Rapport de projet : Tableau virtuel interactif

Baptiste Saleil Geoffrey Mélia Julien Pagès Kevin Bollini

9 mai 2012

Table des matières

Table des figures

Remerciements :

Nous tenons à remercier tout particulièrement M. William Puech (enseignant chercheur au LIRMM à l'université Montpellier 2, responsable de la formation du Master informatique IMAGINA) sans qui ce projet n'aurait pas pu se faire, pour avoir accepter de nous encadrer, pour son aide et son implication.
Nous souhaitons aussi remercier M. Benoit Lange (doctorant au LIRMM) pour ses conseils et sa présence au cours de ce projet.

Chapitre 1

Introduction

1.1 Présentation

Les cinq dernières années ont marqué un grand renouveau dans les interfaces entre hommes et terminaux (écrans tactiles, contrôles vocaux, détection de mouvements). Cet état de faits nous a amenés à penser une application mettant en scène l'une de ces nouvelles façons d'interagir.
L'objectif du projet issu de cette réflexion est donc de créer un tableau virtuel, avec lequel une ou plusieurs personnes peuvent interagir en réalisant directement des gestes comme sur une toile réelle, par reconnaissance des mouvements detectés par une webcam.

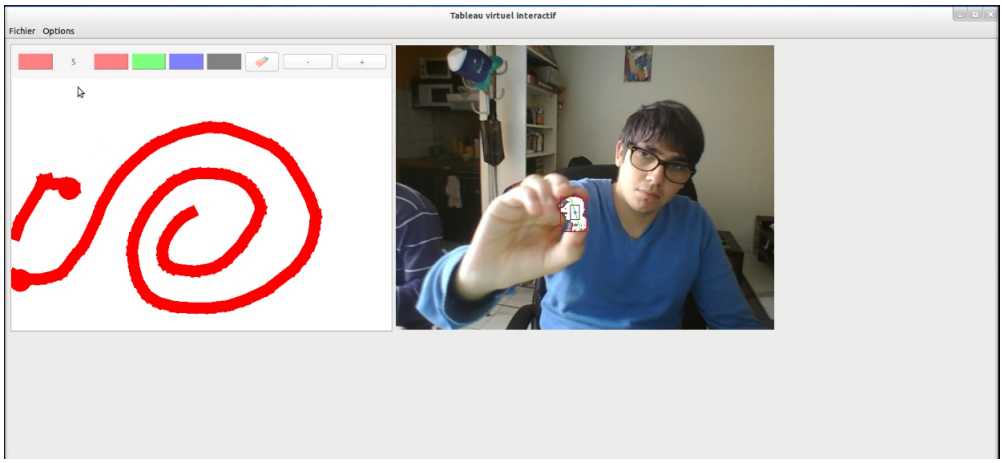

FIGURE 1.1 – Aperçu de l'application

1.2 Contexte

Ce projet est réalisé dans le cadre des TER (Travaux d'Étude et de Recherche) de 1ère année de Master informatique à L'Université Montpellier 2.
Le TER est un projet qui se déroule sur un semestre, généralement choisi par les étudiants parmi une liste de projets qui leurs sont proposés.
Ce genre de projet apporte beaucoup aux étudiants. Il leur permet d'une part de travailler une longue période sur un seul et même sujet afin d'obtenir quelque chose de complet, d'abouti.
D'autre part, le TER (comme son nom l'indique) apporte une bonne expérience dans le domaine de la recherche. En effet, un travail préalable doit être effectué sur l'existant (Est-ce que d'autres projets de ce genre existent ? Si oui qu'ont-ils de différent ?,...), mais également la faisabilité (Est-ce possible à faire ? Quels algorithmes existent pour répondre à tel ou tel besoin ?, ...).

La particularité de ce projet réside dans le fait que nous avons nous-même proposé ce sujet à un enseignant-chercheur.
Nous avons décidé de proposer ce sujet pour diverses raisons, dont voici les pricipales :

- Tout d'abord, c'est un projet que nous avions en tête depuis plus d'un an, qui a donc eu le temps de murîr et d'être au niveau d'un TER.
- Deuxièmement, notre groupe vient de deux formations distinctes : AIGLE, formation orientée génie Logiciel ; et IMAGINA, spécialisée dans le jeu vidéo et l'image. Ce projet représentait une manière opportune de lier ces deux spécialités
- Enfin, ce projet était pour nous une bonne occasion de réaliser une application fonctionnelle et distribuable, et par conséquent nous permettre d'explorer une autre facette du développement d'applications : la mise en production.

