peut etre que positive, car les valeurs sont strictement positives. Il est impossible d'avoir des vitessses négatives.

Une route a forcément un type, qui est ou national, ou l'autoroute. Il n'existe pas d'autres type de route dans la base de données.

La longueur d'une route est forcément positive, il est impossible que cette valeur soit négative ou meme nulle.

Chaque route a forcément un nom, il n'existe pas de route qui n'ont aucun nom.

Pour le type trajet:

Invariants:

context t:trajet **inv**:
t.duree > 0

context t:trajet **inv**:
t.distance > 0

Justifications:

La durée d'un trajet ne peut pas etre négative, c'est impossible. La duree ne peut etre nulle, sauf si on est déjà arrivé à destination. C'est donc pour cela que la durée est strictement positive.

Comme pour la route, la distance d'un trajet ne peut pas etre négative ou nulle, cela est impossible, donc la disance du trajet est strictement positive.

Pre et post conditions:

context calculerDistance(routes Route) : Integer

pre: routes.size() > 0

post: result > 0

context calculerTemps(routes Route) : Integer

pre: routes.size() > 0

post: result > 0

context calculerPrix(routes Route) : Integer

pre: routes.size() > 0

post: result >= 0

context calculerVolumeCO2(routes Route, vehicule Vehicule) : Integer

pre: routes.size() > 0 and vehicule <> null

post: result > 0

context notifierParametresNonRespectes()

context notifierRouler2Heures()

Justifications:

Pour calculer la distance, il faut faire une somme sur la distance des routes à parcourir, c'est pour cela que le tableau contenant les routes ne doit pas etre vide. Le résultat renvoyé sera forcément positif, puisqu'on ne peut pas parcourir de distance négative ou nulle.

Pour calculer le temps, il faut faire une somme sur le temps de parcour des routes à parcourir, en fonction de leur limitation de vitesse, puisque l'on suppose que l'utilisateur roulera toujours à la vitesse maximale. C'est pour cela que comme pour le calcul de la distance, le résultat renvoyé ne peut ni etre négatif, ni etre nul.

Pour calculer le prix, on effectue une somme des prix des routes à parcourir, c'est pour cela que le tableau des routes ne doit pas etre vide. On ne peut parcourir de routes à prix négatif, mais on peut parcourir des routes gratuites. Le résultat renvoyé sera donc forcément positif ou nul.

Pour calculer le volume de CO2 expulsé, on a besoin de la distance, donc d'un tableau de route et d'un véhicule pour appliquer les formules de calcul. Le résultat renvoyé sera strictement positif, car il est impossible de ne pas émettre de CO2 lors d'un trajet.

Pour les fonctions notifierParametresNonRespectes et notifierRoulerHeures, nous n'avons pas établi de pré et post conditions. Il s'agit de fonctions void, qui ne renvoient aucun résultat, donc pas de post conditions. Nous n'avons pas trouvé de pré conditions.

Pour le type Utilisateur:

Invariants:

context u:utilisateur **inv**:
u.pseudo.size() > 0

context u:utilisateur **inv**:
u.password.size() > 0

context u:utilisateur **inv**:
u.historique.size() >= 0

Justifications:

Un pseudo ne pouvant etre un caractère vide, la taille du pseudo est forcément strictement positive.

Un mot de passe ne pouvant etre un caractère vide, la taille du mot de passe est également strictement positive.

L'historique d'un uilisateur ne peut pas etre de taille négative, car il n'existe pas de tableau de taille négative. Par contre, l'historique peut etre vide ou contenir des données, donc sa taille peut etre positive ou etre nulle.

Pre et post conditions :

context Utilisateur::saisirVehicule(nom: String, c: Gabarit, c: Carburant): Vehicule

pre: (c.gabarit = Gabarit::citadine or
c.gabarit = Gabarit::monospace or
c.gabarit = Gabarit::smart or
c.gabarit = Gabarit::2 roues or
c.gabarit = Gabarit::camionnette or
c.gabarit = Gabarit::camion or
c.gabarit = Gabarit::tout-terrain) and
(c.carburant = Carburant::essence or
c.carburant = Carburant::diesel) and
nom.size() > 0

