Polytech Nice-Sophia

Rapport modélisation objet d'un simulateur de robot V1

2014 - 2015

GUILLOT Romain - PUJOL Jonathan (SI4 G3) - 1

Introduction

Pour ce projet, nous avons à modéliser puis développer l'ensemble des comportements d'un robot. Le robot se déplace (avance, tourne) à travers une carte représentée par une abscisse "x" et une ordonnée "y". Il peut rencontrer des plots, sur lesquels sont posés (ou pas) des objets. Le robot peut évaluer la taille du plot devant lequel il se trouve. Il pourra aussi saisir les objets sur les plots, les peser, se déplacer avec, et les poser sur de nouveaux plots. De plus, le robot pourra être mis en pause.

Les actions du robot sont : avancer, tourner, rencontrerPlot, evaluerPlot, saisir, peser, poser, figer.

Le robot présente des comportements différents selon l'état dans lequel il se trouve, en effet, il pourra évaluer un plot que lorsque qu'il est à vide face à celui-ci, ou encore il ne peut peser un objet que lorsqu'il l'a au paravent saisi.

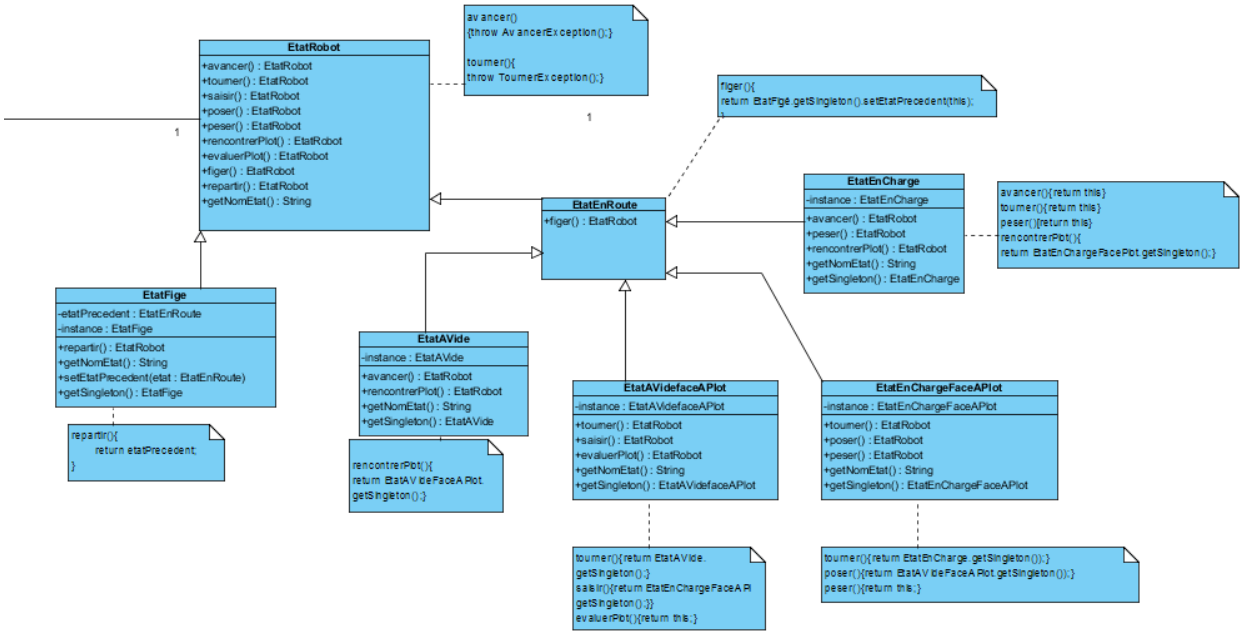
Les états du robot sont : aVide, aVideFacePlot, enChargeFacePlot, enCharge, figé.

Les schémas de conception utilisés

Pattern état :

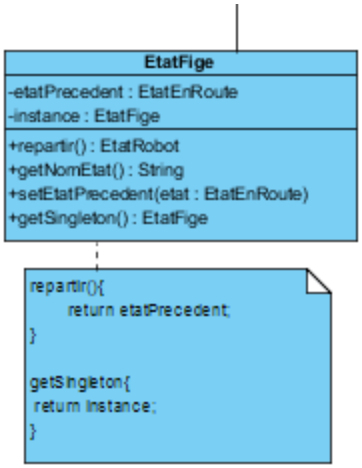
Nous avons utilisé le pattern état afin de modéliser l'état dans lequel se trouve le robot. EtatRobot est la classe mère et ses classes filles sont l'ensemble des états possibles du robot. Le robot possède un attribut qui est son état courant. Pour chaque action du robot, celui-ci appellera la méthode correspondante dans l'état :

- Si l'action n'existe pas dans cet état, une exception sera levée.
- Si l'action existe, alors on n'effectue rien OU on change d'état.



