- « généralités » : c'est un peu banal comme titre de section (surtout la première). Un truc un chouia plus créatif serait bienvenu...
- « on va un peu plus loin » -> Nous allons aller plus loin en…
- GOF : Tout le monde ne sait pas qu'il s'agit du « Design Patterns » …
- Pas d'accord avec l'utilisation du mot « état » par rapport à la commande : il s'agit de paramètres ou de données afférentes à cette commande. Il s'agit de valeurs transitoires (qui peuvent être certes mémorisées) mais qui ne sont pas symptomatique de l'état de la commande. Les états de cette commande seraient plutôt des choses du genre « awaiting », « done », « undone ».
- Première phrase de le section « découplage spatial… » : attention, espace avant la parenthèse et tu peux traduire « scope » par « portée ».
- Dans le cas du « découplage spatial » : sans aller jusqu'à la séparation physique, la séparation entre modules (même co-localisés) mais que l'on veut garder faiblement couplés est une motivation suffisante. Le module d'execution a alors juste à connaitre l'interface de Command sans se soucier de la création de nouvelles commandes. On est en plein dans « l'open-close » principle de Robert Martin (et c'est cool).
- Remarque Geek : En théorie en UML, on devrait utiliser une « Collaboration Paramétrée » pour représenter les rôles des patterns (rien à voir avec les diagrammes de collaboration). Le hic, c'est qu'à peu près personne ne connait les Collaborations Paramétrées. Donc, c'est cool avec des stéréotypes ;)
- « découplage spatial : command remote » : J'ai un eu de mal avec la transition du point 3 au point 4 ; tu n'expliques pas comment le RemoteInvocator reçoit la commande. De plus le point 3 (execution de la commande) est en fait corrélé au dernier de la cinématique. Je pense qu'il faudrait reformuler le point 3 (genre : l'implémentation de l'execution est adossée à des definitions abstraites d'environnement…), car on croit que c'est à cette étape que l'on exécute la commande ! Et expliquer comment la commande est passée au RemoteInvokator et par qui.
- « découplage temporel » : En fait, on peut aussi se servir du processeur de commande afin d'introduire un niveau d'asynchronisme (léger différé) entre le déclenchement de la commande et son execution. C'est dans l'air du temps (par exemple : CQRS + Vertex)
- « instant ultérieur » -> « moment ultérieur ».
- J'attirerai l'attention sur le fait que pour l'undo, la Command doit conserver tout ce qui nécessaire pour jouer la commande à l'envers, ce qui est une différence notable avec la simple exécution.
- Sur le diagramme UML, je fais généralement apparaitre 2 relations entre Conversation et Commande : une relation « done » et une relation « undone ». Mais ça, c'est surtout utile pour permettre le « redo » !
- Section « undo » : cela donne un aspect bizarre à la structuration de l'article ; j'avais déjà l'impression d'être en plein dans la question de l'undo dans la sous-section « découplage temporel » de la section précédente… Mais je n'ai pas d'alternative simple à proposer. A moins de renommer « L'undo » en « Implémenter le processeur de commande » : on voit alors que l'on passe du design à l'implémentation...