post: **if** not(Utilisateur.historique.vehicules.includes(result)) **implies**
Utilisateur.historique::enregistrerVehicule() = true and
result = v
else Utilisateur.historique::enregistrerVehicule() = false and result = null
endif

context Utilisateur::saisirParametres(prixMax : Integer, volumeCO2: Integer,
distancePlusCourte : Boolean,
rapideDuree: Boolean) : Parametres

pre: (prixMax > 0 and volumeCO2 > 0) and
((distancePlusCourte = true and rapideDuree = false) or
(distancePlusCourte = false and rapideDuree = true))

post: **if** not(Utilisateur.historique.parametres.includes(result)) implies
Utilisateur.historique::enregistrerParametres() = true and
result = p
else Utilisateur.historique::enregistrerParametres() = false and result = null
endif

context Utilisateur::consulterHistorique(): Historique

post: **if** Utilisateur.historique.size() > 0 **implies** result = Utilisateur.historique
else result = null

Justifications:

Un utilisateur peut saisir un véhicule, à condition de respecter toutes les conditions, à savoir que le gabarit, le type et le nom doivent etre renseignés. Si l'historique ne contient pas ce véhicule, alors le système l'enregistre, sinon non.

Il en est de meme pour les parametres, le raisonement est analogue à la saisie de véhicule.

L'utilisateur peut consulter son historique si et seulement si l'historique n'est pas vide. Sinon, on notifie l'utilisateur que sont historique est vide.

Pour le type véhicule:

Invariants:

context v:vehicule **inv**:
v.gabarit = Gabarit::citadine or
v.gabarit = Gabarit::monospace or
v.gabarit = Gabarit::smart or
v.gabarit = Gabarit::2 roues or
v.gabarit = Gabarit::camionnette or
v.gabarit = Gabarit::camion or
v.gabarit = Gabarit::tout-terrain

context v:vehicule **inv**:
v.carburant = Carburant::essence or v.carburant = Carburant::diesel

context v:vehicule **inv**:
v.nom = nom.size() > 0

Justifications:

Le gabarit d'un véhicule est forcément choisit parmis la liste de gabarits proposés. On ne peut donc pas avoir d'autres types de véhicules.

Le type de carburant est ou "essence" ou "diesel" puisqu'il n'y a que cela comme type de carburant.

Le nom d'un véhicule à renseigner étant forcément plus long qu'une liste vide, la taille de son nom est donc strictement positive.

Pour le type ville:

Invariants:

context v:ville **inv**:
v.nom.size()>0

context v:ville **inv**:
v.taille = Taille de la ville::grande or v.taille = Taille de la ville::moyenne or
v.taille = Taille de la vaill::petite

