2014

UTProfiler

Merouane HARRIGA

Mohamed-Madiou DIALLO

UTC

12/06/2014

Table des matières

Introduction 2
Contexte 2
Objectifs 2
L'architecture 3
Description et justification des choix : 4
Tester Notre application 5
Algorithme de Complétion 6
Conclusion : 9

Introduction

Contexte

Un étudiant d'une Université de Technologie souhaite se doter d'un logiciel lui permettant de gérer, et de choisir ses Unités de Valeur(UV) tout au long de son parcours étudiant.

Objectifs

L'application doit permettre aux étudiants de visualiser, d'éditer, d'ajouter, ou de supprimer les différents types de cursus. Elle doit aussi permettre a un étudiant de saisir sa situation actuelle en précisant le cursus de formation, la liste des uvs aux quelles il a déjà été inscrit avec leur résultat et les équivalences de crédits obtenues. Pour cela on utilise un Framework multiplateforme (Qt) et le SGBD Sqlite.

Pour résumer, l'application permet à l'étudiant de :

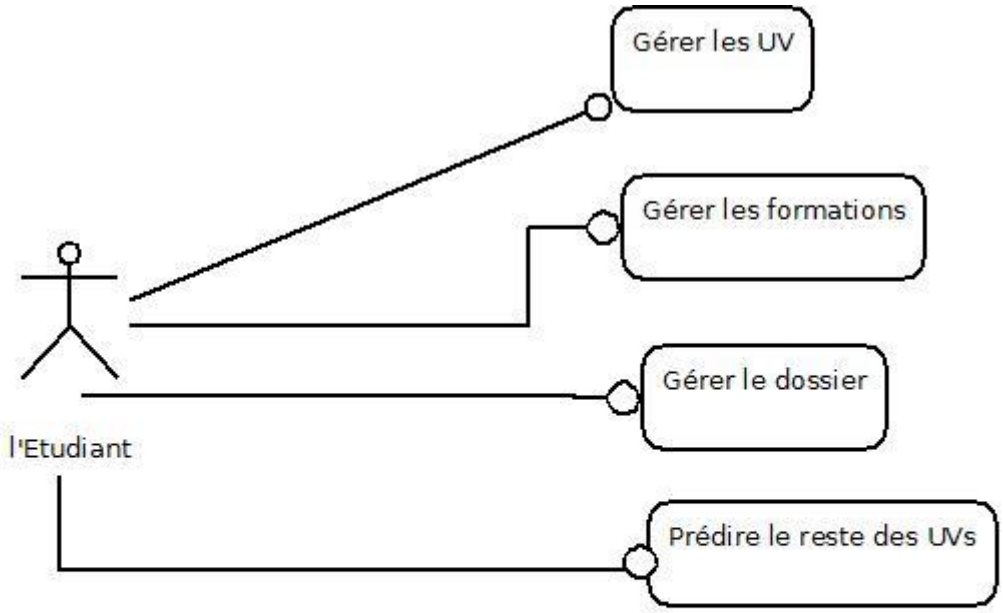
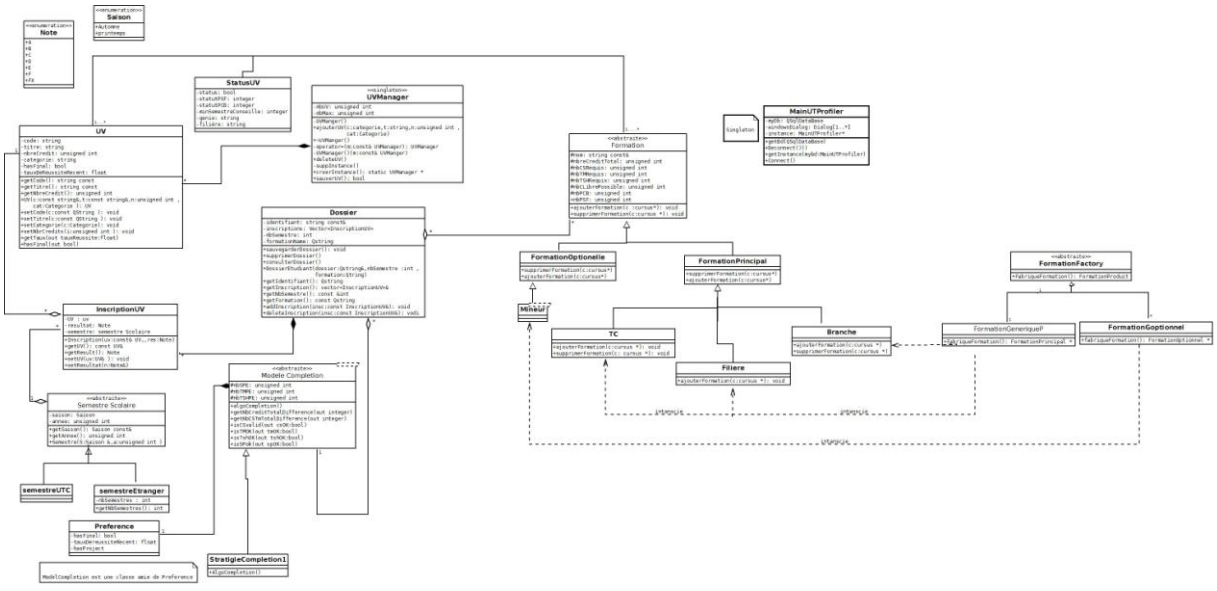