- « on donne des exemples plus loin » : pas terrible comme formulation…
- Tu es un peu succinct dans l'énoncé des 3 variation. Je pense que tu pourrais les citer dès cette introduction, le lecteur se repèrerait mieux.
- La référence à Seam ou Web Flow ne me semble pas utile, elle encombre plutôt le propos qui est l'undo avec ses 3 variations. Restons focalisés à-dessus.
- « le type parameter C… » : Phrase un chouia cryptique tant que l'on n'a pas regardé le code. J'aurais préféré un truc du genre : plutôt que de dépendre directement d'une classe Command, nous utilisons un type paramétré, pour permettre l'utilisation des 3 déclinaisons de Command que nous allons vous présenter. Formulation pas forcément géniale, mais ce serait l'idée.
- On peut certes se passer de la présentation de la classe « TypeString », mais on serait plus à l'aise avec…
- Le test « undo twice » n'a pas beaucoup d'intérêt ici, même si en production c'est intéressant. On peut alléger un poil l'article.
- Le « double undo / 1 commande » est un peu dans ce cas. C'est un poil redondant avec Noop.
- Même remarque sur le double redo
- Je pense qu'on peut ne garder qu'un seul test sur undo-redo (entre celui avec « a » et celui avec « ab »). Je garderais le plus complet, perso.
- Le double undo/redo est marrant, mais redondant pour un article.
- « à un détail technique propre… » -> « à un détail technique **près** propre… »
- Je découvre juste au moment où tu présente les implémentations de tes 3 classes de test comment elles marchent. Je pense que tu aurais pu présenter la classe abstraite au début, avant les cas de test. D'autant que la manière de faire est élégante.
- « Couer de l'implémentation » : Tu utilises 2 tacks, donc ce serait définitivement bien de le montrer sur le diagramme UML.
- Paramètres S et C : Tu es un chouia cryptique, mais ça passe peut-être… Mon extrapolation, c'est que dans les autres cas, on instancie la classe avec le même type concret pour S et C… Mais pour les nouveaux venus au pattern Command, c'est pas sûr que ce soit compréhensible.
- Petites questions sur ReplayConversation : 1) Une grosse limite sur le fait que l'on puisse effectivement resetter l'environnement. As-tu des exemples de contexte où cela a du sens et/ou c'est plus simple ? Ca aiderait. 2) Je ne comprend pas pourquoi on doit nettoyer la « redoStack » à cheque exec. J'ai l'impression que ce n'est pas le bon endroit…
- Le coup du lambda anonyme, c'est pas pour les tapettes. Comme ce n'est pas forcément un concours de celui qui a la plus longue (apparemment, ce n'est pas moi), il y a-t-il un moyen d'implémenter cet appel simplement sans en passer par là ?
- Version memento : tu capture 2 mémento au moment de l'exec. Ce n'est pas très « just in time ». Perso je ferais plutôt du lazy, en capturant le before à ce moment-là et l'after (qui sera alors aussi un before à ce moment-là ;) lors de l'undo ! Ainsi, si on ne fait pas de redo, on ne paie pas le prix de ce second memento. Je sais, c'est un peu ch… car comme on ne garde pas les commandes, cela signifierai que le Memento « before » est aussi l'Originator de l'After...
- « plus lightweight » -> « plus légère » … Quand même ;)
- facteurs défavorables à l'undo : quand tu dis « il faut être sûr » de pouvoir undoer toutes les commandes, ce n'est pas exactement le cas. On peut aussi inhiber la chaine d'undo via un « isUndoable() » : quand on le rencontre lors de l'exec, on flush la stack undo. C'est un peu radical, mais c'est une option, par exemple quand on n'a que quelques commandes exceptionnellles qui ne peuvent être undoées.
- Il vaut mieux parler « d'approche DDD » que de « méthode DDD ».
- Attention, pour moi Event Sourcing est un peu une boite de pandore (intéressante) par rapport au processeur de commande. On peut le voir comme une sorte de forme spécifique de variation sur le Memento. Mais il y aurait beaucoup à dire…
- « ou que le système présente une hysteresis » … euh… là c'est übergeek !
- facteur défavorable à l'undo : la façon de formuler le second point ne convient pas. Il faudrait dire un truc du genre « lorsque les contraintes en mémoire et en performances sont importantes ».
- profondeur de l'undo-redo : une OOME ?? Là, je sèche !
- Tu termines un peu en queue de poisson sur les macro. Tu pourrais donner le code de execute(), par exemple, pour montrer à quel point c'est trivial
- Il faudrait quand même une petite conclusion...

Ressources

- Command Processor : http://www.eli.sdsu.edu/courses/spring01/cs635/notes/memento/memento.html#Heading10