Rapport : Un rapport en format .pdf composé de 2 parties :

– la description de votre architecture;

– une argumentation détaillée où vous montrez que votre architecture permet facilement des évolutions.

Vous pouvez ajouter en annexe de ce rapport des instructions à destination de votre correcteur si nécessaire

(présentation des livrables, instructions de compilation, . . . ). Ce rapport ne devra pas dépasser 10 pages

(annexes comprises).

LO21 ProjectCalendar

Rapport

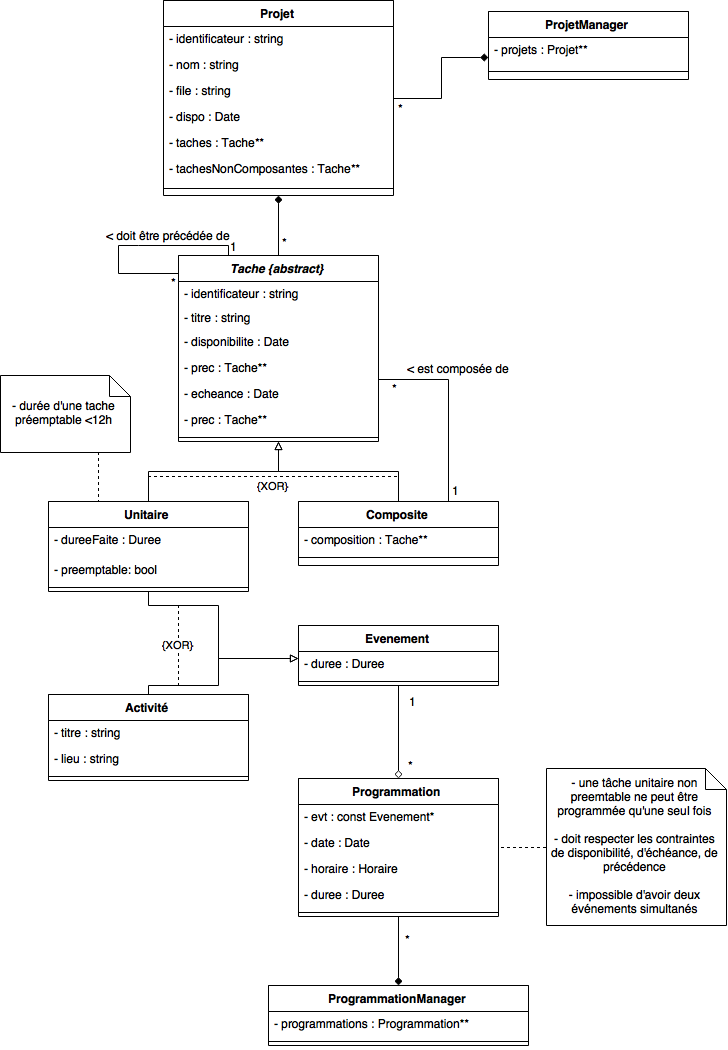
# Introduction

Ce projet de LO21 consiste à créer à l’aide de la Programmation Orientée Objet en C++, une application destinée à mixer des fonctionnalités d’un agenda électronique traditionnel et d’un outil de gestion de projet.

L’agenda en question doit afficher des vues des éléments programmés par semaines, et l’outil de gestion de projet doit pouvoir séparer les tâches des projets en deux catégories : unitaire et composite, ayant toutes deux des contraintes de précédence sur d’autres tâches.

Nous avons utilisé les pages “timing.h” et “timing.cpp” fournies en TD pour manipuler plus facilement les horaires, dates et durées des différents éléments de notre architecture.

# Description de l’architecture



Notre architecture s’oriente en deux parties :

* la gestion de projets ;
* la programmation d’événements.

## Gestion de projets

Les Taches composent des projets et les Projets composent le ProjetManager. Ainsi une tache ne peut pas se trouver dans deux projets à la fois.

