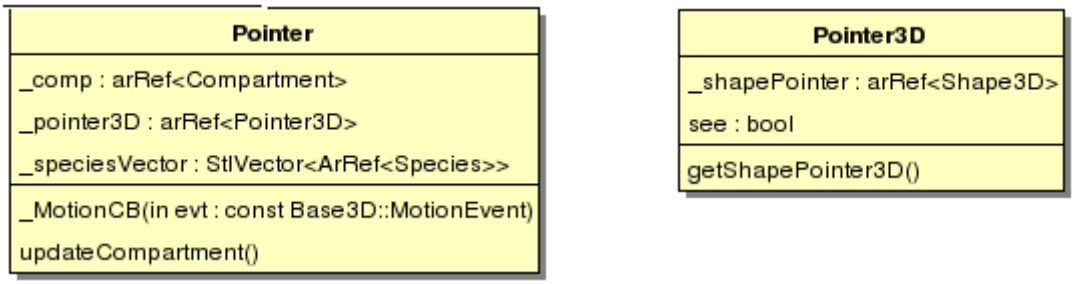
Conception logicielle

Nous allons détailler ici les différentes classes nécessaires à créer afin de coller aux fonctionnalités du logiciel. Certaines classes existent déjà notamment sur le logiciel Endosim, cependant des méthodes et des attributs leur ont été ajouté afin de pouvoir communiquer.
Les méthodes de type get et set ne seront pas décrites afin d'alléger la documentation.

1. Les nouvelles classes

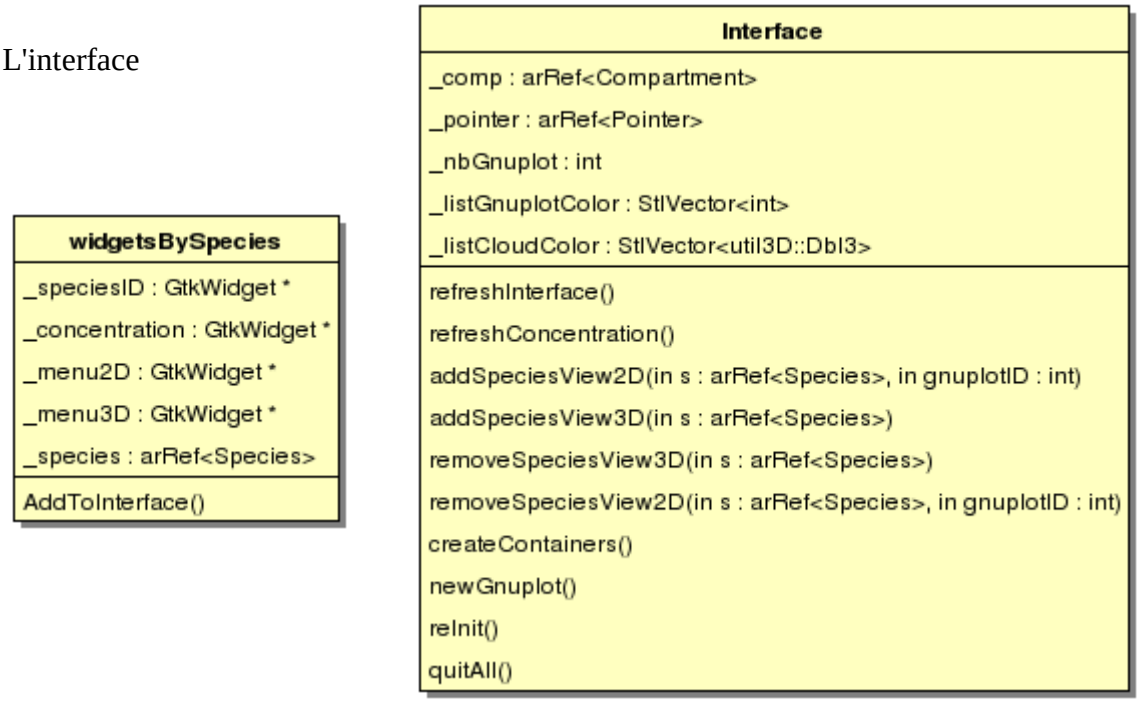
1. Le pointeur 3D



Le pointeur stoque le compartiment dans lequel il se trouve dans la variable *_comp*. Cette valeur est changée dynamiquement quand *_MotionCB* détecte que le curseur a changé de compartiment.
Les espèces présentes dans le compartiment sont stockées dans le vecteur *_speciesVector*.

