École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

5ème année

2009/2010

Projet de fin d'études
Rapport Final

Reconnaissance de logos par iPhone

Encadrants
Mathieu DELALANDRE
mathieu.delalndre@univ-tours.fr
Nicolas SIDERE
nicolas.sidere@univ-tours.fr
Jean-Yves RAMEL
jean-yves.ramel@univ-tours.fr

Etudiant
Taleb MOHAMED EL WELY
taleb.wely@gmail.com
DI5 – 2009/2010

Laboratoire de Reconnaissance de Formes et Analyse d'Images
Université François Rabelais, Tours

Version 3.0 – 29/05/2010


SFR
02:04
iReLog
Reconnaissance de Logos par iPhone
Conception et Développement
Taleb MOHAMED EL WELY
Sous l'encadrement de
Mathieu DELALANDRE
Nicolas SIDERE
Jean-Yves RAMEL
RF AI
POLYTECH
TOURS

Table des matières

Introduction

En dernière année du cycle d'ingénieur du département informatique de l'École Polytechnique de Tours ; nous, étudiants, sommes confronté à la résolution d'un problème technique/technologique et/ou théorique, et ce en vue de l'obtention de notre diplôme.

Sans entrer dans les détails de ce type de problèmes, il faut souligner que ce sont des problèmes dont la complexité et la grandeur sont comparables à ceux qu'on peut rencontrer dans l'industrie.
Certains projets sont même réalisés pour le compte de grands groupes industriels et peuvent/vont être exploités, d'autres dans un but de recherche… La résolution de ce problème s'inscrit dans le cadre d'un projet intitulé projet de fin d'études.

Ce projet de fin d'études vise à mettre l'étudiant dans une vraie situation de gestion de projet en faisant intervenir tant l'aspect organisationnel (délais, ressources, documents…) que technique (solutions, environnements…).

Mais avant d'entrer dans les détails de ce document, certaines définitions et conventions doivent être mises en exergue, et ce pour ne pas avoir des abus de langage.

Ainsi par la suite **projet** signifiera le présent projet de fin d'études dont le thème sera défini un peu plus loin.
Tout projet fait entrer en jeu un maître d'ouvrage et un maître d'œuvre.
Le maître d'œuvre est l'étudiant auteur qui doit proposer et mettre en œuvre une solution pour mener à bien le projet.
Le maître d'ouvrage est le professeur qui a proposé le sujet et sera dénommé **le client** dans la suite du document. Il a pour rôle aussi de valider et de relire tout document avant son officialisation.
Ce même professeur aura aussi comme rôle d'amener des conseils tant organisationnels que technique pour la maîtrise d'œuvre. Dans ce rôle il sera appelé **l'encadreur** par la suite. Il a pour rôle aussi de valider et de relire tout document avant son officialisation.

Ces trois entités travailleront en collaboration tout au long du projet.
N'oublions pas la notion d'**utilisateur** qui est le client du client du projet, en d'autres termes c'est l'utilisateur final.

Une fois ceci dit, Il a été convenu de rédiger en collaboration avec l'encadrant un rapport donnant une vision transversale du projet, permettant ainsi de faire un lien explicite entre les livrables en synthétisant, les objectifs, la solution finale, son degré de finalisation et ce qu'il reste à faire.

1 Rappels importants

1.1 Présentation du projet

Il s'agit, comme il a été clairement défini dans le cahier de spécifications de développer une application iPhone qui sera en mesure de donner à une personne possédant un iPhone toutes les informations sur un logo qu'elle prend en image.
Le schéma ci-contre montre le processus de la reconnaissance d'un logo sur un t-shirt, dès la capture de l'image sur une affiche publicitaire jusqu'à l'obtention des informations sur ce t-shirt :

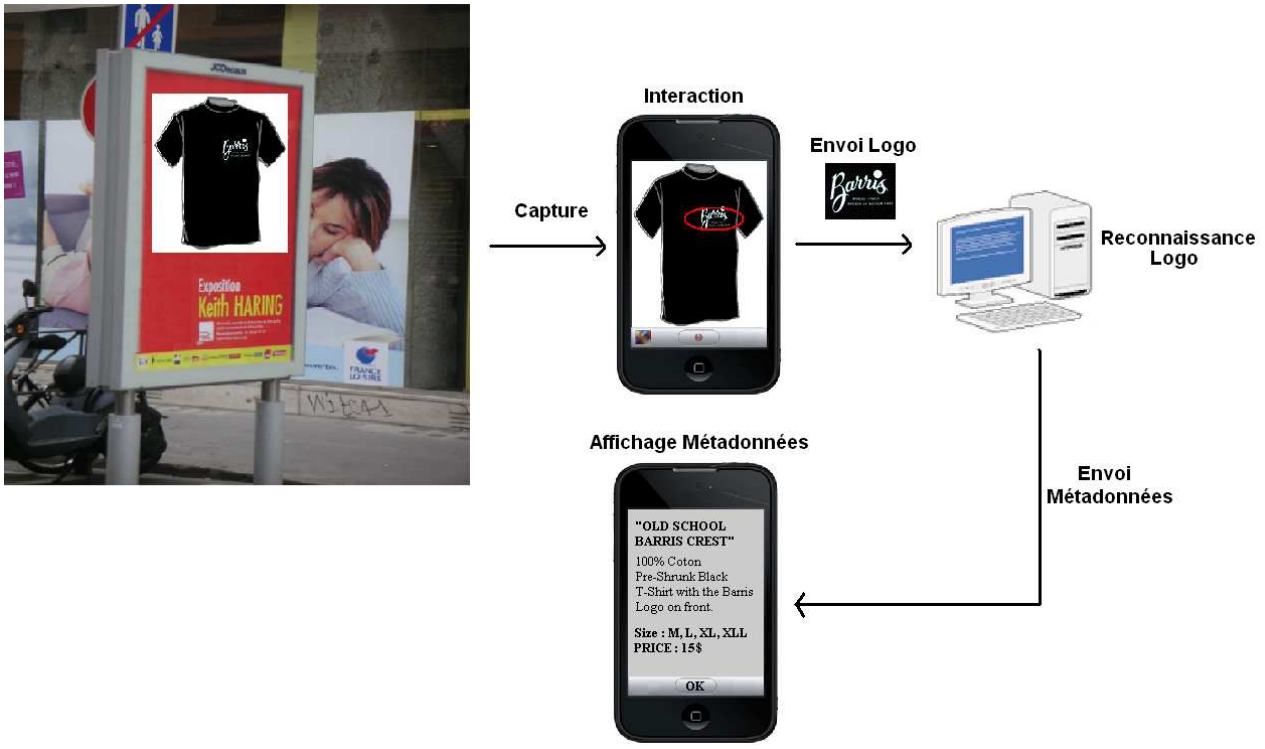


Figure 1 : Processus de reconnaissance de logo

Dans le cas où le logo n'a pas été reconnu, le serveur devra renvoyer l'image à l'application en vue d'un traitement par une seconde interface plus riche en outils avant une seconde tentative de reconnaissance.

Pratiquement, le client souhaite que l'application iPhone soit la plus fluide possible. Celle-ci doit permettre à son utilisateur de pouvoir envoyer l'image le plus rapidement possible en un nombre de gestes très limité. C'est au serveur ensuite de prendre la relève.
Ceci s'explique. Le serveur est beaucoup plus puissant que l'appareil iPhone et son travail est transparent à l'utilisateur. En d'autres termes pour gagner en fluidité on peut ramener le gros du travail au serveur sans lui envoyer par contre une image inexploitable. [Paragraphe tiré du CSPD]

1.2 Objectifs principaux

Dans le cahier de spécifications et plan de développement, il a été précisé que la priorité sera de concevoir et de réaliser un système minimum mais opérationnel. C'est à dire la priorité réside à concevoir et à développer la première interface, la communication entre cette interface et le serveur et le retour des résultats par le serveur, suivi d'une validation par l'utilisateur.

