TP3, INF5153, UQAM, Hiver 2013

Mini-Éditeur
Document de conception détaillée

Version <2.0>
Date : 17/04/2013

Auteur: Moussa SOW

Historique des révisions

Date	Version	Description	Auteur
01/04/2013	1.5	Mise en place du gabarit	Moussa SOW
10/04/2013	1.6	Modification du Diagramme de Classe	Moussa SOW
14/04/2013	1.8	Diagrammes de Séquences	Moussa SOW
17/04/2013	2.0	Dernière Modifications et Corrections	Moussa SOW

Table des matières

Conception détaillée

Introduction

1.1 But du document

L'objet de ce document est de décrire la conception architecturale d'un mini-éditeur de texte avec ses fonctions de bases : insérer, supprimer, sélectionner, copier, couper, coller du texte, annuler l'action, refaire l'action et utiliser des enregistrements (macros). Le système sera décrit à l'aide de diagrammes et d'exemple de l'interface graphique.

1.2 Public visé

Le public visé par ce document est tout d'abord l'équipe de production qui sera en charge de produire ce logiciel. Ce document démontre les concepts de bases rattachés à la conception d'un éditeur de texte. De plus, celui-ci est aussi destiné au client afin de s'assurer que ses besoins sont bien rencontrés avant de débuter la production.

1.3 Définitions, acronymes et abréviations

Éditeur de texte : Logiciel qui permet de faire l'édition de texte. Il comprend les fonctions suivantes : insérer, supprimer, sélectionner, copier, couper, coller du texte, annuler et refaire une action, démarrer un enregistrement, arrêter un enregistrement et rejouer un enregistrement.

Copier : Action de copier du texte qui est préalablement sélectionné à l'aide du curseur. La sélection reste afficher après l'action.

Couper : Action de copier du texte qui est préalablement sélectionné à l'aide d'un curseur. La sélection est supprimée après l'action.

Coller : Action de coller du texte qui est préalablement copié ou couper.

Défaire : Action d'annuler la dernière action fait par l'utilisateur.

Refaire : Action de refaire la dernière action préalablement défaite par l'utilisateur.

MVC : Modèle architectural (Modèle - Vue - Contrôleur)

UC : Use-Case

Texte_Editor : C'est l'éditeur de texte. Il représente la vue principale des utilisateurs. Il contient l'ensemble des boutons (actions) d'édition du texte que l'utilisateur entrera. Il est composé d'un Buffer et d'un Text_Area.

Texte_Area : l'endroit où seront affichées les modifications apportées au texte suite aux actions sur le buffer.

Buffer : Représente les données. Il emmagasinera le texte et c'est sur lui que les actions seront réalisées. Ensuite c'est lui qui affichera les modifications dans le texte Area.

1.4 Références

Référence	Auteurs
Les Design Patterns en Java	**Steven John Metsker**
Les 23 modèles de conception fondamentaux	**et William C. Wake**
INF5153 : Génie logiciel	**Naouel Moha**
Chapitre 5 – Conception détaillée – Patrons de conception	

Tableau 1: Documents référencés

1.5 Structure du document

Ce documents en divisé en 5 sections. Premièrement, une description sommaire du système à concevoir sera décrite à l'aide des détails fournis par le client. Deuxièmement, un modèle de domaine démontrera les concepts importants du domaine ainsi que leurs relations.
Troisièmement, un diagramme de cas d'utilisations illustrera les fonctionnalités dont les différents acteurs auront accès. Chacun de ces fonctionnalités seront approfondit à l'aide de diagrammes de séquences. Quatrièmement, une description du modèle architectural démontrera les représentations (les vues) de l'architecture du système. Le système et les sous-systèmes y seront décrits. Cinquièmement, une capture d'écran sera présentée ainsi qu'une courte description.

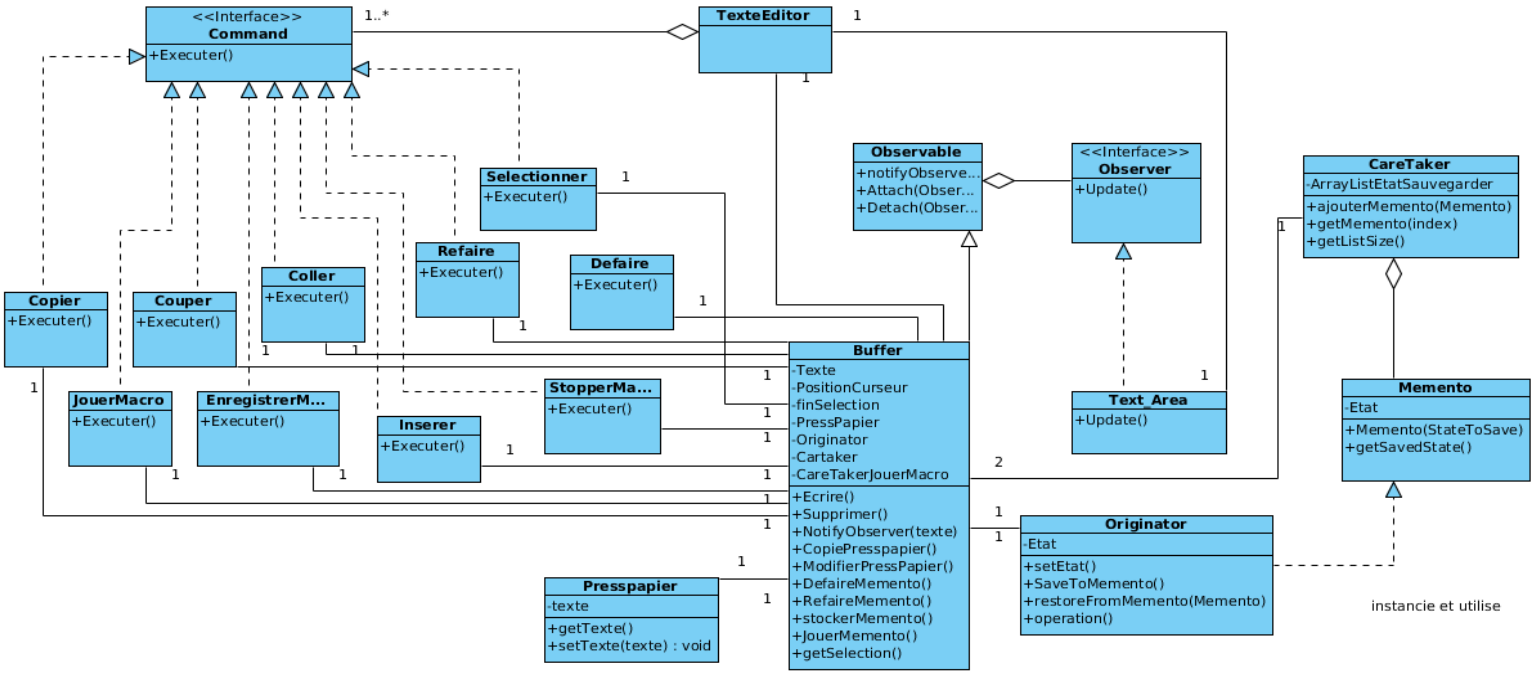
Description sommaire du système

Le système porte sur le développement d'un mini-éditeur de texte (tel que Notepad) qui dispose des actions d'édition de base : *sélectionner, copier, coller, couper, insérer, supprimer, défaire* et *refaire* une action. On pourra aussi enregistrer puis rejouer les actions de l'utilisateur.