L'apparence du pointeur se trouve dans l'attribut *_pointer3D* du type **Pointer3D**. Cette classe définit la forme 3D du pointeur ainsi que sa visibilité via l'attribut see.

2. L'interface

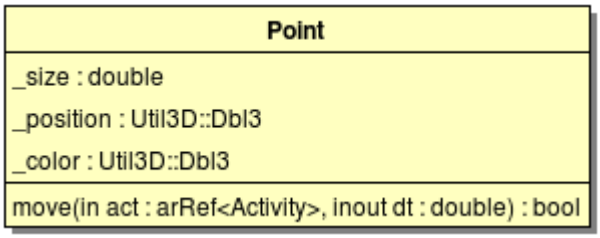


La classe **Interface** est le cS ur de fenêtre de commande. En effet, c'est elle qui instanciera une fenêtre Gtk, lui donnera ses formes et son contenu. L'interface est instanciée dans le main une fois que le pointeur (*_pointer*) a été créé. En effet, l'interface comprend un champ *_comp* qui correspond au compartiment pointé par le pointeur.
L'interface sera en mesure d'instancier des objets **Gnuplot**. Un compteur *_nbGnuplot* permettra de stocker leur nombre.
La méthode *refreshInterface*() permettra à la fenêtre de mettre à jour ses lignes (une par espèces) lors d'un changement de compartiment afin de n'afficher que les espèces présentes dans le compartiment.
La méthode *refreshConcentration*() est une activité permettant d'actualiser le label *_concentration* de chaque **widgetsBySpecies** afin d'avoir une vue dynamique directement sur l'interface de la variation de la concentration par espèce.
La méthode *createContainer*() permet de créer les widgets nécessaires pour une espèce lors du parcours des espèces présentes. Chaque instance correspond à une espèce et donc à une ligne de l'interface.

Les callbacks:
Toutes les autres méthodes de la classe Interface sont des fonctions de type CallBack utilisées lorsqu'une action est effectuée sur la fenêtre.
quitAll() : Appelée lors d'un clic sur le bouton Quitter. Elle permet de quitter l'application en entier.
reInit() : Appelée lors d'un clic sur le bouton Réinitialiser. Elle décoche tout. Supprime les gnuplots et les nuages de points.
newGnuplot() : Appelée lors d'un clic sur le bouton Nouveau gnuplot. Cela crée un gnuplot vide et permet de rajouter son ID dans le widget _menu2D de la classe **widgetsBySpecies** correspondant.
Les fonctions de type *addSpecies* et *removeSpecies* (2D ou 3D) sont appelées lors des changements d'état des sélectionneurs de l'interface. Elles permettent de rajouter ou de supprimer les espèces concernées dans les listes des espèces à visualiser (classes **ViewGnuplot** et **ViewCloudPoint**).

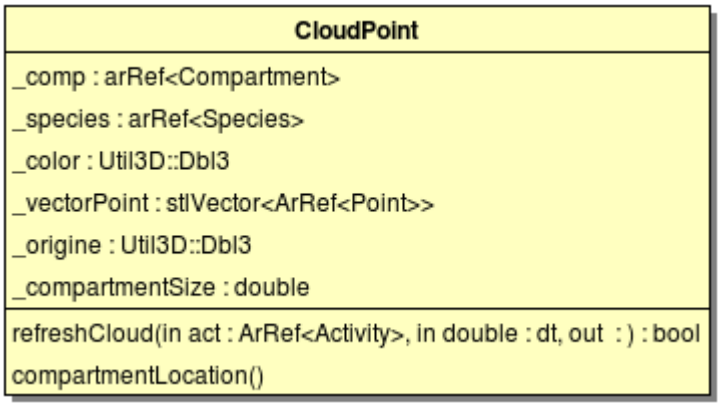
3. Les nuages de points

Les nuages de points sont constitués de plusieur**s Point.**
Ces points sont en fait physiquement représentés par des sphères.



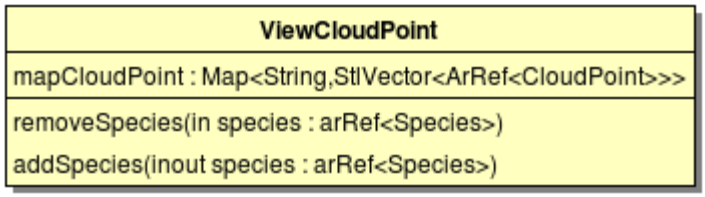
Un **Point** est caractérisé physiquement par trois attributs. *_size* contient la taille de la sphère, le champ *_position* caractérise sa position par rapport au repère de l'**Object3D** du **Compartment** dans lequel le nuage de points évolue. La couleur est aussi paramétrable, d'où le champ _color qui n'est rien d'autre que les valeurs RGB sur trois doubles.
Chaque Point est en activité de période dt. L'activité est la méthode move() qui permet aux point de changer de position aléatoirement dans le nuage de points.