L'objectif a été rempli et l'application ainsi réalisée permet de capturer une image et de colorier la partie du logo. Notons que tous les types de coloriage sont pris en compte : faire un cercle ou un quadrilatère autour du logo, colorier entièrement le logo…
Suite à ce coloriage l'application réalise un certain nombre de traitements (que nous verrons plus tard dans le présent rapport) afin de bien tirer avantage de cette interaction avec l'iPhone.

Nous ferons le point à la fin de ce document sur l'utilité de cette interaction.

1.3 Objectifs secondaires

Toujours dans le cahier de spécifications et plan de développement, il a été défini un certain nombre d'hypothèses sur quelques développements de fonctionnalités.
Ces développements, comme convenu, sont conditionnés par l'avancement du projet.
En effet si le serveur ne reconnait pas l'image, il a été prévu de développer une fonctionnalité permettant à l'utilisateur de renseigner les métadonnées concernant son image requête.

A l'heure où ce document est rédigé, cette fonctionnalité n'a pas été encore développée, vu les retards que l'installation de l'environnement de travail ont occasionnés.
Cependant, comme que les objectifs principaux sont remplis et qu'il reste une semaine de développement il se peut que cette fonctionnalité voit le jour.

1.4 Autres documents

Ce document est en étroite relation avec trois autres documents :

- Le cahier des spécifications et plan de développement (CSPD)
- Le cahier des livrables d'analyse (CLA)
- Le cahier des tests (CT)

Ainsi tout au long de celui-ci le lecteur pourra être amené à visiter l'un d'eux pour avoir plus d'informations.

Les sections suivantes vont présenter les modifications apportées au planning, à la gestion de projet et à la conception de l'application.
En effet le développement d'applications chez Apple, qui était méconnu par les parties prenantes du projet a nécessité la revue de toutes ces parties.

Commençons justement par étudier l'installation de l'environnement de développement ainsi que de tous les outils qui ont été nécessaires pour la réalisation de l'application.

2 Technologies

J'ai testé plusieurs technologies afin de choisir celle la mieux adaptée pour la réalisation des objectifs et ce dans les délais.

2.1 Cabrera

La méthode Cabrera consiste à jailbreaker l'iPhone et développer avec des outils libres tel que Eclipse. Le langage de programmation utilisé est le C++. L'auteur de cette méthode explique sur son site comment arriver à installer une application développée sur Eclipse dans un environnement Unix/Windows.
Cependant cette méthode présente plusieurs inconvénients dont on peut citer :

- Elle n'est pas mise à jour régulièrement et fonctionne rarement, de ce fait elle n'est pas fiable.
- Aucun outil de développement adéquat n'est présent. Par exemple pour développer une simple interface graphique il faudra écrire tout le code, ce qui peut prendre énormément de temps…

2.2 Open Source

Il existe d'autres méthodes semblables à la méthode précédente. Par exemple sur le site de la communauté de développeurs de Google, plusieurs projets sont présents. Cependant ils présentent les même inconvénients que celle de Cabrera.
J'ai passé 4 semaines à tester ces deux méthodes. Les résultats n'étaient pas satisfaisants pour ne pas dire médiocres.

2.3 Adobe

Adobe a introduit dans sa suite CS5, un plugin permettant de compiler des applications iPhone natives.
Ce plugin n'a pas été testé dans le cadre de ce projet vu le retard occasionné par les deux méthodes précédentes.
Il a connu un succès temporaire car la qualité des applications obtenues était satisfaisante.
Le langage de programmation utilisé est l'Action Script.
Cependant, ce projet a été arrêté suite à un problème juridique entre la firme et Apple.

2.4 Apple

Pour développer sous iPhone, il faut absolument avoir un mac, un iPhone et une licence de développement. Pour avoir une licence il faut s'inscrire sur l'un des quatre programmes de développement proposés par Apple.

2.4.1 La licence

Il existe principalement trois types de programmes :

- Développer Program : Permet à une et une seule personne de pouvoir développer sur l'iPhone. Ce programme coûte 79€/an.
- University Program : Permet à une université, école ou institut d'enregistrer ses étudiants pour pouvoir développer des applications non commercialisables. Ce programme est gratuit mais peut prendre plusieurs semaines voire plusieurs mois avant qu'Apple valide les licences.
- Entreprise Program : Permet à une entreprise d'inscrire ses employés au programme. Les applications de ce programme peuvent être commercialisées sur l'AppStore ou distribuées en interne vers des bêta-testeurs. Ce programme coûte 299€/an.

Au début du projet, j'ai passé un mois pour tester une éventuelle possibilité de développer sur iPhone avec des outils Open Source mais en vain. Quelques rares amateurs ont réussi à installer une application sur l'appareil. Mais ces applications sont illégales, non reconnues par Apple et non commercialisables (même si la finalité de ce projet n'est pas commerciale). Un autre inconvénient c'est qu'avec cette méthode on n'a pas accès à l'environnement de développement dédié à l'appareil.
Pour plus d'informations sur ces méthodes, il faut se reporter au gestionnaire du projet (voir CSPD).
J'ai fini par m'inscrire, dans le premier programme car il était impératif de commencer le développement.

2.4.2 Les outils

Sans trop rentrer les détails, XCode est l'environnement de développement principal d'Apple. Il contient toute une suite logicielle permettant facilement de créer des différents types de projets. Il permet aussi de transférer l'application sur l'iPhone pour la tester, même s'il contient un simulateur d'iPhone.
Le simulateur en effet ne peut pas simuler un appareil photo qui était essentiel dans ce projet. Il ne permet pas non plus de détecter les différentes fuites de mémoire. Pour cela XCode fournit un utilitaire Objects Alloc qui permet de voir si l'application consomme beaucoup et si elle libère la mémoire. XCode fournit aussi Interface Builder qui permet de générer rapidement des interfaces graphiques ergonomiques et très riches.

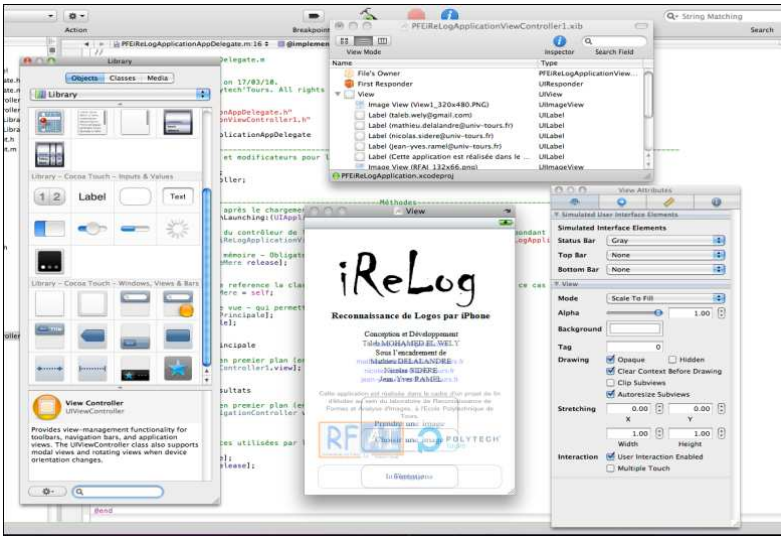


Figure 2 : XCode et Interface Builder

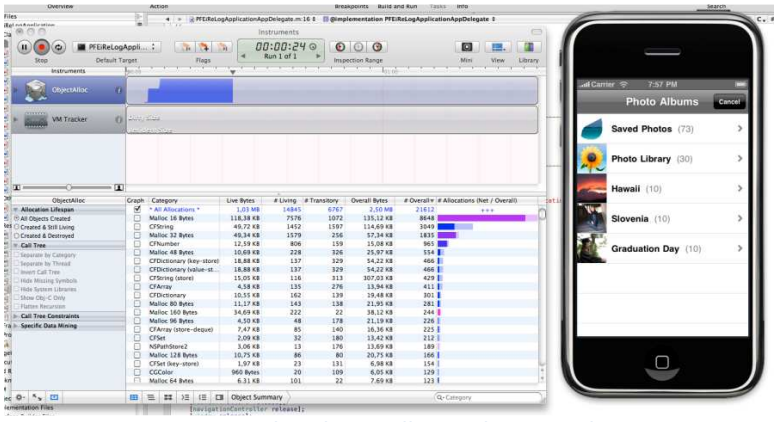


Figure 3 : Outils - Objects Alloc & iPhone Simulator

2.4.3 Commencer à développer sur iPhone

Une fois que la licence est obtenue et installé sur le poste de travail (voir le manuel d'installation) il faut lier l'application, l'iPhone et le poste de travail à l'aide de ce qu'Apple appelle un Provisionning Profile.
Cela commence par aller sur le portail de développement et inscrire le téléphone et l'application, fournir la licence et télécharger le Provisionning Profile.

