CONDUCTEUR TRAVAUX DIRIGÉS
MINI-ÉDITEUR

Master 1 Info

Noël Plouzeau

Organisation

Les objectifs de ce travail dirigé sont les suivants :

- présenter une technique de conception à objets simple ;
- montrer les différents moyens de coordination des objets ;
- montrer la construction parallèle des diagrammes statiques et dynamiques.

Sujet

Il s'agit de construire un mini-éditeur de texte offrant les concepts et les fonctions suivants :

- le texte est contenu dans un *buffer* (zone de travail) ;
- il existe une notion de sélection de texte, avec des commandes utilisateur permettant de déplacer le début et la fin de la sélection ;
- copie de la sélection dans le presse-papier ;
- copie de la sélection dans le presse-papier puis effacement de la sélection ;
- remplacement (« collage ») de la sélection par le contenu de presse-papier ;
- l'interface homme-machine est d'un type quelconque (textuelle ou graphique)

Version initiale

Le premier travail est de créer un diagramme statique :

1. à partir des concepts de l'énoncé ;
2. sachant que le type d'application suggère l'application du patron commande (*Command*) ;
3. en employant la notation UML standard pour la référence à un patron de conception.

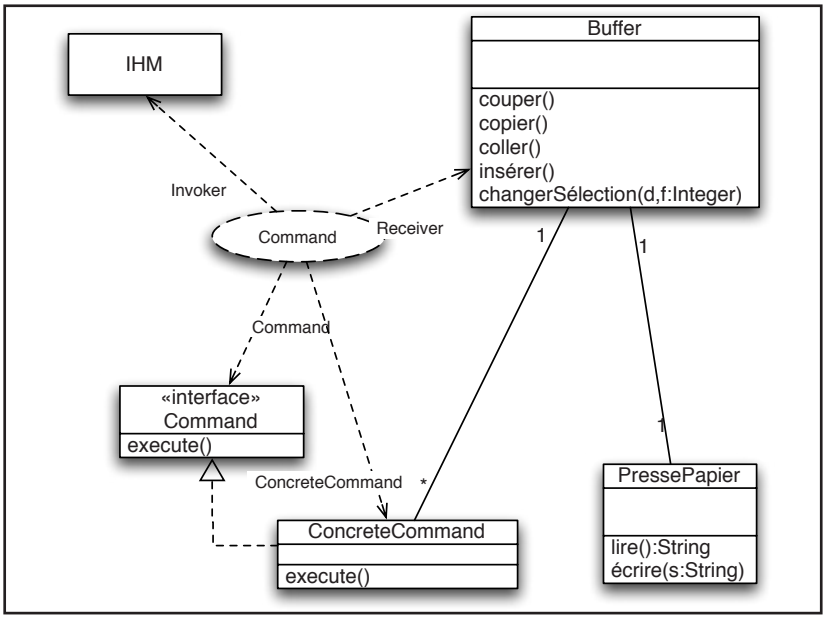
Il faut ensuite construire un premier diagramme dynamique, en prenant comme exemple la commande *Couper*.

Dans un deuxième temps, demander aux apprenants d'adapter le principe pour obtenir une mise en œuvre de la commande *Insérer*. Cette commande remplace la sélection par un fragment de texte entré depuis l'interface homme-machine.

Architecture statique initiale.