Chapitre 2

Analyse et conception

2.1 Étude de l'existant et faisabilité

Faisabilité

Notre groupe ayant lui-même proposé ce sujet, celui-ci s'accorde parfaitement à nos formations et spécialités. Le choix de ce sujet a donc été réalisé en fonction de nos expériences, savoir-faire et affinités.
Ce projet s'inscrivant dans le cadre d'un TER de notre formation, l'étude de la faisabilité n'incluera pas certains critères comme l'étude de marché, le contexte économique ou le besoin réel. Nous nous sommes concentrés notamment sur les compétences techniques et de gestion. Pour les parties financement et analyse des coûts, il va sans dire que nous ne disposions d'aucun financement et n'avons donc utilisé que des outils gratuits.

Existant

La vision par ordinateur et particulièrement le suivi d'objets, sont des domaines connus et pour lesquels il existe de nombreux travaux. Nous pouvons nous inspirer de certains de ces travaux afin de proposer des fonctionnalités plus pertinentes, éviter certains écueuils ou plus directement utiliser des outils existants comme la librairie OpenCV.

- Exemples dans le jeux vidéo : Kinect, Eye-toy, CamSpace (techniques pour l'IHM, par exemple, mouvements continus ou immobilité sur une zone).
- Exemples issus de thèses et de projets de recherche (techniques de programmations).

FIGURE 2.1 – Exemple du Kinect

2.2 Gestion du projet

2.2.1 Choix stratégiques

Lors de ce projet, nous avons distingué deux parties majeures. La première est une bibliothèque de suivi d'objets. La seconde, une application qui utilise cette bibliothèque pour réaliser un logiciel de dessin interactif.
Ce choix présente plusieurs intérêts :

- Premièrement, ce découpage permet de bien différencier les tâches. Le développement de la librairie requiert des connaissances en traitement d'images pour tracer des couleurs/objets, appliquer des filtres, manipuler les images, etc. Ceci correspond parfaitement à la formation de deux membres du groupe(formation IMAGINA à l'UM2). Le développement de l'application requiert quant à lui des connaissances en génie logiciel, pour l'architecture de l'application, en IHM, pour la réalisation de l'interface (logicielle et gestuelle), ou encore en réseau pour la mise en place de l'architecture client/serveur. Ces différents points correspondent également parfaitement à la formation des deux autres membres du groupe. (Formation AIGLE à l'UM2). Ainsi, les tâches peuvent bien se répartir en fonction des connaissances et spécialités de chacun.

- Deuxièmement, l'intérêt de ce découpage est d'avoir deux projets à la fois complètement indépendants et complémentaires. En effet, la librairie est développée offre des fonctionnalités de suivi utilisables par n'importe quelle application. L'application se charger d'utiliser les fonctionnalités de traitement d'images de la librairie en ajoutant une interface graphique, une couche réseau, une connexion aux webcams, la récupération des images etc. La librairie et l'application sont donc deux projets développés en parallèle et très modulaires. Notre application peut très bien utiliser une autre librairie de traitement d'images, et la librairie peut très bien être utilisée par d'autres applications qui offriraient des fonctionnalités totalement différentes de la nôtre.

2.2.2 Diagramme de Gantt

Étant donné que nous avons proposé notre propre sujet, la première étape a été de rédiger un cahier des charges pour définir ce que nous voulions faire et de quelle manière.
Après ceci, nous avons commencé à effectuer des recherches du côté bibliothèque et application, pour savoir quels outils et techniques utiliser pour arriver à nos objectifs. Nous avons réservé plusieurs semaines pour effectuer des recherches et perfectionner notre idée du projet. Nous avons donc lu des articles de recherches sur des thématiques proches, effectuer des essais d'outils etc. Cette période d'analyse et de conception sert aussi à penser l'application en imaginant des cas d'utilisations (use cases) et en réalisant le diagramme de classes.