context v:ville **inv**:
v.longitude > 4 and v.longitude > 8

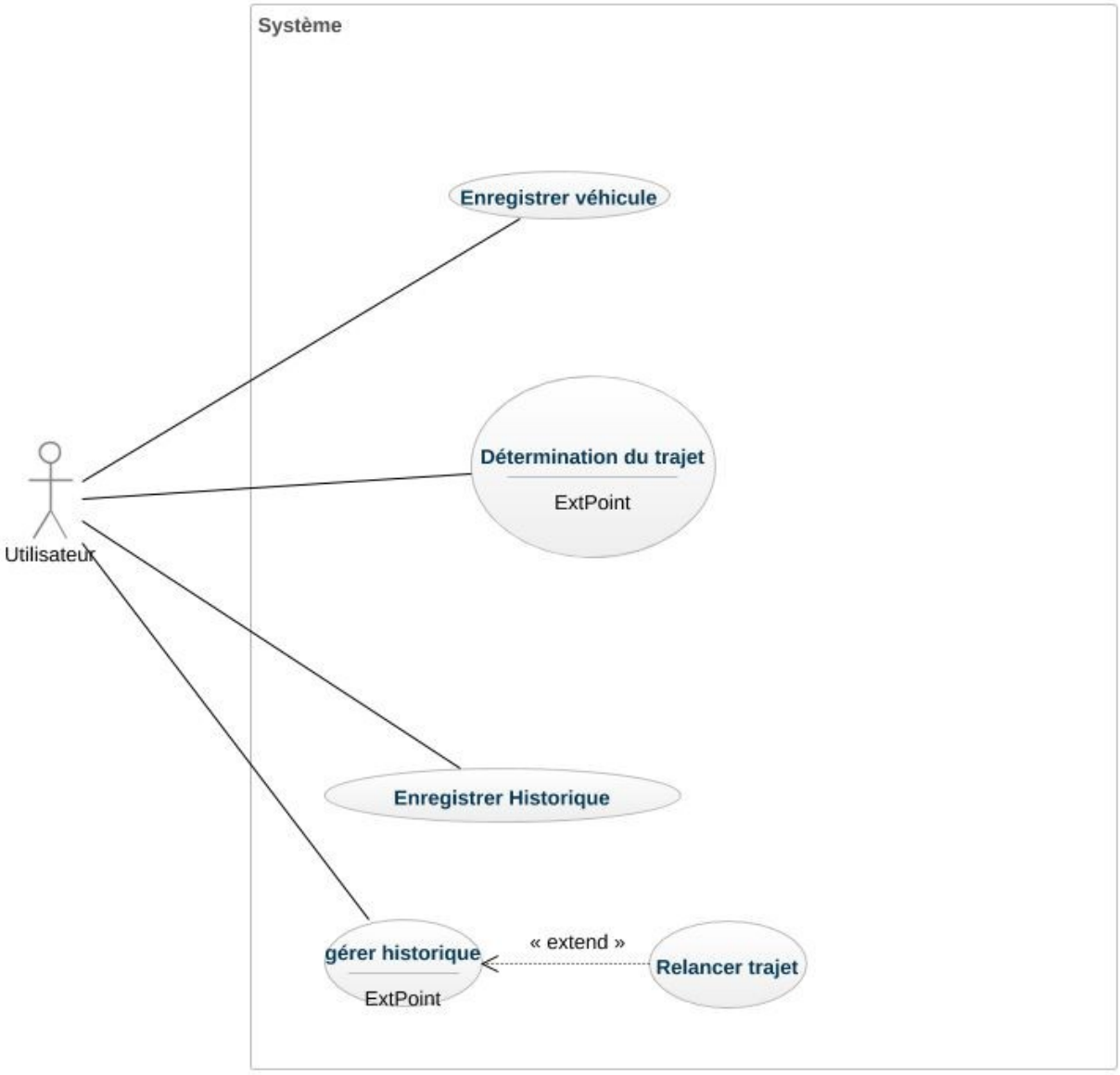
context v:ville **inv**:
v.latitude > 42 and v.latitude < 51

Justifications:

Toute les villes dans la base de données ont un nom, il est donc impossible de rencontrer des villes sans nom.

La taille d'une ville ne peut etre que "petite", "moyenne" ou "grande", donc on impose un choix parmis ces trois.

Les valeurs des latitudes et longitudes maximales ont été tirées de wikipedia et correspondent aux extrémités Nord, Sud, Est et Ouest des coordonées de la France métropolitaine, puisque la base de données ne comprend que des villes situées en France métropolitaine.


Système
Enregistrer véhicule
Détermination du trajet
ExtPoint
Utilisateur
Enregistrer Historique
gérer historique
ExtPoint
« extend »
Relancer trajet

