Université de Technologie de Compiègne

Programmation et Conception Orientées Object
LO21

Rapport du projet PluriNotes

Auteurs :
M. Aurélien Roy
M. Thibault Subreville
M. Xiang Li

Correcteur :
M. Idir Benouaret

Printemps 2017

Tables des matières

Table des figures

Introduction

Dans le cadre de l'enseignement de la programmation et conception orientées objet de l'UTC, nous sommes amenés à réaliser un gestionnaire de notes : "PluriNotes". Il est destiné à éditer et gérer un ensemble de notes (des mémos) qui peuvent correspondre à du texte, des images, des vidéos, du son...
Ce projet consiste à nous permettre de connaître et de manipuler la conception de certains Design Patterns, de former l'architecture, de mettre en oeuvre toutes les fonctionnalités demandées, ainsi qu'à implémenter une interface graphique.
Le projet a été réalisé en C++ avec l'IDE Qt.
En plus de présenter le contexte et les objectifs de ce projet, le présent rapport a pour but d'exposer nos réflexions ainsi que nos choix de conceptions et de développement, accompagné d'un aperçu de l'interface utilisateur.

1 Nature et contraintes du projet

Contexte: Projet à but pédagogique, effectué dans le cadre de l'UV LO21 de l'UTC.

Maître d'ouvrage: Antoine Jouglet en tant que responsable de l'UV et Idir Benouaret en tant que chargé de TD et correcteur.

Intitulé du projet: Concevoir et développer l'application PluriNotes, destinée à éditer et gérer un ensemble de notes (des mémos).

Résultats attendus: Logiciel intuitif, code source, document Doxygen, rapport de rendu, vidéo explicative des fonctionnalités du logiciel.

Public ciblé: Le logiciel de gestion de notes est utilisable pour tout public. Il a été conçue de sorte à être intuitif.

Délais: Le projet a commencé début mai 2017, et se finit le 15 juin de la même année.

Moyens logiciel: IDE Qt Creator, Doxygen logiciel de génération de documentation du code source.

Supports: Polycopié de cours de LO21, documentation Qt, internet.

Equipe de projet:

- Thibault Subreville, Génie Informatique 02 à l'UTC.
- Aurélien Roy, Génie Informatique 02 à l'UTC.
- Xiang Li, Génie Informatique 01 à l'UTC.

2 Choix de Conception

Avant de commencer le codage du projet, nous avons établi un modèle conceptuel UML, disponible en annexe, permettant de définir l'architecture du programme par des relations entre les différentes classes.

2.1 Gestion des notes

Note: Nous avons choisi de disposer d'une Classe Abstraite Note, qui est la classe base de tous les types de mémo (Article,Task, Image, Sound,Video). Une Note correspond en réalité au contenu d'une note (une version de note). Un objet Note sera donc recrée à chaque nouvelle version d'une note pour représenter le nouveau contenu.

NoteHolder: Cette classe représente une note de manière plus générale, indépendamment de sa version. Il contient entre autre l'identifiant de la note ainsi qu'une référence de type Note vers la dernière version de son contenu.

NoTextualNote: Comme nous avons trois types de notes similaires à gérer Sound, Video, Image, nous avons ajouté une classe hérité de la classe Note dans le but de factoriser le code. Cette classe permet aussi l'évolution vers l'agrandissement de l'application et la gestion d'autres types de notes non textuelles.

NotesManager: Cette classe permet la gestion des notes de toute l'application (relation de composition). Dans le but de renforcer la protection du code nous avons utilisé le design pattern Singleton pour n'en avoir qu'une seule instance. Le design pattern Iterator est aussi employé pour préserver le principe d'encapsulation. Elle contient aussi une sous classe NotesModelHolder jouant le rôle d'adapteur pour s'interfacer avec les différentes classes de Qt pour la visualisation des données.

2.2 Gestion des relations

Pour la partie gérant les relations entre notes, nous avons choisi d'utiliser la programmation générique. On peut ainsi gérer des relations entre objets, indépendamment du type d'objet. Dans le cas de Plurinotes, nous utilisons des relations entre NoteHolder, nous aurons donc un objet de type RelationsManager<NoteHolder>. Mais grâce à la modularité des templates class, rien ne nous empêche d'intégrer un système de relations entre d'autres types d'objets dans un autre projet, en gardant exactement le même code.

Le seul inconvénient d'utiliser la programmation générique ici est que cette fonction du langage n'est pas bien gérée par Qt. Ainsi une classe générique ne peut pas hériter de QObject, rendant plus compliqué la gestion des signals et des slots.