Par la suite, la majeure partie du temps est réservée au développement en lui même, en prenant en compte la date de présentation du projet pour avoir une réalisation complète.
Bien entendu ces temps de développement s'accompagnent de réunions avec tout le groupe de projet ainsi que de travail pour lier nos deux sous-parties et les tester ensemble.
Des documents comme le rapport ou la documentation furent produit au fur et à mesure du projet, pour rester en adéquation avec le travail accompli. Voici le rétroplanning qui illustre le déroulement chronologique du projet :

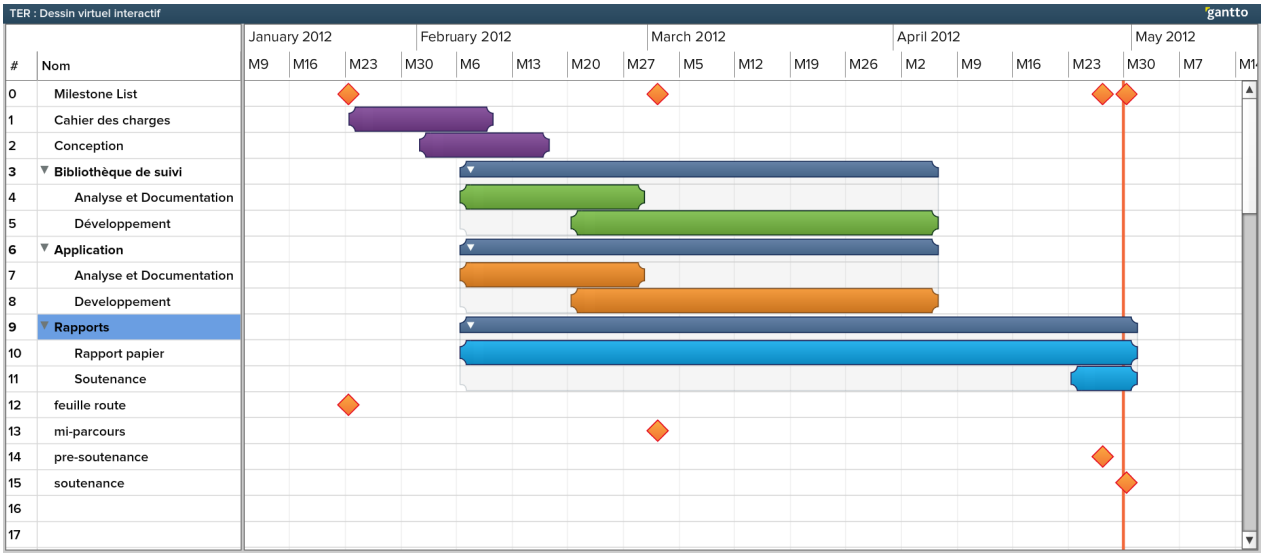

FIGURE 2.2 – Rétroplanning

2.3 Outils utilisés

Durant la période de recherche et au cours du projet nous avons utilisé différent outils (au sens large) pour nous aider dans la réalisation. Nous les détaillons ici dans cette section.

Outils collaboratifs

Tout d'abord nous avons utilisés des outils pour nous aider à synchroniser notre travail entre les deux sous-groupes.

 Subversion : Un gestionnaire de versions qui nous permettait de synchroniser notre travail, de partager le code source et de s'échanger quelques documents complexes (ce rapport par exemple).

 Gobby : Logiciel libre pour l'édition collaborative, qui nous permets de développer à distance sur le même document simultanément.
Bien sûr ces outils ne sont pas suffisants, nous avons donc fait des réunions régulières pour organiser le travail et décider des orientations, par ailleurs, nous avons fais beaucoup d'échanges pour que tout le monde soit toujours au courant des différentes communications importantes.

Outils techniques

Langage C pour la bibliothèque : ce langage est bien adapté pour écrire une bibliothèque car il nous permets de faire des fonctions, ce qui était dans notre idée ce qu'il fallait pour cette bibliothèque. Par ailleurs c'est un langage bien adapté pour faire du traitement de l'image car il est de bas niveau, et très rapide. De plus nous avons utilisé par la suite des librairies (OpenCV par exemple) qui ont été écrites initialement en C.
Langage C++ pour l'application : c'est un langage objet ce qui nous aidait pour la conception, il est également très rapide et c'est un langage avec lequel nous sommes à l'aise et nous avons de l'expérience, ça nous évitait donc de dévoir réapprendre et donc de perdre du temps précieux.

