Outils et méthodes pour le développement
TP – Conception Orientée-Objet - Mini éditeur

Molinier Camille

28/09/2022

J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée.

Table des matières

1 Introduction

Le but de ce TP est de concevoir un mini éditeur de texte. Cet éditeur doit permettre d'effectuer les commandes d'un éditeur de texte classique (comme le bloc note natif de Windows par exemple). Au vu du cahier des charges, on peut dors et déjà dire que l'on va utiliser le patron de conception *command*. Celui-ci généralise les fonctionnalités sous forme d'appels successifs de commandes.
Ce rapport traite de toute la partie conception du projet, il sera question de la définition des cas d'utilisations, puis on verra les digrammes de séquences associés, ce qui donnera naissance au diagramme de classe. Le rapport sera séparé en deux parties, une pour chaque version du projet, ce qui permettra de finir le rapport par étudier le passage de la conception de la version 1 à celle de la version 2.

2 Version 1

La première version du projet doit permettre d'écrire du texte, le supprimer, de copier une sélection ou la couper, puis la coller à une position souhaitée ou en remplacement d'une sélection.

2.1 Diagramme de use case

La première étape de la conception est de partir des besoins clients pour définir des use cases. Ici, les besoins sont des use cases. Le diagramme en figure 1 se lit donc très facilement.

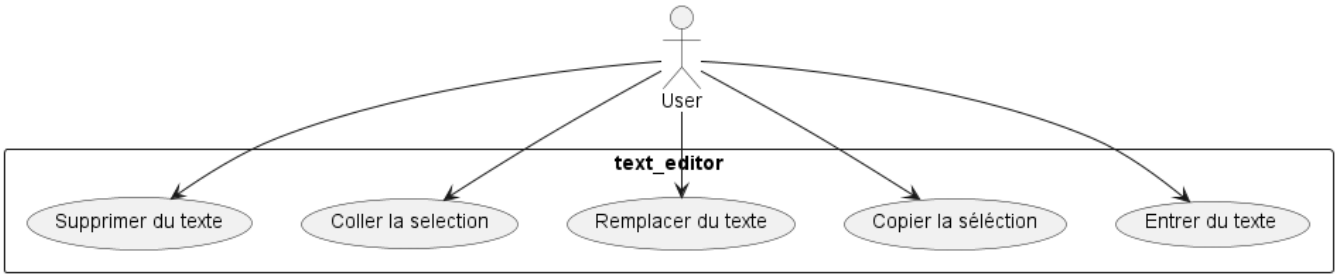

FIGURE 1 – Use case diagram

Même si ce diagramme semble simple et peu utile vu le cahier des charges explicite, il est quand même utile pour fixer une direction dès le départ du projet et permettra de vérifier facilement la concordance avec l'implémentation.

2.2 Diagramme de séquence

La seconde étape permet de décrire les use cases vu précédemment, c'est la passerelle entre les use cases et le diagramme de classe. C'est dans ce type de diagrammes que l'on verra apparaitre le début de notre patron de conception. Celui-ci se compose d'un client qui régit le fonctionnement global de l'application, d'un invoquer, qui préparera les commandes et les exécutera, d'une ou plusieurs commandes qui contiennent les suites d'opérations à effectuer sur le système, et d'un receveur qui sera en charge d'effectuer les modifications de l'état du système.

Le premier use case que l'on va traiter est celui pour couper du texte. Le résultat attendu est la suppression du texte sélectionné après l'avoir sauvegardé dans le presse-papier. Comme on peut le voir sur la figure 2, on retrouve le client (GUI), l'invoker, la commande (Cut) et le receveur du patron de conception. Pour accomplir la fonctionnalité, il reste à ajouter de quoi sauvegarder la sélection (Clipboard) et un conteneur de l'état du système (Buffer).