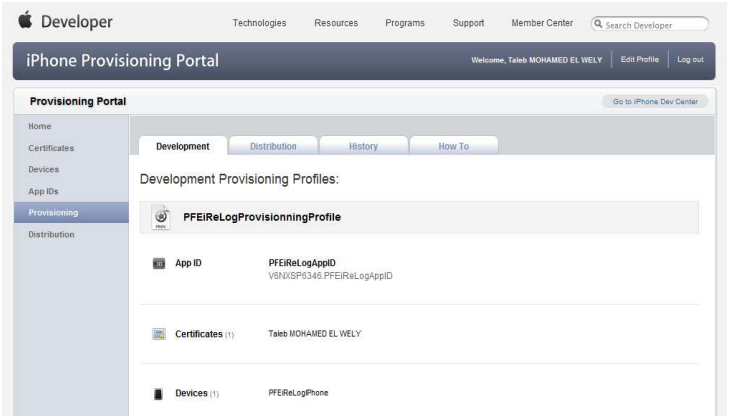


Figure 4 : Création d'un Provisionning Profile

Le schéma ci-dessous illustre le processus d'une distribution Ad Hoc (pas de soumission à l'AppStore).

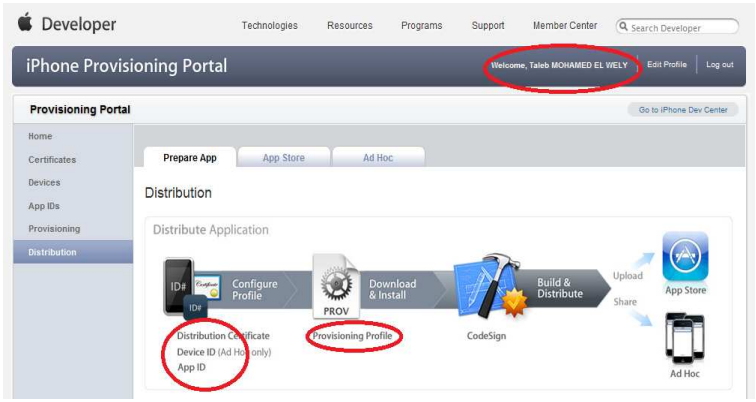


Figure 5 : Processus de distribution d'une application iPhone

2.4.4 Comparatif

Le tableau ci-dessous récapitule les caractéristiques majeures des technologies étudiées.

Technologie	Langage	Plateforme	Licence
Cabrera	Java	Unix/Windows	Freeware
Open Source	C/C++	Unix/Windows	Freeware
Adobe	ActionScript	Windows	Commercial
Apple	**Objective-C**	**Mac OS**	**Commercial**

3 Gestion de projet revisitée

Comme que le développement avait pris du retard à commencer, il fallait revoir toute la partie gestion de projet car le planning ne pourra plus être respecté.
Ainsi dans le CSPD il était prévu de fournir des livrables et des versions de l'application à des dates clés.
Par la suite nous avons, avec le client, adopté une méthodologie itérative.

C'est à dire qu'à partir du CSPD, le développement d'une fonctionnalité est réalisé et le client la teste. Si elle est validée le passage à la suivante est fait, sinon une autre version est produite et le processus reboucle.
Tout ceci bien évidemment dans le respect strict du CSPD et dans un but de rattraper le retard occasionné par l'attente de la licence de développement.

Cette méthodologie a montré ses forces et permit d'avoir un système complet et opérationnel avant la deadline finale du projet. Et pour rappel il serait même envisageable de réaliser les deux fonctionnalités qui étaient conditionnées par l'avancement du projet.
Ci-dessous le diagramme de Gantt initial avec l'état d'avancement des tâches.
En rouge : tache incomplète
En bleu : tâche réalisée

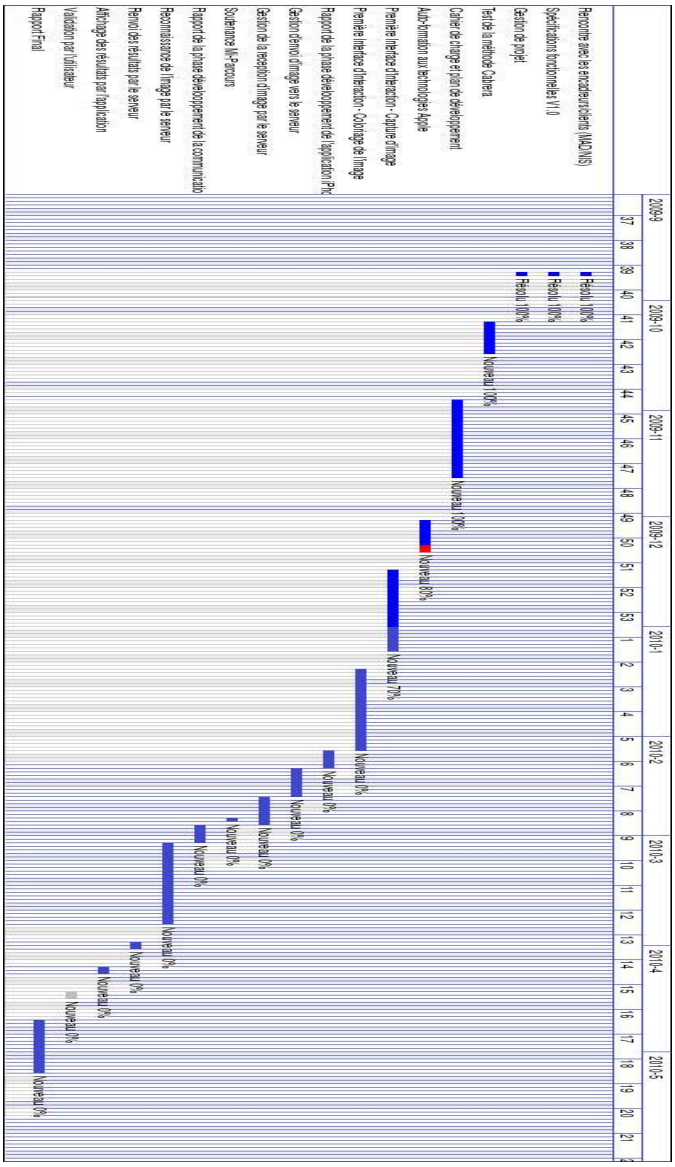


Figure 6 : Avancement du projet

4 Conception revisitée

Dans cette partie nous nous focaliserons plus sur la partie iPhone, la partie serveur ne possédant qu'une seule classe et sera abordée plus en détail dans le cahier des livrables d'analyse.

Au cours du développement certaines classes ont été rajoutées à l'application iPhone, d'autres ont été revisitées afin d'assurer le bon fonctionnement de l'application.

Ci-dessous le diagramme des classes final de l'application iReLog.