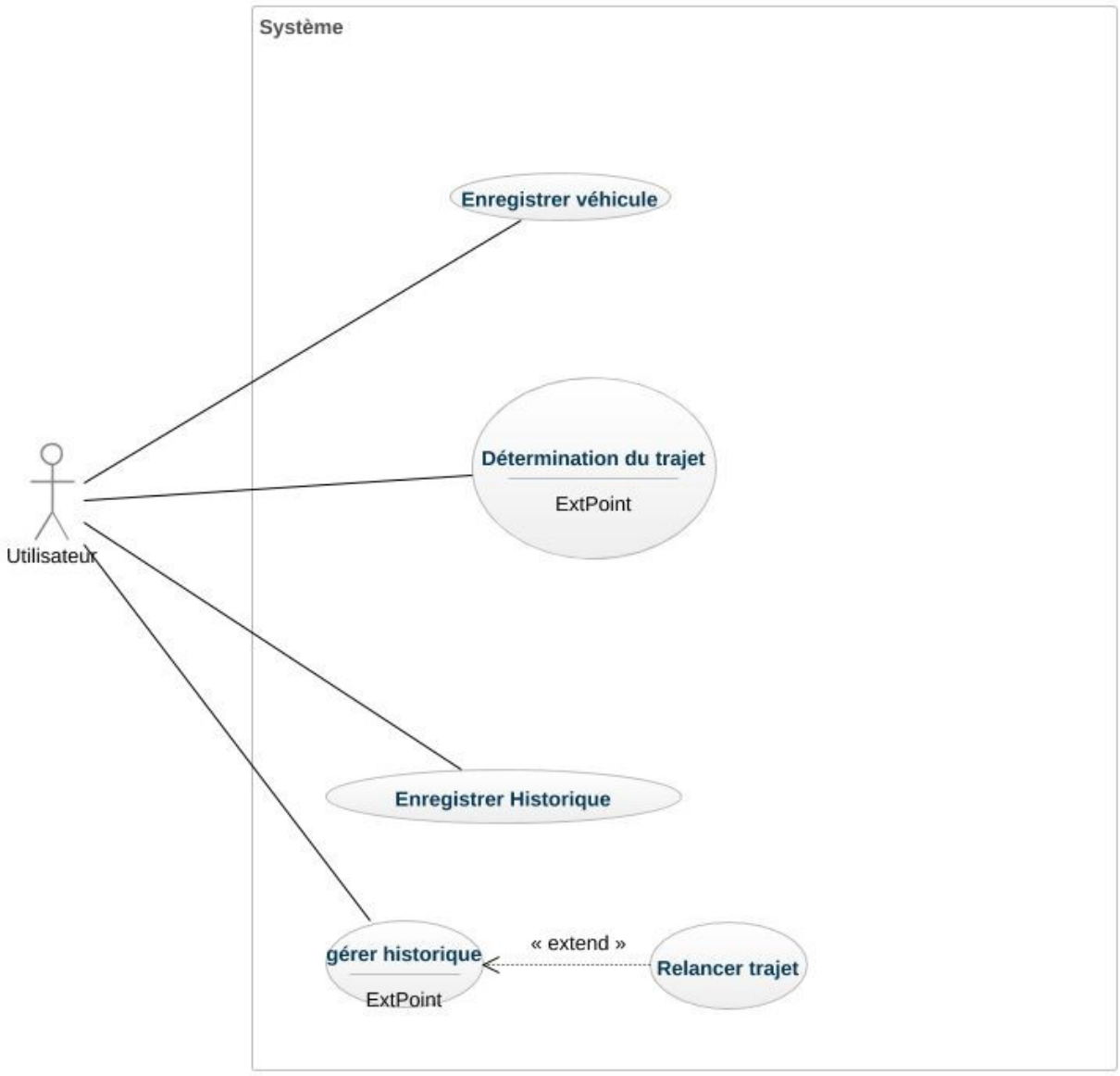


Diagramme de cas d’utilisations. Ici, un seul acteur qui est l’utilisateur, puisque le système gère les fonctions et la base de données. A noter que seules les fonctions les plus pertinentes sont ici montrées, il est inutile d’indiquer les fonctions de saisies de villes ou de paramètres.

Voici la description textuelle des différentes opérations énumérées dans le diagramme des cas d'utilisation.

Nom : Détermination de la validité des parametres
Description : Le système determine si les paramètres du trajet sont valides ou non.
Acteurs : L'utilisateur.
Contexte : Le système doit déterminer si les parametres du trajet sont valides pour déterminer si le trajet choisi est valide ou non.
Entrées et préconditions : Les paramètres non null.
Sorties et postconditions : Renvoie vrai si les parametres sont valides, faux sinon.
Extension : Nom : Destination Valide(Ville d'arrivée)
point de rattachement : Détermine si la ville de destination est valide, bien différente de la ville de départ
Etapes principales de l'extension : 1) Vérifie que la ville de destination est enregistré dans la base de données et qu'elle est différente de la ville de départ.

Scénario principal :

1) L'utilisateur choisit et saisi les paramètres.
2) Le système détermine alors le meileur trajet possible.
3) Le système affiche alors le trajet, ainsi que tous les calculs qui ont été effectué sur le temps de parcours, la distance, le volume d'essence consommé et le prix à payer.

Scénario d'erreur :

1)a) La ville de départ et d'arrivée sont identiques : Pas de trajet possible, car ces 2 villes doivent etre différentes. Reprise en 1.