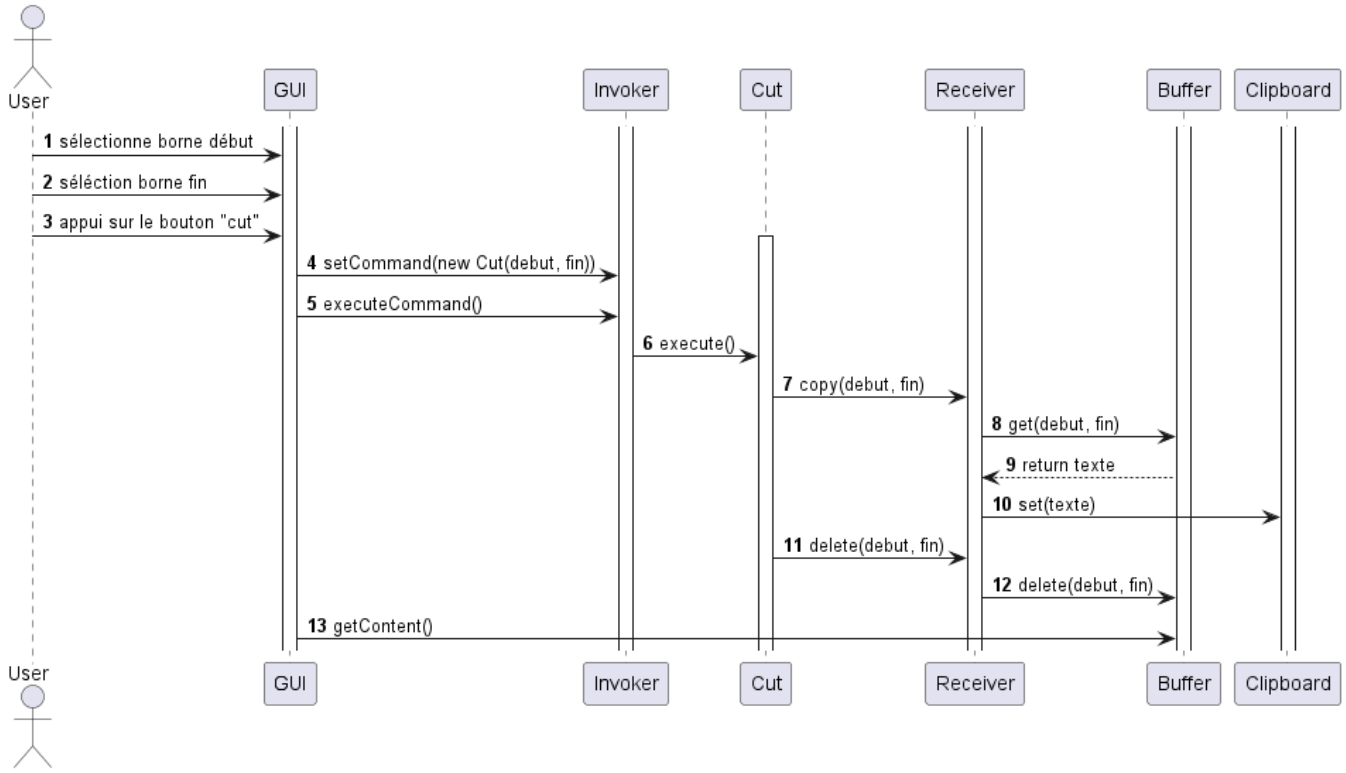

FIGURE 2 – Cut sequence diagram

Le scénario classique de ce use case commence lorsque le client appuie sur le bouton de copie. La GUI récupère les informations sur la borne de début et de fin puis crée et exécute une nouvelle commande de type Cut. Cette commande va copier le contenu du buffer entre le début et la fin dans le Clipborad puis supprimer le contenu du buffer entre la borne de début et de fin. Pour finir, la GUI vient récupérer le contenu du buffer pour actualiser son affichage.

Le second cas que l'on présentera dans ce rapport est celui de l'insertion. On peut remarquer sur la figure 3 que les participants de ce scénario sont sensiblement les mêmes que le précédent, cela est dû à l'utilisation du patron de conception.

Le scénario commence lorsque l'utilisateur appuie sur le bouton "copier". La GUI va récupérer les informations des positions des curseurs puis générer une nouvelle commande de type Insert dans l'Invoker puis lui demander de l'exécuter. Cette commande va demander au receveur de supprimer le contenu du buffer entre les bornes puis d'insérer le contenu du presse-papier dans le buffer à la position de début de la sélection.

