Polytech' Nice Sophia-Antipolis

Simulateur de Robot en C++

Analyse et Conception

Forget - Jiang
23/11/2014

Table des matières

Introduction

Ce rapport a pour but de mettre en avant les patterns choisi pour la conception et l'implémentation d'un simulateur de Robot en C++. Chaque patterns a était choisi dans un but de réutilisabilité, de flexibilité et d'optimisation du code. Voici le diagramme initial du Robot (cf. figure 1) :

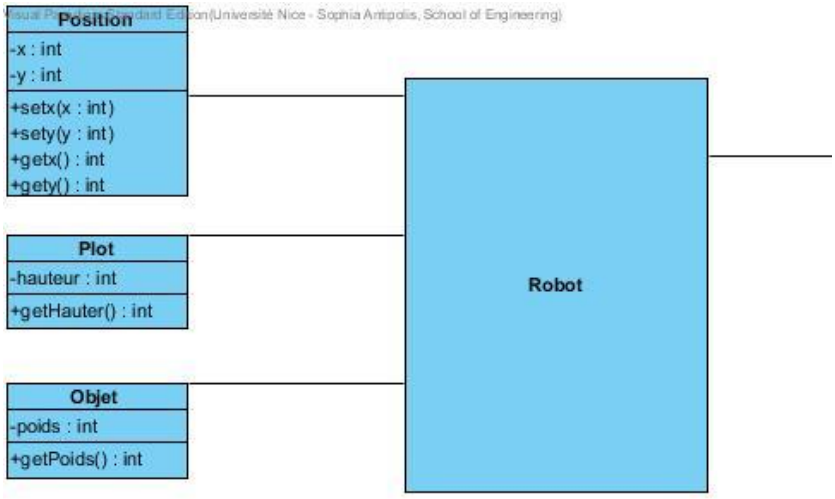

FIGURE 1 : DIAGRAMME DE CLASSE SIMPLIFIE DU ROBOT

Etape 1 – Etat

La principale caractéristique du Robot est qu'il parcourt son environnement et interagis avec diffèrent éléments. De ce fait le robot peut être représenté dans plusieurs états différents ; en route, en charge, face à un obstacle, etc. Pour modéliser ce comportement nous avons choisi le pattern homonyme, à savoir « Etat » (ou State).

Nous avons donc les 2 états principaux, **enRoute** et **fige** qui héritent d'une classe **Etat**, qui reprend toute les fonctions du robot qui dépendent de son état (toutes sauf celles liées à l'affichage). Toute ces fonction lèves par défaut une exception de type **WrongStateException** et serons détaillé dans chaque état, par exemple dans **enRoute** nous avons la méthode **figer()** qui permet de passer à l'état figer et inversement dans figer la méthode **repartir()** qui devra retourner l'état dans lequel on était.

Pour un autre exemple voir le pseudo-code de la méthode tourner dans les différentes classes du diagramme (cf. figure 2).