Le mini-éditeur comporte les concepts et les actions suivantes :

- **Insérer :** l'utilisateur peut entrer du texte ;
- **Supprimer :** l'utilisateur peut supprimer du texte ;
- **Sélectionner :** l'utilisateur peut définir le début et la fin d'une sélection ;
- **Copier :** la copie de la sélection dans le buffer;
- **Couper :** la copie de la sélection dans le buffer puis effacement de la sélection ;
- **Coller :** le remplacement de la sélection par le contenu de la copie présente dans le buffer;
- **Défaire :** l'utilisateur peut annuler la dernière action faite.
- **Refaire :** l'utilisateur peut refaire une action préalablement défaite (annulée)
- **Démarrer un enregistrement (macro) :** l'utilisateur peut démarrer l'enregistrement de ses actions.
- **Arrêter un enregistrement (macro) :** l'utilisateur peut arrêter l'enregistrement de ses actions.
- **Rejouer un enregistrement (macro) :** l'utilisateur peut rejouer un enregistrement préalablement enregistré.

Diagramme de classes



Les patrons utilisés sont :

- **Le patron Commande :** Le patron COMMAND est un patron de type comportemental qui sert à encapsuler une requête dans un objet, de façon à être manipulés et étendus comme tout objet et surtout, nous permet de :

  -Supprimer tout couplage entre l'objet qui invoque une opération et celui qui la réalise
  -Faciliter l'ajout de nouveaux objets Command, car cela n'implique pas de modifier les classes existantes.
  -Assurer un service de défaire et refaire (avec utilisation du patron MEMENTO pour une meilleure robustesse)

  Nous l'avons utilisé dans notre contexte parce qu'il permettra à notre éditeur de texte d'envoyer des requêtes à des objets sans connaître l'opération qui est demandée ni le destinataire de ces requêtes. Les différentes actions (Copier, couper, coller…) seront découplés et, nous pourrons facilement implanter un service d'annulation (unExecute() dans l'interface Commande) qui se chargera de défaire une action. Encore, le patron Commande peut être combiné à d'autre patron comme le Composite (pour exécuter un nombre quelconque de commande) ou le Mémento (Pour mieux gérer le défaire et refaire) afin de rendre notre application plus cohésif et moins couplé. De plus, ce patron nous permet d'ajouter très facilement de nouvelles actions (boutons, opérations) à notre éditeur de texte sans baisser la cohésion ni augmenter le couplage.

- **Le patron Observateur :** Le patron OBSERVATEUR est un patron de type comportemental qui permet de créer une interdépendance entre un et plusieurs objets de façon à ce que lorsque l'un (l'observable) est modifié, les autres (ceux qui l'observe) qui dépendent de lui soient notifiées et se mettent à jour. Il permet entre autre de :

  -Découpler les vues du modèle
  -Découpler les objets de sorte que les modifications de l'un d'entre eux puisse affecter un nombre quelconque des autres sans qu'il soit nécessaire pour l'objet modifier de connaître les détails des autres.

  Nous l'avons utilisé dans notre cas afin de séparer les outils graphiques des données. C'est-à-dire

qu'on veut pouvoir modifier le Buffer, y faire toutes les modifications demander par actions de l'utilisateur, et de l'afficher ensuite aux yeux de l'utilisateur dans le texte area de notre éditeur de texte qui sera notifié de tout changement.

- **Le patron Memento :** Permet de restaurer un objet à un état antérieur en enregistrant son état interne sans violer l'encapsulation. Il permet de :

  - Mémoriser l'état interne d'un autre objet

  -Ne pas violer l'encapsulation

  -Acquière et délivre à l'extérieur une information sur l'état interne d'un objet afin que celui-ci puisse être rétabli ultérieurement dans cet objet .

Les participants de ce patron sont:

– Originator
- Possède un état à sauver/restaurer
- Est capable de créer des objets Memento contenant une photo de son état interne courant
- Seul à avoir le droit d'accéder à l'état interne du Memento pour revenir à un état antérieur

– Memento
- Mémorise l'état interne d'un originateur

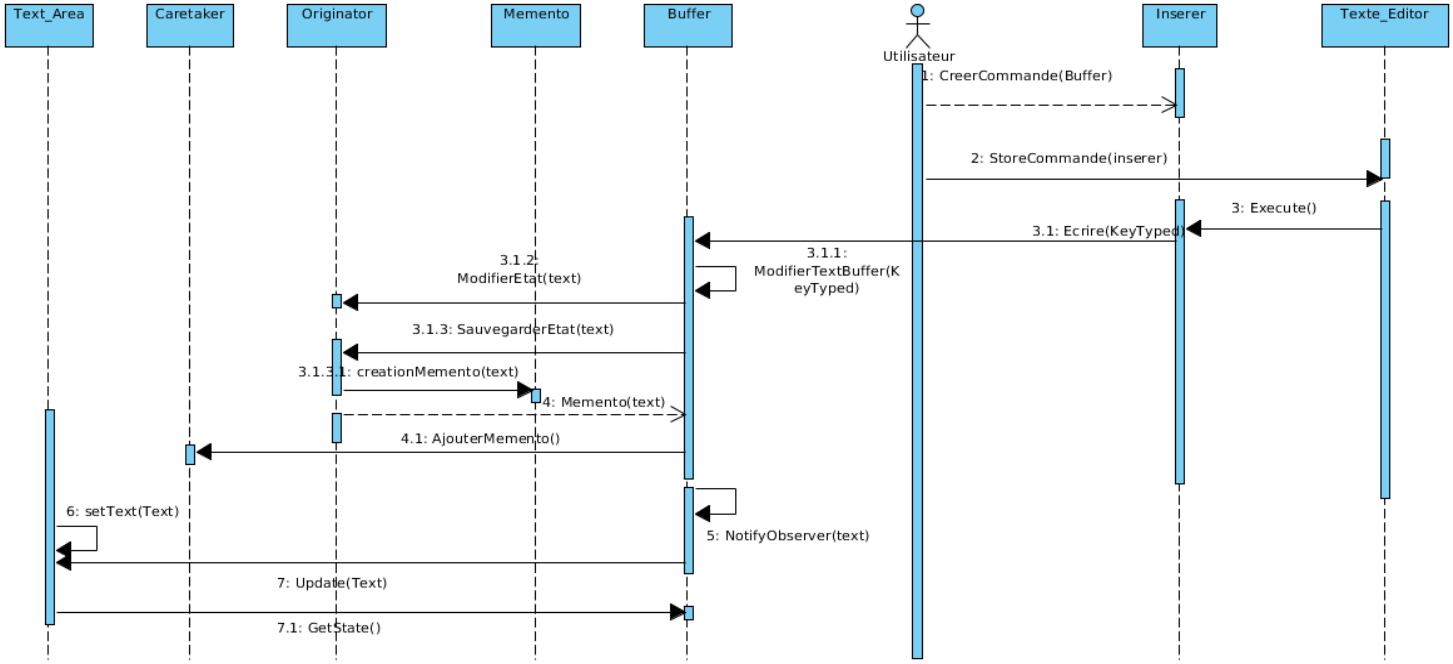
– Caretaker
- Capable de stocker des objets Memento et de les récupérer
- N'agit jamais sur Memento, ni ne l'examine. Il ne fait que passer le Memento aux autres objets

