TP4: Conception à base de patrons I

Par : Yanis Toubal - 1960266

Jin Yu TUNG - 2023222

Groupe 05

LOG2410 – Conception logicielle

Polytechnique Montréal

Remis le 17 novembre 2019

2.1.a) L’intention du patron Composite.

Le patron Composite est un patron structurel. Le but de ce patron est de définir des objets pouvant être composé à partir d'objets de base et de traiter tous ces objets de la même façon. On peut représenter un patron composite avec un diagramme en arbre comme on peut le voir sur la figure 2.1 b) où tous les objets sont des feuilles. Cela se traduit dans notre cas par l'utilisation de la classe "TeamMember" qui est la classe de base et de Team qui est composé de plusieurs TeamMember.

2.1 b)

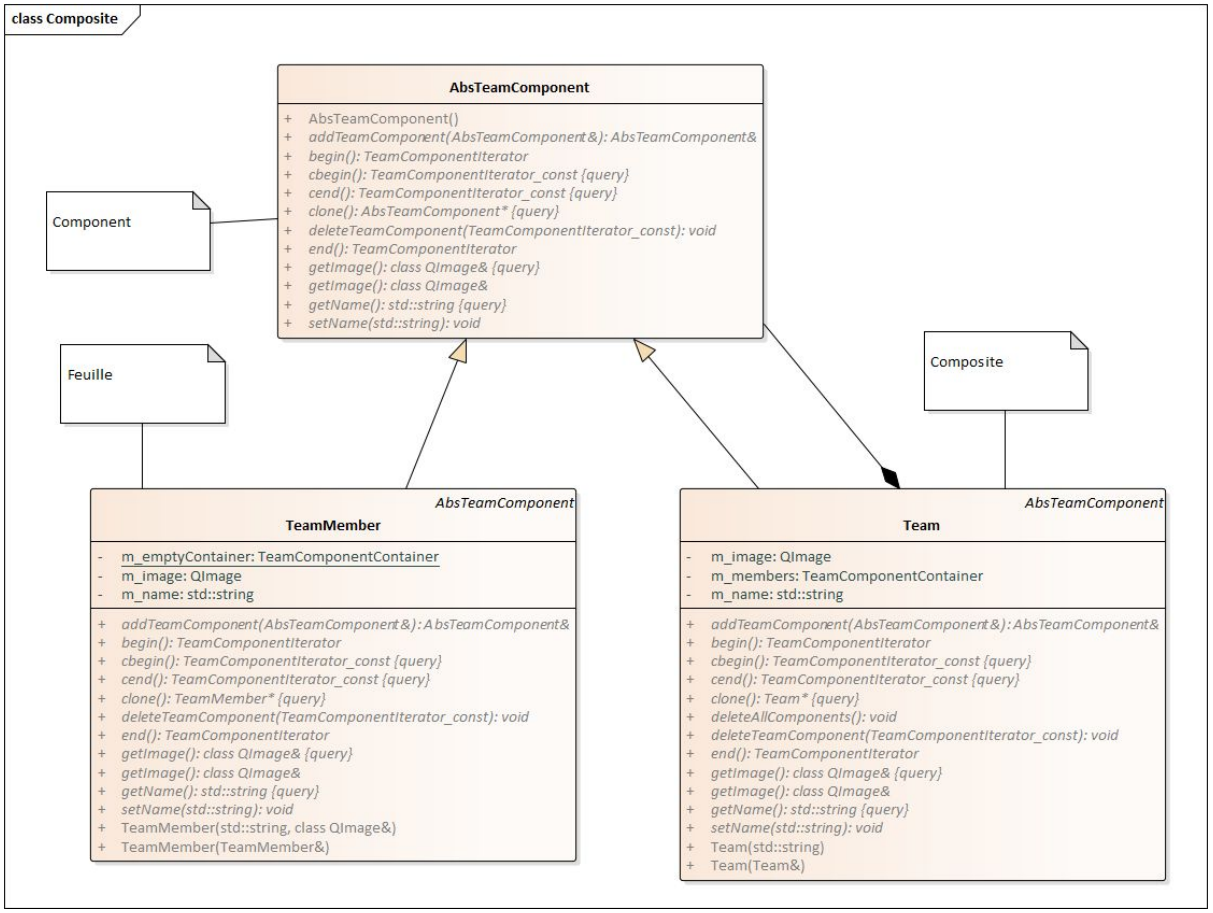

Diagramme UML du patron composite

“AbsTeamComponent” joue le rôle de component dans le patron de composite qui définit les composites et les objets de base de façon uniforme.

“TeamMember” est un objet feuille qui n'a pas d'enfant. Il est la composant de base de Team.

“Team” répresente la composite de TeamMember. Une team est composée de plusieurs TeamMember.

3.1.a) L’intention du patron Decorator.

Le patron Decorator est un patron structurel. Le but du patron Decorator est de permettre de modifier un seul objet de façon dynamique sans modifier les autres objets de la même classe et sans avoir à faire usage de l'héritage. Son principal avantage est qu'il permet d'être flexible aux niveaux des responsabilités en permettant d'ajouter ou d'enlever des responsabilités en cours d'exécution du programme. Dans notre code, le patron Decorator est représenté dans la classe "TeamMemberRole" qui a un accès dynamique vers un objet de type "TeamMember".