Pour ce projet nous utilisons plusieurs librairies pour nous aider :

 Qt : librairie d'IHM (et bien plus) en C++, nous aide pour faire les interfaces graphiques, pour la partie réseau également où elle offre des abstractions agréables.

 OpenCV : librairie de traitement de l'image, nous aide à exploiter la webcam et pour le traitement de l'image avec plusieurs fonctionnalités.

2.4 Analyse

2.4.1 Cas d'utilisations

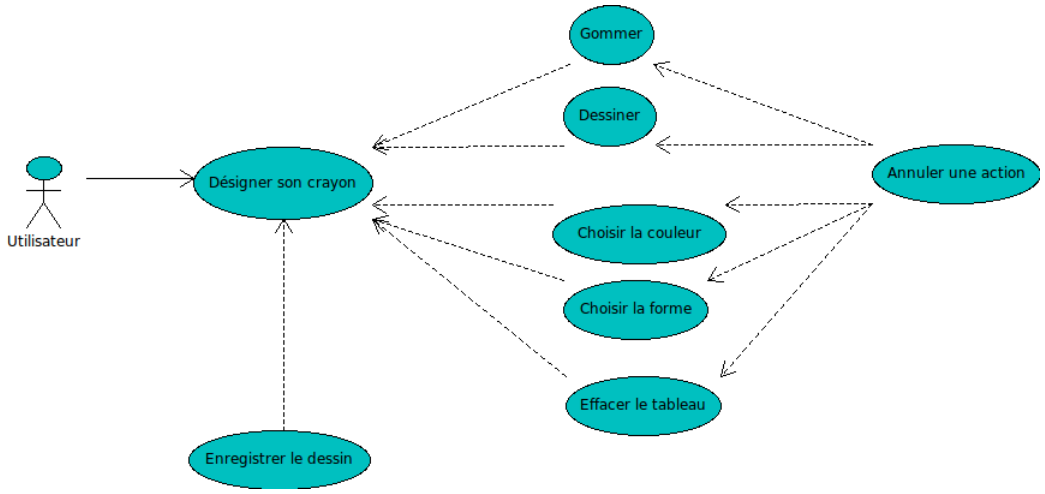

FIGURE 2.3 – Cas d'utilisation dessin

2.4.2 Diagrammes de séquences

Les diagrammes qui suivent expliquent le fonctionnement de l'application pour une utilisation locale, et une utilisation en réseau. On peut voir sur ce diagramme que le fonctionnement en local est assez basique. Dans un premier temps, l'utilisateur