3)b) Le prix dépasse celui imposé par l'utilisateur. Reprise en 2 pour recalculer un trajet qui respecte au mieux la contrainte. Si ce n'est pas possible, notifie l'utilisateur.

Nom : Détermination du trajet
Description : Le système determine le meilleur trajet en fonction des paramètres.
Acteurs : L'utilisateur.
Contexte : Le système essaie de choisir le trajet correspondant au mieux aux contraintes imposées par l'utilisateur, en fonction des paramètres saisis.
Entrées et préconditions : Ville de départ, ville d'arrivée, les paramètres.
Sorties et postconditions : Pas de sortie, affiche le trajet, avec les villes à parcourir, pareillement pour les routes, ainsi qu'une estimation du temps de trajet, de la distance à parcourir et du volume d'essence consommé et de CO2 expulsé.
Extension : Nom : Calcul (Distance, temps, cO2, argent….)
point de rattachement : Effectue les calculs pour et du trajet
Etapes principales de l'extension : 1) Effectue tous les calculs.

Scénario principal :

1) L'utilisateur choisit et saisi les paramètres.
2) Le système détermine alors le meileur trajet possible.
3) Le système affiche alors le trajet, ainsi que tous les calculs qui ont été effectué sur le temps de parcours, la distance, le volume d'essence consommé et le prix à payer.

Scénario d'erreur :
1)a) La ville de départ et d'arrivée sont identiques : Pas de trajet possible, car ces 2 villes doivent etre différentes. Reprise en 1.

3)b) Le prix dépasse celui imposé par l'utilisateur. Reprise en 2 pour recalculer un trajet qui respecte au mieux la contrainte. Si ce n'est pas possible, notifie l'utilisateur.

Nom : enregistrer historique
Description : Le système enregistre le trajet effectué, avec les paramètres, le véhicule et le trajet.
Acteurs : L'utilisateur.
Contexte : L'utilisateur, une fois arrivé à destination, autorise ou non le système a enregistrer le trajet effectué.
Entrées et préconditions : Trajet, véhicule, paramètres. Il faut que le trajet ait été effectué correctement.
Sorties et postconditions : Pas de sortie, enregistre juste les données dans l'historique si autorisé.

Scénario principal :

1) L'utilisateur confirme au système qu'il est arrivé à destination.
2) Le système demande l'autorisation d'enregistrer ou non ce trajet dans l'historique.
3) Si oui, le système enregistre effectivement dans l'historique. Ne fait rien sinon.
4) Sortie du navigateur.

Scénario d'erreur :
3)a) Enregistre si l'utilisateur a refusé. Dans ce cas, supprime, reprise en 4).

3)b) Le système n'enregistre pas, alors que l'utilisateur a voulu que ce soit enregistré. Le système enregistre alors dans l'historique. Reprise en 4)

Nom : Enregistrer véhicule
Description : L’utilasateur enregistre un véhicule
Acteurs : L’utilisateur
Contexte : L’utilisateur, enregistré ou non, a cliqué sur cette fonctionalité
Entrées et préconditions : L’utilasateur a bien renseigné le gabarit et le type d’essence du véhicule. Ce véhicule ne doit pas avoir déjà été enregistré.
Sorties et postconditions : Nouvel objet “Véhicule” créé, et bien enregistré dans les paramètres. Il faut que le véhicule soit bien intégré dans l’historique.

Scénario principal :

1) L’utilisateur rentre le gabarit du véhicule.
2) L’utilisateur saisit le type de carburant.
3) Le système vérifie que le véhicule n’est pas déjà enregistré dans l’historique.
4) Le système enregistre le véhicule saisit par l’utilisateur.
5) Le système permet la suite des choix du trajet.

Scénario d’erreur (a):
3)a) Si le véhicule a déjà été enregistré, alors ne l’enregistre pas et le signale à l’utilisateur, pour qu’il le choissise directement dans l’historique.
Reprise à 5

Nom : Consulter l’historique
Description : L’utilisateur consulte son historique de trajets, de véhicules et de paramètres
Acteurs : L’utilisateur
Contexte : L’utilisateur vérifie tous les trajets effectués avec quel véhicule en fonction de quels paramètres.
Entrées et préconditions : Aucune entrée, il faut qu’il y ait un historique déjà crée.
Sorties et postconditions : Pas de sortie, affiche juste les données.
Extension: Nom: Relancer Trajet(trajet)
Point de rattachement: Relance un trajet à partir de l’historique.
Etapes principales de l’extension: 1) L’utilisateur choisit un trajet dans l’historique.
2) Le programme relance le trajet sélectionné.