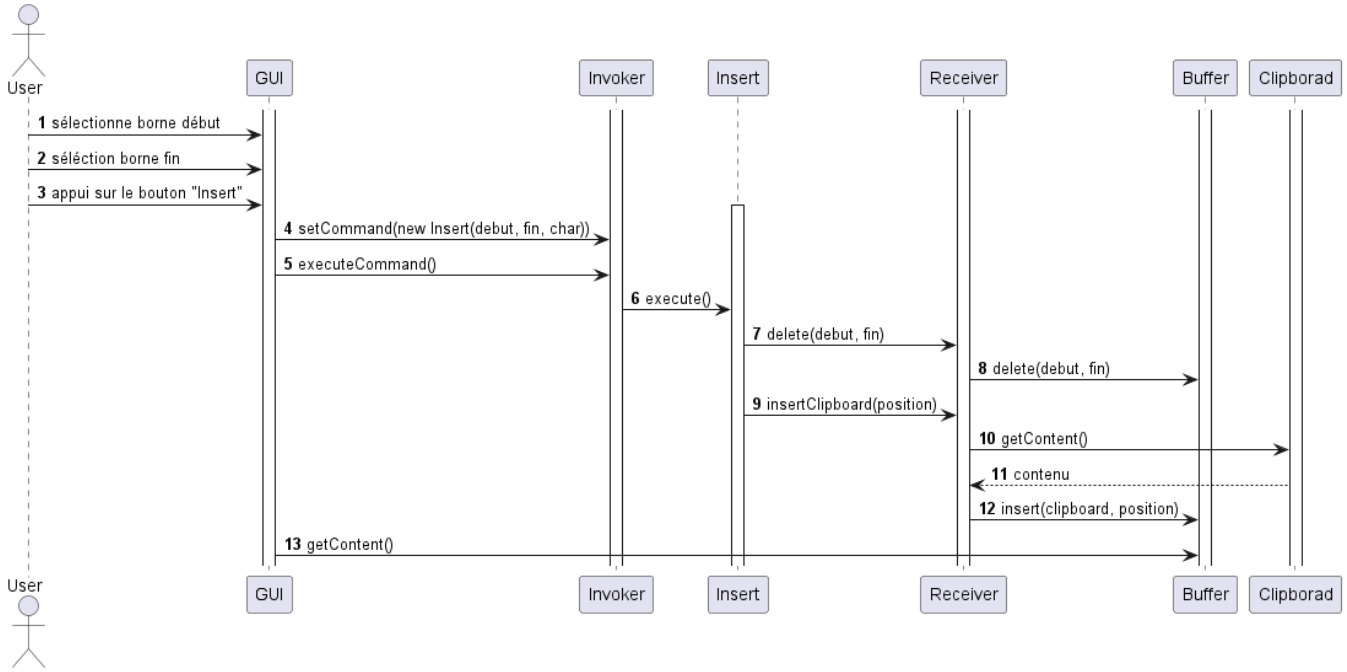

FIGURE 3 – Insert sequence diagram

2.3 Diagramme de classes

Les diagrammes de séquences ont permis de faire apparaitre une grande partie des classes dont on aura besoin pour l'implémentation du projet ainsi que la plupart des relations entre elles.