Dans notre cas, nous utiliserons ce patron afin de sauvegarder les états de notre buffer(le texte du buffer) afin de pouvoir y accéder ultérieurement sans brisé l'encapsulation. Il nous permettra entre autre d'implémenter les services Défaire et Refaire et les macros Enregistrer et Jouer car on le fais en affichant les états précédents de notre buffer.

Diagrammes de séquence

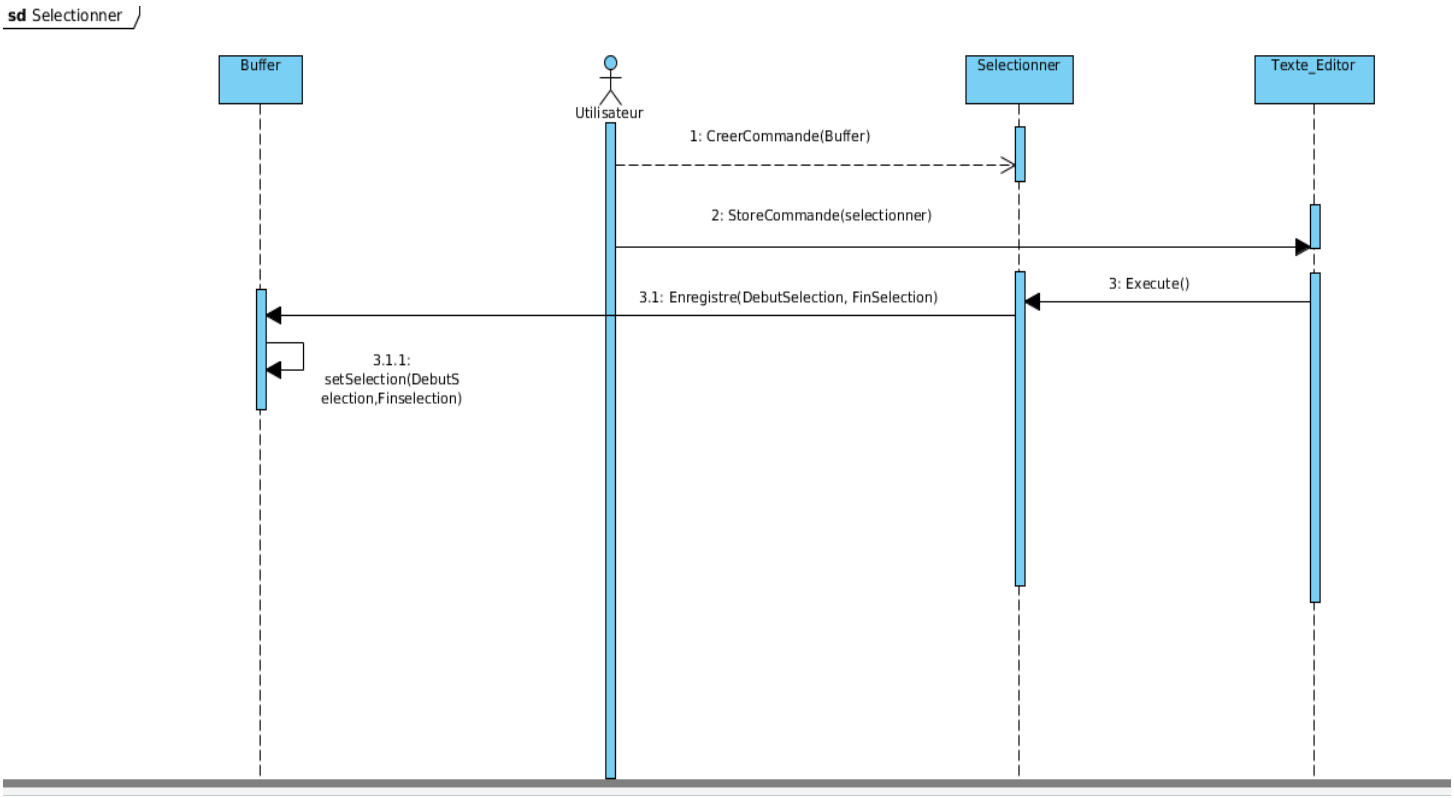
Insérer Texte

L'utilisateur entre du texte au clavier. Le texte_Editor crée la commande de saise et grâce aux KeyListeners, la commande envoie au Buffer les caractères entrés au clavier et modifie le texte du Buffer. Le buffer demande à l'originateur de sauvegarder son état dans le careTaker et notifie alors l'observateur Texte Area de son changement qui se met à jour.