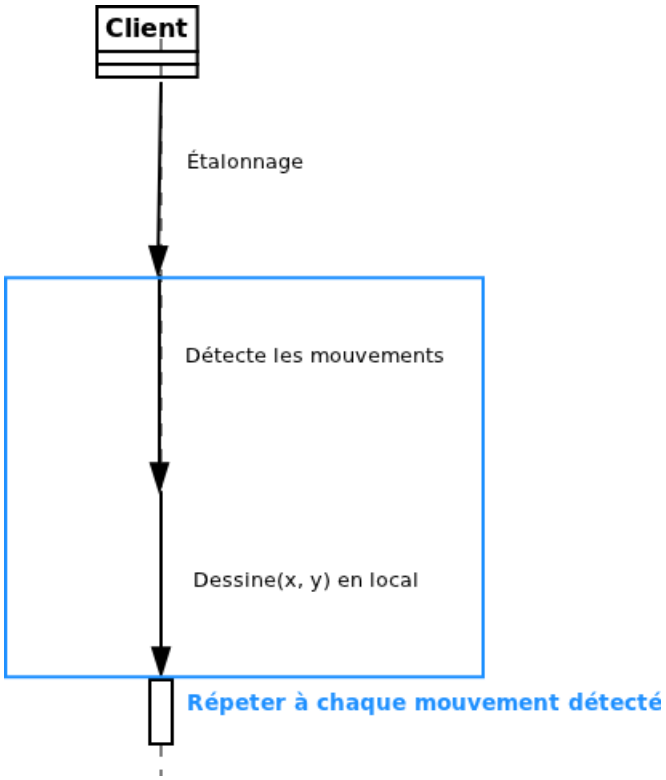

FIGURE 2.4 – Diagramme de séquence local

étalonne l'objet avec lequel il veut dessiner. Ensuite, lorsque l'utilisateur déplace l'objet en question, l'application **détecte** ce déplacement, et **dessine** à la nouvelle position de l'objet. Ce schéma de détection-dessin est répété jusqu'à fermeture de l'application.
Ainsi, l'utilisateur verra se dessiner chacun des mouvements qu'il effectue avec son objet.

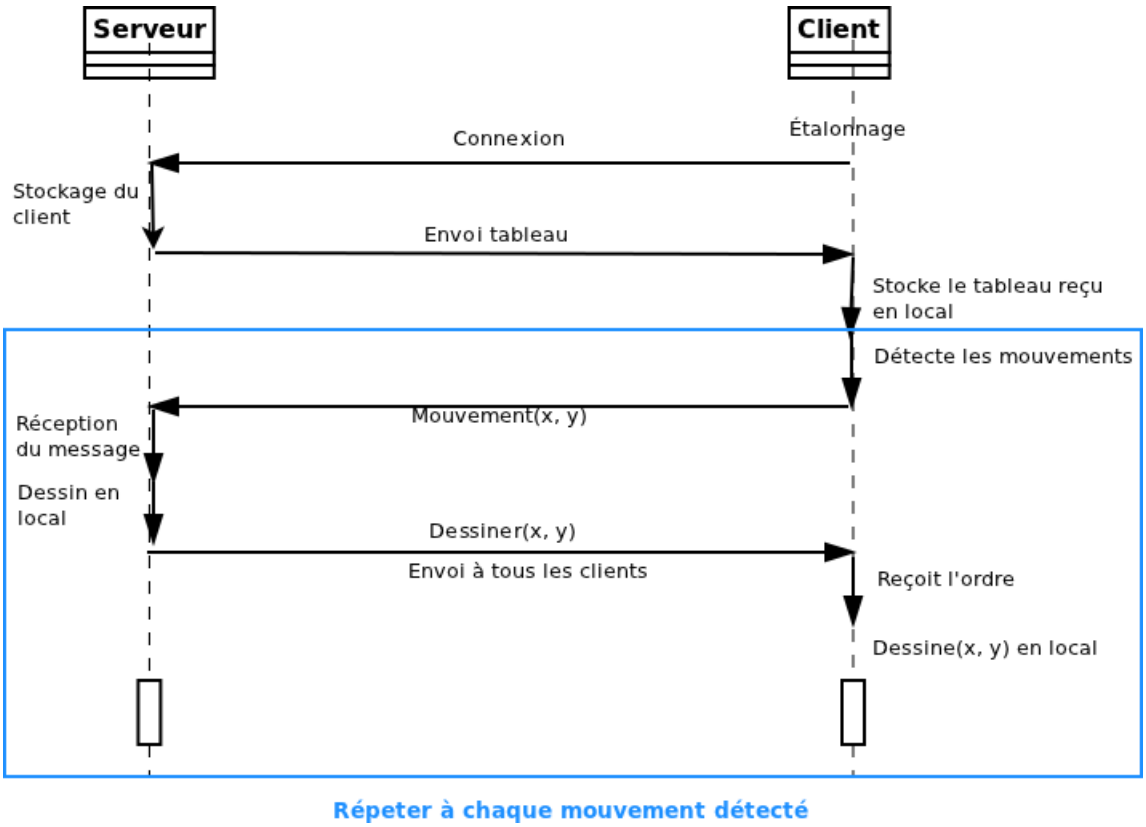

FIGURE 2.5 – Diagramme de séquence réseau

En réseau, l'utilisation suit le même schéma, mais un autre acteur entre en jeu, le **serveur**.
Comme pour une utilisation locale, l'utilisateur comment par **étalonner** l'objet à suivre. Puis, la connexion s'effectue auprès du serveur, qui va renvoyer le dessin courant au nouveau client. Une fois en possession du dessin, le client le stocke et peut rentrer dans la boucle de détection-dessin :
Comme en local, l'application va **détecter** le mouvement de l'objet suivi. La nouvelle position est envoyée au serveur qui va dessiner sur son propre tableau, puis renvoyer à tous les clients l'ordre de **dessiner** à cette même position. Chaque client verra donc ce que les autres clients dessinent.
Ainsi, l'utilisateur verra se dessiner chacun des mouvements qu'il effectue avec son objet, mais égelement ceux de tous les autres clients.