Un nuage de points correspondant à la concentration d'une espèce dans un compartiment, nous avons la classe CloudPoint suivante.



Dans cette classe, nous identifions le compartiment d'évolution (*_comp*), l'espèce concernée (*_species*), la couleur du nuage de points (*_color*), la liste des Point qui constituent le nuage (*_vectorPoint*).
Afin de pouvoir placer les différents points dans les limites du compartiment, nous récupérerons l'origine du compartiment et la taille d'un compartiment, nous stockerons ces valeurs dans _origine et *_compartmentSize*. Ces champs seront mis à jour via la fonction compartmentLocation() qui aura la charge de récupérer ou calculer ces données.
Le nuage aura aussi une activité (*refreshCloud*()) afin de mettre à jour le vecteur de points *_vectorPoint*. En effet, cette méthode permettra d'ajouter ou de supprimer des points du vecteur en fonction de la variation de la concentration.

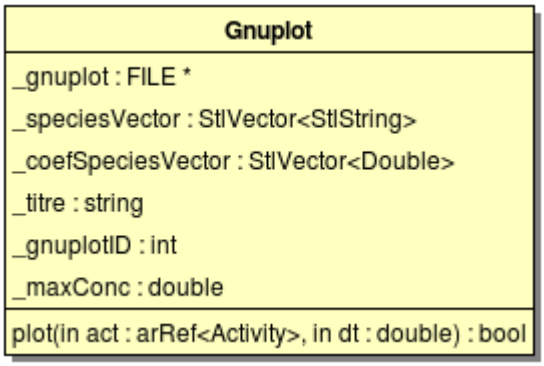
Enfin, l'ensemble de tous les nuages de points présents dans la simulation est stocké dans la classe **ViewCloudPoint** qui contient une map prévue à cet effet *_mapCloudPoint*. La clef de celle-ci représentera le nom de l'espèce chimique associée au nuage, la valeur sera le vecteur de tous les nuages de points concernés (1 nuage de points / compartiments / espèces).



Les méthodes removeSpecies et addSpecies seront directement appelées par les méthodes *addSpecies3D* et *removeSpecies3D* de l'interface. La map contient donc l'ensemble des nuages de points visibles, l'interface peut en ajouter ou en supprimer via ses callbacks.

4. Les courbes Gnuplots

La classe Gnuplot permet la création d'un graphique sur lequel on visualise la ou les concentrations présentes dans le _speciesVector.
Le FILE * *_gnuplot* est un flux permettant de stocker la commande qui sera utilisée pour créer un gnuplot et pour le configurer afin que les espèces souhaitées soit toutes représentées.
Chaque Gnuplot sera différencié par un identifiant : *_gnuplotID*, et aura un titre qui lui sera propre : *_titre* du type Gnuplot_ID.
La concentration maximum *_maxConc* est calculée constamment afin d'ajuster l'échelle du graphique.
Chaque **Gnuplot** possède une activité de période dt. *plot*() qui sert à rafraichir le graphique et se base sur les fichiers créés par **Historique**. Cela est nécessaire pour simuler une représentation continue de l'évolution des espèces chimiques.