Scénario principal :

1) L’utilisateur fait appel à la foncitonnalité.
2) L’historique s’affiche, du plus récent au plus ancien.
3) Choix de l’affichage dynamique : Du plus récent au plus ancien, inversement, en fonction des paramètres ou alors du trajet.
4) Fermeture de l’historique.

Scénario d’erreur :

1)a) Il n’y a pas d’historique, donc rien ne s’affiche, l’utilisateur est invité à faire au moins un trajet. Reprise en 4.

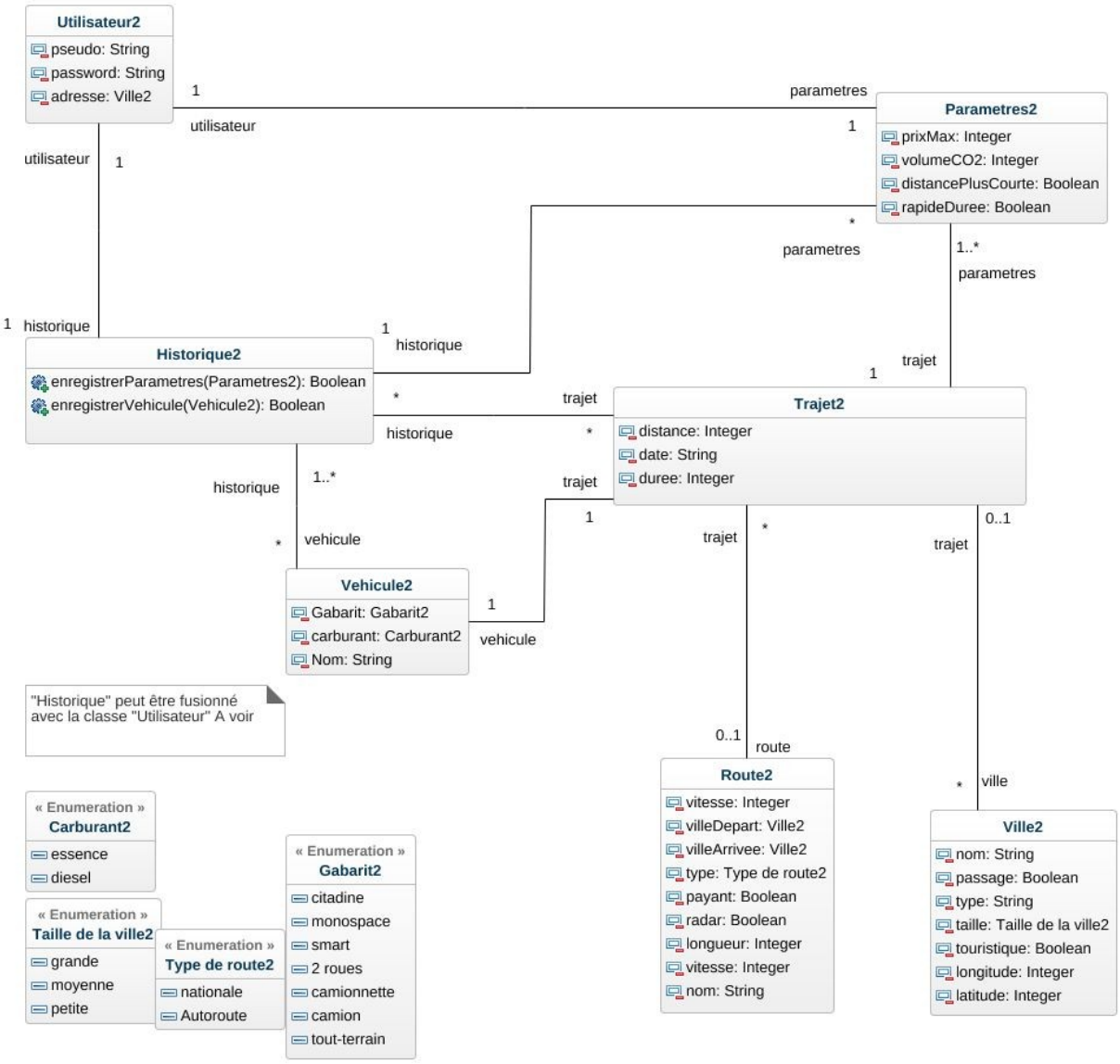


Diagramme de classes dans le cas de l’analyse. Ici, seules les classes et les attributs, ainsi que les liens entre les classes sont représentés. Nous avons choisis des types énumérés pour les types de carburant, taille de la ville, les types de routes ainsi que les gabarits des véhicules pour éviter toute mauvaise saisie de paramètres, plutot que d’utiliser un type String. A noter que ce diagramme peut etre modifié, puisque ce diagramme ne prend pas en compte les troncons, ce qui posera problème pour la lecture de la base de données.

