IFT3051 - Projet d'informatique

Qualité des Programmes et Patrons de Conception

Présenté à :

Yann-Gaël Guéhéneuc et

Houari A. Sahraoui

Par :

Mehdi El Moutaouakkil

Denise Gbetibouo

Emmanuelle Orcel

Ives Toe

UNIVERSITÉ DE MONTRÉAL

Département d'Informatique et de recherche opérationnelle

Table des matières

Liste de figures et des tableaux

Contexte

Bien qu'étant la plus importante, la qualité d'un logiciel est la caractéristique la plus difficile à définir.

On lui connaît trois principales définitions :

- Selon Iso «*la qualité c'est l'ensemble des traits et des caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou explicites.* »

- Pour Crosby «*La qualité correspond au degré selon lequel un client perçoit qu'un logiciel répond aux multiples attentes* »

- Mais pour Pressman « *la qualité est la conformité aux exigences explicites, à la fois fonctionnelles et de performance, aux standards de développement explicitement documentés et aux caractéristiques implicites qui sont attendus de tous logiciels professionnellement développés* »

Toutes ces définitions montrent clairement que la qualité est une caractéristique importante pour une entité logicielle. Il s'avère donc important de la mesurer afin de savoir si le logiciel est «*bon* » mais aussi pour justifier son rapport qualité /prix.

La qualité est une notion très vaste, elle contient des sous-caractéristiques telles que : la maintenabilité, la portabilité, la facilité d'utilisation. Des techniques telles que GQM (Goal Question Metric) ont permis de définir des procédés de mesure des métriques des sous-caractéristiques qui interviendront dans la mesure de la qualité.

Sachant qu'une utilisation abusive de patrons de conception dégrade la maintenabilité d'un programme et les métriques orientées objets, il s'avère intéressant d'analyser les liens susceptibles d'exister entre la présence de patrons de conception dans un programme et la qualité du programme.

Objectifs

Le travail en question dans ce projet consiste à analyser plusieurs versions du logiciel DrJava et d'en extraire les patrons de conception qu'auraient pu appliquer intentionnellement ou pas les développeurs.

Après avoir identifié les patrons de conception, le travail consiste à corréler l'introduction, la suppression ou la modification de patrons de conception avec les demandes de changements (bogues, améliorations, restructurations) soumises à l'équipe de développement de Dr. Java. « La corrélation entre les demandes de changements et l'introduction, la suppression ou la modification de patrons de conception permettra de mieux comprendre les intentions des développeurs et les forces qui poussent à l'utilisation de patrons de conception ».

Le logiciel à analyser permettra d'une part d'approfondir les connaissances de la programmation orientée objet étant donné que le langage de programmation est JAVA. Mais aussi de maîtriser l'environnement de travail Eclipse qui est *la* plate-forme de développement de référence dans l'industrie et le milieu universitaire.

L'identification des patrons de conception permettra d'élargir les connaissances de ces derniers et par la suite de voir leur impact sur la qualité des programmes orienté objet.

