FALI Assam
ROUSSEAU Théo

PPIL
Rapport du projet de synthèse

Introduction

L’objectif du projet de PPIL de cette année a été la mise en place d’un logiciel permettant de dessiner diverses formes géométriques à l’aide d’une communication client-serveur. En effet, le client, écrit en C++, se charge de la gestion des coordonnées des figures géométriques, coordonnées qui sont ensuite envoyées par requête au serveur via un protocole TCP/IP. Celui-ci, écrit en Java, a pour rôle d’afficher les figures à l’aide de l’Active Rendering. Pour réaliser ce travail, nous avons mis en place différentes étapes détaillées ci-dessous.

Diagrammes

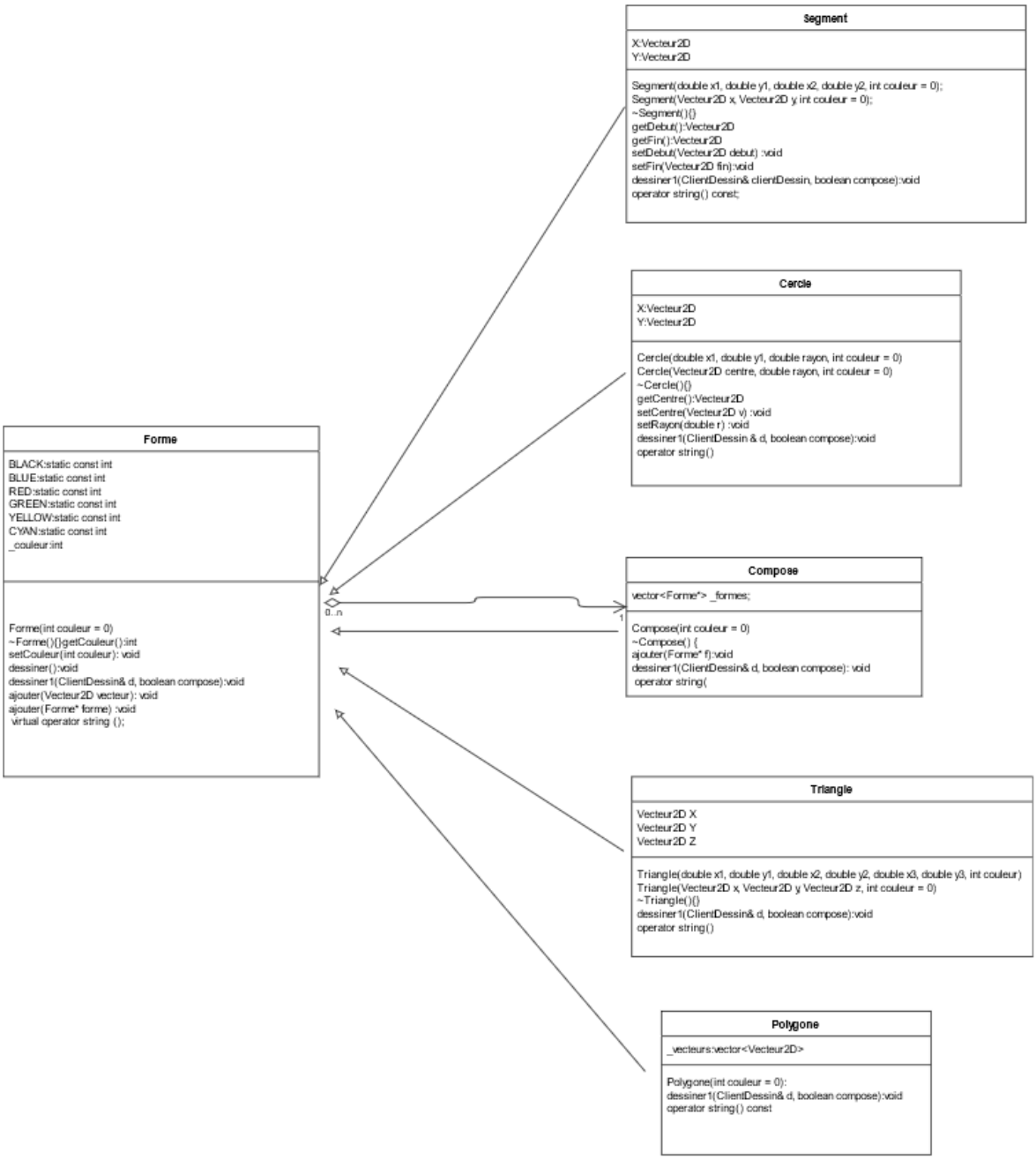

Figure 1 : UML du client

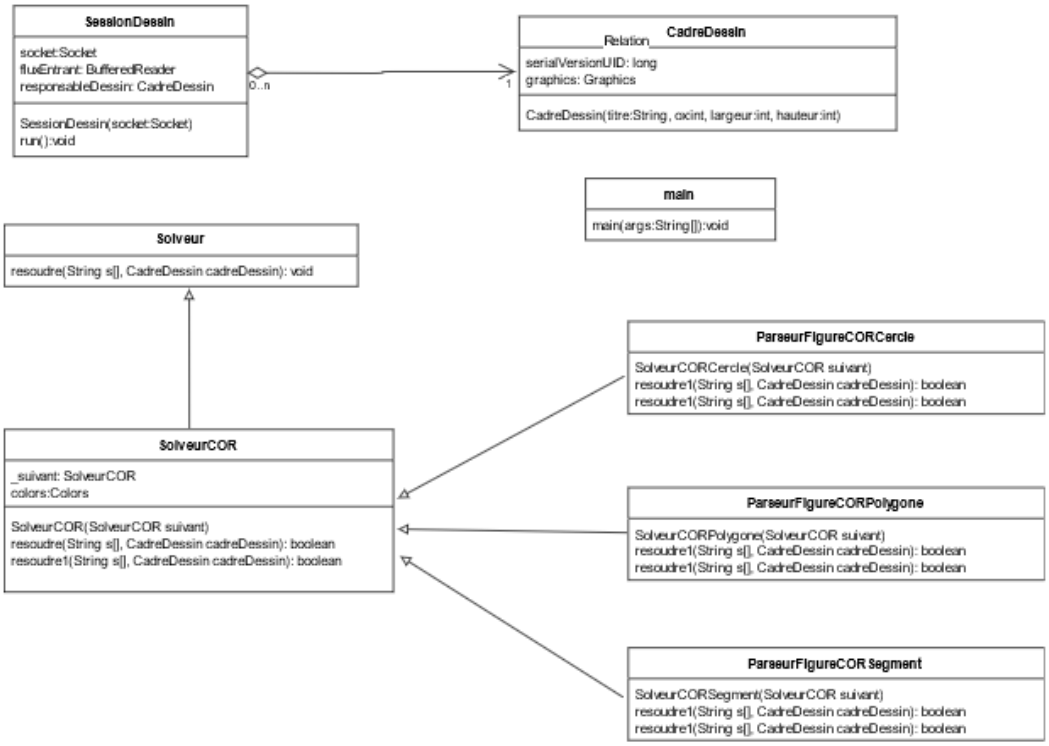

Figure 2 : UML du serveur

Développement du projet

Dans un premier temps, nous avons travaillé ensemble sur les figures les plus simples (segment, triangle et cercle) afin de comprendre le fonctionnement de base du client/serveur. Ensuite, nous nous sommes réparti les tâches afin d'optimiser au mieux le travail.

Je me suis chargé de finaliser les classes de figures ainsi que de créer celle pour les polygones et les figures composées. Je me suis aussi occupé de la partie Java et de Chain of Responsability (côté Java uniquement).

Assam s'est quant à lui occupé des transformations des figures et de la partie visiteur.

Assam	Commun	Théo
-Transformation des figures -Partie visiteur	Classe Segment	-Classes Cerlce, Triangle, Polygone et Compose -Partie Java -Chain of Responsability

Classes Segment, Triangle et Cercle

Pour réaliser ces classes, nous nous sommes inspirés des classes Croix et Rond de la correction du dernier TP. Ainsi, la classe Segment hérite de la forme et use de la fonction « traceSegment » de la classe ClientDessin afin de tracer un trait. Côté serveur, le trait est tracé via la fonction drawLine d'Active Rendering.

Les classes Triangle et Cercle fonctionnent d'une manière similaire, à la différence que le serveur à recours à drawPolygon pour les triangles et à drawEllipse pour les cercles.

Classes Polygone et Compose

Ces deux classes possèdent un vector d'objets en paramètres : vector de Vecteur2D pour Polygone et vector de Formes pour Compose.