Voici quelques classes utilisées dans notre partie Relations :

RelationsManager: Il s'agit de la classe jouant le rôle de façade dans le système de relations. Elle permet d'effectuer toutes sortes d'actions (création d'une relation, association et dé-associaiton de deux notes, génération d'un arbre...)

Relationship: Relationship désigne un type de relations. Elle est caractérisée par un nom et type d'association. On possède également une classe fille BidirecitonnalRelationship pour représenter les relations fonctionnant dans les deux sens.

Association: Une association est un couple de deux objets, aussi caractérisée par un type de relation (Relationship) et un label.

2.3 Interface graphique

Mainwindow: Mainwindow est la fenêtre principale de l'application.

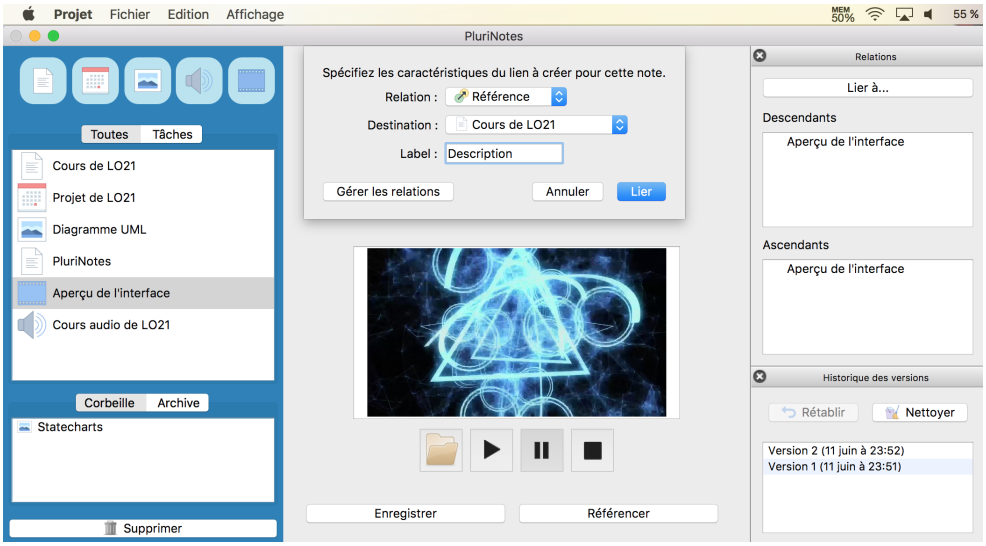

Figure 1: Aperçu de l'interface de l'application PluriNotes

Mainframe: Cette classe correspond à la partie centrale de l'application (coeur de l'application).

Relateddockview: Ça appartient ou non à la mainframe, qui affiche à droite de mainframe, qui nous donne une visualisation des relations intuitives. Cependant, il est de type dockview, il est un peu parallèle par rapport au mainframe.

NotesListView: Cette classe domine la partie gauche de l'application, qui nous fait lister les notes actives, archivés, sursis. De plus, cette dernière, nous offre des boutons afin de créer une nouvelle note ou vider la corbeille...

NotesProxyFilter: Cette classe joue le rôle de filtre sur les notes. On peut ainsi obtenir uniquement les notes actives, ou bien les notes supprimés...

Noteframeview: Cette classe s'occupe de l'affichage d'une note en particulier (zone au centre de l'application).

NoteInterface: C'est une classe base, qui facilite la conception sur laquelle se base les autres interfaces de Notes. Elle est principalement utilisé pour la factorisation du code, i.e.: signaux, slots, boutons...

Les classes hérités de NoteInterface: A l'intérieur de ces classes, nous avons défini et détaillé les méthodes et slots afin de maintenir l'interaction entre l'interface et NotesManager et les classes dérivées.

2.4 Gestion des médias

L'application PluriNotes permet la lecture des formats picturales, audios et vidéos. Nous avons implémenté un lecteur audiovisuel fonctionnant au coeur de l'application. Il est donc à tout possible de mettre en pause ou d'arrêter une vidéo.
L'application gère ainsi de nombreux formats comme le mp4, avi, wav, gif...

