Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

[TP] Prise en main du client C++

Formation ANR-Digidoc

Corentin Lallier

Juin 2012

LaBRI

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

Présentation du TP

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

But du TP

Le but de ce TP est de manipuler les classes importantes qui permettent d'utiliser les algorithmes et de les rendre disponibles à distance.

Pour cela nous allons voir deux façons d'utiliser les algorithmes :

1. Une application : packagée et complète, basée sur Qt Gui.
2. Un plugin : un algorithme simple, sans Gui, disponible localement et à distance.

Structure du TP

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

Nous allons créer des projets basés sur CMake, nous utiliserons donc le protocole de compilation vu précédemment, basé sur CMake. Pour chaque projet sont fourni la partie applicative, c'est à dire :

- La structure du project : sources, CMakeList pour la compilation, etc
- La structure de base de l'application : la classe principale
- Une GUI basique (si nécessaire)

Notions clés

- Page
  - Image
  - Annotations (ex : mesures)
  - Attachments (ex : fichier de VT)
- PageCollection
- Repository : stocke les infos relatives aux serveurs
- DoQuBookApp : permet la création d'une application
- DoQuBookDaemon : démon qui permet de déclarer ses plugins au serveur.
- AlgorithmOnTask : Binarizer, OCR, etc

Création d'une application

Dans ce premier TP, nous créerons une application autonome permettant de récupérer une liste d'images du serveur puis de leur appliquer un traitement. Récupération de la structure du projet à partir du disque dur externe.

Ce projet contient déjà la partie applicative, c'est à dire :

- La structure du project : sources, CMakeList pour la compilation, etc
- La structure de base de l'application : la classe principale, héritant de DoQuBookApp
- La méthode principale : runApp (QMap<QString, QString> args)
- Une GUI basique

Création d'une application

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

Dans ce premier TP, nous allons :

- Voir la partie connexion au serveur : Repository et ApplicationContext
- Voir une interaction simple avec un gestionnaire de ressources, ici le PageCollectionService
- Voir l'appel d'un plugin via le PluginLoader

Création d'une App

Connexion au serveur digidoc

- Ouvrir le projet workspace/Formation_App (récupéré à partir du disque dur externe)
- Ouvrir le main.cpp ou ouvrir le CMakeList dans QtCreator
- Le code à modifier se trouve dans la méthode runApp()
- présentation des autres méthodes :createGui, addImage

```
// Connexion au serveur
QString repoName = "digidoc";
DoQuBook::RepositoryPtr repo =
    DoQuBook::ApplicationContext::instance()->
                                        getRepository(repoName);
```

Création d'une App

Interaction avec le PageCollectionService


```
// Interaction  avec le  PageCollectionService et recuperation des pages
DoQuBook::PageCollectionServicePtr collectionService =
                                                repo->getPageCollectionService();
QList<PagePtr> pages;
```

Création d'une App

Récupération des pages d'une collection


```
// recuperation de la premiere collection
if (collectionService->getAll().size() > 0)
    firstCollectionName = collectionService->getAll().at(0);
// recuperation de toutes les pages
if (firstCollectionName.compare("") != 0 )
    pages = collectionService->getPages(firstCollectionName);
```

Création d'une App

Chargement des plugins grâce au PluginLoader

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

```
// Chargement des plugins locaux
PluginLoader* loader = PluginLoader::instance();
loader->loadPlugins();
```

Création d'une App

Chargement d'un plugin de binarisation

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

```
// Recuperation d un plugin
BinarizerPtr bin =
        loader->getPlugin<Binarizer>("labri.BinarizationOtsu.v1.0");
```

Création d'une App

Application du plugin sur la liste de pages

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

```
// binarisation et ajout a l interface
foreach(PagePtr p, pages) {
    addImage(p->getImage(), p->getName());
    addImage(bin->binarize(p->getImage()),bin->algorithmName());
}
```

Création d'une App

Compilation et exécution

compilation via la procédure CMake

- Dans le répertoire du projet workspace/Formation_App
- mkdir build
- cd build
- cmake .. -G "MinGW Makefiles"
- cmake –build .
- execution de FormationApp

Création d'un plugin

But : Créer un plugin permettant de calculer des mesure sur une liste d'images du serveur. Ici cette mesure sera simplement le nombre de composantes connexes de l'image calculée via OpenCV. Récupération de la structure du projet à partir du disque dur externe.

De même que dans le TP précedent, existence de la partie applicative :

- La structure du project : sources, CMakeList pour la compilation, etc
- La structure de base du plugin
- La méthode principale : QMap<QString, double> measure(QList<QImage>, ...)