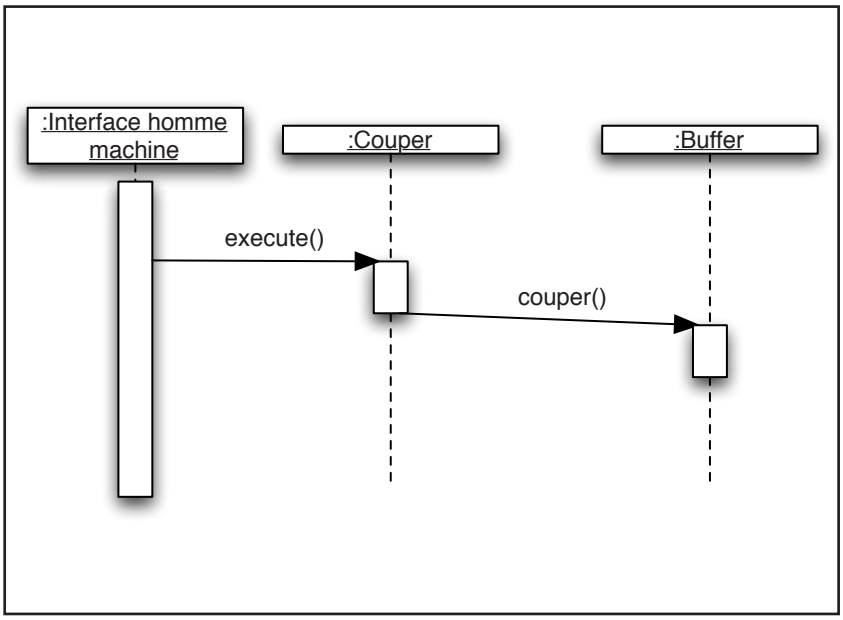
Noter l'emploi de l'ellipse tiretée issue de la notation UML. Cette ellipse indique que l'on instancie un patron de conception.

Ce diagramme UML devra être complété au fur et à mesure.



Ce diagramme met en évidence le rôle central des objets Command.

Toute interaction dans le sens IHM vers Buffer est interceptée par une commande.



Récupération de données depuis l'interface homme-machine

Les apprenants doivent réaliser que l'opération *insérer(t)* de la classe *Buffer* demande une donnée qui est disponible dans l'interface homme-machine. Faire analyser les différents cas, leur impact sur l'architecture existante et leur faisabilité :

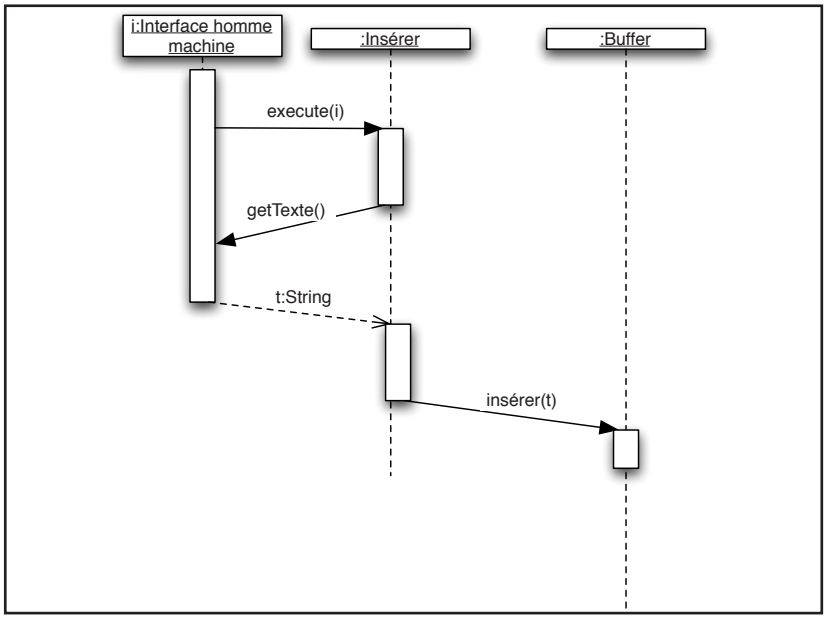
1. la donnée est transmise en paramètre du message execute() ;
2. la donnée est transmise de l'IHM vers la commande, à l'initiative de l'IHM ;
3. la commande réclame la donnée à l'IHM.

Seule la troisième possibilité est viable.

Enregistrement des commandes

Il s'agit de rajouter la possibilité d'enregistrer les actions faites par l'utilisateur. Certains paramètres employés lors de l'exécution initiale des commandes devront être mémorisés, tandis que d'autres ne le seront pas :

Diagramme de séquence montrant comment une commande requérant spécifiquement un paramètre peut l'obtenir de l'Invoker



– le texte inséré lors de l'exécution de la commande d'insertion de texte depuis l'interface homme-machine *sera mémorisé* ;

– la position de la sélection (son début, sa fin) *ne sera pas mémorisée*.