### ProjetManager

ProjetManager est un singleton, géré par un handler qui est l’unique élément capable d’instancier et de détruire le ProjetManager. La classe possède un tableau de pointeurs de Projets qui crée les instances de Projet, et un itérateur qui peut parcourir ces projets.

### Projet

Projet est une classe comportant un id généré automatiquement par l’application, un nom et une date de disponibilité donnés par l’utilisateur et un tableau de pointeurs de Taches qui crée les instances de Tache, et un itérateur qui peut parcourir ces taches. Il peut aussi renvoyer une date d’échéance qui dépendra uniquement des dates d’échéance des Taches qui composent le Projet.

Un Projet pourra être exporté en XML, il représentera alors tous ces attributs et les attributs des Taches qui le composent.

### Tache

Tache est une classe abstraite, Unitaire et Composite héritent d’elle. Elle a un tableau de pointeurs de Taches qui représentent les contraintes de précédences et un itérateur qui parcourt ce tableau. Ce tableau ne créera pas de nouvelle Tache, il faudra ajouter des pointeurs de Taches déjà existantes.

Une Tache possède aussi un id généré automatiquement, un nom, une date de disponibilité et une date d’échéance qui sont donnés par l’utilisateur, et un pointeur vers le projet parent. La date de disponibilité ne doit pas être inférieure à celle du projet parent.

### Composite

Composite hérite de Tache, elle a un second tableau de pointeurs de Taches qui montre des contraintes de composition. Ce tableau ne créera pas de nouvelles Taches, il faudra ajouter des pointeurs de Taches déjà existantes.

### Unitaire

Unitaire hérite de Tache. Elle possède une durée (durée totale) donnée par l’utilisateur, un bouléen préemptif qui permettra de savoir si la Tache peut être programmée en plusieurs fois (et qui sera forcément égal à TRUE si la durée de la Tache dépasse 12h), mais aussi une durée faite (initialisée à 0) et une durée restante (qui sera la soustraction de la durée totale avec la durée faite) qui permettront de savoir le temps restant à programmer pour la Tache.

Unitaire hérite aussi d’Evénement décrit plus bas.

## Programmation d’événements

### ProgrammationManager

ProgrammationManager est un singleton dont l’instance est retournée à l’aide d’une structure handler.

Elle possède un tableau de Programmations qui la composent, elle est donc responsable de leur création et destruction par les fonctions d’ajoutProgrammation. Dans le cas d’une programmation d’un événement Activité, c’est cette même fonction qui créera l’activité.

### Programmation

Programmation compose ProgrammationManager. Une Programmation possède une date, un horaire, une durée et un pointeur vers un Evénement. Le principe de la programmation est d’associer date, horaire et durée à un événement.

### Evénement

Evénement est une classe abstraite dont hérite Activité et Unitaire. Elle est composée uniquement d’une durée. Elle sert à pouvoir faire des Programmations à la fois d’Activités et de Taches Unitaires.

### Activité

Activité hérite d’Evénement. Elle possède un nom et un lieu.

# Argumentation

## Protection de l’architecture

On observera d’abord que notre architecture est protégée.

En effet, ProjetManager et ProgrammationManager sont des singletons, ce qui empêche d’avoir plusieurs entités de ces classes, ce qui serait absurde.

Ensuite, la plupart des attributs des classes sont privés et ne peuvent être modifiés que par des méthodes de la classe.

Les constructeurs par recopie ont souvent été interdits afin d’empêcher des situations où deux éléments ont le même identificateur, ou deux programmations sont identiques par exemple.

Les classes Tache et Evenement sont abstraites, empêchant ainsi de les instancier, ce qui conduirait à la programmation d’un élément n’ayant qu’une durée par exemple.

## Evolution de l’architecture

Notre architecture peut facilement évoluer.

En effet, les classes Tache et Evenement étant abstraites, on peut aisément créer de nouvelles classes héritant de l’une, de l’autre ou des deux afin de complexifier les structures de projets avec un nouveau type de Tache ou afin d’introduire un nouveau type d’Evénement ayant une liste de personnes présentes par exemple.