# Projet : conception

Lomàn Vezin
Raphaël Vock

28 mai 2019

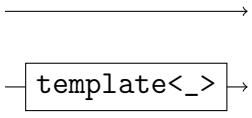
## 1 Lexique

| Langue de Shakespeare | Langue de Molière |
|---|---|
| *Drawable* | Dessinable |
| *Canvas* | Support à dessin |
| *Beam* | Faisceau |

## 2 Représentation schématique des hiérarchies

Ci-dessous deux schémas représentant les hiérarchies relationnelles des classes.

### 2.1 Légende



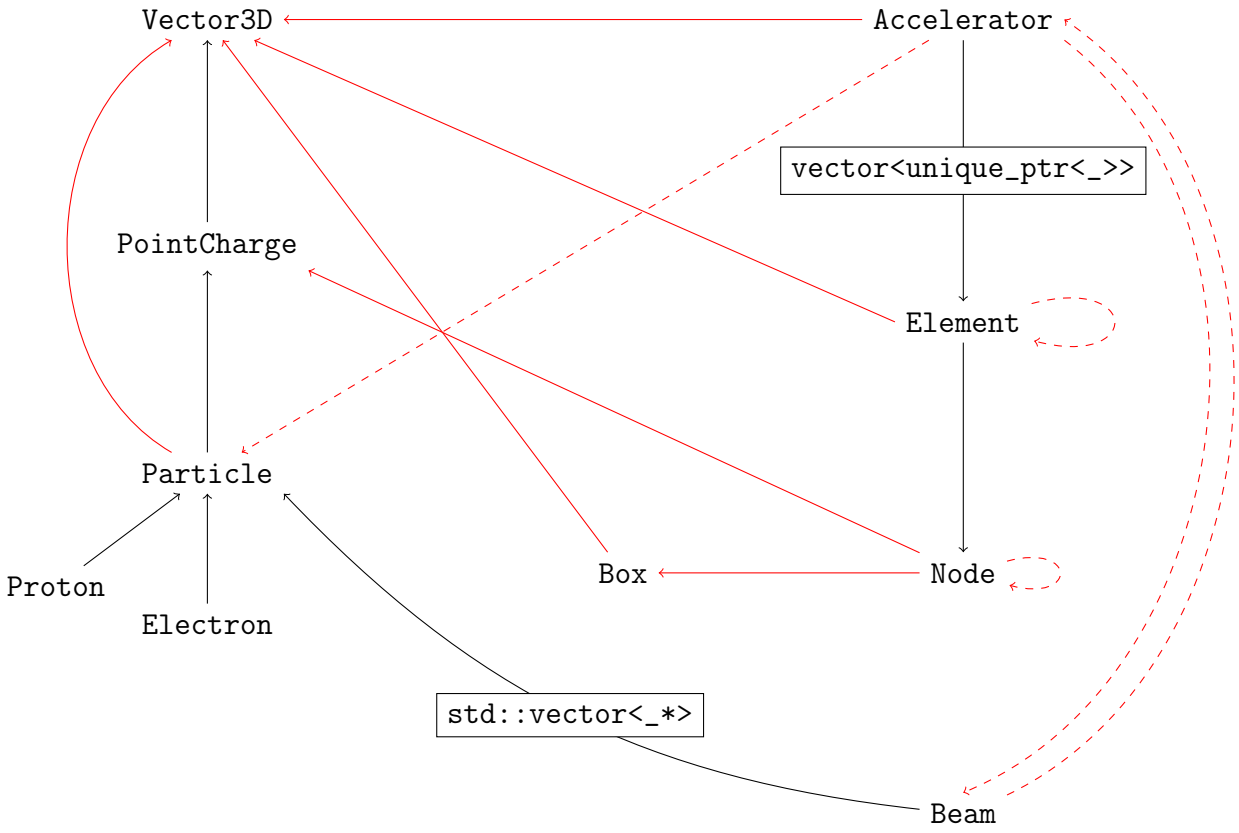

hérite de

hérite de *via* `template<_>`

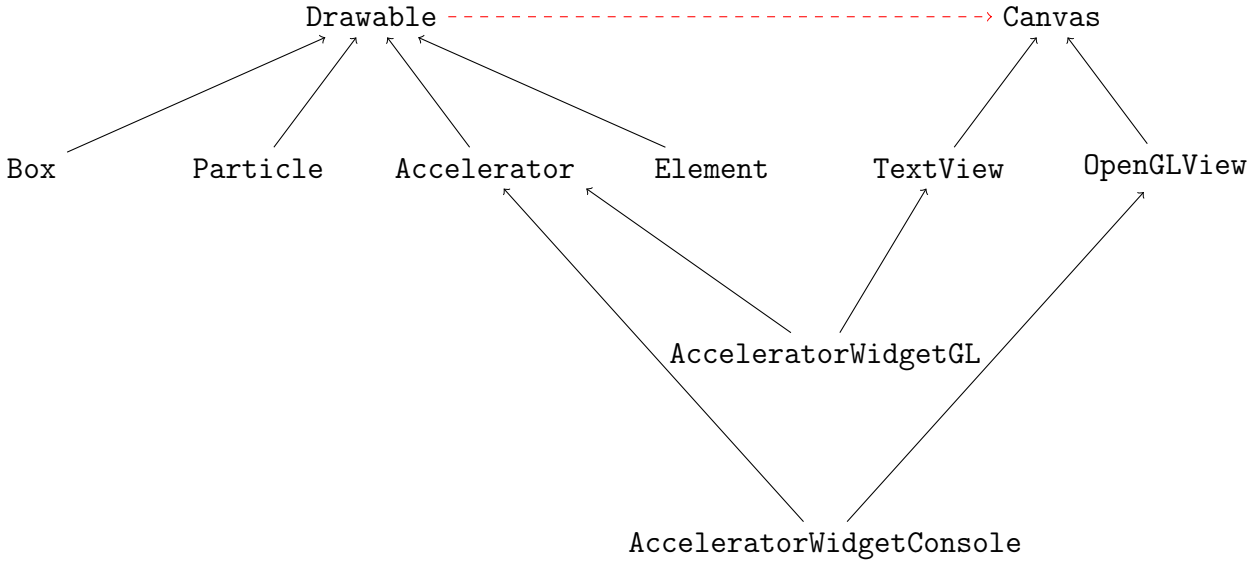
contient une ou plusieurs instances de

contient un ou plusieurs pointeurs vers des instances de

## 2.2 Des classes liées aux objets de la simulation




## 2.3 Des classes liées à la représentation de la simulation



# 3 Quelques éclaircissements

En grande partie, les classes et les héritages suivent directement dans la lignée des instructions du projet. Nous proposons ici de tenter d'éclaircir les points de divergence.

1. La classe `PointCharge` est une "généralisation" d'une particule ; c'est la formalisation de la notion de *charge ponctuelle relativiste* dont l'utilité est primordiale dans le fonctionnement de l'algorithme de Barnes–Hut (cf. fichier réponses §8) dans laquelle les instances de `PointCharge` représentent les "charges ponctuelles moyennes." En effet, la classe `Particle` contient plusieurs attributs qui ne sont pas pertinentes. Dans l'optique d'affiner la conception et de réduire l'empreinte mémoire (la performance de cet algorithme est clé) nous avons donc introduit cette super-classe dont les attributs sont la position, la charge et le facteur gamma.