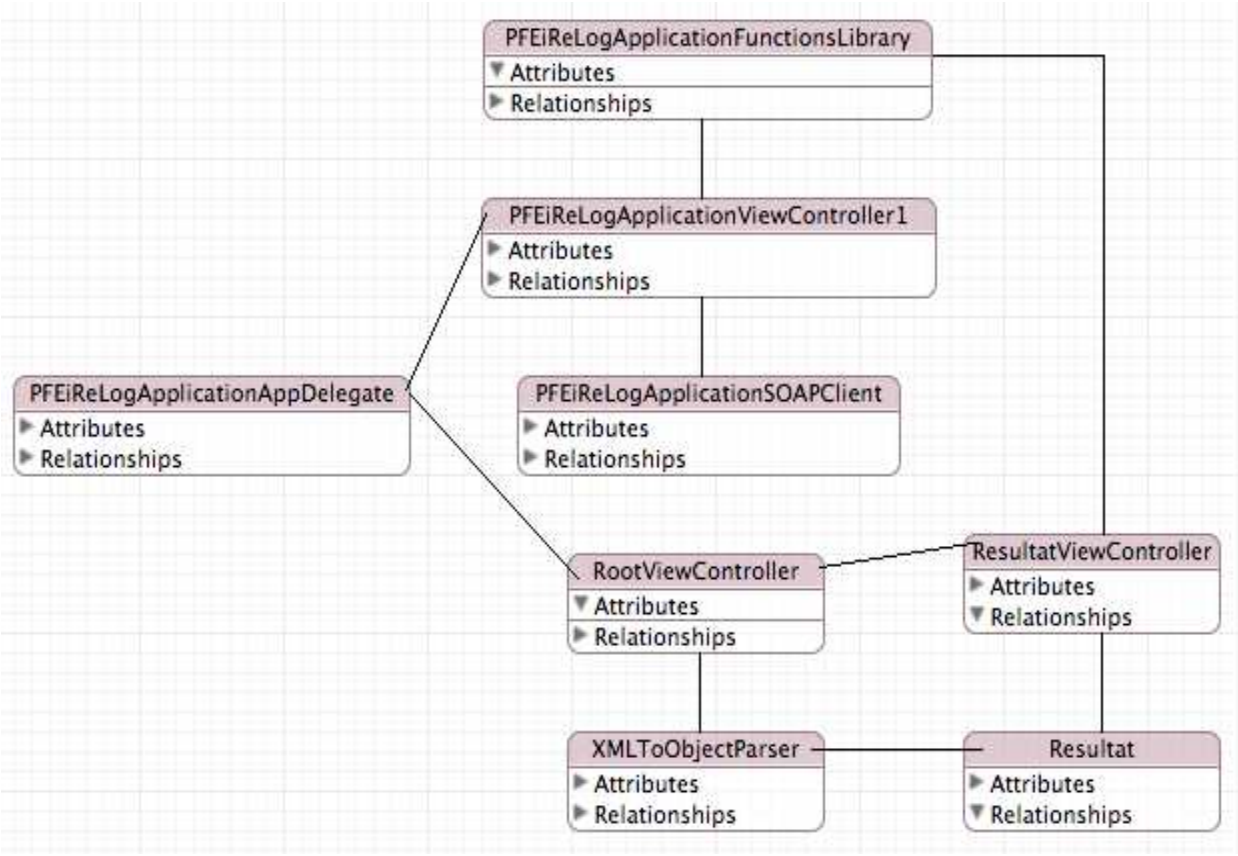


Figure 7 : Diagramme des classes d'iReLog

La classe PFEiReLogApplicationAppDelegate est la classe délégué de l'application, c'est elle qui crée l'objet fenêtre qui abritera les vues.

Ce diagramme est détaillé dans le cahier des livrables d'analyse. Il s'agit ici de montrer les classes développées pour avoir une application fonctionnelle.

5 Système réalisé

Dans cette partie nous allons plus rentrer dans les détails du système réalisé et de son fonctionnement.
Le système est composé de plusieurs entités communiquant entre elles. Toutes ces entités ainsi que leurs caractéristiques seront détaillées plus tard dans ce document.
Son séquencement est expliqué un peu plus loin.

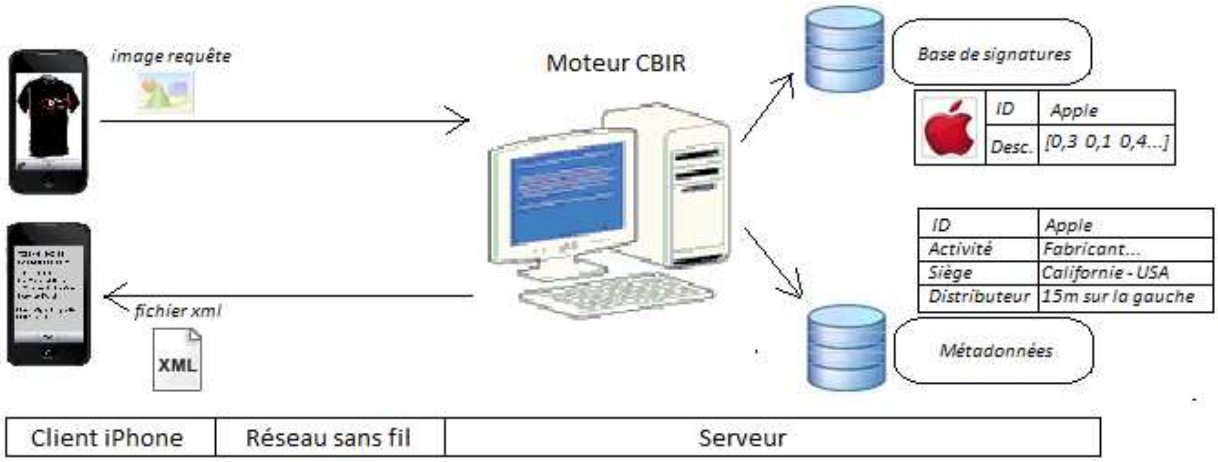


Figure 8 : Système réalisé complet

Le système est divisé en trois parties. Pour chaque partie, seront présentés les objectifs, l'étude et la réalisation.

- L'interface d'interaction (iPhone du dessus présent sur l'image ci-dessus)
- Le serveur (partie droite de l'image)
- L'interface d'interaction (iPhone du dessous présent sur l'image ci-dessus)

Afin de mieux suivre les étapes de la réalisation, nous allons remettre le schéma ci-dessus au début de chacune des trois parties et encadrer la partie concernée.

5.1 Interface d'interaction

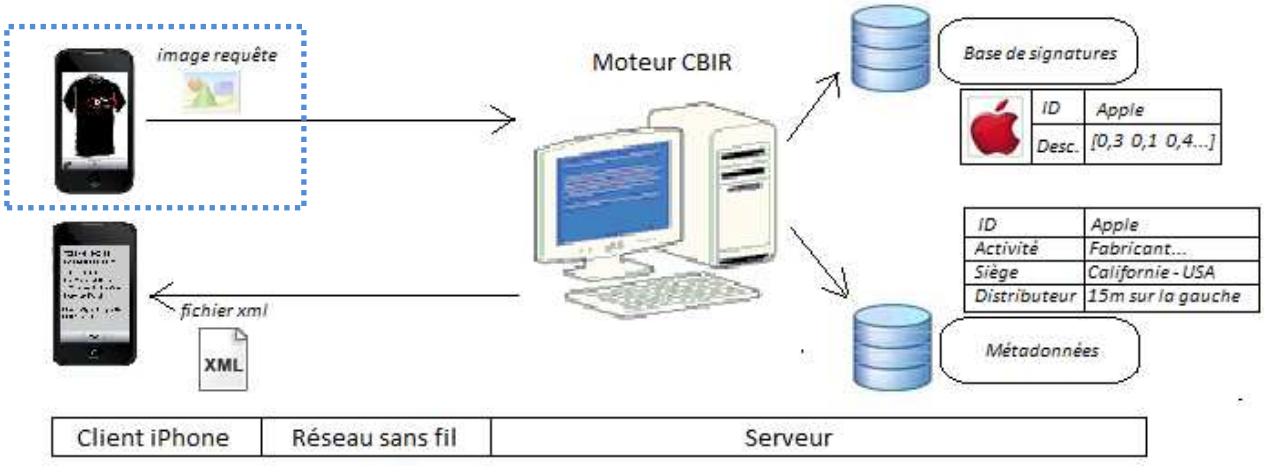
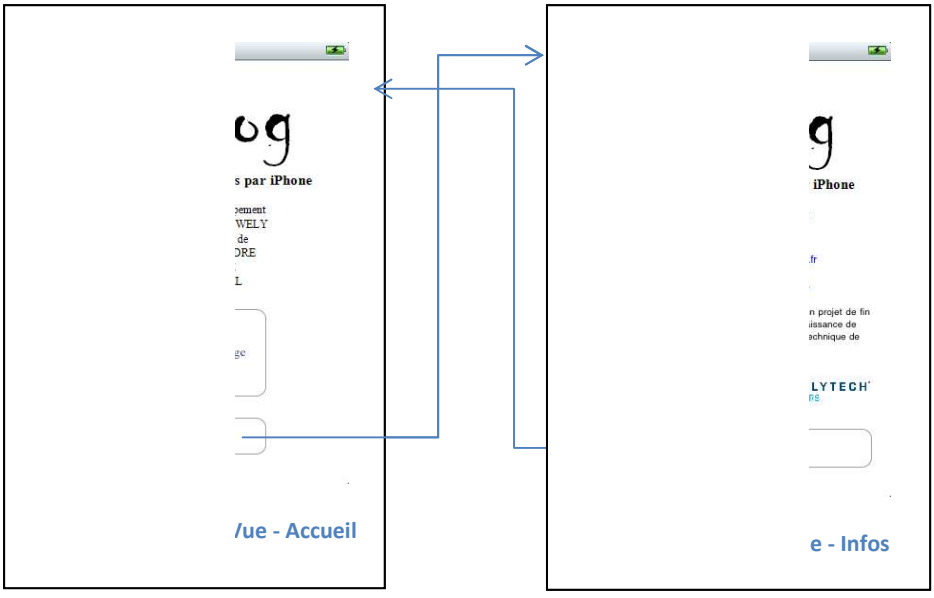