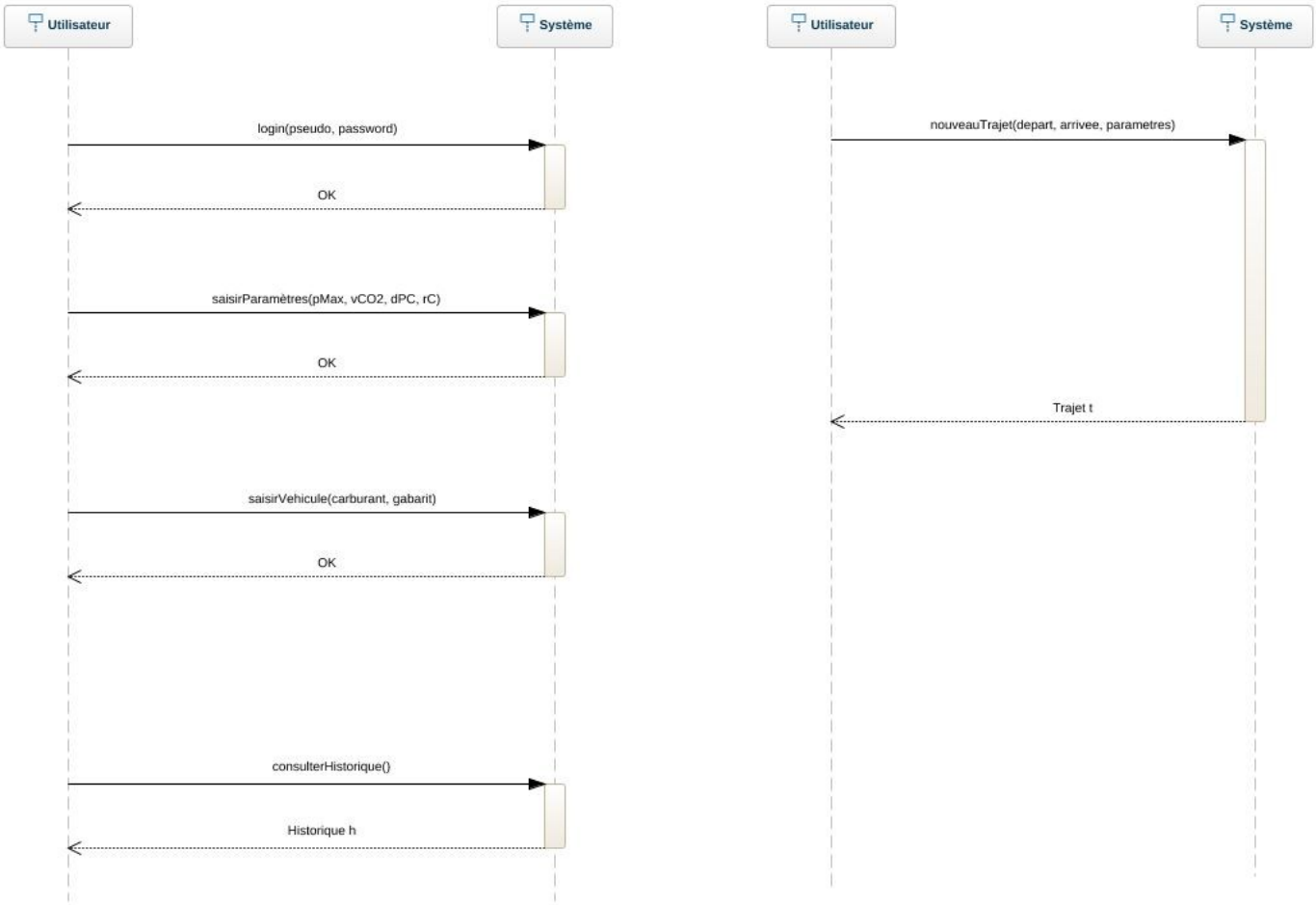


Diagramme de séquence montrant le déroulement lors du démarrage du programme, que ce soit la connection, la saisie de paramètres, de véhicules ainsi que la consultation de l'historique.
L'appel à la fonction nouveau Trajet doit renvoyer un nouveau trajet.