Il s'agit de discuter les différentes questions suivantes :

1. quel objet capte l'exécution d'une commande pour provoquer son enregistrement ;
2. comment l'état d'une commande est-il sauvegardé ?

La réponse à la première question est guidée par la découpe en trois parties de l'architecture (IHM, contrôle, commande) représentée par l'emploi du patron de conception *Command*. On peut considérer que l'on va superposer deux instance du patron de conception *Command* :

1. la première instance gère l'exécution effective des commandes, selon l'architecture vue à la section précédente ;
2. la seconde instance gère l'enregistrement de l'exécution.

La réponse à la seconde question peut se faire de plusieurs façons :

– soit par clonage de chaque commande avant son exécution (chaque objet *Command* est donc exécuté une fois pendant l'enregistrement) ;

– soit par stockage de l'état de l'objet *Command* à l'aide du patron de conception *Memento* (il n'existe qu'un seul objet instance d'une classe *ConcreteCommand* donnée).

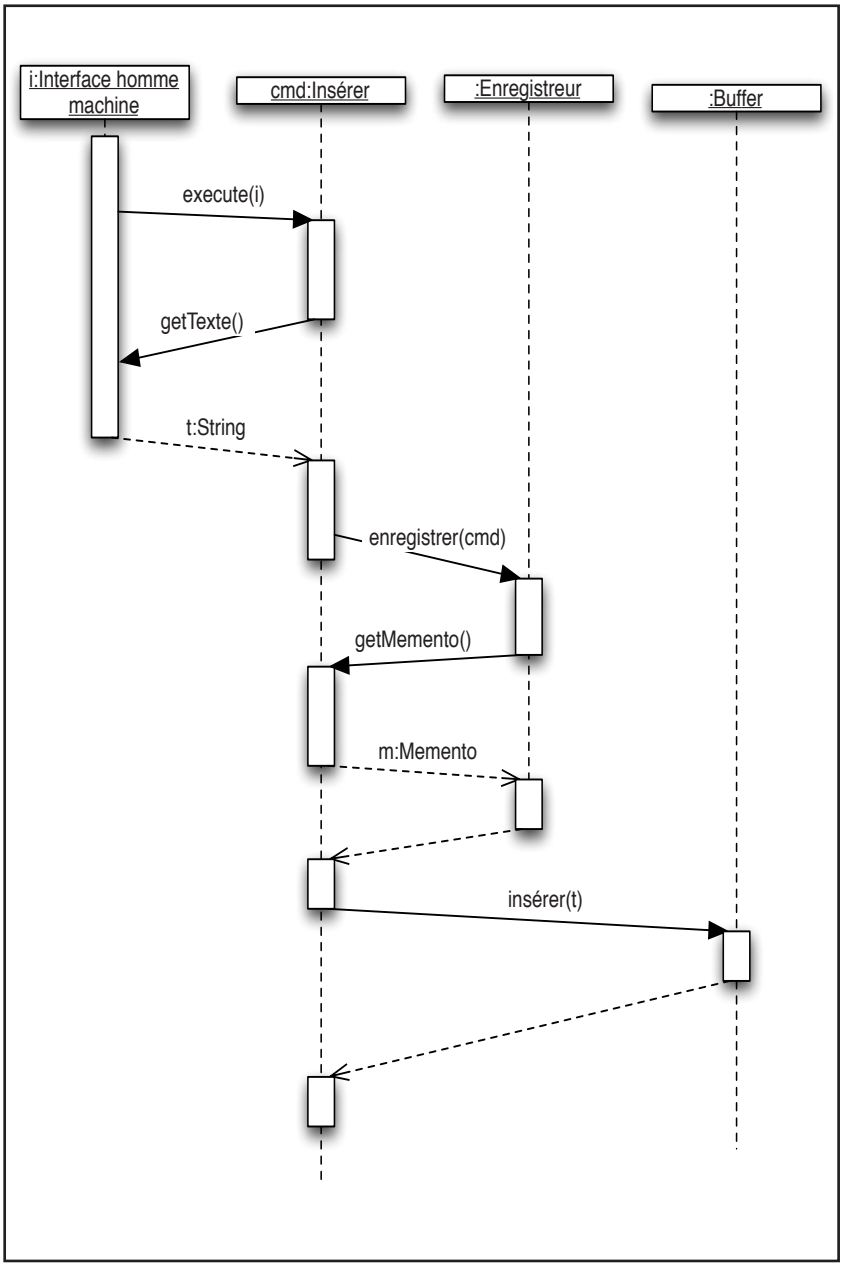
Variante avec emploi de Decorator

Le patron *Decorator* permet d'étendre les possibilités des commandes sans modifier la commande elle-même et sans mettre deux *Receiver*. Pour cela, on « chaîne » deux objets commande. Par exemple, la classe *Insérer* est complétée par une classe *InsérerEnregistrable* qui est responsable de la mise en œuvre du rôle *Originator*.

Fonction défaire-refaire

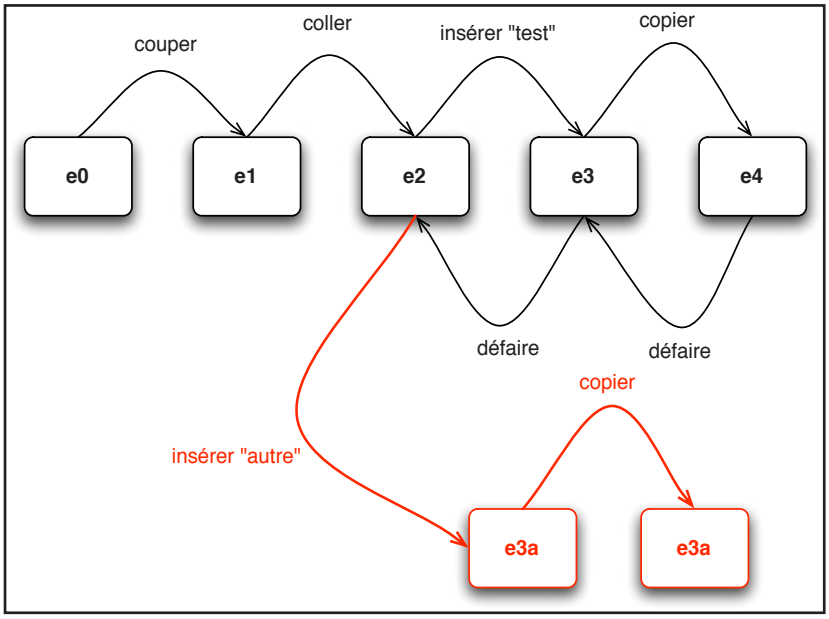
La fonction défaire-refaire à mettre en œuvre permet d'annuler un nombre quelconque de commandes puis d'en refaire un nombre quelconque. Chaque