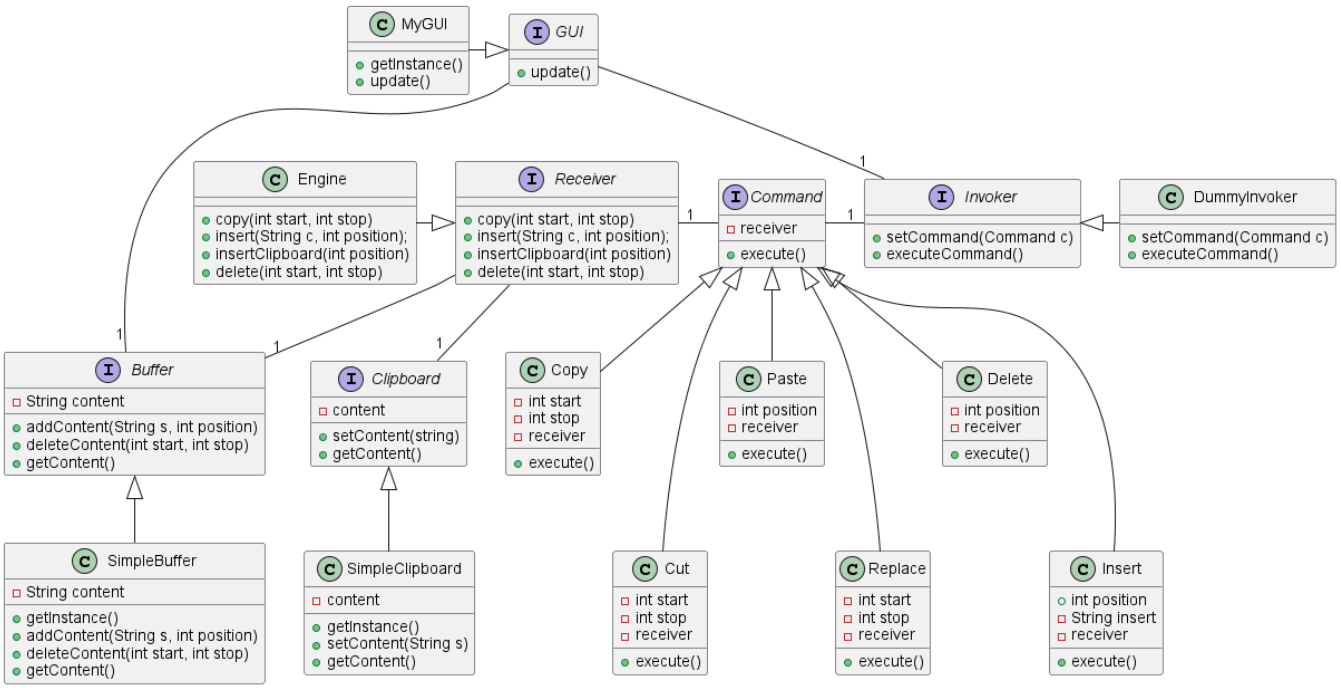

FIGURE 4 – Class diagram

La figure 4 montre le diagramme de classe de la première version. Le squelette du patron de conception command est entièrement en interface, on trouve GUI, Invoker, Command et Reciever. On rajoute les deux interfaces Buffer et Clipboard qui régissent le buffer et le presse-papier avec leurs opérations de manipulations. Le fait d'avoir choisi des interfaces permet le remplacement des implémentations. Par exemple, la GUI peut être faite avec un affichage dans le terminal dans un premier temps, puis on peut la changer pour une interface graphique avec java swing, la seule modification à faire dans le code sera dans la déclaration de l'objet de type GUI.

Les diverses interfaces ont toutes leur implémentation. L'interface Command est la seule à posséder plusieurs implémentations, comme chaque commande correspond à une fonctionnalité (pour respecter le concept de single responsabilité). Cela permet de rajouter des fonctionnalités facilement, il suffira d'ajouter une nouvelle classe implémentant l'interface Command.

Concernant les cardinalités, une GUI a un Invoker, l'Invoker a une Command, la Command a un Reciever qui possède un Clipboard et un Buffer. La GUI possède un lien avec le Buffer pour pouvoir accéder au contenu.

3 Version 2

La seconde version du projet doit permettre d'anuller ou refaire une commande ainsi que d'enregistrer des commandes utilisateurs. Le tout doit toucher le moins possible à la première version.

3.1 Diagramme de use case

Cette seconde version reprend toutes les fonctionnalités de la première version et rajouter les fonctions Undo, Redo ainsi que la possibilité de jouer ou enregistrer des scripts. Un nouvel acteur intervient dans les nouveaux cas d'utilisation, il s'agit des fichiers. Ils interviennent lors de la réalisation ou la lecture de scripts et permettent d'écrire ou lire les fichiers scripts contenant les commandes utilisateur.