A chaque fois que l'on souhaite dessiner un polygone il suffit d'ajouter autant de formes que voulu et d'appeler la fonction dessin qui fait appel à la fonction tracePolygone de ClientDessin. Dans cette dernière, une itération a lieu sur le vector de Vecteurs2D et l'on récupère ainsi les coordonnées de chaque vecteur présent à l'intérieur. La requête envoyée au serveur contient également en information le nombre de vecteurs présents le vector afin que le serveur puisse mieux prendre en charge la requête.

La classe Compose fonctionne d'une manière similaire : une itération dans le vector _formes permet de dessiner chaque forme présente à l'intérieur en appelant pour chacune sa fonction dessin1. Afin de n'ouvrir qu'une seule fenêtre de dessin, j'ai rajouté un booléen dans les paramètres. Celui-ci est instancié à false si la forme n'est pas dans un groupe et va donc appeler la fonction et a true sinon. De cette manière, le client sait s'il doit faire appel ou non à la fonction ouvreFenêtreGraphique de ClientDessin.

Partie Java

Pour gérer le côté serveur, j'ai eu recours à Chain of Responsability. En effet, j'ai mis en place une classe Solveur qui contient une fonction résoudre(String s, CadreDessin cadreDessin). Cette fonction prend donc en paramètres la requête reçue par le serveur ainsi que le cadre qui servira à faire la fenêtre où seront dessinés les figures.

La classe SolveurCOR hérite de la classe Solveur. Elle contient deux membres : un SolveurCor _suivant qui servira donc à faire le lien avec la solution suivante ainsi qu'un tableau de Color que j'ai implémenté à l'aide du module Color de java.

Le serveur contient donc cinq classes de solution : ParseurFigureCercle, ParseurFigureTriangle, ParseurFigureSegment et ParseurFigurePolygone.

Tous les parseurs fonctionnent d'une manière similaire. C'est-à-dire que la fonction résoudre commence par vérifier si le nom de l'opération

correspond bien à la classe de la solution via le premier élément de la requête et renvoie false si ce n’est pas le cas.

Si l’opération correspond bien au parser, alors celui-ci va en récupérer les coordonnées ainsi que le numéro de la couleur et tracer le dessin dans le CadreDessin.

Pour les polygones, le parseur reçoit le nombre de vecteurs présents dans le vector de la forme et fait une itération en fonction afin de récupérer les coordonnées de chaque forme.