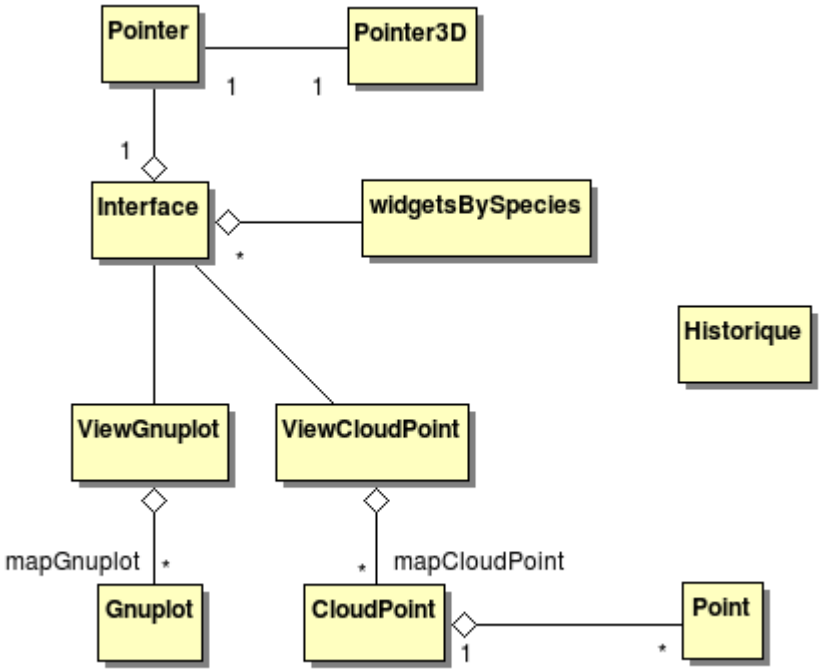
5. L'historique

L'historique permet de garder une trace de l'évolution de la concentration de chaque espèce chimique pour chacun des compartiments.

Historique a besoin de connaître l'architecture de la simulation afin d'avoir accès à tous les compartiments, d'où l'attribut *_constituent*. Une activité est associée afin de pouvoir écrire une ligne supplémentaire sur chaque fichier à un intervalle de temps *dt*.

2. Le diagramme de classe

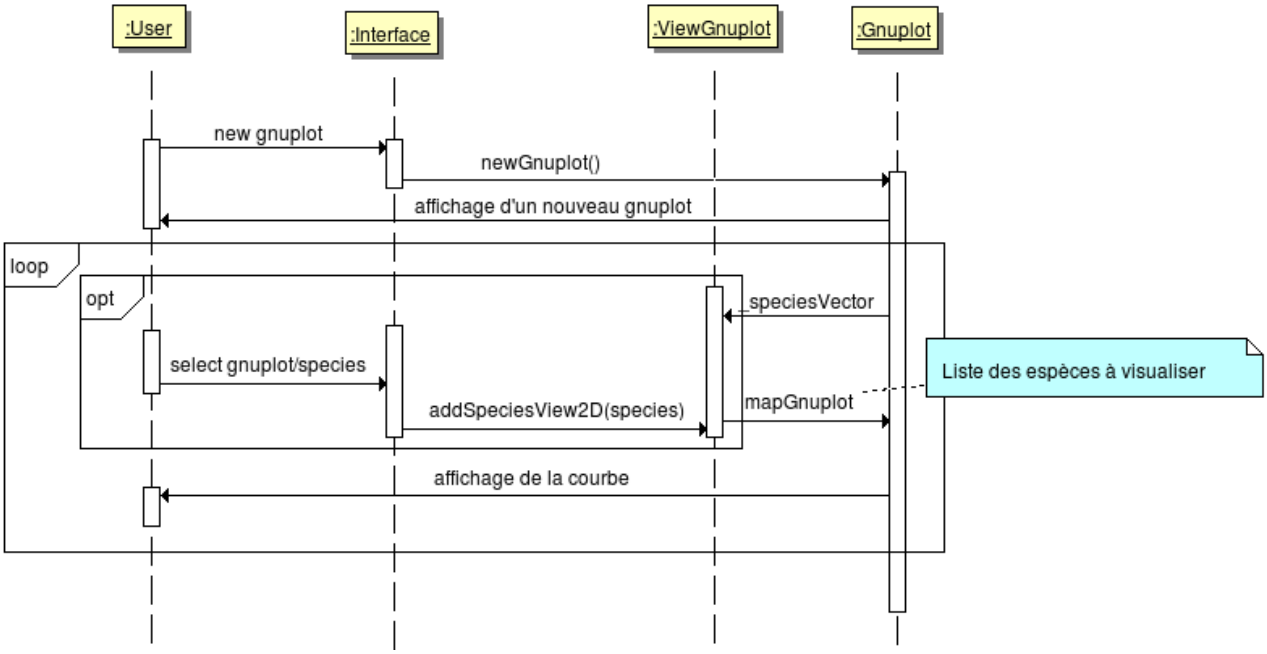
Le diagramme ci-dessous présente les liens de connexité entre les classes. Nous remarquons bien que la classe Interface tient une position centrale. On distingue trois branches correspondant aux fonctionnalités pointeur, vision de la concentration en gnuplot et vision de la concentration en nuages de points. La classe historique à un rôle indépendant, et n'a besoin que de l'environnement ReISCOP actuel pour fonctionner.