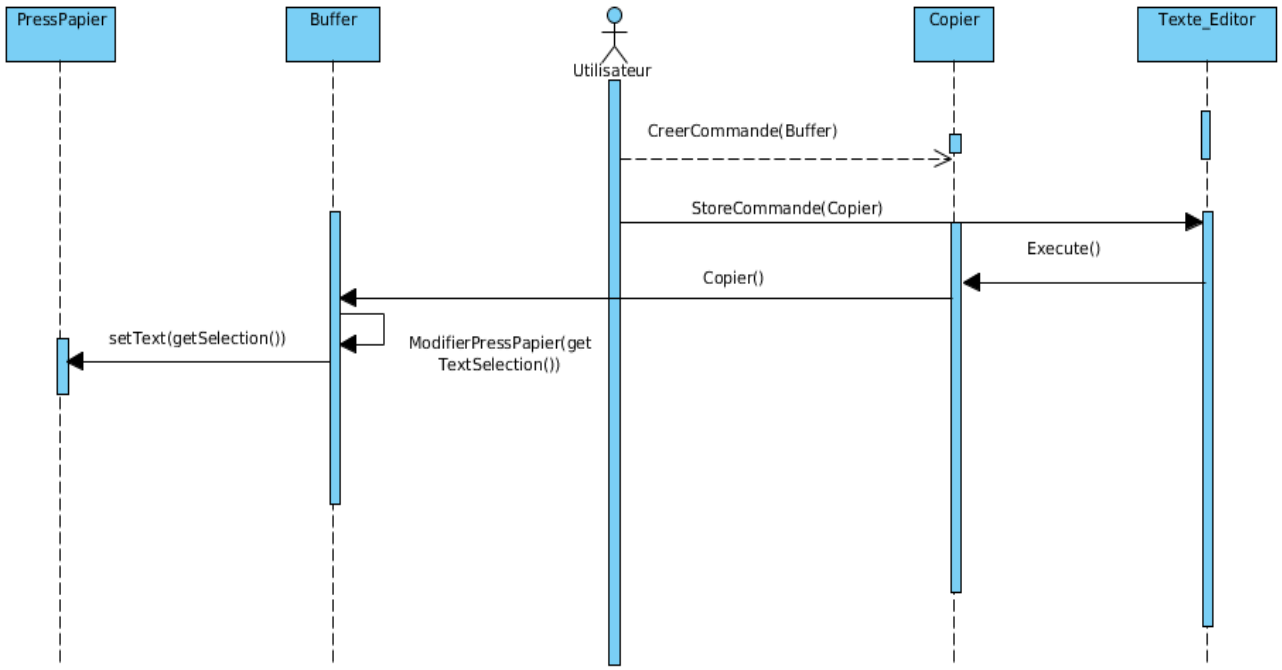
Sélectionner Texte

L'utilisateur sélectionne du texte. Le texte_Editor crée la commande de Sélection et grâce aux écouteurs du curseur, la commande envoie au Buffer le début et la fin de la sélection de l'utilisateur.



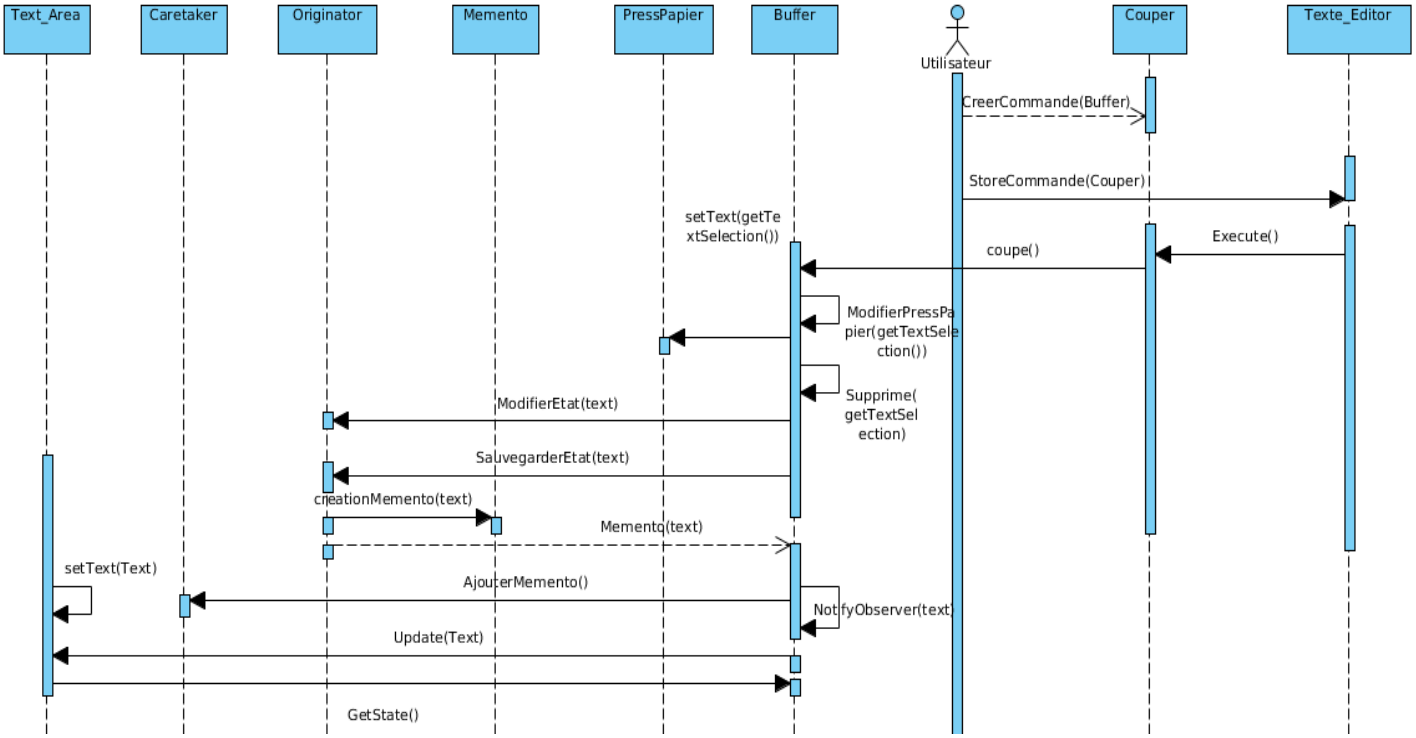
Copier Texte

L'utilisateur click sur le bouton Copier. Le texte_Editor crée la commande de Copie. Il exécute la commande qui quant à elle, copie la sélection du texte du buffer et la stock dans le pressePapier. Si rien n'est sélectionné, rien n'est copié.



