Rapport de projet LO21

Camille Beaudou et Justine Marlow

Printemps 2017

Table des matières

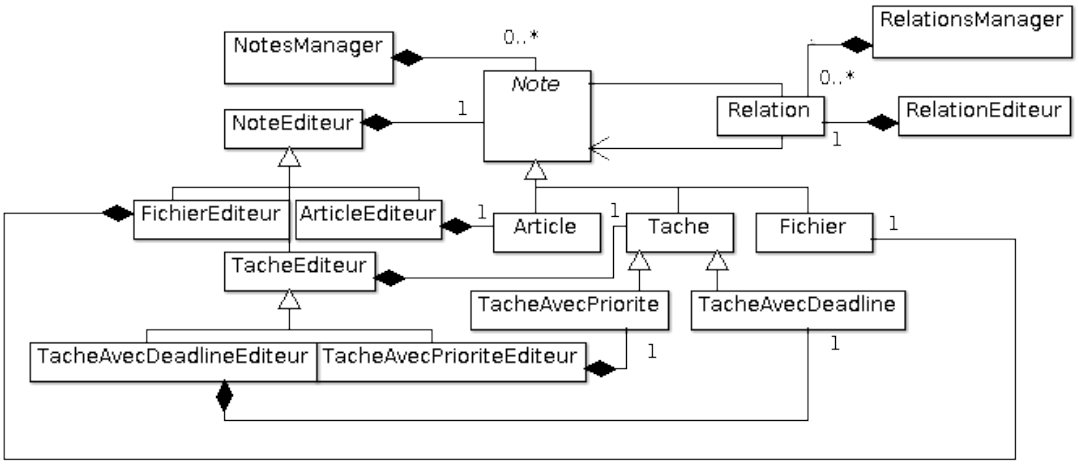
Chapitre 1

Description de l'architecture

1.1 Présentation globale

Nous avons décidé, pour répondre au cahier des charges posé par le sujet du projet *PluriNotes* d'implémenter l'ensemble des classes de l'architecture décrite dans l'UML (1.1).

Figure 1.1: UML : Architecture globale du projet



Cet UML est volontairement constitué d'uniquement les noms des classes, par souci de clarté et de lisibilité. Ces différentes classes, que nous expliciterons dans la suite de ce rapport, peuvent être classées dans différentes catégories qui guideront notre description du projet.

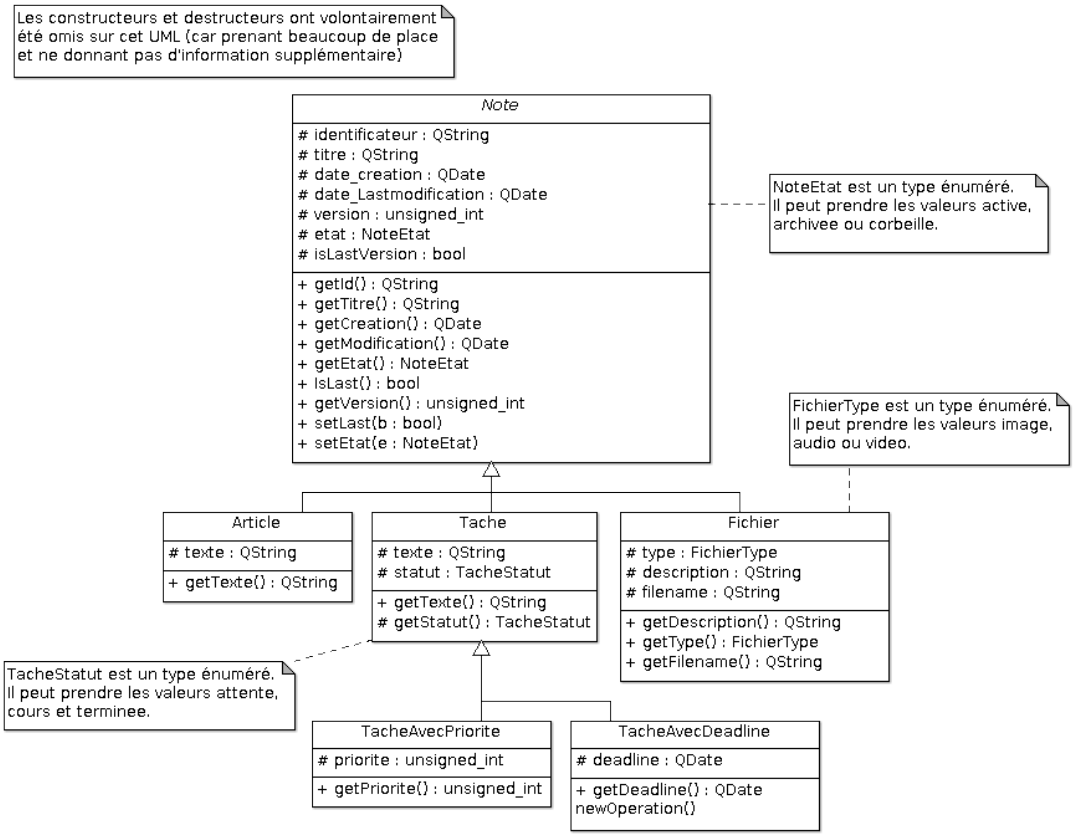
- La classe Note et ses classes filles
- La classe Relation
- Les classes NotesManager et RelationsManager
- Les classes Editeurs