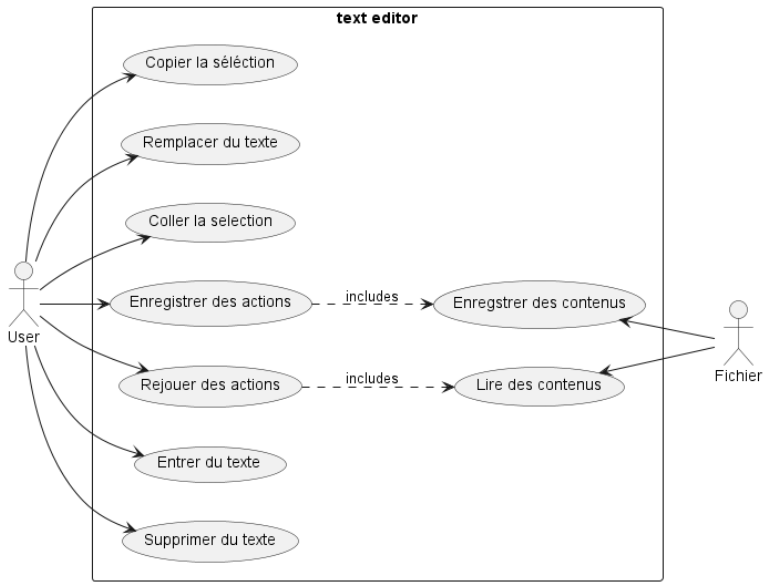

FIGURE 5 – Use case diagram

3.2 Diagramme de séquence

Les nouvelles fonctionnalités introduisent de nouveau comportement de l'application. L'ajout des nouvelles fonctionnalités demande l'ajout d'un nouveau patron de conception, memento. Celui-ci régit la capture d'états du système aux moments voulus. Il se compose d'un Originator qui déclenchera les sauvegardes ainsi que les restaurations, d'un CareTaker qui stockera les sauvegardes et d'objets Memento qui contiennent une sauvegarde de l'état du système.

Le premier cas que l'on va traiter est celui de la fonction Undo. Cette fonctionnalité permet d'annuler la dernière commande exécutée, cela revient à revenir au dernier état du système. Ce scénario commence lorsque l'utilisateur appuie sur le bouton "Undo". La GUI va transmettre une nouvelle commande de type Undo à l'invoker puis l'exécuter. Lors de son exécution, la commande Undo va commencer par demander à l'Originator de restaurer l'état précédent du système. L'Originator récupère l'état dans le CareTaker puis renvoie l'objet Memento à la commande Undo. Celle-ci va demander au Reciever de supprimer l'intégralité du contenu du Buffer puis d'y insérer le contenu du Memento. Enfin, la GUI mettra à jour son affichage en récupérant le contenu du Buffer. L'objet Memento n'est pas supprimé au cours de la procédure, car il doit encore pouvoir être appelé en cas de commande Redo.