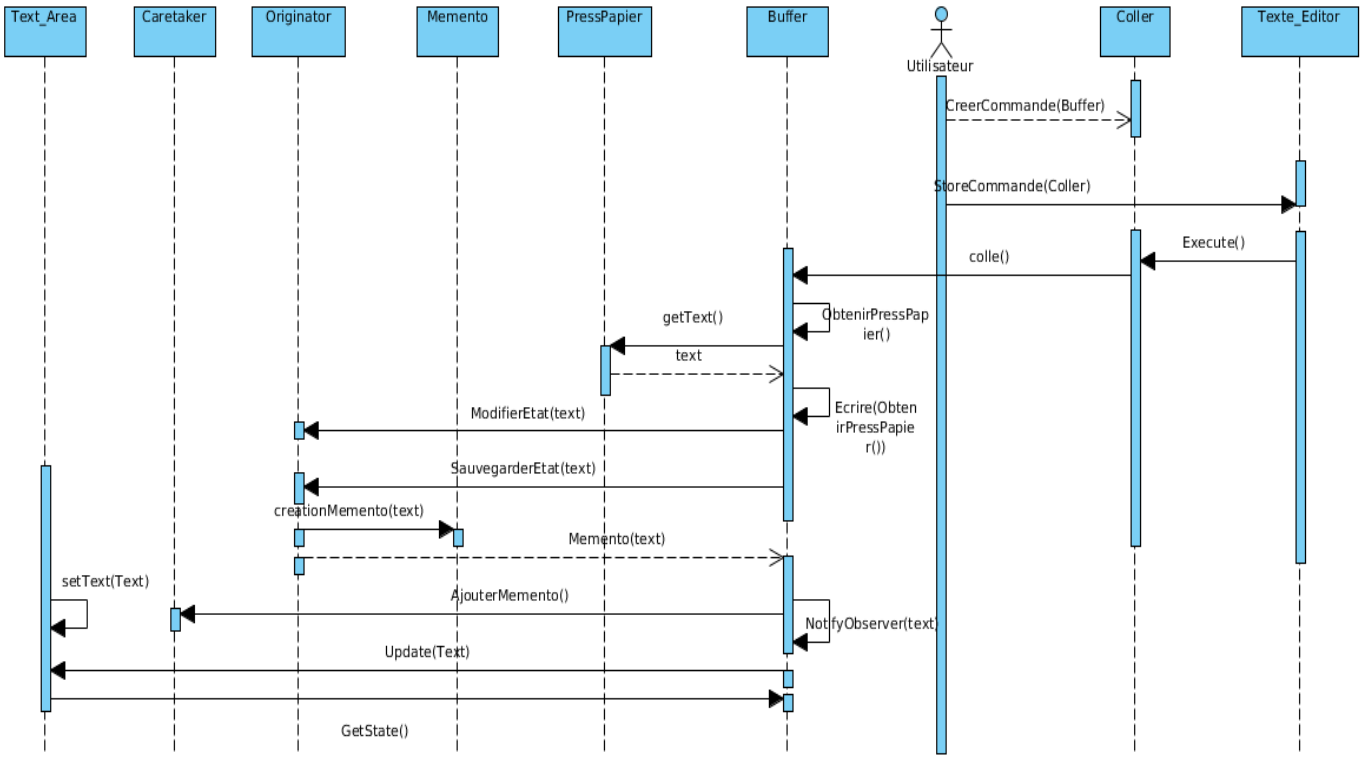
Couper Texte

L'utilisateur click sur le bouton Couper. Le texte_Editor crée la commande Couper. Il exécute la commande qui quant à elle, copie et supprime la sélection du texte du Buffer et la stock dans une variable. Si rien n'est sélectionner, rien n'est couper. Le buffer ensuite demande a l'originator de sauvegarder son état dans le caretaker et notifie sont observeur qui est text_Area afin qu'il puisse afficher la modification du texte du Buffer en modifiant son texte.



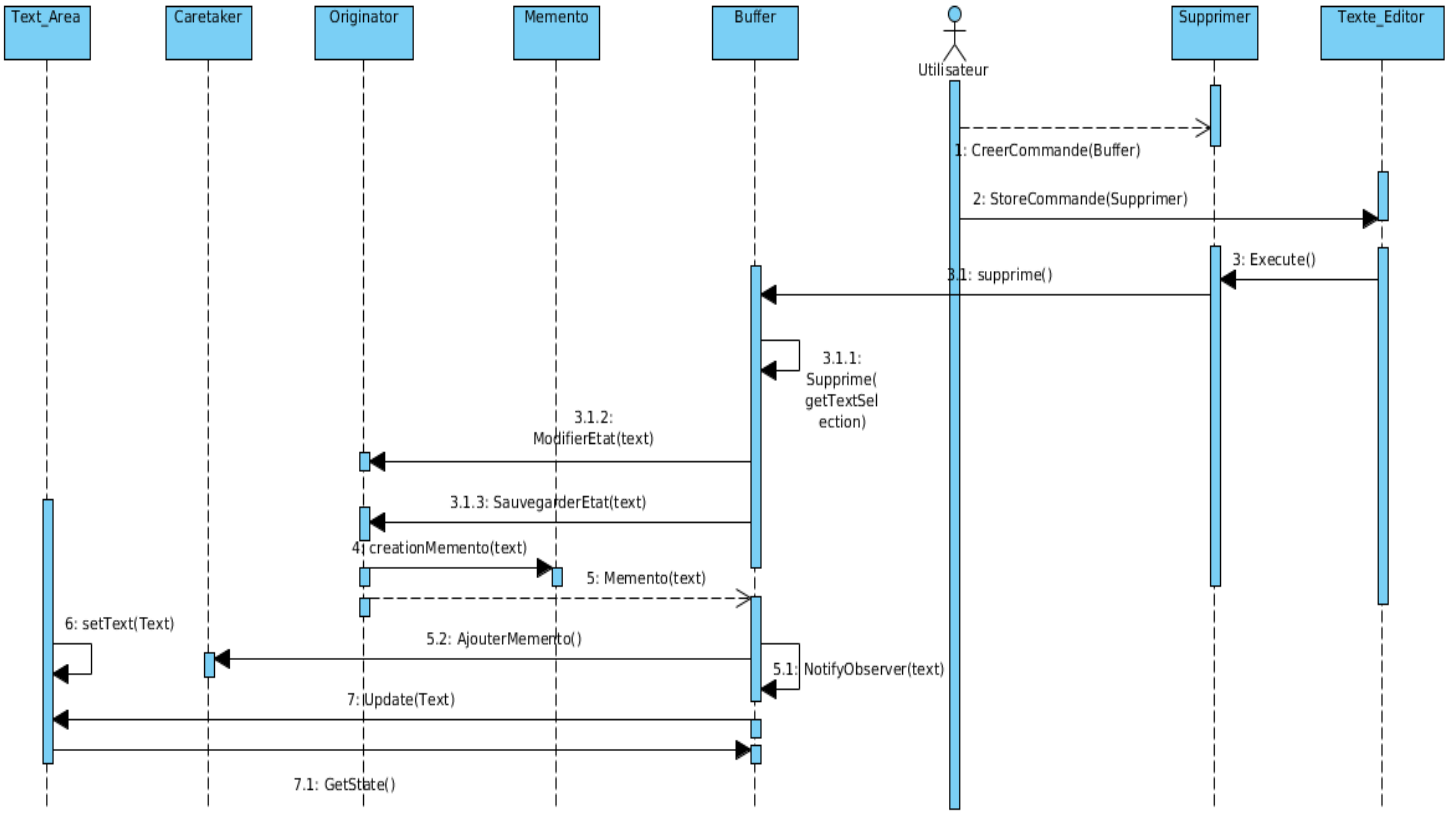
Coller Texte

L'utilisateur click sur le bouton Coller. Le texte_Editor crée la commande Coller. Il exécute la commande qui quant à elle, colle le texte précédemment copié ou coupé du Buffer. Si du texte est sélectionner, il sera supprimer et remplacé. Le buffer ensuite notifie sont observeur qui est le text_Area afin qu'il puisse afficher la modification du texte du Buffer en modifiant son texte.