3 Argumentation sur le choix de l'architecture

3.1 Respect des bonnes pratiques de codage

La déclaration des destructeurs des classes abstraites comme étant virtuelle garantit aussi un respect du principe de substitution. Le principe d'encapsulation a été respecté : les attributs sont privés et ne sont accessibles que par l'emploi des méthodes de classe. Nous utilisons également les méthodes const. La génération de certaines classes n'est possible que par leurs "Manager" (design pattern Factory).
Tout cela permet de garantir une meilleure fiabilité de notre programme.
Pour les classes "Manager", nous avons utilisé le principe de singleton dans le but d'empêcher plusieurs instance des ces classes.

3.2 Gestion des versions

Les différentes versions d'une note sont gérées par la création successive de nouveaux objets Note, représentant le contenu d'une note. Un NoteHolder contient uniquement une référence vers la dernière version d'une note. Le reste des versions n'est pas chargé en mémoire mais reste dans la base de données.
Les anciennes versions sont chargées dès que l'utilisateur souhaite revenir à une ancienne version.

3.3 Annuler et Rétablir

Afin de pouvoir proposer une fonction "Undo/Redo", nous avons implémenté le design pattern Memento et Command. Précisément, dans la classe memento nous sauvegardons l'état d'une note (titre, text, description...), et les éléments servant à reconstituer une relation.
Cette fonctionnalité d'annuler et rétablir est disponible pour le changement d'état d'une note (mise en archive, en corbeille...) mais aussi pour la création/destruction des relations (autre que les références).

3.4 Persistance des données

Pour l'enregistrement des données de l'application, nous avons décidé d'utiliser une base de données intégré nommée SQLite. Nous n'avons pas choisi l'enregistrement via un fichier XML, car:

- La base de données offre une meilleur gestion des données pour un grand volume de données. Cette architecture permet la réalisation des évolutions plus facilement.
- Le choix des versions se réalise avec une simple requête SQL, et est plus rapide que de chercher dans un simple fichier.
- Le XML est un format qui tend à disparaître au profit d'autres formats. De plus, il existe certains problèmes de compilation avec les flux XML sur Qt et les distributions MAC et Linux.

Dans tous les cas, la partie persistance est détachée du reste du programme. Cela permet par exemple de rajouter d'autres méthodes de persistance (XML, CSV, ...), en plus de SQLite, sans modifier les autres classes du projet.

3.5 Lisibilité

Nous avons créé des dossiers dans le but de classer les fichiers de code en fonction de leurs fonctionnalités. Nous faisons également pris grand soin à nous faire comprendre par les autres programmeurs, en ajoutant des commentaires à côté de la déclaration des

méthodes les plus compliquées/importantes. De plus, une documentation Doxygen a été générée à partir du code source, cette doc est fournie avec le rapport.

3.6 Extensibilité

Nous avons séparé l'interface graphique (vues), les parties logiques (gestion des notes et relations), et la persistance (SQLite) afin de respecter le modèle MVC et d'avoir un code modulaire. Cela rend possible l'extraction des parties du code fonctionnelle sans grandement les modifier. Grâce à la programmation générique, nous avons aussi laisser la possibilité de pouvoir contenir d'autres objets que des notes sans modifier l'intérieur des classes.

3.7 Praticité

Lors de la conception de notre application, nous avons fait en sorte de la rendre la plus "user friendly" possible. Un menu d'aide a était introduit pour permettre à un utilisateur de comprendre au mieux les fonctionnalités du logiciel.

4 Utilisation

- ctrl+M : Menu d'aide
- ctrl+Z : Annuler
- ctrl+Y : Rétablir
- ctrl+R : Afficher/Enlever la vue des relations (vue réctrable)
- ctrl+V : Afficher/Enlever la vue des versions (vue réctrable)
- ctrl+Q : Quitter l'application

Pour supprimer une relation, il suffit de cliquer sur la relation (à droite de l'interface) et d'appuyer sur la touche du clavier "Suppr".
Pour plus de détails, veuillez vous référer à la vidéo de présentation de l'application PluriNotes qui accompagne ce document.

5 Retour et Conclusion

Ce projet a été réalisé en suivant scrupuleusement le cahier des charges, et toutes les fonctionnalités sont présentes et plus encore.
Les relations d'interdépendances entre les différents membres de notre architectures nous ont contraints, à plusieurs reprises, d'effectuer des changements au cours du développement. Une architecture, même bien pensée au départ, est souvent remaniée dans la phase de codage, car nous n'avions pas pensé à toutes les contraintes que ce soit de système d'exploitation (fonction non portable) mais aussi à la gestion de l'historique de commandes...
Néanmoins, nous en retirons une satisfaction à rendre une application fonctionnelle et ergonomique.