Création d'un plugin

Sommaire

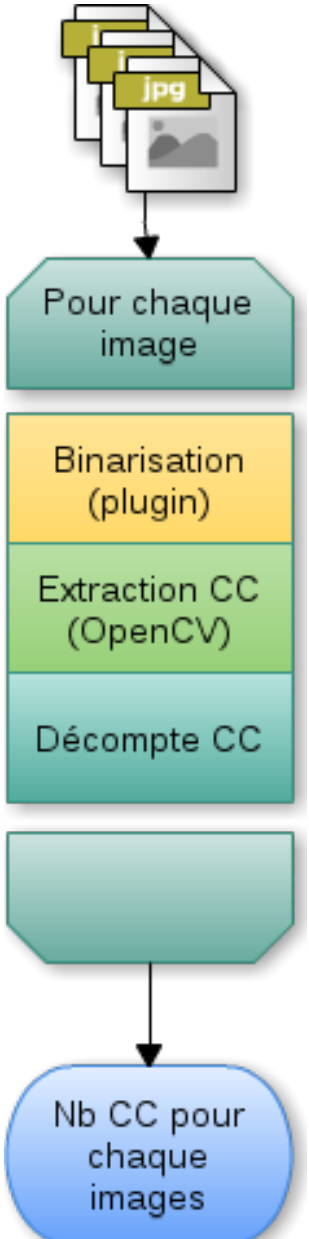
Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion



Création d'un plugin

Parcourt des images

- Ouvrir le projet workspace/Formation_Plugin (récupéré à partir du disque dur externe)
- Ouvrir le Formation_Plugin.cpp ou ouvrir le CMakeList dans QtCreator
- Le code à modifier se trouve dans la méthode measure()

```
// mesures de sortie
QMap<QString, double> measures;

//  ...

// parcourt de la liste des images
foreach (QImage i, images) {
}
```

Création d'un plugin

Appel d'un plugin de binarisation

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

```
// Appel d un plugin pour la binarisation de l image
PluginLoader *loader = PluginLoader::instance();
BinarizerPtr bin =
        loader->getPlugin<Binarizer>("labri.BinarizationSauvola.v1.0");
QImage iBin = bin->binarize(i);
```

Création d'un plugin

Conversion en cv : :Mat 8bits

Declaration du converter

```
// convertisseur cv::Mat <-> QImage
DoQuBookOpenCVConverter converteur;


// conversion QImage vers cv::Mat
cv::Mat ibinCVMat = converteur.getCvMat(iBin);

// conversion vers image 8UC1 (le converter retourne une image 8UC3)
cv::Mat g_gray =
                    cv::Mat::zeros(ibinCVMat.rows, ibinCVMat.cols, CV_8UC1);
cv::cvtColor( ibinCVMat, g_gray, CV_BGR2GRAY );
```

Création d'un plugin
contours de l'image


```
// stockage des regions
std::vector<std::vector<cv::Point> > contours;

// recuperation des contours de l image
cv::findContours(g_gray, contours, cv::RETR_LIST,
                                    cv::CHAIN_APPROX_SIMPLE)
```

Création d'un plugin

Calcul et recuperation des mesures

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

```
// id de l image
QString id = "image"+QString::number(imageId++);
// recuperation des mesures
measures[id] = contours.size();
//maj id
imageId++;
```

Création d'un plugin
Compilation et exécution

Test du plugin via un test unitaire

compilation via la procédure CMake

- Dans le répertoire du projet workspace/Formation_Plugin
- mkdir build
- cd build
- cmake .. -G "MinGW Makefiles" -DWANT_UNIT_TESTS=on
- cmake –build .
- cmake –build . –target install
- ./test/TestFormation_plugin
- le test se déroule : la ligne "image0 : 23431" donne le nombre de CC dans l'image utilisée par les tests.
- Il est maintenant possible d'utiliser ce nouveau plugin dans Formation_App

[TP] Prise en main du client C++

Corentin Lallier

Sommaire

Introduction

Présentation des ressources importantes du serveur

Création d'une application

Création d'un plugin

Conclusion

Déclaration d'un nouveau plugin (voir le header)

- organisationName ()
- algorithmName ()
- version()
- description ()

Type de plugins

MEasurer : Map<String, Double> measure (List<Image>)

OCR : String ocr(Image)

Filter Image filter(Image)

etc ...

Création d'un plugin
Conclusion

- Connexion, récupération des informations, etc est pris en charge en amont sans intervention de l'utiliateur, le but est de se concentrer sur l'écriture de l'algorithme.

Pour conclure

Ce que nous proposons

- Serveur de stockage/partage
- Interfaces de partage d'images et de traitements
- Interface web (prototype)

Ce que vous avez appris

- Création d'un plugin / App
- Gestion des différentes ressources
- Execution à distance des plugins
- Interface web
- Interface REST

Merci de votre attention !