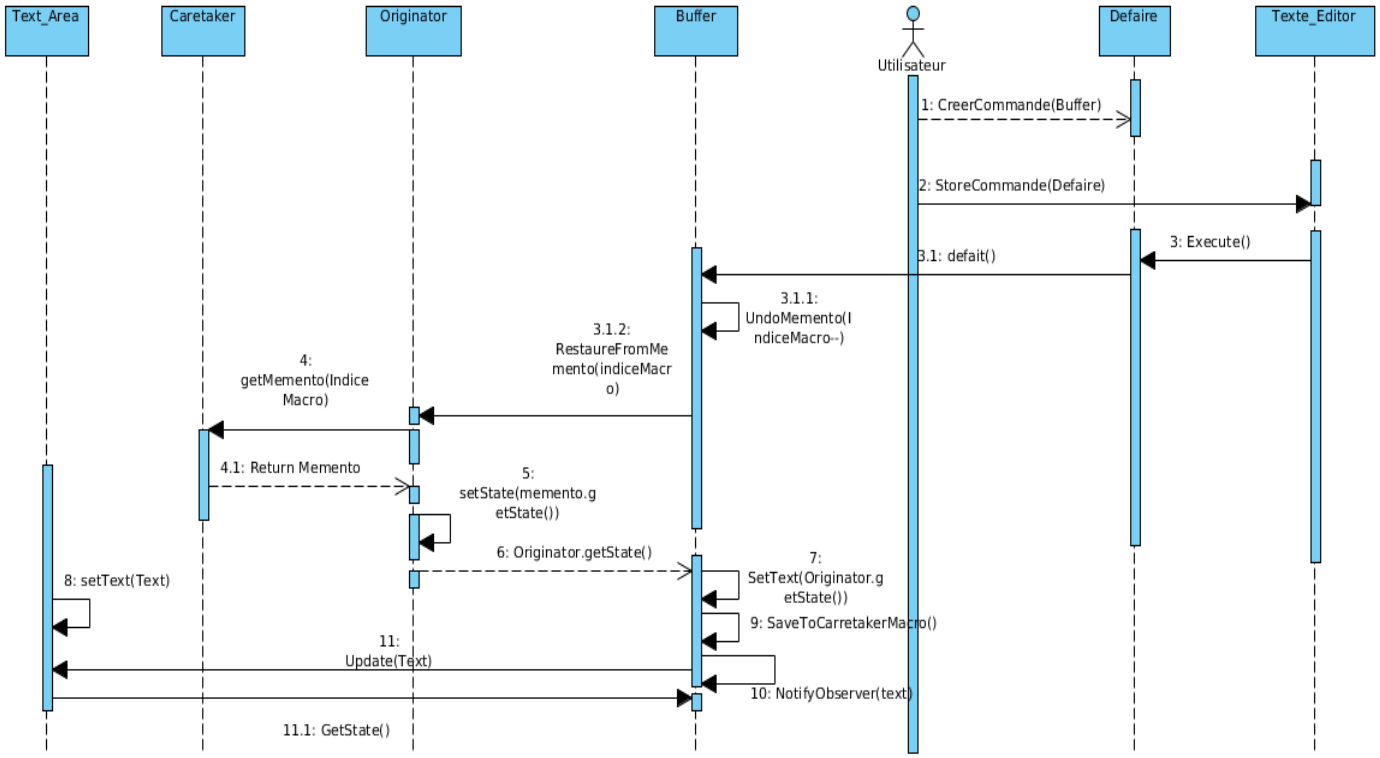
Supprimer Texte

L'utilisateur supprime du texte (touche backspace du Clavier). Le texte_Editor crée la commande de suppression. Il exécute la commande qui quant à elle, supprime le caractère à la position du curseur dans le Buffer. Si du texte est sélectionné, il sera supprimé. Le buffer ensuite notifie sont observeur qui est le text_Area afin qu'il puisse afficher la modification du texte du Buffer en modifiant son texte.