Nous allons détailler pour chacune de ses catégories les diagrammes UML des classes et les choix d'implémentations que nous avons réalisé.

1.2 Présentation de la classe Note et de ses classes filles

Nous avons choisi d'implémenter les classes *Note*, *Article*, *Tache*, *TacheAvecPriorite*, *TacheAvecDeadline* et *Fichier* en utilisant l'héritage. Nous avons donc suivi l'UML (1.2).

Figure 1.2: UML : Architecture de la classe Note et de ses classes filles



1.2.1 Abstraction de la classe Note

Nous avons choisi d'implémenter la classe *Note* comme une classe abstraite. En effet, compte tenu du sujet du projet, nous avons considérer qu'une *note* était soit un *article*, soit une *tâche* (ou *tâche avec priorité* ou *tâche avec deadline*), soit un *fichier*. En effet, la possibilité de gérer une simple *note*, autrement dit un *id* et un *titre* ne nous a pas paru intéressante ni même utile.

1.2.2 Regroupement des images, des enregistrements audio et vidéo dans la classe Fichier

Comme les images, enregistrements audio et vidéo sont tous trois caractérisés par une description et un fichier, nous avons décidé de les regrouper sous une seule et même classe appelée *Fichier*.

1.2.3 Gestion des versions

Pour gérer les différentes versions d'une note, nous avons décider qu'à chaque modification d'un objet *note*, un nouvel objet *note* avec les nouvelles valeurs des attributs est créé. Les attributs

id et *date_ creation* ne changent pas (ainsi plusieurs objets *note* peuvent avoir le même *id* à la condition qu'il s'agisse de deux versions d'une même *note*). Les attributs *isLast* et *version* sont gérés par l'application.

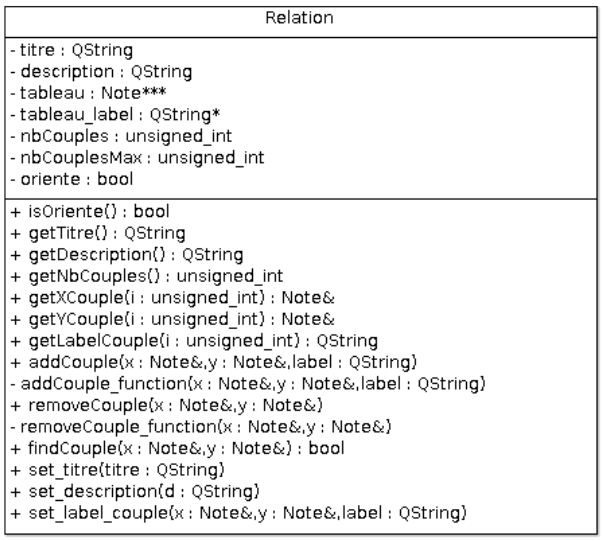
1.2.4 Attributs protected et constructeurs

Les attributs de ces différentes classes sont tous dans le domaine *protected* pour permettre aux classes filles d'accéder à ces attributs. Il a été choisi de ne pas faire apparaître les constructeurs (ni les destructeurs) dans cet UML. Toutefois il est important de stipuler qu'ils sont également dans le domaine *protected* afin que la gestion de la création et de la destruction d'objet *Note* ne puissent se faire que via le *NotesManager* (cf section 1.4).

1.3 Présentation de la classe Relation

Nous avons implémenter la classe *Relation* en tant que classe-association : elle permet d'associer des objets *Note* entre eux. Ainsi, une relation a pour attribut un tableau bi-dimensionnel de *Note**. Nous avons donc suivi les UML (1.3) (à nouveau, nous avons préféré ne pas indiquer les constructeurs et destructeurs comme ils n'apportent pas plus d'information et par souci de clarté et de lisibilité).