2. La décision de faire hériter `PointCharge` (et donc `Particle`) de `Vector3D` permet de voir les particules comme des points de l'espace munis de structure supplémentaire, et de ce fait d'obtenir un polymorphisme avec les diverses méthodes "géométriques" des éléments. Par exemple, les méthodes liées au système de coordonnées locales d'un élément devraient, en toute généralité, pouvoir être appelés sur des points quelconques de l'espace et non que des particules. En revanche, c'est très contraignant de devoir faire recours à un getter à chaque fois que nous voulons l'appeler sur (la position d') une particule. Dès lors, en faisant hériter la classe `Particle` de `Vector3D`, nous contournons ce problème.
3. Les sous-classes `Electron`, `Proton` permettent tout simplement de s'abstraire des grandeurs physiques qui les définissent et de définir une affichage caractéristique à chaque type de particule. À l'avenir on ajoutera des sous-classes correspondant à d'autres types de particules subatomiques.
4. La classe `Node` est l'implémentation de l'arbre décrit dans §8.
5. La classe `Box` permet de représenter des pavés de l'espace et leur utilité se trouve dans l'algorithme de Barnes–Hut. Elles s'accompagnent de méthodes servant à les subdiviser, à décider si un point y appartient, etc.
6. Les classes `AcceleratorWidgetX` doivent être comprises comme "des instances d'`Accelerator` qui vivent dans un `X`" (où `X` est une sous-classe de `Canvas`). Concrètement, c'est pour pouvoir interagir directement avec à la fois le support à dessin et son contenu depuis le `main`, d'où l'héritage double.