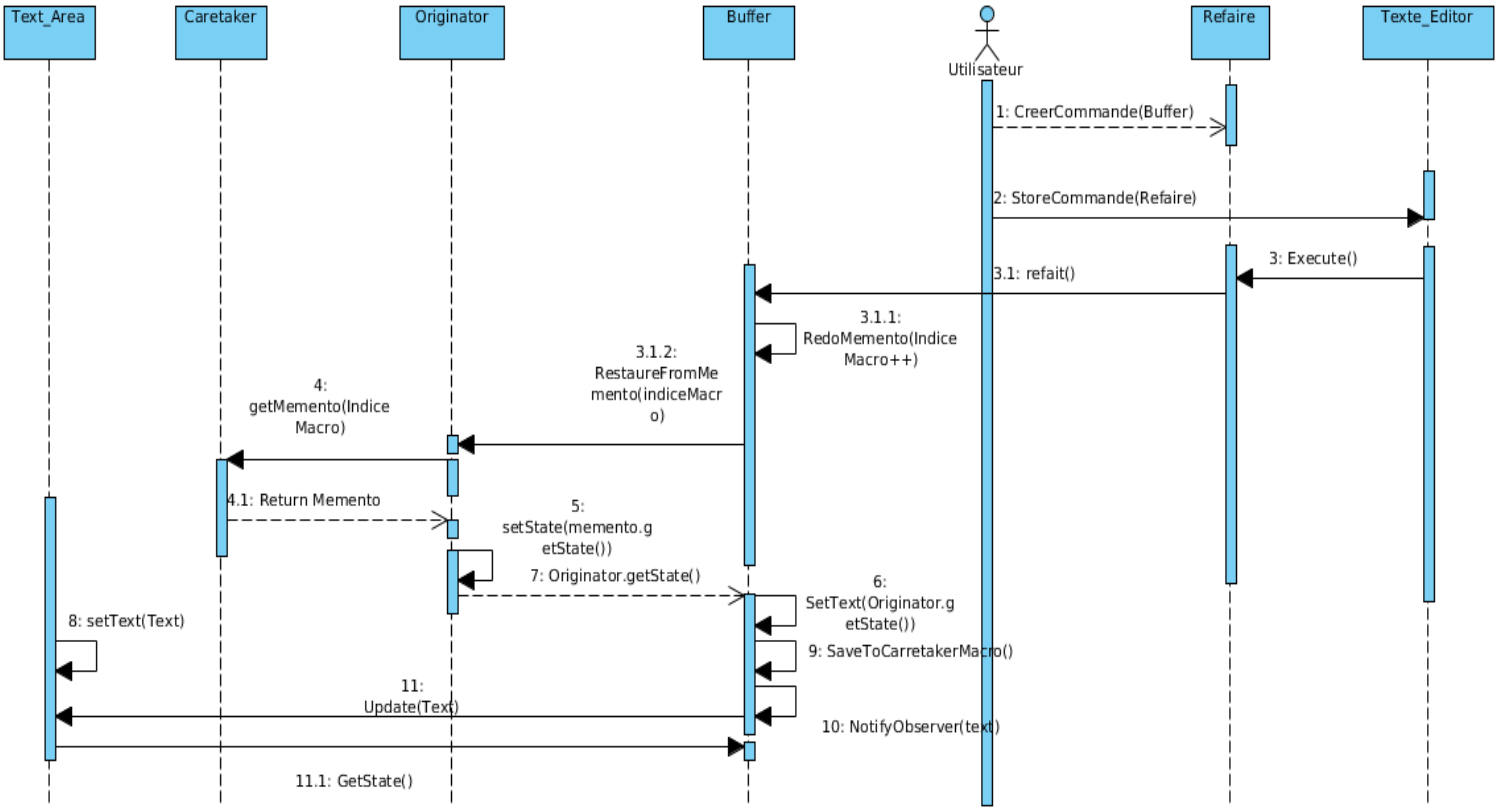
Defaire

L'utilisateur click sur le bouton Defaire. Le texte_Editor crée la commande Defaire. Il exécute la commande qui quant à elle, affiche l'état du texte du Buffer précédemment sauvegarder dans le Caretaker. IndiceMacro est initialisé à Carretaker.getSize().



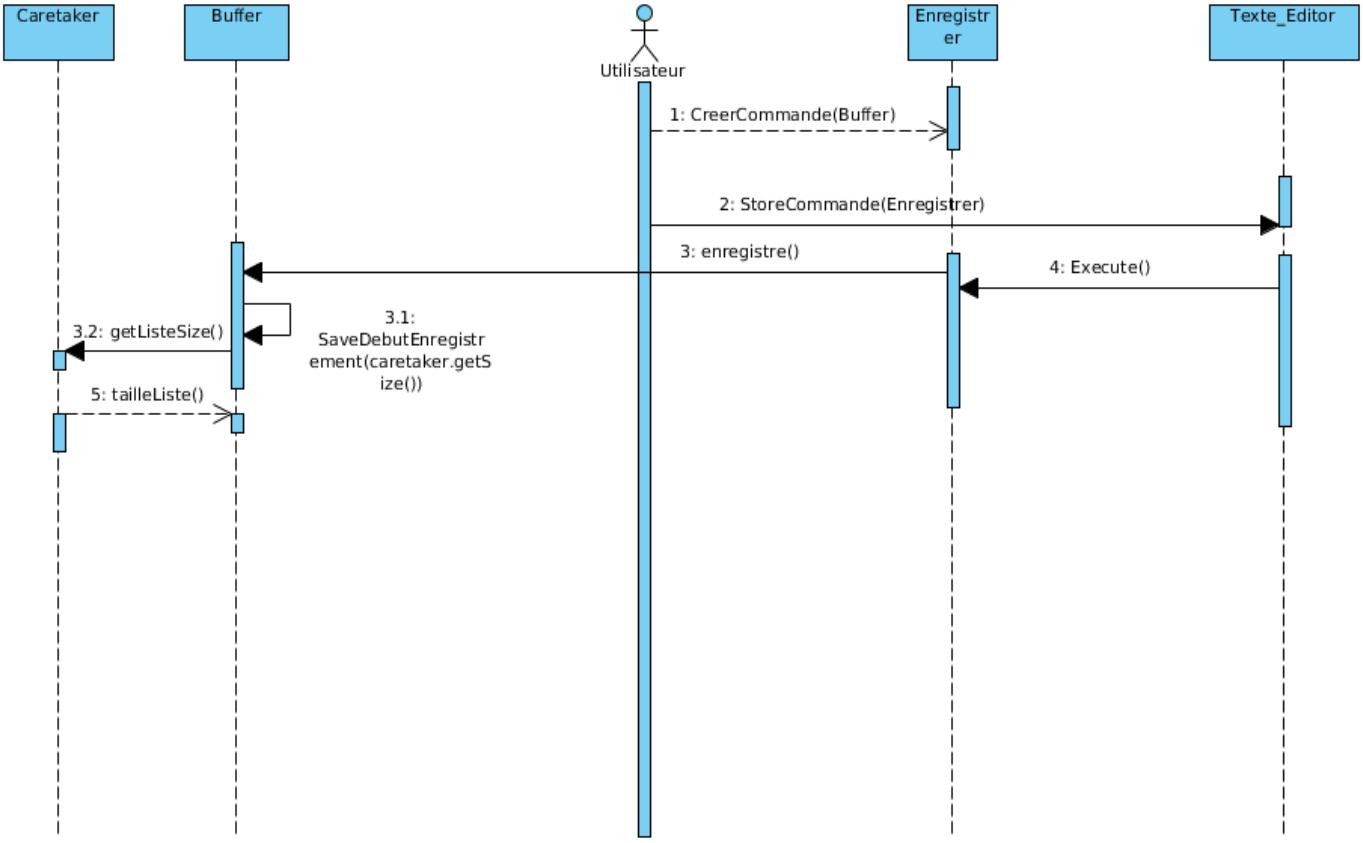
Refaire

L'utilisateur click sur le bouton Refaire. Le texte_Editor crée la commande Refaire. Il exécute la commande qui quant à elle, affiche l'état du texte du Buffer sauvegarder dans le Caretaker qui à été précédemment Défait. IndiceMacro est initialisé à Carretaker.getSize().