6 Annexes

N.B.: Nous avons omis certains attributs et méthodes dans les diagrammes UML suivant dans un souci de visibilité, l'idée est de présenter l'architecture global du projet.

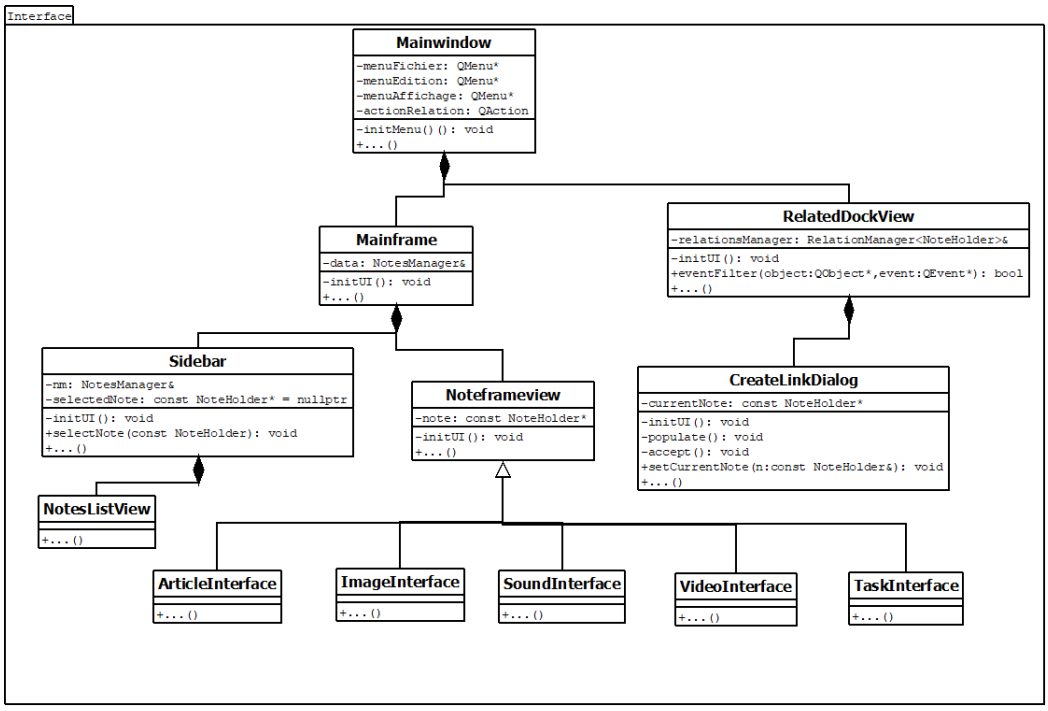

Figure 2: UML simplifié de l'interface PluriNotes