3.1.b)

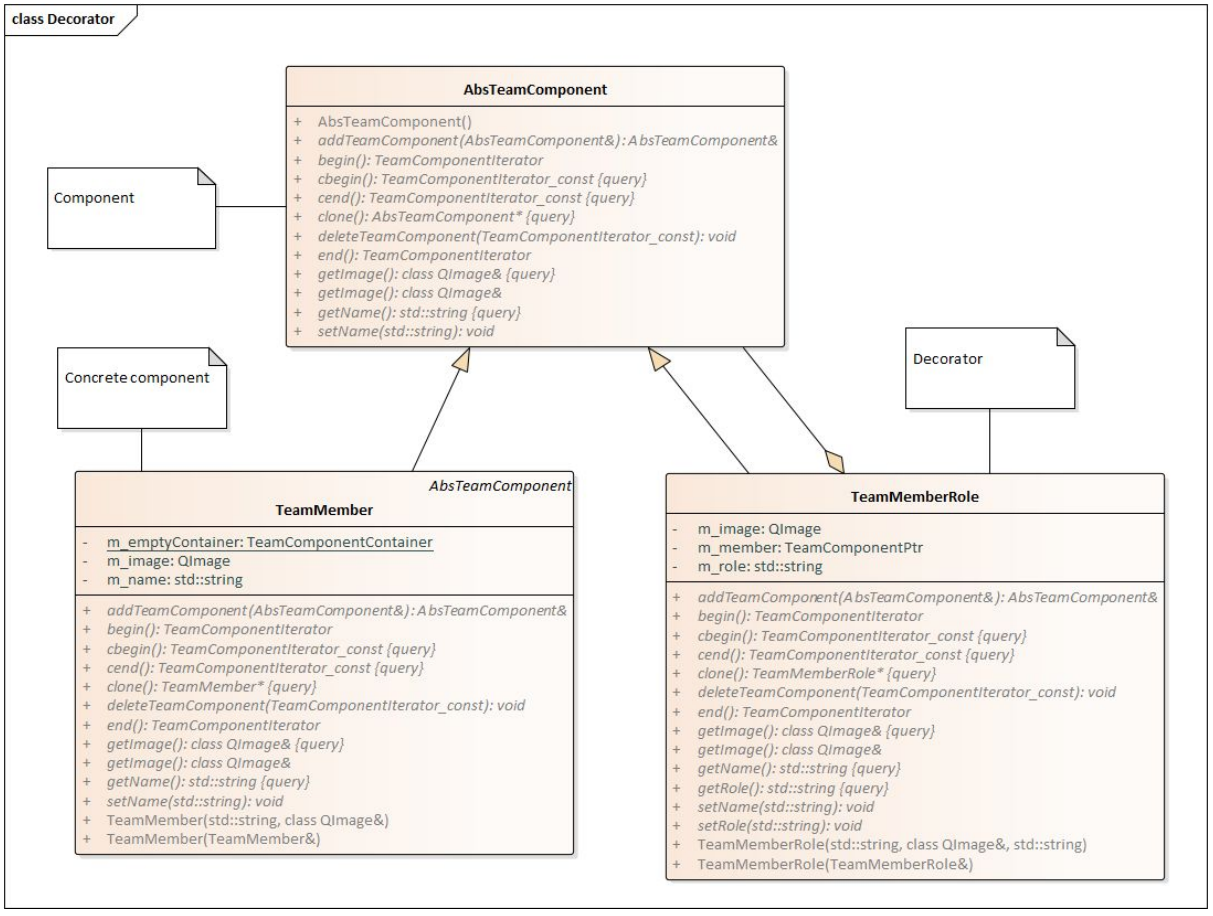

Diagramme UML du patron decorator

“TeamMemberRole” est le decorator. On crée une association dynamique vers les TeamMember à traver cette classe. Un objet de TeamMemberRole peut modifier l'état d'un TeamMember grâce à cette association.

“TeamMember” est le concrete component du patron Decorator. La classe Decorator peut appeler les méthodes de “TeamMember”.

4)