Diagramme de cas d'utilisation

L'architecture

Pour avoir un aperçu global de notre travail, voici notre diagramme UML proposé.



Description et justification des choix :

✓ Présentation générale des classe

Dès le début on a essayé de préparer le plan pour l'algorithme de l'auto complétion.

Merci de bien vouloir voyez notre fichier SQL qui implémente bien notre architecture aussi. Je note aussi qu'il est très complet, le diagramme ci-dessus est plus ou moins ce que notre application gère sous QT. Comme on a essayé de suivre un peu les méthodes agiles on l'a pas respecté à 100/100.

Pour la classe on a commencé par définir les critères qui peuvent nous aider facilement pour choisir et une UV parmi les autres distinguer parmi les autres :

- Taux de réussite récent
- Si l'UV a un final
- Si l'UV a un médian
- Si elle a un projet
- Si elle a un TP ou TD
- Une description, si l'étudiant veut chercher en se basant sur ce qui est enseigné (nom des technologies par exemple, JAVA…)

Pour la classe formation on a définit ce qu'on a jugé nécessaire pour la valider :

- Nombre de crédit total requis
- Nombre de crédit CS total requis
- Nombre de crédit TM total requis
- Somme de crédit CS/TM requise
- Nombre de crédit SP requis
- Nombre PSF requis (si la formation n'est pas une filière PSF=0)
- Nombre PCB requis
- Nombre de Semestre à effectuer dans la situation normale
- Matrice TSH…

Maintenant on a plusieurs **critères qu'**on peut les utiliser. Nous avons trouvé que c'est crucial de noter tout d'abord qu'une UV peut exister dans plusieurs formations, elle a un semestre conseillé et peut être fortement conseillée dans une formation et optionnelle dans une autre. Elle peut être aussi obligatoire pour la validation dans une formation (dans le cas d'une filière). On a ajouté tout ça dans la classe d'association qui relie la formation par l'UV. On l'a nommé « UVStatut » dans notre diagramme de classe.

Au niveau de dossier, pour une raison de simplicité on considère qu'un dossier contient plusieurs inscriptions (UV + note) et une formation. Bien évidement il est caractérisé pas son IDD. On a utiliser un vecteur pour gérer les Inscriptions

Notre classe Modèle complétion nous permet de calculer la situation de l'étudiant la comparer avec les contraintes de la formation pour chaque critère. Compléter cette différence en se basant sur le critère préféré par l'étudiant. Cela permet d'utiliser plusieurs algorithmes de complétion, l'ajout d'un autre algorithme se fait donc en créant une classe qui hérite de la classe **Modèle complétion** et en définissant un nouveau comportement à la méthode de complétion. Cela facilite donc le choix dynamique des traitements à exécuter, aisance d'inter changer les algorithmes, le code devient facile à maintenir en cas de changement ou d'évolution.

✓ *Le choix du design pattern :*

Les design patterns que notre architecture propose sont le design pattern Singleton pour la classe MainUTProfiler qui permet de gérer la connexion à la base de données ainsi que tous les dialogues avec nos fenêtres.

Le « **abstract factory** » pour créer des formations car on sait pas quel formation va créer à priori. On a distinguer deux types de formations : Principale (TC, branche…) et Optionnelle (mineur…). Donc chaque usine sera responsable sur la création de ses formations (Produits). faciliter toute la tache pour ajouter une nouvelle formation. Cela nous a donc permis de créer d'objets regroupés en famille sans devoir connaître les classes concrètes destinées à la création de ces objets. L'ajout d'une nouvelle formation autre que (TC, BRANCHE, FILIERE,Mineur) se fera en créant une classe qui hérite de Formation Principal ou Optionnelle.

La distinction entre les formations principales et optionnelles n'est pas obligatoire mais si un jour on modifie un de ces deux types on va gagner au niveau de la maintenance.

L'algorithme d'auto complétion est implémenté suivant le design pattern **Stratégie**. Selon le critère que l'étudiant choisis (Taux de réussite de l'UV, si l'UV a un final.. il doit être capable de retrouver facilement une bonne solution)

Tester Notre application

Pour tester notre solution il faut spécifier le chemin physique du fichier UTProfiler.db dans la fonction connectDb dans MainUTProfiler.