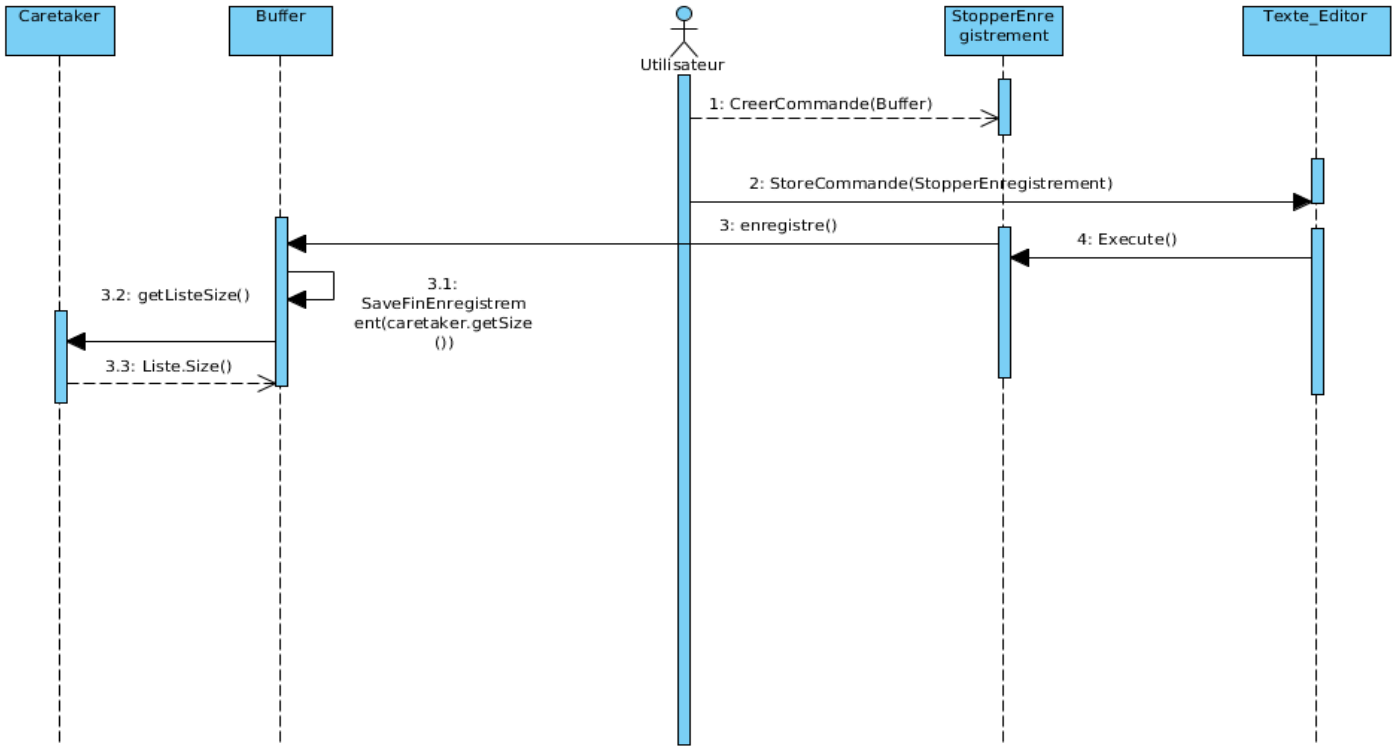
EnregistrerMacro

L'utilisateur click sur le bouton Enregistrer. Le texte_Editor crée la commande EnregistrerMacro. Il exécute la commande qui quant à elle,sauvegarde l'index du Caretaker où sont sauvegarder l'état du texte du Buffer afin de pouvoir rejouer à partir de cette position.



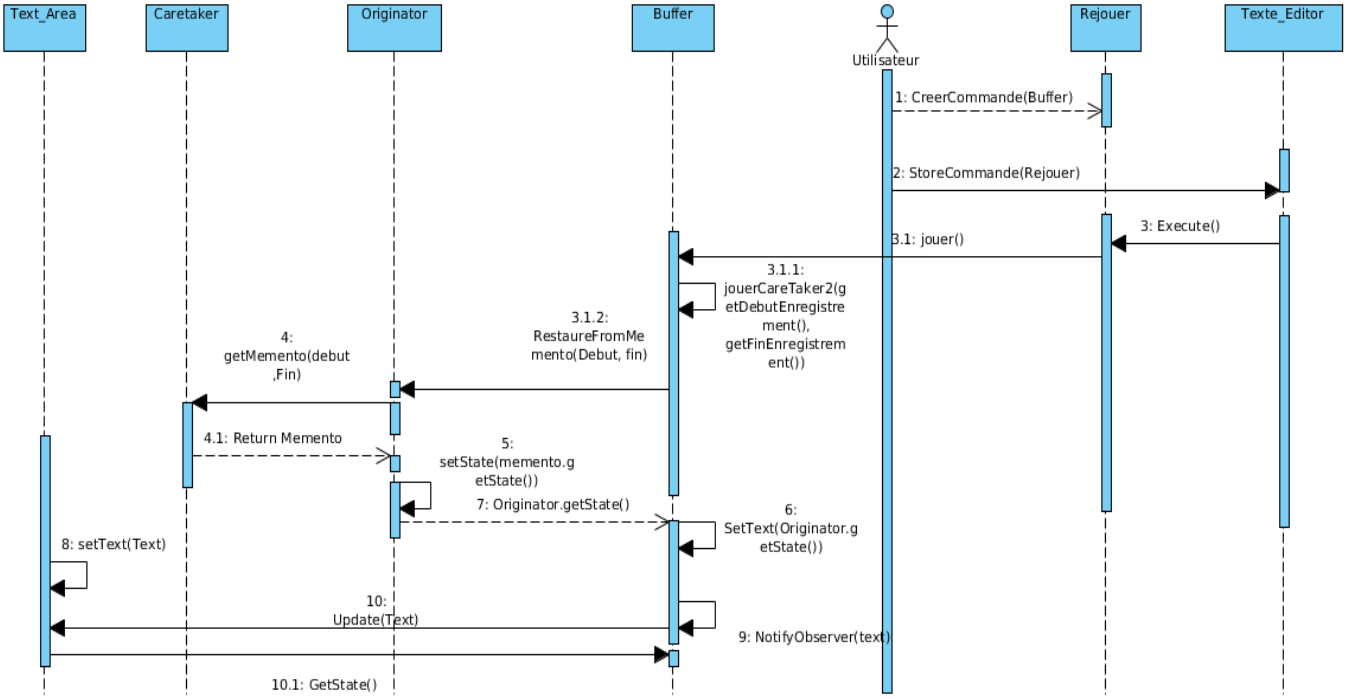
StopperEnregistrementMacro

L'utilisateur click sur le bouton Enregistrer. Le texte_Editor crée la commande EnregistrerMacro. Il exécute la commande qui quant à elle,sauvegarde l'index du Caretaker où sont sauvegarder l'état du texte du Buffer afin de pouvoir rejouer jusqu'à cette position.



Rejouer

L'utilisateur click sur le bouton Rejouer. Le texte_Editor crée la commande Rejouer. Il exécute la commande qui quant à elle, affiche tous les états sauvegarder du texte du Buffer dans le caretaker de l'index du début d'enregistrement jusqu'à la fin.



Appendices

EXPRESSIONS, ACRONYMES	DÉFINITONS
Write()	**Méthode qui permet au buffer de modifier son texte.**
Action()	**Signifie l'exécution de la commande.**
GetState()	**Permet à l'observeur de connaitre l'état de l'objet qu'il observe (patron observateur).**
notifyObserver()	**Permet à l'observable (objet observé, c'est le Buffer dans notre cas) de notifier ses observeurs (Text_Area dans notre cas) lorsqu'il change d'état.**
l'observable	**Objet dans le cadre du patron de conception Observateur qui est observé par des observeurs**
L'observeur	**Objet dans le cadre du patron de conception Observateur qui observe l'observable**