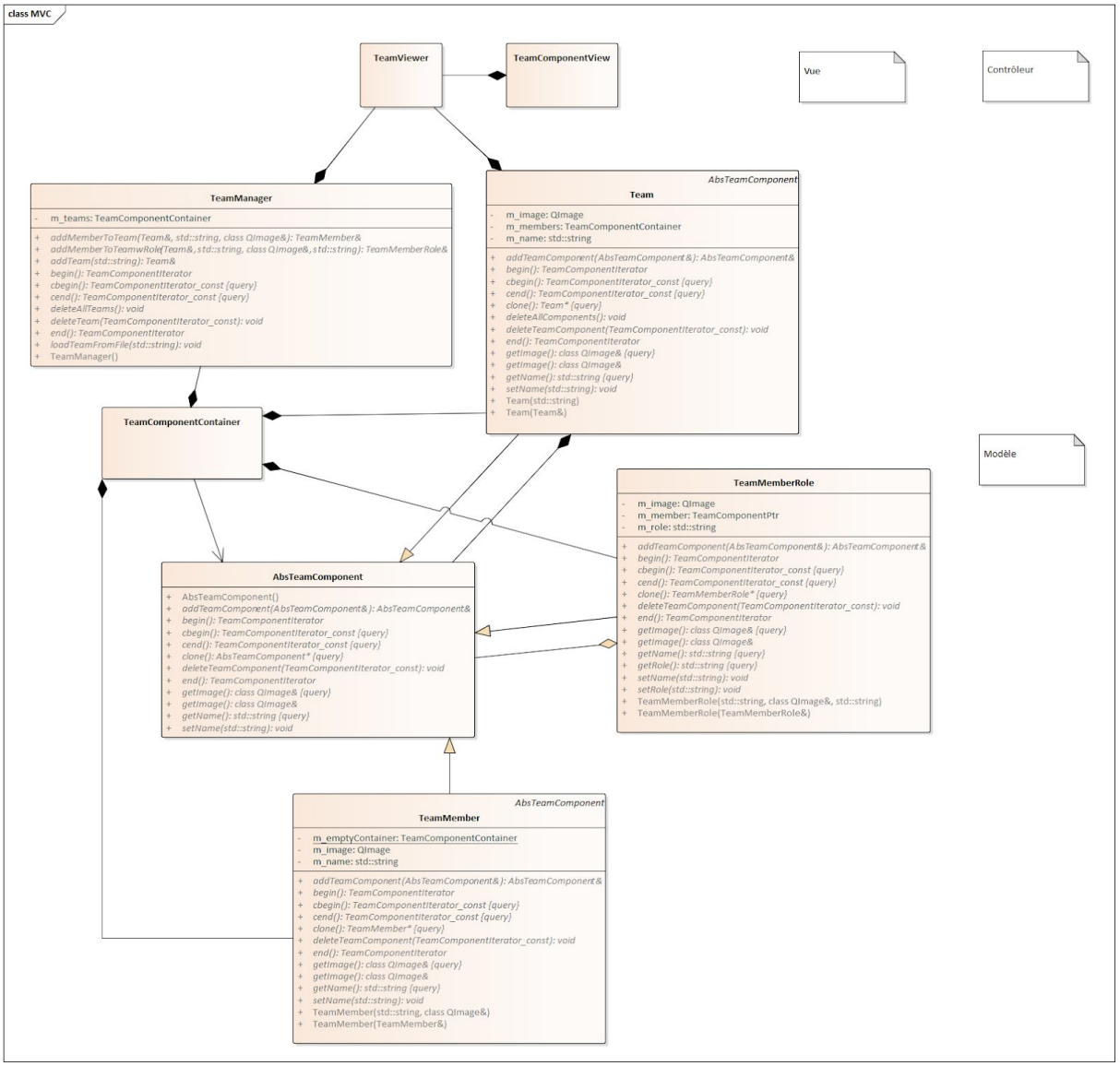

Diagramme UML du patron architectural Modèle-VueContrôleur

Sous-Vue: TeamComponentView
Vue-Contrôleur: TeamViewer
Modèle: TeamComponentContainer, AbsTeamComponent, TeamMember, TeamMemberRole, Team, TeamManager

L'application TeamViewer ne respecte pas l'architecture MVC, car la bibliothèque Qt n'utilise pas le modèle MVC, mais plutôt Modèle-Vue et c'est de cette façon que le code a été séparé. Dans notre cas, la vue et le contrôleur sont mis ensemble dans la classe TeamViewer.

Tandis que TeamComponentView est une sous classe de la vue et que toutes les autres classes font parties du modèle.

Vue:
La classe “TeamComponentView” est une sous-classe appartenant à la vue qui gère l’affichage d’un TeamMember. Elle est ensuite utilisé dans la classe “TeamViewer” pour afficher une équipe dans l'application qui peut être composée de plusieurs membres d’équipes ou encore de plusieurs sous-équipes.

Vue-Contrôleur:
La classe “TeamViewer” joue le rôle de la vue, mais aussi du contrôleur. Elle gère les évènements et l’interface. Elle est en charge de modifier la vue et le modèle selon les actions de l'utilisateur.

La classe TeamView lie les buttons de la vue à des slots dans la méthode createActions() ce qui va permettre de modifier le modèle pour ensuite mettre à jour la vue, en appelant le slot associée à l'option, à chaque fois que l'utilisateur clique sur une des options du menu. La classe updateActions() va aussi dans ce sens en permettant d'activer certaines options qui sont désactiver par défaut.

La méthode loadFile(AbsTeamComponent& team, const QString &, const QString &) sert à changer l’état du modèle alors que la méthode saveFile(const QString &fileName) est en charge de sauvegarder l’état du modèle. Ces deux méthodes font partie des tâches du contrôleur.

Modèle:
Les autres classes font partie de la logique et des données de notre modèle.