Bien qu'on a bien reçu à trouver une solution judicieuse pour la question 4, nous n'avons pas eu le temps pour l'implémenter. Veuillez nous excuser.

Algorithme de Complétion

Calcule de la situation actuelle : Ces méthodes nous permettes de retourner les crédits qui nous manque pour un certain critère ou bien la non validité d'une contrainte pour la validation du dossier

getDiffCreditsTotal() : on retourne nombre de crédit total dans le dossier- Le nombre de crédit requis par la formation

getDiffCreditsCS() : On retourne le nombre de crédits CS dans le dossier - le nombre de crédits CS requis par la formation

getDiffCreditsTM() : On retourne le nombre de crédits TM dans le dossier - le nombre de crédits TM requis par la formation

getDiffCreditsCSTM() : on retourne diffCreditsCS + diffCreditsTM – totale CSTM requis par la formation

getDiffCreditsTSH() : On retourne le nombre de crédits TSH - le nombre de crédits TSH requis par la formation

isTSHValide() : on retourne true si le nombre de ligne et le nombre de colonne sont respectés

getDiffCreditsSP() : On retourne le nombre de crédits SP dans le dossier - le nombre de crédits SP requis par la formation

Si(formation==filière) {

getDiffCreditsPSF() : On retourne le nombre de crédits PSF dans le dossier - le nombre de crédits PSF requis par la formation

getDiffCreditsPCB : On retourne le nombre de crédits PCB dans le dossier - le nombre de crédits PCB requis par la formation

}

Une fonction helper qui verifie si un dossier est valide :

Bool isDossierValid(){

bool expression= **getDiffCreditsTotal**() >=0 && **getDiffCreditsCS()**>=0 **&& getDiffCreditsTM**()>=0 && **getDiffCreditsCSTM()**>=0 **&& getDiffCreditsTSH()**>=0 **&& isTSHValide()**>=0 **&& getDiffCreditsSP()**>=0

***If*(formation==filière)**

***Retourner expression &&* getDiffCreditsPCB** >=0 **&& getDiffCreditsPSF() <0**

Else

Retourner expression

}

Une fonction helper qui retourne le nombre d'UV raté :

int numberMissedUV(){

c=0

pour chaque inscription dans le dossier :

si(note= Fx ou note=Abs ou note=F)

c++

retourner c

}

```
// on suppose que l'étudiant ne veut pas les UV qui ont un final pour S1
AlgoCompletionS1(){

        nombreDeSemestreRestant= nombreSemestreSituationNormale- nombreSemestreActuelle
        // toujours strictement supérieur
        nombreUVaProposerAprioriParSemestre= [[(nombreSemestreSituationNormale-
        nombreSemestreActuelle)*6]+ numberMissedUV()]/nombreDeSemestreRestant
        UVS=  Selectionner les UVs qui n'ont pas de final
        indicateurDeNombreUV= false
        If( totale de crédits UVS -  getDiffCreditsTotal() >0 && ! isDossierValid())  {
                While( nombreDeSemestreRestant>0){
                        nbUV= nombreUVaProposerAprioriParSemestre
                        While(nbUV >0 && ! isDossierValid()){
                                indicateurDeNombreUV=true
                                If(getDiffCreditsCS()<0 || getDiffCreditsCSTM()<0 ){
                                        ✓ Proposer une CS parmi UVS
                                        ✓ nbUV -- (on décremente)
                                        ✓ on retire l'uv de UVS
                                        ✓ indicateurDeNombreUV=false
                                }
                                If(getDiffCreditsTM()<0 || getDiffCreditsCSTM()<0){
                                        ✓ Proposer une TM parmi UVS
                                        ✓ nbUV -- (on décremente)
                                        ✓ on retire l'uv de UVS
                                        ✓ indicateurDeNombreUV=false
                                }
        ……..//Pareil pour le reste des critères…….
                        If(indicateurDeNombreUV==true){
                                nbUV -- // Sinon, on risque de créer une boucle infini dans le
                                cas ou UVS contient que des UV des catégories déjà validé
                                par l'étudiant
                        }

                        }//  deuxième while
                        nombreDeSemestreRestant --
                }// fin du 1ere while

        }
        Else{
                //Impossible de trouver une solution avec ce critère
        }
        If(! isDossierValid()){

                // Insuffisant Proposer un autre critère...
        }
}
```

Conclusion :

Pour conclure, ce projet nous a permis de mieux appréhender Qt et de se familiariser avec les designs patterns itération, singleton, abstract factory , stratégie de la création et l'utilisation d'une base donnée associé au Framework Qt. en partant d'un cahier des charges duquel nous avons dû extraire les informations utiles puis modéliser une solution sous forme d'UML.
Lors de notre modélisation, nous avons dû faire certaines hypothèses concernant des parties du sujet que nous avons précisées dans ce rapport.