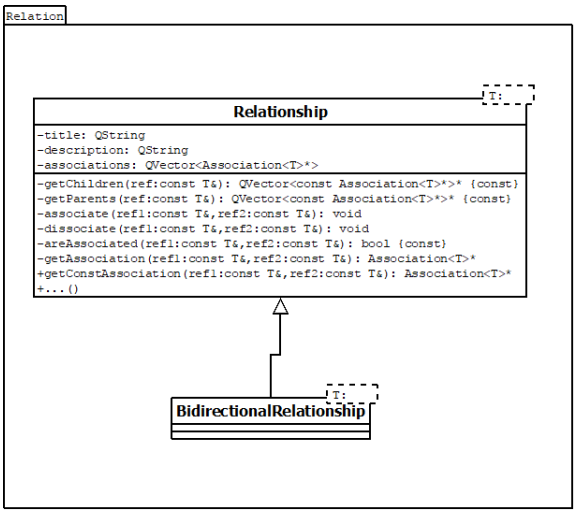

Figure 3: UML simplifié du gestionnaire de relation

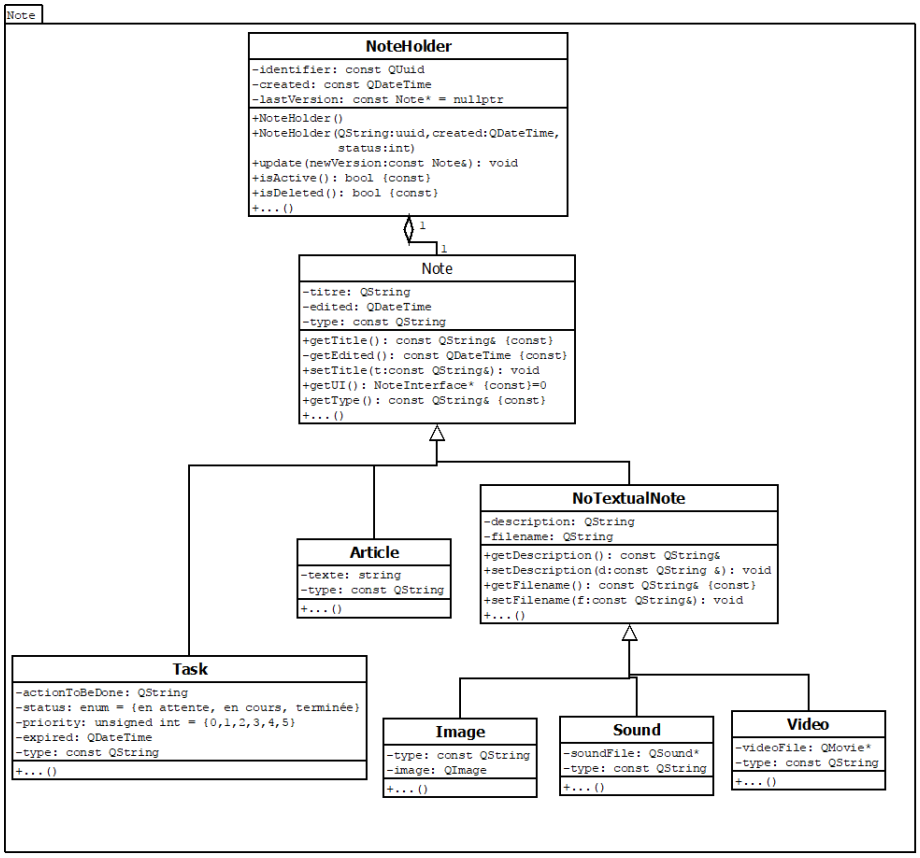

Figure 4: UML simplifié des notes

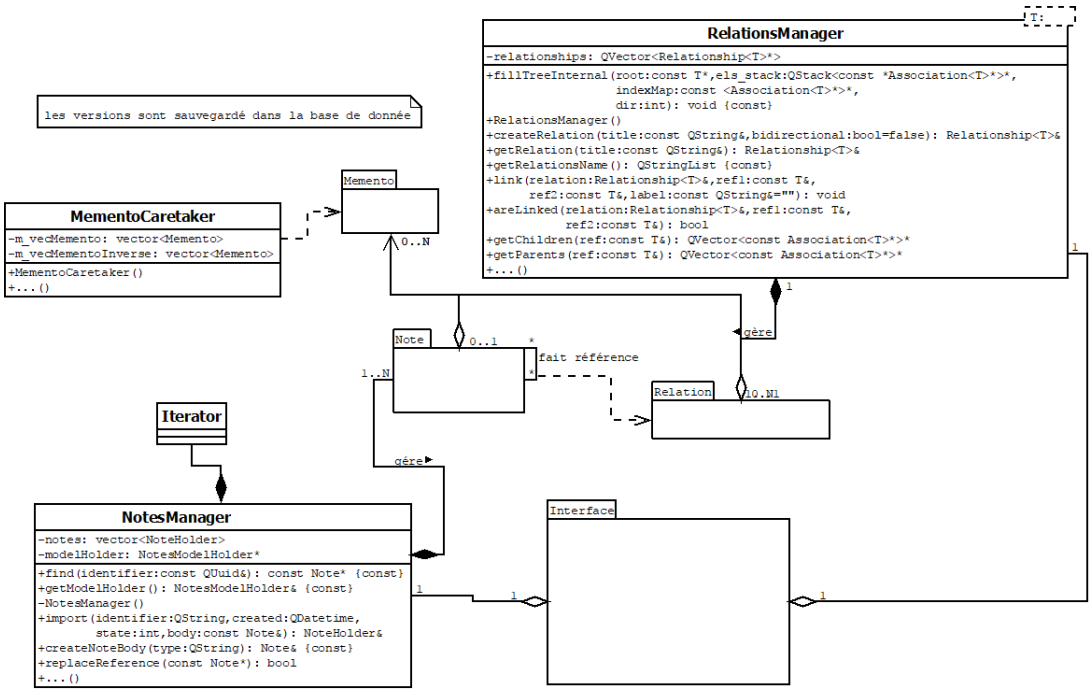

Figure 5: UML simplifié du projet du PluriNotes par packages