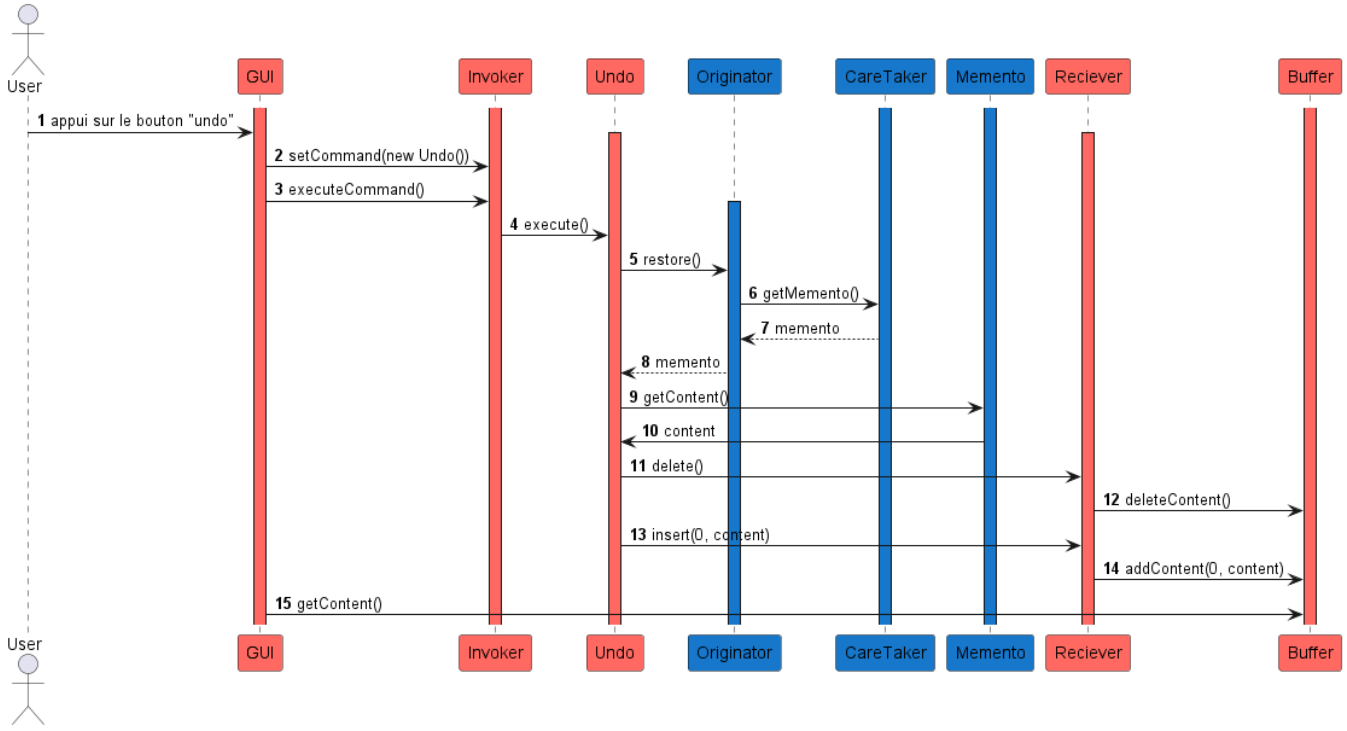

FIGURE 6 – Undo sequence diagram

Le second cas que l'on va traiter est la création de scripts. Cet enchainement doit permettre à l'utilisateur d'enregistrer des commandes pour pouvoir les rejouer plus tard. Le scénario commence lorsque l'utilisateur appuie sur le bouton "start". La GUI va transmettre l'instance de la commande script à l'Invoker puis l'exécuter une première fois. Ceci va déclencher la récupération du Memento courant en guise de point de départ. L'utilisateur peut alors faire toutes les commandes qu'il souhaite jusqu'à l'appui une seconde fois sur le bouton "start". La GUI va alors transmettre à nouveau l'instance de la commande script à l'Invoker puis l'exécuter une seconde fois. Lors de cette seconde exécution, la commande Script va demander au CareTaker de lui donner la liste de tous les Memento depuis le point de départ défini précédemment. Enfin, la commande Script va pouvoir sauvegarder la liste dans le fichier et ainsi finir le scénario. ce cas d'utilisation fait apparaitre le comportement particulier de la commande Script et explique pourquoi le choix de l'utilisation du patron singleton a été fait.