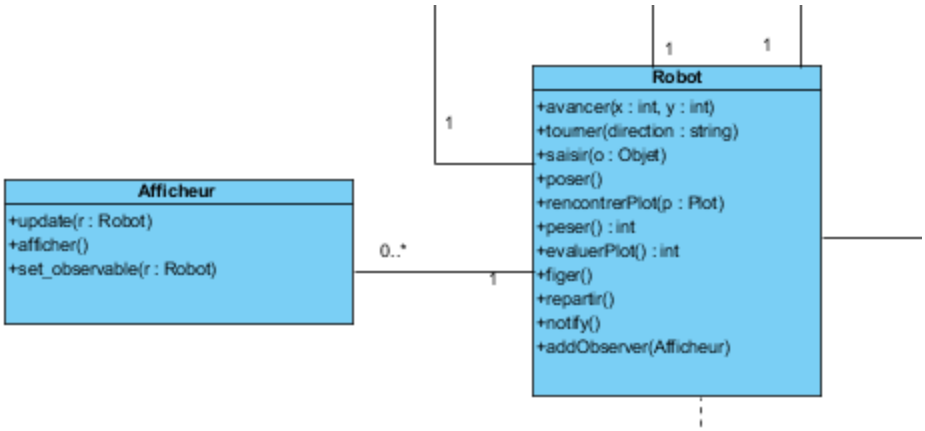
Pattern singleton :

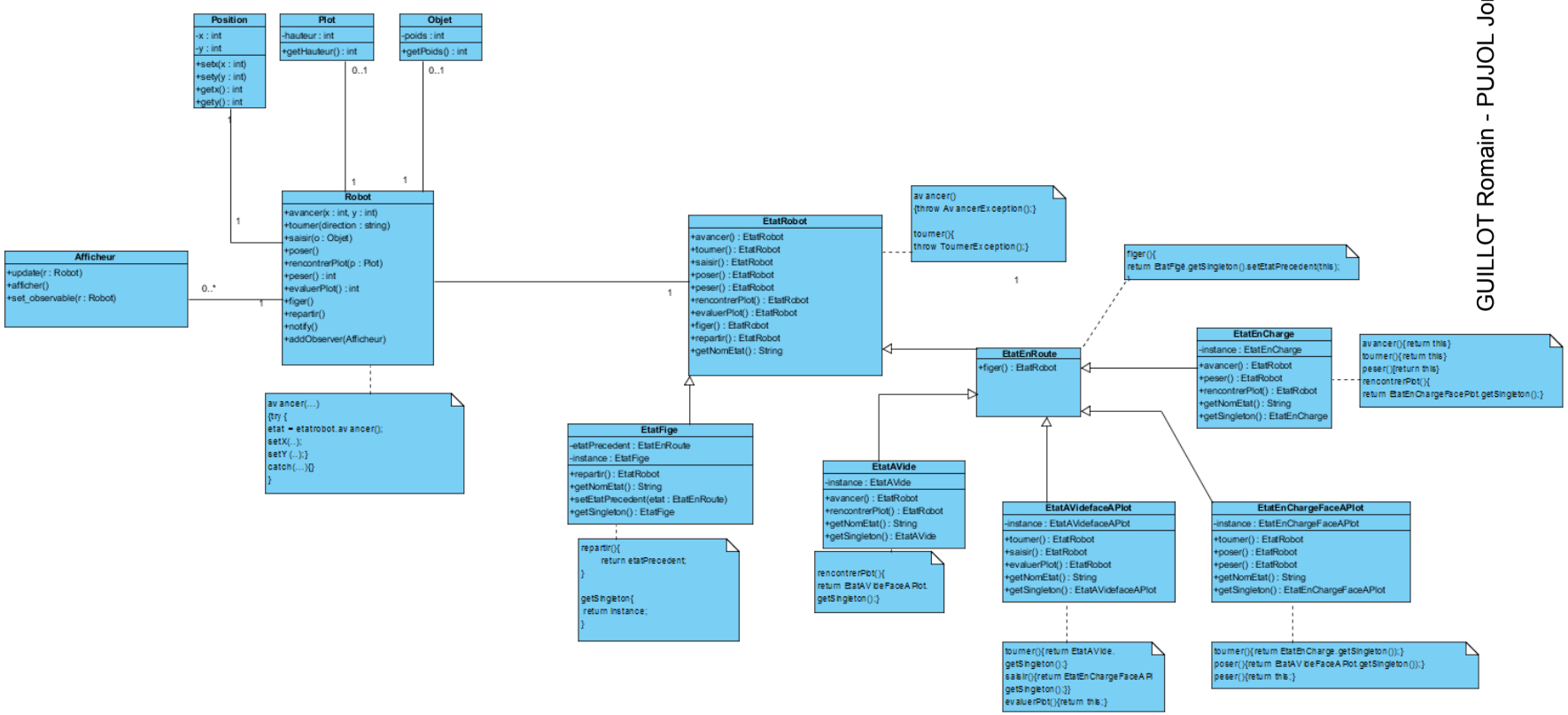
Nous avons utilisé le pattern singleton car de nombreuses instances des classes d'états allaient être créees à chaque changement d'état du robot. Un singleton est associé à chaque état instanciable du robot. L'appel de la méthode getSingleton sur un état permet de créer son instance dans le cas où cet état n'a jamais été créée, ou de retourner l'instance déjà créée de cet état.