On peut donc voir que pour l'utilisateur, le fonctionnement est identique qu'il dessine seul, ou avec d'autres personnes :

- Il étalonne l'objet
- Il déplace l'objet
- l'objet dessine sur le tableau

Cependant, derrière, le fonctionnement n'est pas le même. En réseau, chaque mouvement passe par le serveur qui se charge de faire le lien avec tous les clients de l'application.

2.4.3 Diagrammes de classes

La bibliothèque fonctionne grâce à une structure de données utilisée par deux fonctions enveloppes.
Le curseur est représenté en mémoire sous la forme d'une strucure de données "Cursor".
Cette structure de données est engendrée par la fonction "calibration" de la librairie. Elle est ensuite utilisée par la fonction "track", qui permet la mise à jour des informations relatives au curseur par rapport à une image.

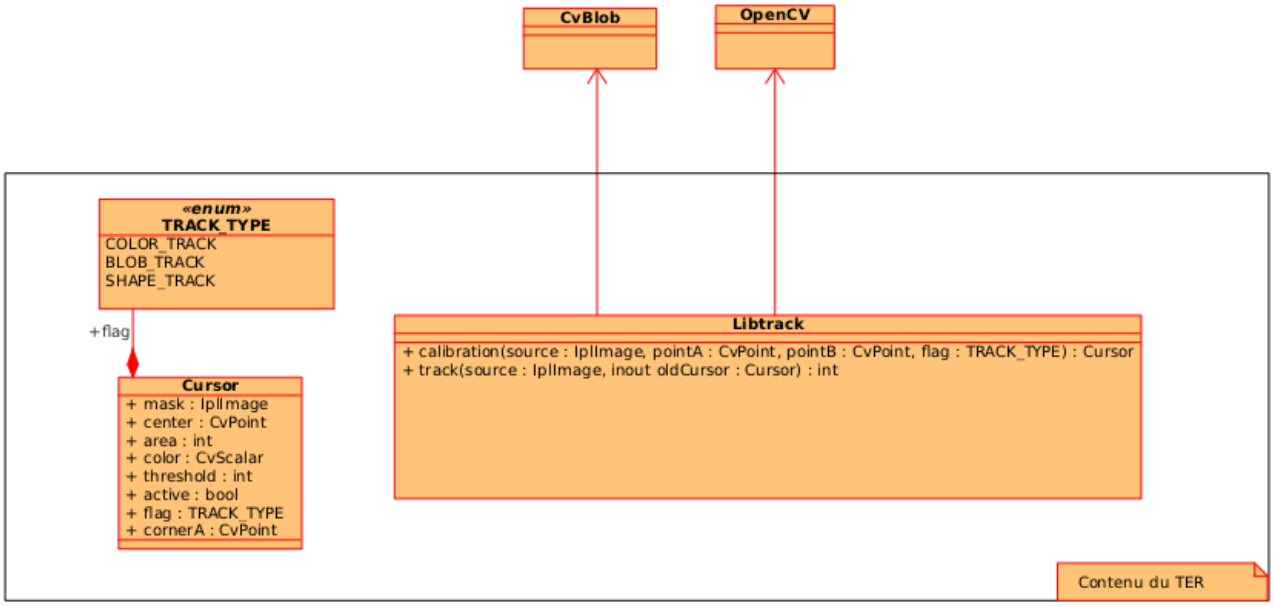

FIGURE 2.6 – Architrecture de la bibliothèque

La bibliothèque dépend de deux bibliothèques externes :
– OpenCV, Spécialisée dans le traitement des images
– CvBlob, librairie issue d'OpenCV utlisée pour la recherche d'objet dans des images binaires