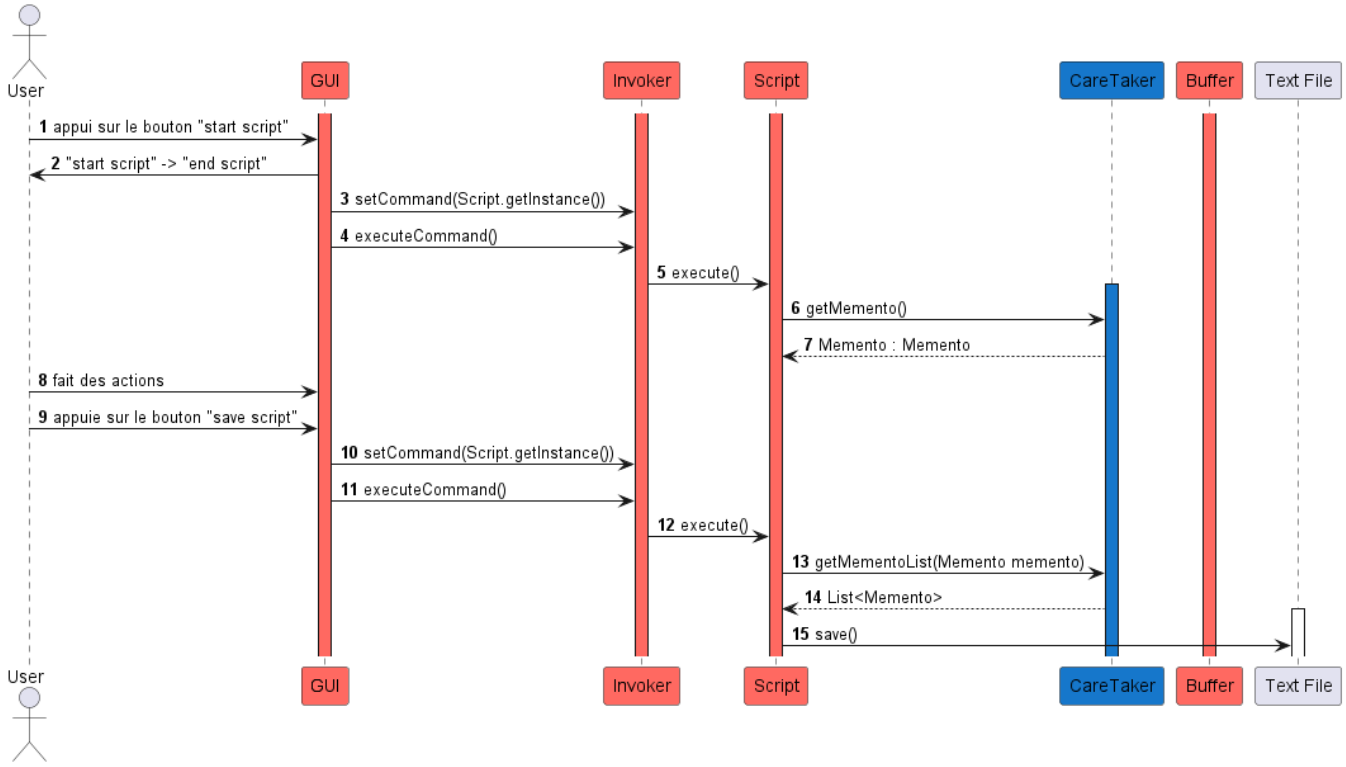

FIGURE 7 – Script sequence diagramm

3.3 Diagramme de classes

Les diagrammes de séquence vus précédemment permettent de voir les ajouts à faire sur le diagramme de classe de la V1, un nouveau patron de conception doit être ajouté et quatre nouvelles commandes aussi. Le diagramme commence à devenir tentaculaire, c'est pourquoi un code couleur a été mis en place :

rouge : interfaces du patron Command
orange : implémentations du patron Command
bleu clair : interfaces du patron Memento
bleu foncé : implémentations du patron Memento