Figure 9 : Système réalisé complet – Partie Interface d'interaction

5.1.1 Capture et coloriage

Une application iPhone est composée de plusieurs vues de type UIView qui sont contenues dans une fenêtre de type UIWindow. Au démarrage le délégué de l'application commence par créer une fenêtre et une vue qu'il affiche. Cette vue correspond à la vue d'accueil.

C'est la première vue d'iReLog, elle permet à l'utilisateur d'avoir le choix de visualiser les informations concernant l'application ou de capturer directement une image en vue d'une reconnaissance.
C'est un objet de type UIView et contrôlé par un objet de type UIViewControler. Celui-ci est un contrôleur de vue basée sur un modèle MVC (voir CSPD).
Un appui sur le bouton « Informations » permet de rendre invisible les boutons et d'afficher un texte descriptif de l'application ainsi que d'autres informations.
Un appui sur le bouton « Retour » permet de revenir sur la vue d'accueil.

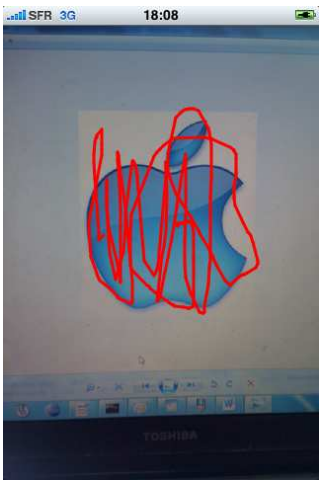


Cette vue a été faite via l'outil Interface Builder d'XCode. Son implémentation a durée 1 journée de développement.
Un appui sur le bouton « Prendre une image » permet de dire au contrôleur de la vue de tout masquer, d'afficher, toujours sur cette même vue, un objet de type UIImagePicker permettant de capturer une image et donner la main à son contrôleur qui est de type UIImagePickerControleur.

C'est là que l'utilisateur pourra capturer une image. Une fois l'image capturée elle est figée (conformément au CSPD) afin que l'utilisateur puisse colorier la partie où le logo est présent.
Apple a implémenté toute la logique métier dans la classe UIImagePickerControleur fin de prendre l'image, après appui sur l'icône au milieu de la vue ci-dessous.
Ce contrôleur possède une fonction membre permettant de retourner un objet image qui est l'image capturée.

re 12 : iPhone - Vue – Capture d'image

L'image est ensuite figée dans un objet de type UIImageView. Un objet UITouch est ensuite instancié pour permette l'interaction avec l'écran. L'utilisateur peut ensuite commencer à colorier la partie l'intéressant.

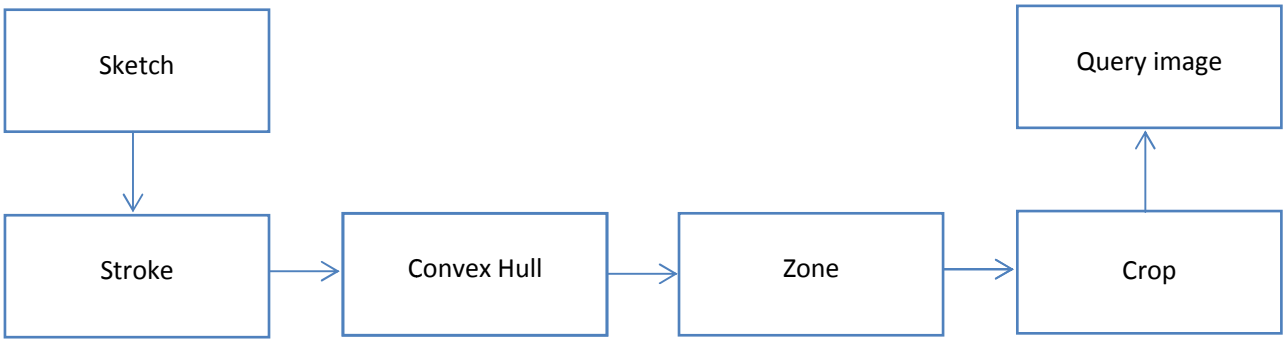


re 13 : iPhone - Vue – Coloriage d'image

5.1.2 Constitution de l'image requête

Pour des raisons ergonomiques et sous la demande du client, l'utilisateur final devra avoir les résultats avec le moins de clics possibles. Dans le CSPD une partie traite de cette problématique.
La solution, étudiée et réalisée, consiste à détecter la fin du coloriage, c'est à dire le moment où l'utilisateur relâche l'écran et de lancer les traitements nécessaires d'une façon transparente à l'utilisateur.

L'application commence par récupérer l'image (sketch) obtenue dans la section, et les coordonnées des points effleurés par l'utilisateur suite à son coloriage (stroke).
L'application commence par calculer une enveloppe convexe (Convex Hull) du nuage des points effleurés afin de déterminer le contour du logo.
Elle réalise ensuite un crop basée sur les extremums de ce nuage de points avant de générer une image requête (Query image) prête à l'envoi au serveur.
Le schéma ci-dessous illustre ce traitement.



5.1.2.1 Calcul de la Convex Hull

Le calcul de l'enveloppe convexe été réalisé par l'implémentation d'une variante de l'algorithme de Graham Scan : La Monotone Chain.
L'algorithme commence par trier les points selon les abscisses croissantes, en cas d'égalité entre deux points le tri se fait sur les ordonnées croissantes.
On calcule ensuite les valeurs des xMin, xMax, yMin et yMax. Deux droites Lmax et Lmin relient les points P—et P+- d'un coté et P-+ et P++ d'un autre côté.
L'algorithme calcule ensuite l'enveloppe convex haute pour les points en dessus de Lmax et finit par calculer celle du dessus.
La dernière étape consiste à faire la somme entre ces deux enveloppes

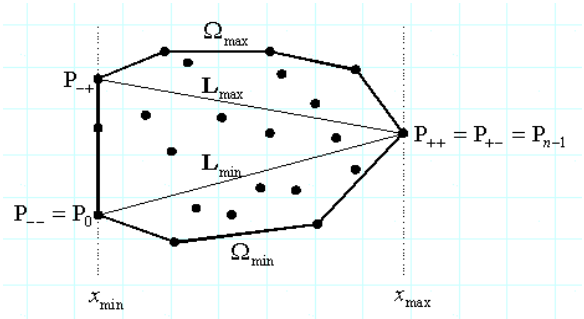


Figure 14 : Convex Hull

5.1.2.2 Calcul de la zone du logo

Une fois l'enveloppe convexe calculée l'étape suivante consiste à récupérer tous les points situés à l'intérieur de cette enveloppe.
Ceci est fait via un algorithme qui scanne toute la zone et puis à l'aide d'un test d'inclusion détermine si le point en cours est à garder.