Pattern Observeur/Observable :

Nous avons utilisé le pattern observeur/observable afin de pouvoir afficher des informations à jour sur l'état courant du robot. Le robot est observable et l'afficheur est l'observeur. Lors des modifications faites sur le robot (son état/la valeur de ses attributs), l'afficheur est notifié et pourra afficher les nouvelles informations sur le robot. Le robot peut avoir de 0 à n observateurs. Il peuvent être ajoutés dynamiquement grâce à la méthode addObserver.




Position
-x : int
-y : int
+setx(x : int)
+sety(y : int)
+getx() : int
+gety() : int
Plot
-hauteur : int
+getHauteur() : int
Objet
-poids : int
+getPoids() : int
0..1
0..1
1
1
1
Robot
+avancer(x : int, y : int)
+tourner(direction : string)
+saisir(o : Objet)
+poser()
+rencontrerPlot(p : Plot)
+peser() : int
+evaluerPlot() : int
+figer()
+repartir()
+notify()
+addObserver(Afficheur)
Afficheur
+update(r : Robot)
+afficher()
+set_observable(r : Robot)
0..*
1
avancer(....)
{try {
etat = etatrobot.avancer();
setX(...);
setY (...);}
catch(....){}
}
EtatRobot
+avancer() : EtatRobot
+tourner() : EtatRobot
+saisir() : EtatRobot
+poser() : EtatRobot
+peser() : EtatRobot
+rencontrerPlot() : EtatRobot
+evaluerPlot() : EtatRobot
+figer() : EtatRobot
+repartir() : EtatRobot
+getNomEtat() : String
1
avancer()
{throw AvancerException();}
tourner(){
throw TournerException();}
figer(){
return EtatFige.getSingleton().setEtatPrecedent(this);
EtatEnRoute
+figer() : EtatRobot
EtatEnCharge
-instance : EtatEnCharge
+avancer() : EtatRobot
+peser() : EtatRobot
+rencontrerPlot() : EtatRobot
+getNomEtat() : String
+getSingleton() : EtatEnCharge
avancer(){return this}
tourner(){return this}
peser(){return this}
rencontrerPlot(){
return EtatEnChargeFacePlot.getSingleton();}
EtatFige
-etatPrecedent : EtatEnRoute
-instance : EtatFige
+repartir() : EtatRobot
+getNomEtat() : String
+setEtatPrecedent(etat : EtatEnRoute)
+getSingleton() : EtatFige
repartir(){
return etatPrecedent;
}
getSingleton{
return instance;
}
EtatAVide
-instance : EtatAVide
+avancer() : EtatRobot
+rencontrerPlot() : EtatRobot
+getNomEtat() : String
+getSingleton() : EtatAVide
rencontrerPlot(){
return EtatAVideFaceAPlot.
getSingleton();}
EtatAVidefaceAPlot
-instance : EtatAVidefaceAPlot
+tourner() : EtatRobot
+saisir() : EtatRobot
+evaluerPlot() : EtatRobot
+getNomEtat() : String
+getSingleton() : EtatAVidefaceAPlot
tourner(){return EtatAVide.
getSingleton();}
saisir(){return EtatEnChargeFaceAPl
getSingleton();}}
evaluerPlot(){return this;}
EtatEnChargeFaceAPlot
-instance : EtatEnChargeFaceAPlot
+tourner() : EtatRobot
+poser() : EtatRobot
+peser() : EtatRobot
+getNomEtat() : String
+getSingleton() : EtatEnChargeFaceAPlot
tourner(){return EtatEnCharge.getSingleton());}
poser(){return EtatAVideFaceAPlot.getSingleton());}
peser(){return this;}