Enregistrement d'une commande insérer *lors de son exécution.*



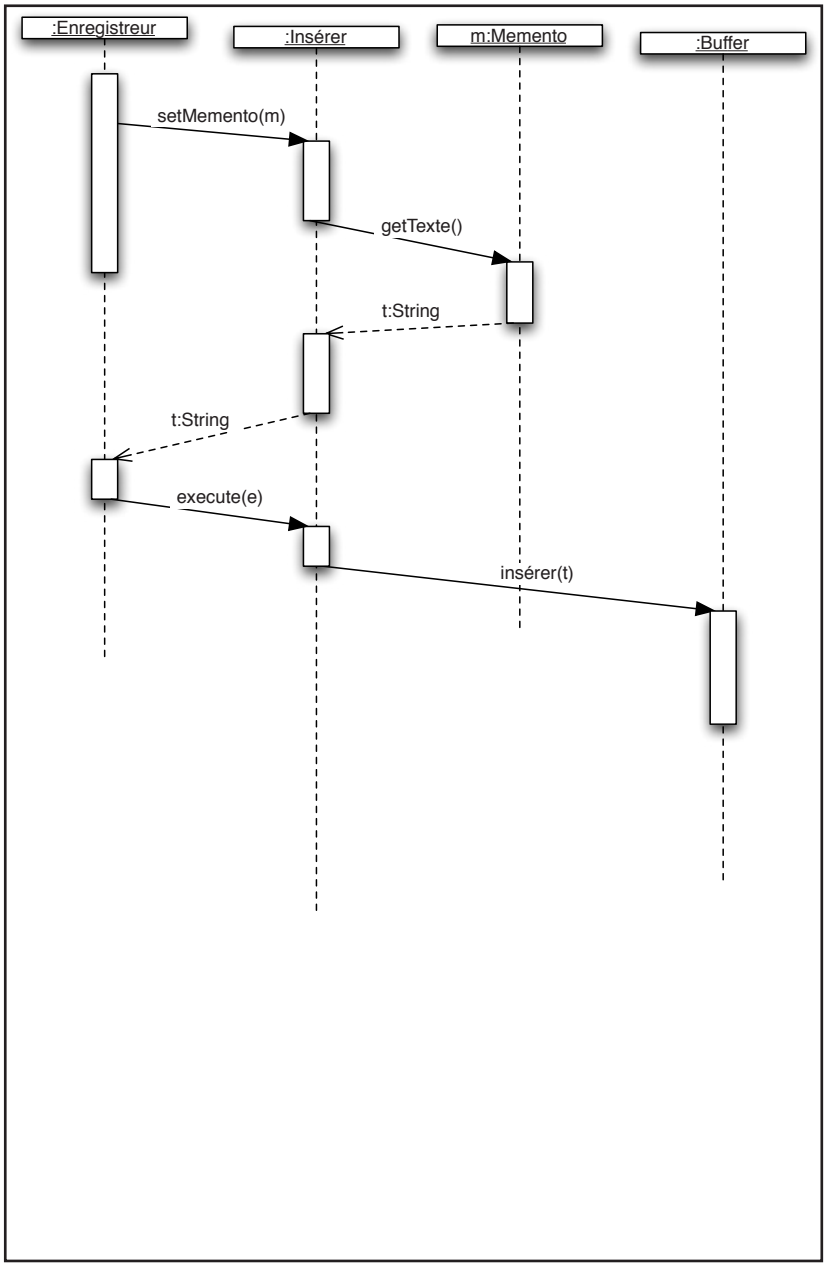
Séquence d'états obtenue lors de l'exécution d'une séquence d'action par l'utilisateur.

En rouge, le chemin suivi lors de deux action défaire suivi d'une autre action.



Réexécution de la commande insérer.

L'enregistreur rétablit l'état de la commande avant de provoquer son exécution.



exécution d'une commande fait passer le logiciel d'un état à un autre. L'annulation d'une commande ramène dans l'état précédent.