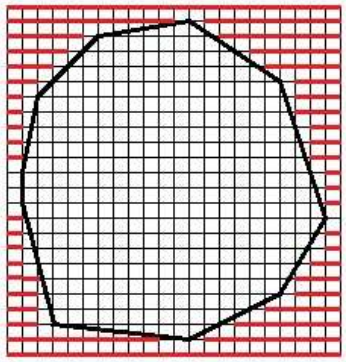


Figure 15 : Zone du logo

Vu le nombre de points, le choix s'est porté sur ces deux algorithmes qui sont simples à implémenter et répondent parfaitement aux besoins.

5.1.3 Traitement : Envoi de l'image requête vers le serveur

Pour envoyer l'image requête au serveur, plusieurs alternatives sont possibles.
Au début le choix s'est porté sur un envoi via SOAP, sauf que l'iPhone n'intègre pas nativement SOAP. En plus SOAP n'est pas très bien adapté à la problématique car les résultats obtenus sont beaucoup plus légers que l'enveloppe SOAP elle-même.
L'utilisation de SOAP est justifiée si les données transférées sont très lourdes et que l'enveloppe est négligeable devant celles-ci.
Une autre méthode, celle implémentée, consiste à envoyer les images (capturée et requête) avec une méthode http POST et de récupérer un fichier XML contenant les résultats.

5.2 Le serveur (Moteur CBIR)

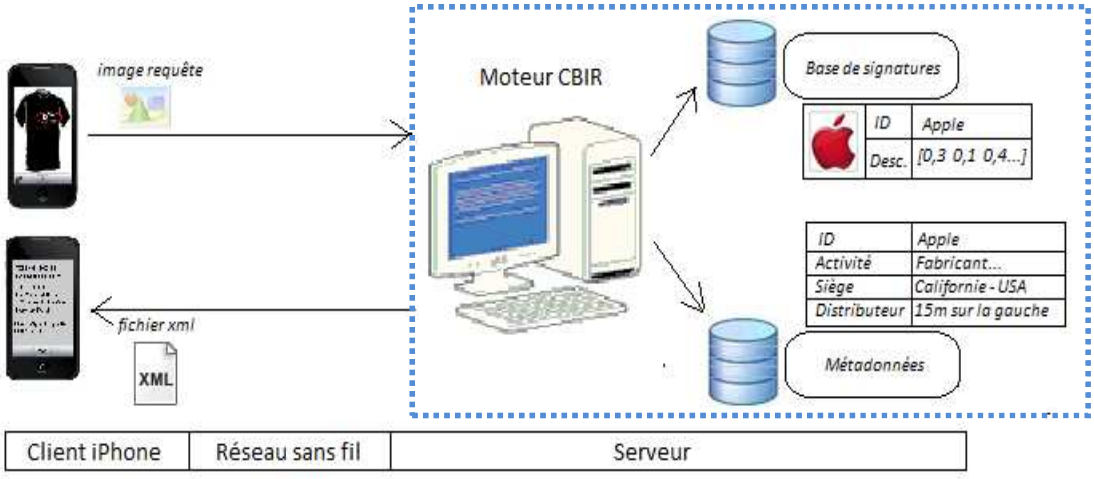


Figure 16 : Système réalisé complet – Partie Serveur

5.2.1 Caractéristiques

Le serveur qui a été déployé est un serveur Apache-PHP-MySql. Ce serveur répond parfaitement aux besoins du système. Il est constitué d'un serveur de fichiers qui contient les images modèles du système.
Il contient aussi une base de données MySql contenant les métadonnées concernant les images modèles.
Dans cette base chaque entrée est constituée de quatre champs :

- Identifiant de l'image
- Nom de l'image
- Description de l'image
- Chemin de l'image sur le serveur de fichiers.

L'identifiant sert aussi à retrouver la signature de l'image dans la base des signatures (voir figure 7).

5.2.2 Fonctionnement

Une fois l'image requête reçu du client (iPhone ou PC), le serveur la renvoie à l'aide d'un protocole XML-RPC vers le moteur CBIR qui calcule sa signature, la compare à toutes les signatures dans sa base de signature et renvoie les identifiants des signatures les plus similaires ainsi que leurs scores.

Le serveur reçoit ensuite les métadonnées des images dont l'identifiant est renvoyé par le moteur CBIR, récupère les métadonnées concernant ces images de la base des métadonnées et génère un fiche XML qu'il renvoie ensuite au client.

Ci-dessous un exemple de ce fichier XML suite à une reconnaissance d'un logo APPLE (le même de la figure 9)

```
<?xml version="1.0" encoding="UTF-8"?>
<Resultats>
   <Resultat>
      <Nom>Apple 06</Nom>
      <Description>Apple rouge</Description>
      <Chemin>http://192.168.0.1/PFEiReLogServer/BanqueImages/IMG_016_APPLE_06.jpg</Chemin>
```

```
      <Score>65.825678484401</Score>
   </Resultat>
   <Resultat>
      <Nom> Apple 07</Nom>
      <Description>Logo Apple jaune</Description>
      <Chemin>http://192.168.0.1/PFEiReLogServer/BanqueImages/IMG_017_ APPLE _07.jpg</Chemin>
      <Score>41.738087749208</Score>
   </Resultat>
   <Resultat>
      <Nom> Apple 01</Nom>
      <Description> Logo Apple gris petit</Description>
      <Chemin>http://192.168.0.1/PFEiReLogServer/BanqueImages/IMG_011_ APPLE _01.jpg</Chemin>
      <Score>39.632882394776</Score>
   </Resultat>
</Resultats>
```

5.2.3 Moteur CBIR

5.2.3.1 Fiche d'identité

Le moteur CBIR utilisé est un moteur Open Source et gratuit nommé ISK-daemon. ISK-daemon utilise des algorithmes d'ondelettes. Il est basé sur le document intitulé "Fast Image Multiresolution Interrogation" par Charles E. Jacobs, Adam Finkelstein et David H. Salesin. ISK-daemon a été conçu et écrit par Ricardo Cabral.
Pour plus d'information voici le lien du site d'ISK-Daemon : http://server.imgseek.net/

5.2.3.2 Fiche technique

L'algorithme de recherche d'images similaires utilise une approche de décomposition multi-résolution en ondelettes pour résoudre le problème de la détermination des k-images les plus similaires à une cible parmi une base d'images pré-indexées.
Parmi certains de ses avantages, cette approche autorise la cible à avoir une résolution différente de la base d'images pré-indexées.
La durée et le stockage de cette méthode sont indépendants de la résolution des images de la base.

Pour chaque image les informations de signature calculées par ISK-daemon peuvent être extraites à partir d'une version d'ondelettes compressées de l'image, permettant à la base de signatures d'être créée facilement à partir d'un ensemble d'images compressées (basse résolution).

Théoriquement, La requête pour les images similaire a une complexité linéaire avec le nombre d'images de la base. Par exemple si le temps d'une requête vers une base de 10 000 signatures est de 5 secondes, et que la base évolue et contient 100 000 images, le temps d'une requête devrait être de 50 secondes.

5.2.3.3 Fonctionnement

Le moteur est implémenté dans un serveur communiquent sur le port 31128 du protocole http. Il ne contient pas malheureusement de base de données pouvant contenir des métadonnées sur les images indexées.
Le serveur du moteur se contente de calculer la signature de l'image et de lui associer un identifiant pour être retrouver plus tard. Ce qui justifie la mise en place d'un serveur de fichiers et d'une base de métadonnées.

Par ailleurs il faut à priori enregistrer 25 images dans la base pour pouvoir envoyer des requêtes de similarité entre images. Il parait que ce nombre est paramétrable dans le fichier *settings* du moteur, ce que je n'ai pas vu, peut-être ça sera possible dans une version prochaine.