Le diagramme qui suit vous présente l'architecture de notre application.
Avant de commencer l'application, nous avons décidé de passer un long moment sur la réflexion pour avoir une architecture solide et modulable.
Nous avons dans un premier temps identifié les différents modules dont serait composée notre application et quel était précisément le ou les rôles de chacun.
Ensuite, nous avons réfléchi à quel était le meilleur moyen dont ces différents modules pouvaient communiquer en évitant toute redondance ou communication superflue.
Une fois cette réflexion faite, nous avons dessiné le diagramme suivant :

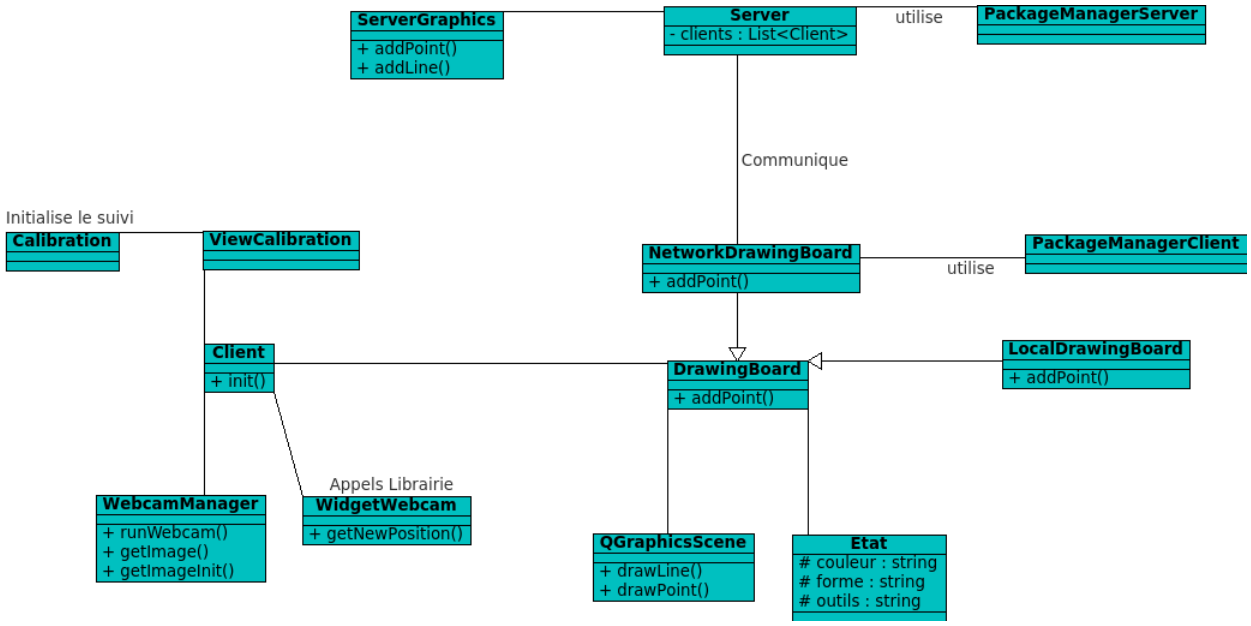

FIGURE 2.7 – Diagramme de classe de l'application

On peut identifier quatre modules principaux :

- **Le module d'étalonnage** (Calibration et ViewCalibration). Ce module sert à tout ce qui touche à l'étalonnage de l'objet. Une fois qu'il est effectué, le module va appeller le module *client* pour lui signaler que l'étalonnage est fait, et lui envoyer les informations qui correspondent (objet à suivre, type de track, dessin en local ou réseau, etc...)
- **Le module client** (Client). Ce module est le coeur de l'application, il a principalement deux rôles. Il va dans un premier temps construire toute l'interface graphique (tout ce qui est visuel) qui sera présentée à l'utilisateur. Ensuite, c'est ce module qui va assurer la communication entre tous les modules. Il est le seul intermédiaire entre les différents modules locaux.
- **Le module de dessin** (DrawingBoard, LocalDrawingBoard et NetworkDrawingBoard). Ce module a également deux rôles. Il va s'occuper de toute la partie dessin (Stocke la couleur et taille du pinceau, dessine les points, lignes, etc...). Aussi, pour une utilisation en réseau, c'est lui qui va se charger de communiquer avec l'application serveur.
- **Le module serveur** (Server, ServerGraphics). C'est ce module qui met en relation les différents client, et qui synchronise le dessin. Il recoit les ordres de dessin, les stocke, et les renvois à tous les autres clients.