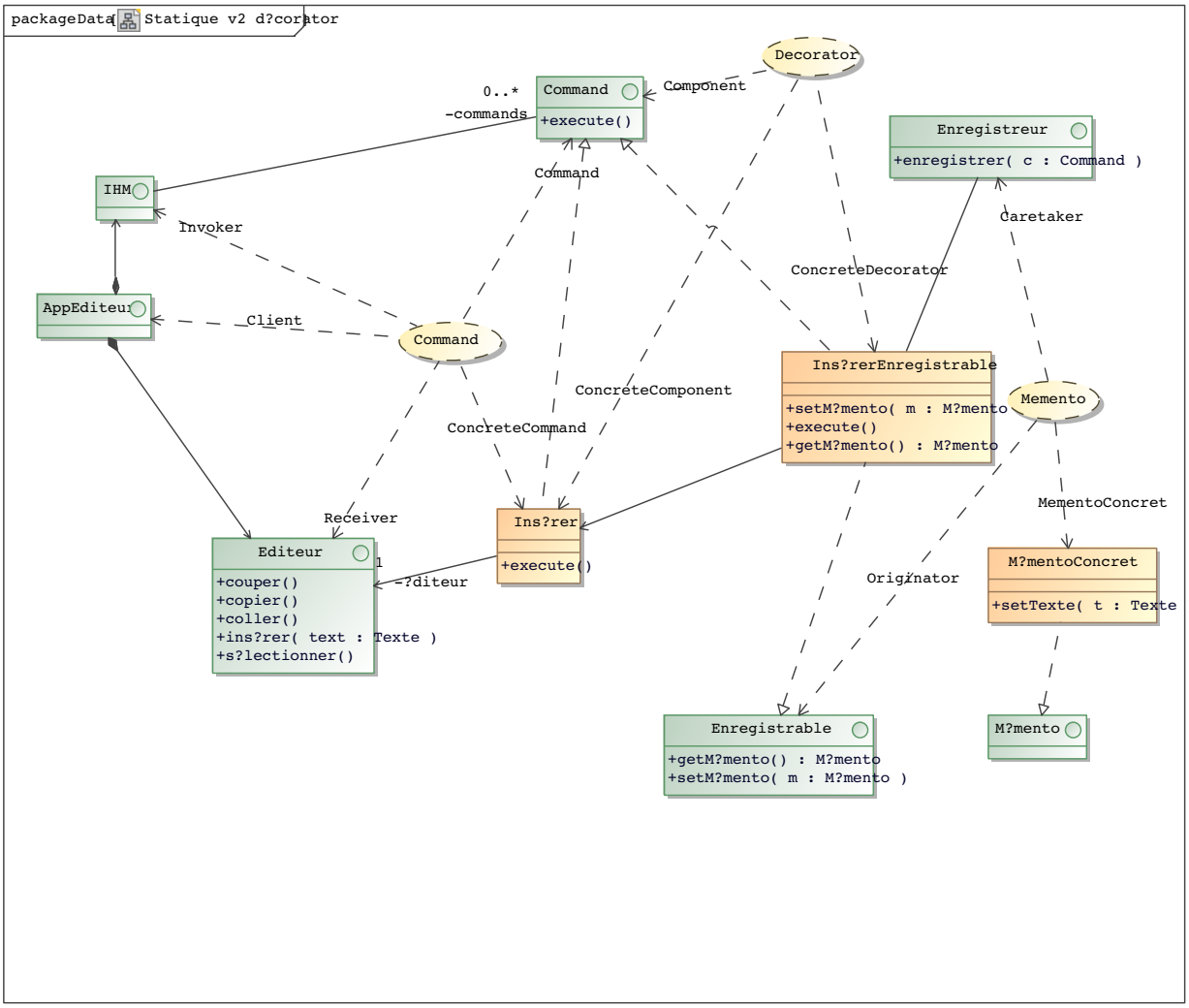
La représentation de l'évolution de l'état de l'application peut être représenté par une séquence d'état et d'action sur des transitions entre ces états (voir figure *infra*).

Stratégies de stockage

Il existe plusieurs possibilités pour être capables de restaurer n'importe quel état de la chaîne des états de l'application :

1. stockage de tous les états de la chaîne d'état ;
2. stockage de l'état initial et enregistrement systématique de toutes les actions ;
3. stockage de l'état précédent l'état courant ;
4. stockage de l'état à intervalle régulier (par exemple toutes les k actions, ou juste avant une action coûteuse en temps de calcul) ;
5. conjonction des deux dernières solutions.

Emploi du patron de conception Decorator.



La solution n° 1 requiert beaucoup d'espace mémoire. La solution n° 2 est coûteuse en temps de calcul. La solution n° 3 permet de faire efficacement une action *défaire* à partir de l'état courant, mais une deuxième action *défaire* ramène à la solution n° 2. La solution n° 4 permet de ne rejouer les actions que depuis la dernière sauvegarde. La solution n° 5 est un compromis entre les autres solutions.