Le moteur supporte en termes de protocoles de communication entre autres : XMPL-RPC, SOAP, JSON...
Dans notre cas, et pour rappel, le serveur web communique avec le moteur CBIR via un protocole XML-RPC.

5.2.4 Développement d'extension : Ajout d'images et de métadonnées

Cette partie concerne le client du projet qui est l'administrateur du système entier.
Ce mode lui permet d'alimenter sa base d'images modèles.
Il se connecte au système via un navigateur et entre les informations ainsi que le chemin sur le disque de l'image à rajouter.
Le serveur web rajoute l'image dans le serveur de fichiers, enregistre ses métadonnées dans la base des métadonnées et calcule et insère sa signature dans la base des signatures du moteur CBIR. Le lien entre ces trois bases est fait via un identifiant unique et propre à chaque image qui est le même dans les trois bases.

Figure 17 : Enrichissement de la base des images modèles

5.3 Interface des résultats

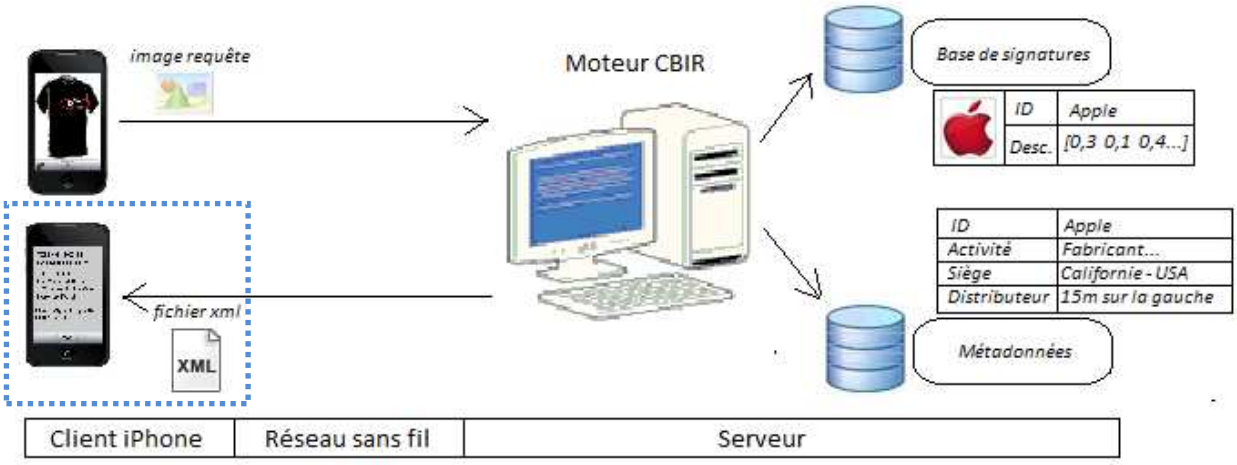


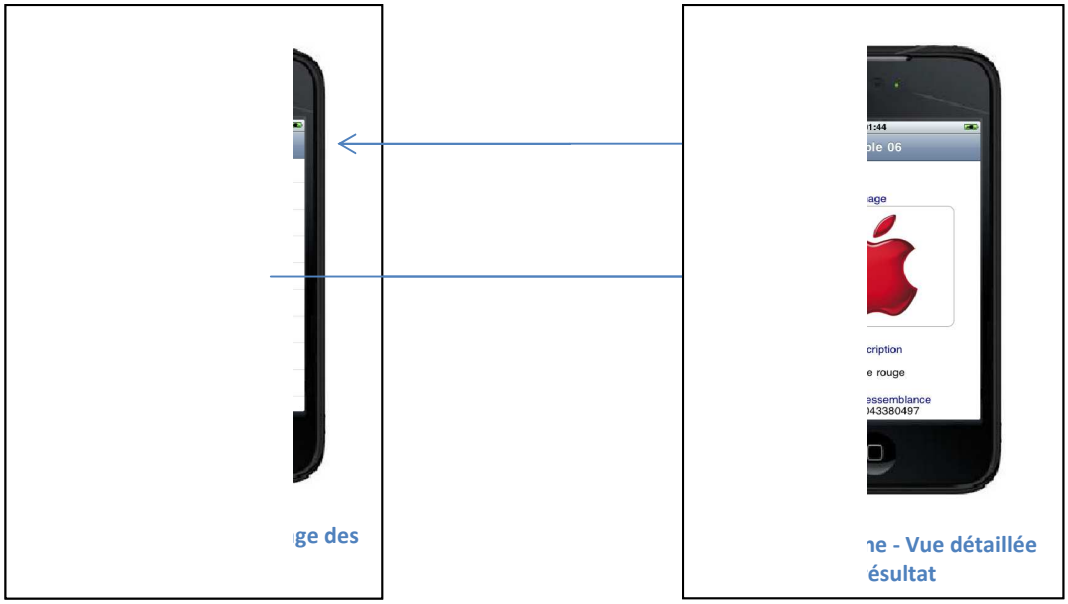
Figure 18 : Système réalisé complet – Partie Interface des résultats

5.3.1 Parsage du fichier XMLresultats

Cette tâche est effectuée XMLToObjectParser qui parse les informations dans le fichier XML renvoyé par le serveur en objets de type Resultat. Celle classe possède quatre attributs qui sont les même que les balises de chaque nœud des résultats du fichier XML (nom, description, chemin et score).

5.3.2 Affichage des résultats

Une fois l'application a traité les informations du fichier XML et les parser en objets qui lui sont propres, elle affiche la vue des résultats. Ceci est fait par la classe RootViewControler. Cette classe crée une TableView dans laquelle elle affiche les attributs *nom* de chaque objet Resultat.
Un appui sur une cellule permet de basculer vers un écran détaillant le résultat sélectionné.
Ceci est fait via un design pattern particulier qu'Apple a implémenté dans la classe RootViewControler. Il est accompagné d'une animation d'entrée de la nouvelle vue et de disparition à gauche et progressive de la vue des résultats initiale.



6 Tests et expérimentations

6.1 Portail des tests

Cette partie a été réalisée dans le but de tester les résultats sans interaction avec l'iPhone.
En effet l'idée de l'interaction est que l'utilisateur aide le moteur en lui indiquant juste les informations dont il besoin pour calculer la signature de l'image requête.
Qu'est ce qui se passera si toutefois on envoie au serveur une image brute contenant d'autres informations en plus du logo ?

Cette interface peut être utilisée du poste « Client PC » (voir figure 7). Le scénario de fonctionnement est le même que dans la section précédente à ceci près qu'on envoie l'image brute au serveur en vue d'une reconnaissance.

Figure 27 : Portail de tests

Pour l'instant il faut cliquer sur le bouton « GénérerXML » et voir les résultats dans un fichier xml à la racine du site. Ceci devrait être remplacé dans les jours à venir par une page affichant les résultats.
Malgré que ça ne rentre pas dans les objectifs du CSPD mais la réalisation de cette page facilitera l'analyse des résultats et permettra ainsi une prise de recul quant aux avantages et inconvénients de l'interaction et ainsi le but principal de ce projet.

L'application iPhone envoie deux images contrairement à ce qui a été dit dans la section précédente.
Pour des raisons de clarté ce constat n'est souligné que dans cette section.

Ces deux images sont :
L'image requête, exactement celle citée dans la section précédente et l'image originale brute, c'est-à-dire avant interaction avec l'iPhone.

La réalisation de quelques tests a prouvé que les résultats sont loin d'être similaires. Ce qui est logique.
Un test a été fait. Une image brute contenant un logo d'Apple a été capturée.
Le score de reconnaissance de cette image avec interaction iPhone était de 63% environ avec l'image modèle.
La reconnaissance de la même image sans interaction avec l'iPhone était aux environ de 30%.