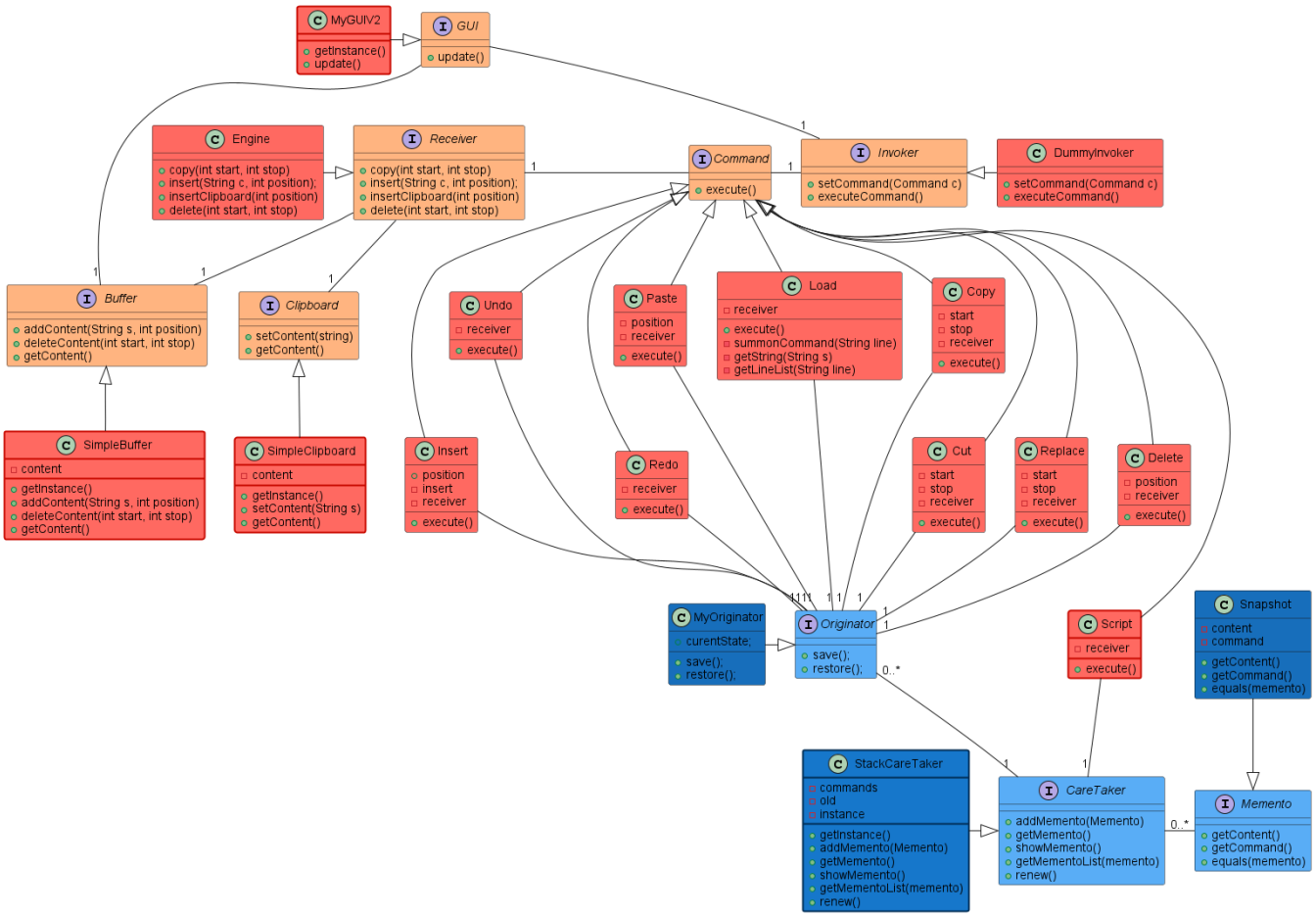

FIGURE 8 – Class diagram

Comme pour la V1, l'extension du diagramme de classe sera d'abord faite en interfaces, on y retrouve les interfaces Originator, CareTacker et Memento. Pour le patron Memento, utiliser les interfaces est pertinent notamment avec le CareTacker car il existe beaucoup de méthodes de stockage. Si l'implémentation en cours n'est pas bonne (mauvais choix de structure de donnée par exemple), on peut simplement la remplacer par une autre implémentation en ayant juste les déclarations à changer (ou uniquement dans la Factory si on avait rajouté ce patron), le reste du projet ne verra pas de changement grâce au contrat.

Les nouvelles interfaces ont toutes leur implémentation et doivent respecter les contrats à la lettre. L'interface Command reçoit quatre nouvelles implémentations : Undo, Redo, Script et Load.

Toutes les nouvelles classes donnent lieux à de nouvelles cardinalités. Les commandes ont toutes (sauf script) un lien de cardinalité 1 vers l'Originator. La commande Script a un lien unaire vers le CareTaker. L'Originator à un lien unaire vers le CareTaker et le CareTaker à un lien 0...* vers l'Originator. Cela s'explique part le fait que le CareTaker est unique, mais peut être appelé par plusieurs Originator si besoin. Enfin, le CareTaker a un lien 0...* vers Memento, car il peut stocker autant de Memento que souhaité, mais peut aussi être vide.

4 Conclusion

Dans ce rapport, on a vu toute la partie conception de l'application. On est parti d'un cahier des charges explicite que l'on a traduit en diagramme de Use Case. Ce diagramme a donné lieu à la description séquentielle de certains des cas d'utilisation qui ont permis de mettre en lumière les différents objets dont le projet va avoir besoin. Ces objets ont ensuite étés regroupés dans un diagramme de classe faisant apparaitre les différents partons de conception (Command, Memento et Singleton).

L'étape suivante est de traduire cette conception en implémentation. Si la conception est bien faite, l'implémentation devrait être plus rapide que si l'on s'était lancé directement dans le code.

Toutefois, cette conception n'est qu'une vision possible du projet. C'est pourquoi il est possible de l'améliorer, par exemple avec l'utilisation du patron Factory qui aurait profité au maximum de la conception actuelle avec toutes ses interfaces.