3. Diagrammes de séquences

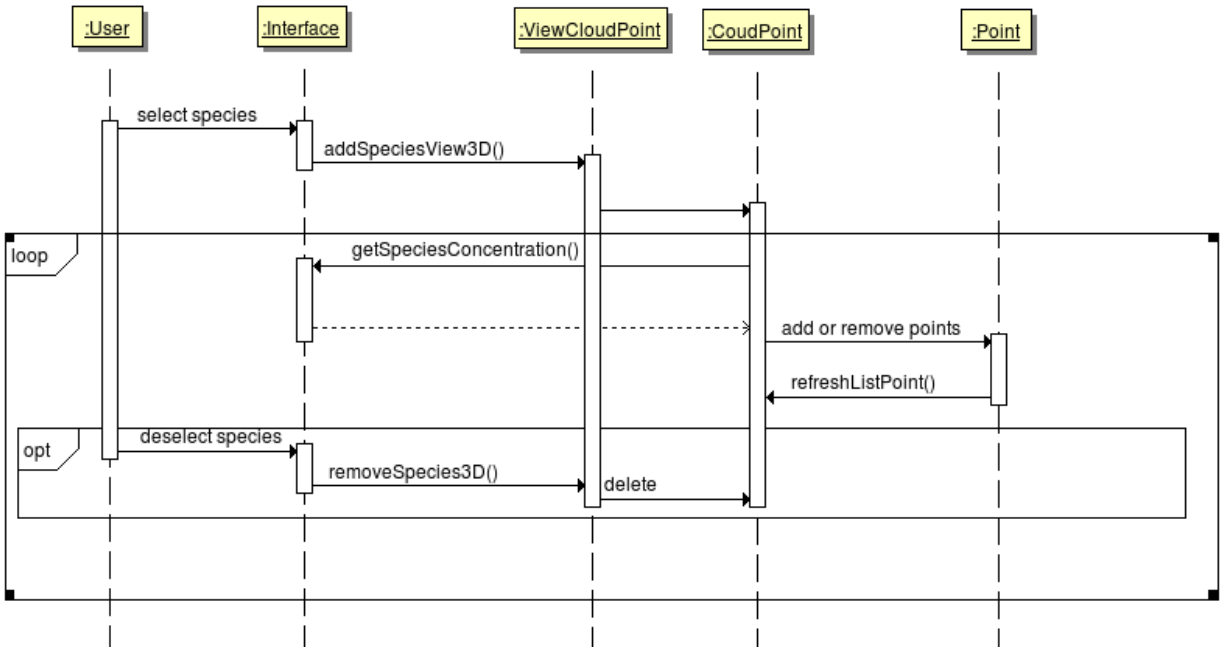
a) La gestion des gnuplots.

Au lancement du programme, une instance d' **Interface** est lancée, celle-ci crée automatiquement une instance de **ViewGnuplot.** Lorsque la méthode *newGnuplot()* est appelée (par un clic sur le bouton « nouveau gnuplot »), une instance de **Gnuplot** est créée. Pour l'instant **Gnuplot** est vide. Si l'utilisateur sélectionne une espèce à voir sur le gnuplot précédemment créé, l'espèce en question est envoyée à **viewGnuplot.** Lors du prochain passage de la boucle *replot*(), **Gnuplot** prendra en compte les espèces ajoutées pour les intégrer sur le graphique. Lors d'un pression sur le bouton quitter, l'interface appelle la fonction *quitAll()* qui ferme le programme et détruit toutes les instances de **Gnuplot.**



b) La gestion des nuages de points.

Lorsque l'utilisateur sélectionne une espèce pour une visualisation de sa concentration en nuages de points, l'appel à la méthode *addSpeciesView3D()* est effectué. Celle-ci permet de rajouter une instance de **CloudPoint** dans la map présente dans **ViewCloudPoint**. Ensuite, le **CloudPoint** questionne l'interface pour savoir quelle est la concentration de l'espèce. Suivant la réponse, il ajoute ou supprime des points dans sa liste de **Point**. Si l'utilisateur désélectionne l'espèce, la méthode *removeSpecies3D*() est appelée, supprimant alors le nuage de points associé.



4. Le diagramme d'états

Ce diagramme présente les différents états du logiciel. L'état stable est **AvancementSimulation**, en effet c'est dans cet état que la simulation vit via une boucle. Lors de chaque pas, *simulationLoop*() ainsi que *refreshHistorique*() et *move*() permettent d'effectuer les différentes réactions, de sauvegarder les concentrations des différentes espèces à cette date et de changer aléatoirement la position de chaque point. Si une action est effectuée, alors on est susceptible de passer à un autre état. Un clic sur un élément de l'interface, mettra automatiquement le système dans l'état souhaité avant de reprendre son état stable. De même, si lors d'une modification de la position du pointeur, celui-ci change de compartiment, alors le système actualise interface avant de continuer sa boucle.