6.2 Résultats

Pour tester l'intérêt de l'application réalisé, plusieurs tests ont été réalisés.
Rappelons quelques définitions :
Image Capturée : Image brute capturée par l'iPhone
Image Requête : Image résultante du coloriage par l'utilisateur
Image Modèle : Image type sur la quelle la reconnaissance est réalisée

Dans la base d'images il y'a trois classes Logitech, Polytech, Rabelais.
Pour chaque classe le serveur dispose de 10 images. Pour réaliser les tests j'ai soumis 10 images capturées et 10 images requêtes de chaque classe au portail des tests.
Ci-dessous les résultats :

Classe	Logitech	Polytech	Rabelais
Images Capturées	7/10	2/10	1/10
Images Requêtes	10/10	8/10	9/10

Le tableau précédent montre le nombre de bonnes détections. On voit très bien, par exemple, qu'avec le coloriage de la zone du logo neuf tentatives de reconnaissance ont réussis au lieu d'une sans coloriage pour la classe Rabelais.

Classe	Logitech	Polytech	Rabelais
Images Capturées	15,39%	4,23%	2,87%
Images Requêtes	51,22%	27,68%	26,97%

Dans le même registre et avec une approche on prouve l'utilité du coloriage en voyant le tableau ci-dessus. Celui-ci en effet nous montre le taux de ressemblance. IL est basé sur le tableau précédent.

Pour l'obtenir il a suffit de faire la somme du premier résultat de chaque reconnaissance réussi (0 en cas d'échec) des 10 essais sur chaque classe et pour les deux types d'images.

Et pour une deuxième fois il prouve que le coloriage augmente la réussite de la reconnaissance.
Par exemple, la classe Logitech, qui a pourtant été bien reconnue rien qu'avec l'image capturée, présente un taux de ressemblance plus de trois fois meilleur en introduisant le concept de l'interaction (de 15 à 50%).

Nous pouvons, donc, conclure que l'interaction a nettement augmenté les chances de reconnaissance de logos, et que dans le cas échéant augment le taux de ressemblance, ce qui peut aider si le nombre d'image dans la bases augmente.

7 Problèmes rencontrés

Les problèmes rencontrés durant ce projet ont généralement concerné le début du développement sur iPhone. La licence qui n'a été obtenue qu'après un mois de la date prévue pour ce début a causé un retard d'un mois sur l'ensemble du projet.
Cependant ce retard a été rapidement rattrapé et le projet a été mené à bien avec un résultat satisfaisant, à savoir une application qui permet de prendre un logo via une interaction, l'envoyer au serveur et récupérer et afficher le résultat.

La période d'adoption du langage Objective-C n'a pas posé de grand problème contrairement à ce qu'on pourrait entendre sur le web, car il est vrai que c'est un langage déroutant au premier abord. Mais une fois le développement commencé on se met rapidement dans la logique et la philosophie de ce langage.

Par ailleurs ce langage nécessite une manipulation attentive des variables car la mémoire n'est guère chose facile dans ce langage pour un débutant.
Apple a mis cependant de outils très performant, facile d'utilisation et ergonomique pour aider les développeurs à se lancer dans leur technologie.

Ce développement nécessite obligatoirement un mac comme énoncé dans le début de ce document. Ce mac l'école me l'a mis à disposition ainsi qu'un iPhone. Le problème était qu'il était impossible de travailler souvent pendant le week-end ou des vacances scolaires car l'école était fermée. Ce problème était présent surtout au début des développements, car une fois l'application a pris sa forme, il était possible de travailler en parallèle sur la partie serveur.

Aucune grande difficulté n'a été rencontré à propos du développement du serveur ainsi que le choix du moteur de reconnaissance.

On peut dire en conclusion que le projet aurait pu être fini un peu avant les temps s'il n'y avait pas ce problème de licence et ainsi l'enrichir avec de nouvelles fonctionnalités. C'est ce qu'expose la dernière partie du présent document.

8 Perspectives

Même si l'application est fonctionnelle, il serait très intéressant de lui ajouter quelques fonctionnalités :

- Une fonctionnalité qui permet aux utilisateurs de parcourir leurs iPhone et de sélectionner des images déjà présente sur l'iPhone. Cette fonctionnalité est déjà présente et a été réalisée à ceci près qu'elle n'est pas mis en avant car elle pose quelques petits problèmes expliqué dans le livrable d'analyse.

- Tester le système avec d'autres moteurs CBIR. Le système réalisé comme le montre la figure 7 est totalement composé de briques pouvant être changeable (le client peut être un autre téléphone ou appareil possédant une couverture réseau)

- Ajouter une fonctionnalité qui permet aux utilisateurs de rentrer les métadonnées des images dont la reconnaissance a échoué, et de ce fait ajouter aussi la seconde interface qui a été mise dans la partie hypothèses et qui n'a pu être malheureusement développée.

Néanmoins tout ceci est faisable, et j'estime que la réalisation de ces tâches nécessite moins d'un mois pour leur accomplissement.
Le travail peut être facilement repris car une documentation technique va être réalisée dans les jours à venir, permettant de pouvoir installer le système et de le faire fonctionner comme il fonctionne aujourd'hui.

Conclusion

Ce projet de fin d'études m'a permis de découvrir une nouvelle technologie que j'ignorai totalement. En effet suite à ce projet j'ai un ressentiment personnel de connaitre les trois technologies les plus utilisées aujourd'hui : Windows, Linux, et Apple (même si cette dernière est basée sur un système Unix).
Le projet a montré mes compétences à changer d'environnement de travail et de méthodologie ce qui reflète mon niveau d'adaptabilité, qualité très important pour un ingénieur de nos jours.

J'ai aussi approfondi mes connaissance dans le domaine de la reconnaissance de formes et d'analyse d'images, domaine que j'ignorais il y'a trois ans. C'est un domaine très lié à l'industrie et dont les recherches et découvertes sont très intéressantes tant pour les professionnels que pour le grand public. Rien qu'avoir l'application que j'ai réalisée et dont une seule version existe au jour d'écriture de ce document et qui s'appelle VIPR. En effet VIPR sert les utilisateurs aujourd'hui à reconnaitre les objets dans leurs environnements en deux clics : c'est en effet ce que j'ai réalisé dans ce projet.

Une page de conclusion ne peut exprimer mon ressentiment à propos de tout ce que j'ai pu apprendre durant de projet et durant ce cursus d'ingénieur dont ce projet est marque la fin. Ce qui me ramène à remercier les personnes qui m'ont aidé à atteindre ce stade :
Mes remerciements vont en premier à Mathieu DELANDRE et Nicolas SIDERE, sans qui ce projet n'aurait pu être ce qu'il est aujourd'hui. Je leurs remercie pour les conseils, les remarques, les critiques et surtout leur disposition et le temps qu'ils m'ont consacré à répondre à mes diverses questions.
Je tiens à remercier aussi l'ensemble du corps professoral de l'Ecole Polytechnique de Tours qui m'a dispensé les enseignements nécessaires afin de venir ingénieur en informatique.
Et sans oublier bien sur ma famille qui réside en Mauritanie et à qui je dédie ce travail.
Merci pour vous tous et à tous ceux que je n'ai pas cité.

Références

Ci-dessous une liste de liens que j'ai trouvés utiles. Évidemment ce n'est qu'une liste très exhaustive contenant ceux que j'ai jugés les plus intéressantes et qui peuvent aider toute personne voulant reprendre le travail que j'ai et que j'aurai à réaliser ou tout simplement voulant se mettre sur la programmation iPhone.
Liens fonctionnels à la date : 25/12/2009.
http://developer.apple.com/iphone/index.action
http://www.datasprings.com/Resources/ArticlesInformation/iPhoneSDKGettingStartedExampleCode.aspx
http://pwar.info/download/Developper_pour_iPhone_-_Antony_Gardez.pdf
http://www.otierney.net/objective-c.html
http://www.dotnetguru.org/articles/dossiers/threads/multithreading.htm

Tous les liens et références sont à trouver sur l'outil dédié à la gestion du présent PFE :
http://srv-dev-unix.di-epu.univ-tours.local/redmine/projects/pfe20-taleb
Ces liens concernent toutes les parties du PFE et sont régulièrement mis-à-jour.

Autre référence très intéressante le livre *Programmation iPhone 3.0* de *Thomas Sarlandine* : bibliothèque de l'école polytechnique de Tours.