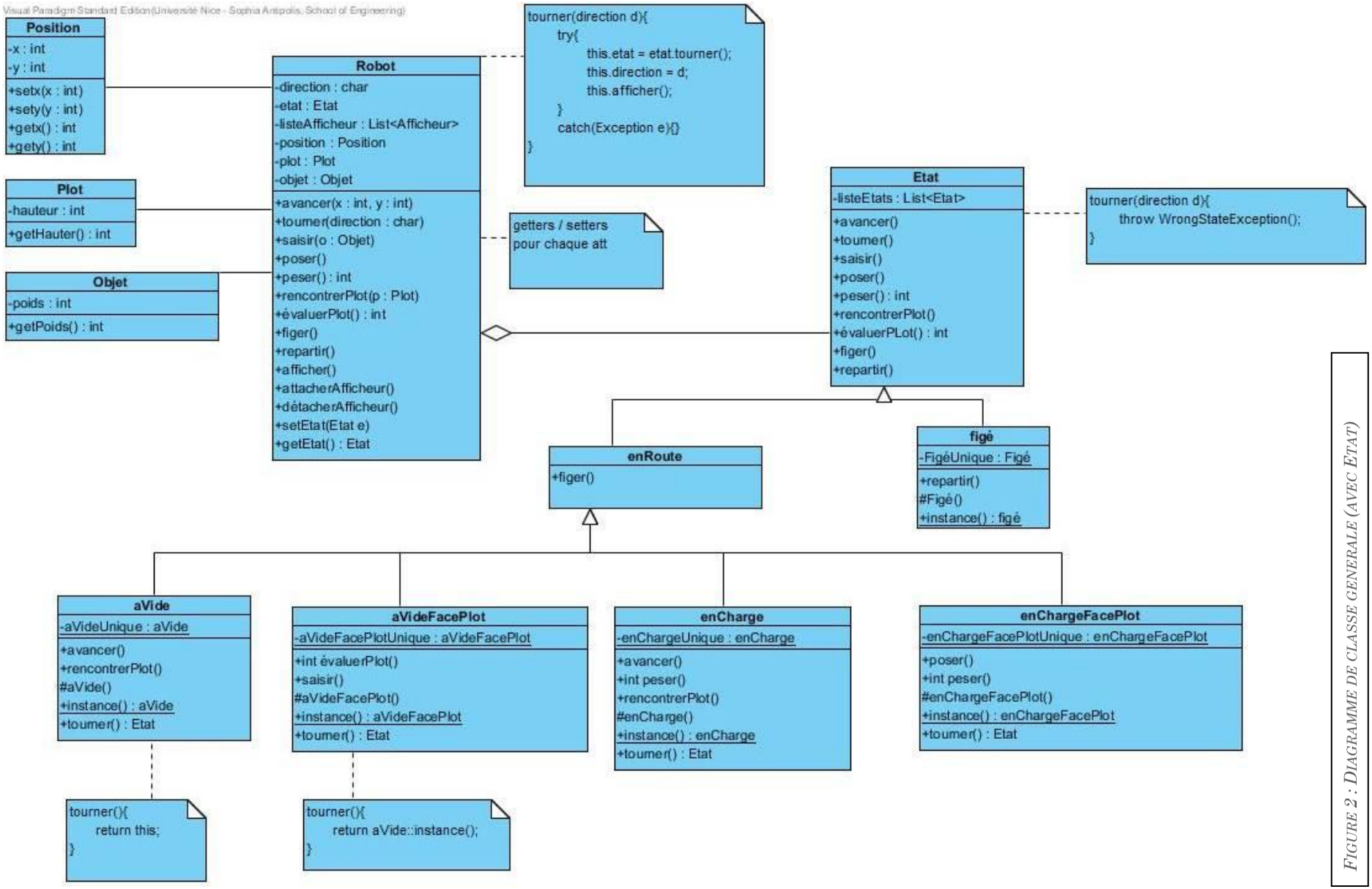

FIGURE 2 : DIAGRAMME DE CLASSE GENERALE (AVEC ETAT)

Etape 2 – Observer

Dans la partie précédente nous avons parlé de fonctions d'affichage. En effet il faut être capable régulièrement (ou sur demande) d'afficher l'état du Robot et d'éventuel détails (direction, poids d'objet, hauteur de plot, etc.). Toujours dans un souci de flexibilité du code, il faut garder à l'esprit que cet affichage peut changer ou être multiple, par exemple la sortie standard, un fichier, … Pour résoudre ce problème nous avons choisi le pattern « Observer » afin de rendre l'affichage indépendant du Robot. Nous avons donc une classe mère **afficheur** avec deux filles ici pour l'exemple une **afficheurConsole** et une **afficheurFichier**. Ces classes reprennent donc la fonction **afficher()** du Robot, il faut donc ajouter à ce dernier un attribut **listeAfficheur** (la liste des afficheur « abonné » au Robot) ainsi qu'une méthode pour pouvoir ajouter ou retirer un afficheur de cette liste (cf. figure 3).

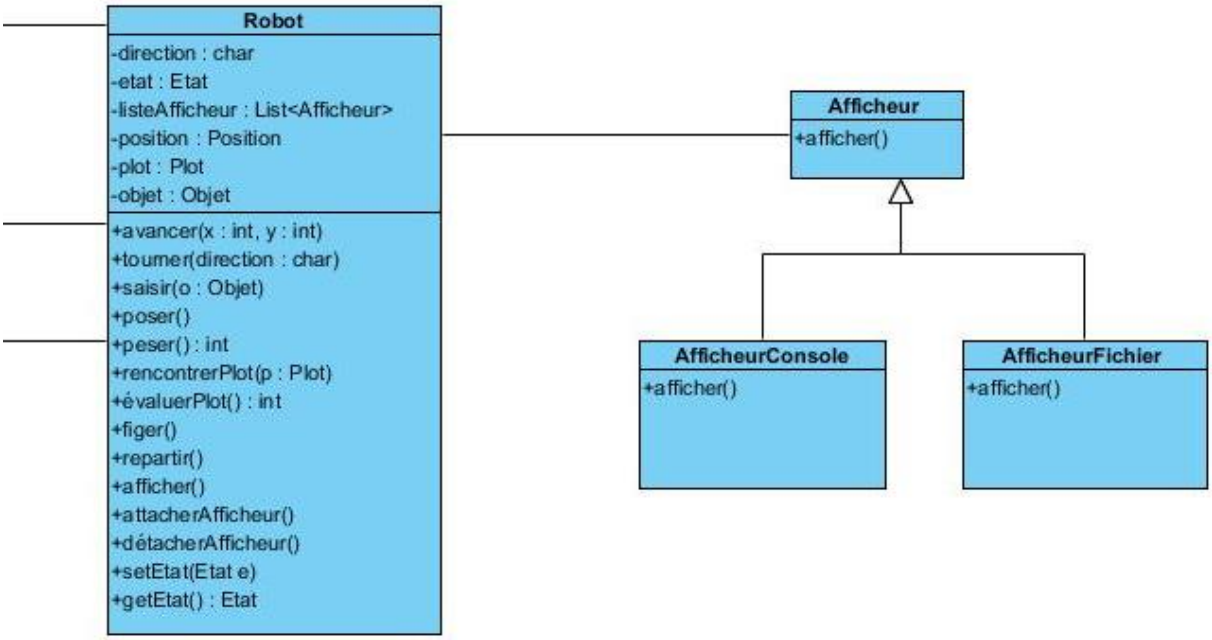

FIGURE 3 : DIAGRAMME DE CLASSE DU ROBOT (AVEC AFFICHEUR)

Etape 3 – Singleton

Nous faisons à nouveau référence à la première étape dans laquelle nous avons mis en place des classes pour chaque état du Robot. Lors de l'utilisation du système il ne faut pas recréer un nouvel état à chaque fois que le Robot en change, et c'est dans ce but que nous avons implémenté le pattern « Singleton ». Ce pattern nous permet dans chaque classe d'avoir un constructeur qui ne va créer qu'une instance de chaque état (cf. figure 2).

Remarque : Suite à un problème technique nous n'avons pas pu implémenter ces solutions dans un code qui compile, donc l'archive fournit contient une version antérieur de notre simulateur de Robot.