Figure 1.3: UML : Architecture de la classe Relation



1.4 Présentation des classes NotesManager et RelationsManager

Les classes *NotesManager* et *RelationsManager* ont été implémentées dans le but de gérer totalement les objets dont elles sont responsables, c'est à dire leur création, leur utilisation et leur destruction. Ainsi, les constructeurs et destructeurs de la classe *Note* et ses classes filles (resp. *Relation*) sont dans le domaine *protected* et il existe une amitié entre ses classes et la classe NotesManager (resp. *RelationsManager*). Nous avons donc suivi les UML (1.4) et (1.5).

Figure 1.4: UML : Architecture de la classe NotesManager

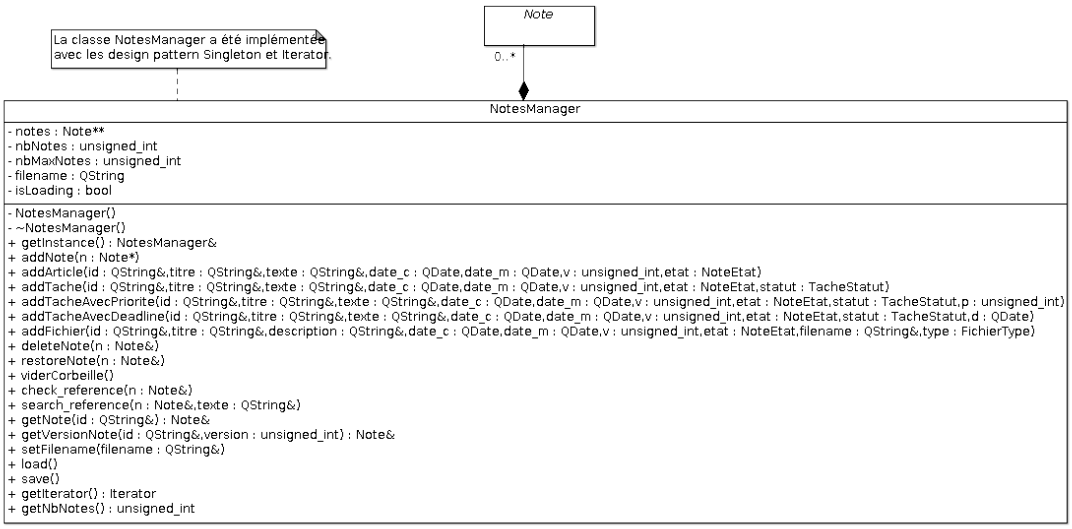
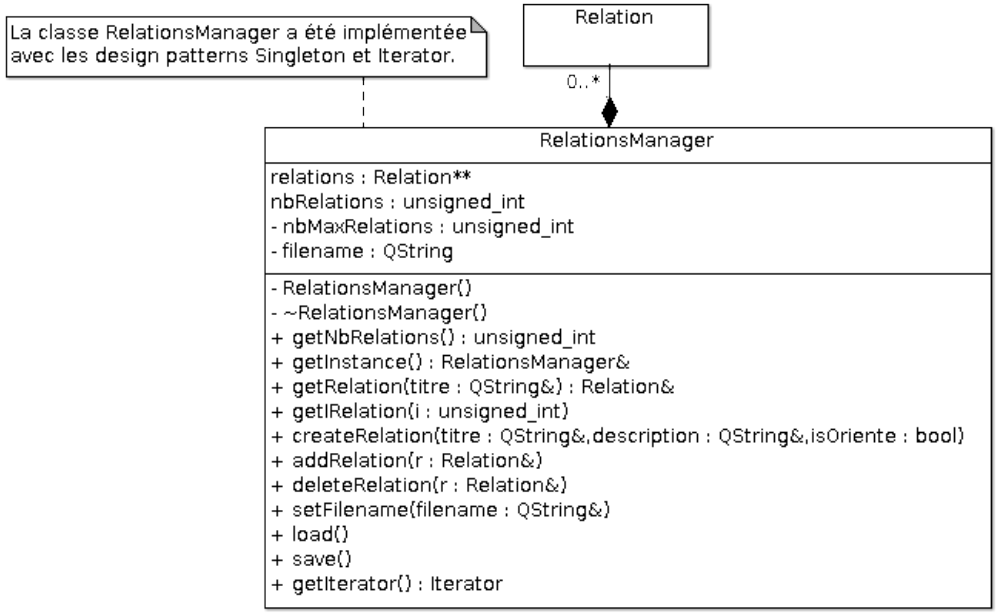


Figure 1.5: UML : Architecture de la classe RelationsManager



1.5 Présentation des classes Editeur

Chapitre 2

Possibilité d'évolution de l'architecture

- Extension à d'autres types de note