La classe **WebcamManager** est la seule classe qui va communiquer avec le **matériel** (ici les webcams). Lorsque l'on veut une nouvelle image de la webcam, le module client va demander à WebcamManager la nouvelle image, et l'envoyer au module qui en a besoin.
La classe **WidgetWebcam**, elle, est la seule classe qui va communiquer avec la **librairie de suivi**.
Les classes QGraphicsScene et Etat représentent respectivement le dessin et l'état actuel du pinceau.
Les classes PackageManager* servent à assembler et désassembler les paquets qui transitent sur le réseau.

Scénario :
Afin d'illustrer la modularité de notre application, on pourrait imaginer le scénario suivant :
- L'étalonnage s'effectu. Une fois terminé, le module d'étalonnage va prévenir le client, et lui donner les informations.

- Chaque X millisecondes, le client va intérroger WebcamManager pour obtenir une nouvelle image.
- Le client va envoyer cette image à WidgetWebcam pour obtenir la nouvelle position. (Grâce à la librairie de suivi)
- Le client va maintenant envoyer cette position à DrawingBoard qui va dessiner le nouveau point en fonction de la couleur la taille, etc...

On voit ici que chaque module rempli bien son rôle et assure le bon fonctionnement général de l'application.

Une fois cette phase de réflexion terminée, toute l'architecture de la librairie, et de l'application était pensée et écrite.
Nous pouvions alors débuter la phase de réalisation, programmer et imbriquer les différents modules, et s'attaquer aux problèmes plus techniques.

Chapitre 3

Réalisation

3.1 Bibliothèque de suivi

Cette partie du développement a pour objectif de fournir un ensemble de fonctionnalités permettant la reconnaissance et le suivi d'un objet, que nous appelerons ici Curseur, à partir d'images.
L'objectif est de développer notre propre bibliothèque pour d'une part avoir un outil remplissant précisément nos objectifs et d'autre part de faire un travail de recherche et développement dans le domaine du traitement de l'image (en particulier le suivi d'objets).
Un certain nombre de librairies existantes proposent des moyens nous facilitant la tâche, notamment grâce à des outils de reconnaissance de formes, de suivis de modèle ou d'évaluation de composantes connexes. Le travail consiste donc à tirer partie de ces bibliothèques en les associant de manière à proposer des méthodes efficaces et simples à utiliser. Cela implique une bonne connaissance des outils existants, de leur compréhension et de leur utilisation conjointes dans le but de faire notre propre bibliothèque.
La Bibliothèque de suivi sert à réaliser le suivi d'un objet, que nous appelerons ici Curseur, avec un minimum de connaissance en traitement des images.

La bibliothèque fonctionne grâce à une structure de données manipulée par deux fonctions enveloppes.
Le curseur est représenté en mémoire sous la forme d'une strucure de données "Cursor".
Cette structure de données est engendrée par la fonction "calibration" de la librairie. Elle est ensuite utilisée par la fonction "track", qui permet la mise à jour des informations relativess au curseur.

Structure de données

Fonctions

3.1.1

3.2 Application

Application :

L'application doit exploiter le plus efficacement possible la librairie de reconnaissance. Nous voulons donc mettre en place un étalonnage de l'objet à suivre le plus précisément possible. Le projet doit ensuite être exploitable et utile, nous devons donc mettre en place un certain nombre d'outils tels que la possibilité de gommer, changer de forme, de couleur etc. Un module réseau est aussi prévu pour une utilisation concurrente et synchronisée. Et bien sûr la fonctionnalité clé de l'application est le tableau virtuel, qui devra donc être exploitable et le dessin devra être fidèle aux mouvements reconnus.

Chapitre 4

Résultats

4.1 Méthodes de suivi

4.2 Application

Chapitre 5

Conclusion

5.1 Difficultés rencontrées

5.2 Perspectives

5.3 Conclusion

Chapitre 6

Références

Première partie

Annexes

Annexe A

Documentation de la librairie