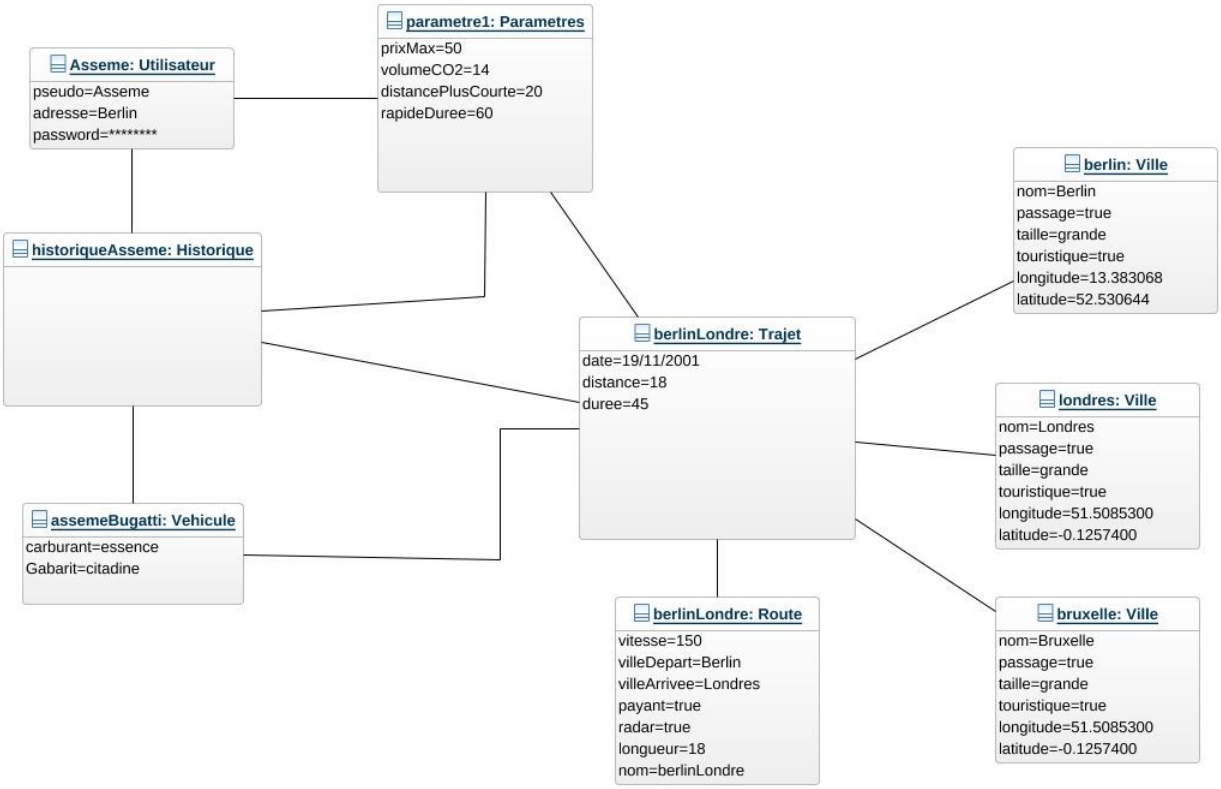


Diagramme d’objets montrant l’état des objets attendu après un appel à toutes les fonctions initialisant un trajet, en supposant que l’historique de l’utilisateur est dans ce cas vide. Les villes et les routes seront toutes enregistrées dans une structure de données adaptée au problème.