L'ensemble des résultats est compilé dans un fichier XML où chaque patron de conception identifié est représenté sous forme d'une micro architecture dont la nomenclature a été définie dès le début du travail par notre responsable de projet Yann-Gaël Guéhéneuc.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
 <program>
  <name version="20020619">DrJava</name >
 <designPattern name ="Proxy">
 <microArchitectures>
<microArchitecture number="400">
        <actor >
        <clients>
        <client roleKind="Class">
        <entity>drjava.model.compiler.CompilerRegistry</entity> </client >
</clients>
 <proxy rolekind="Class">
<entity>drjava.model.compiler.CompilerProxy</entity>
</proxy >
<subject rolekind ="AbstractClass">
<entity>drjava.model.compiler.CompilerInterface</entity>
  </subject >
 <realSubject rolekind="AbstractClass">
<entity>drjava.model.compiler.CompilerInterface</entity>
</realSubject >
</actors >
<comments>CompilerProxy appelle le sujet CompilerInterface par la
méthode compile. Le  Sujet et le RealSubject sont les mêmes entités.
comments>
</microArchitecture >
</microArchitectures>
```

Figure 1. Représentation d'une micro - archictecture identifiée dans la version 20020619 du logiciel DR Java similaire au patron de conception Proxy.

Notre collaborateur Duc-Loc Huynh, qui était membre de la première équipe de recherche et d'analyse des patrons de conception, nous a transmis une méthodologie efficace pour l'identification de patrons de conception dans un programme.

Nous allons tenter de l'expliquer dans les pages qui suivent.

Environnement de travail

La détection de patron de conception dans les versions du logiciel DrJava a été réalisée dans l'environnement de travail fourni par la plateforme ECLIPSE. Les diagrammes de classe du programme ont été réalisés à l'aide du logiciel MICROSOFT VISIO.

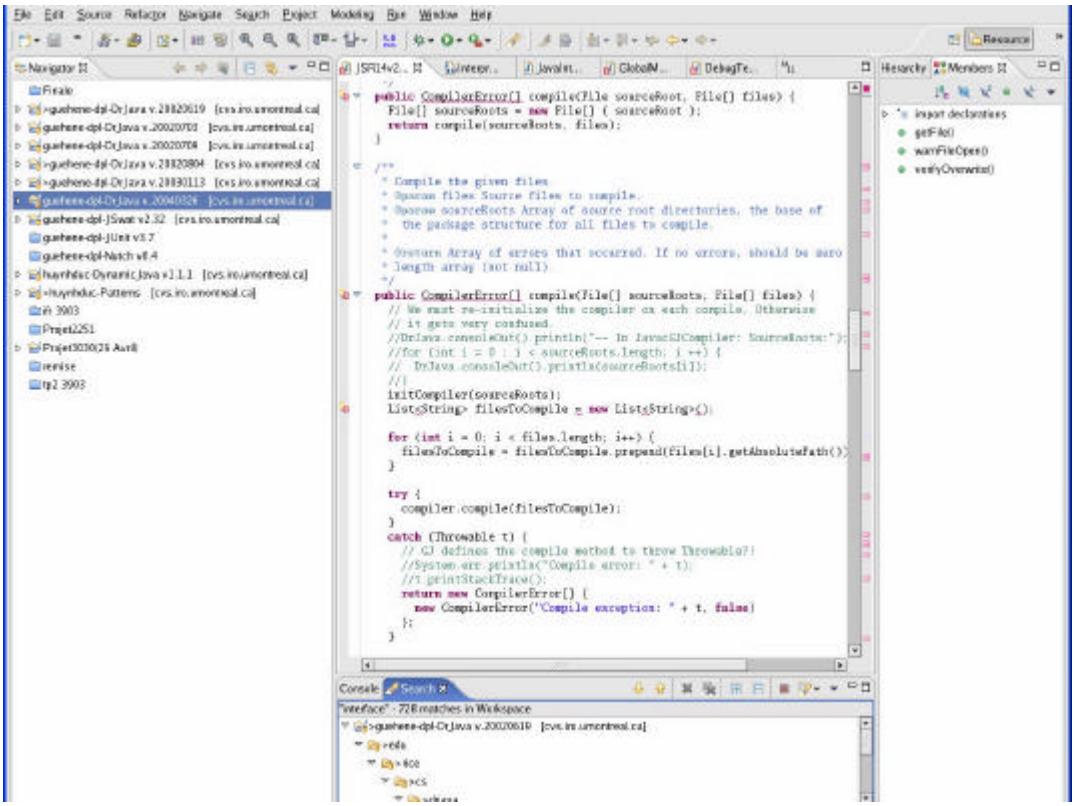


Figure 2. Représentation de l'environnement de travail fourni par la plateforme ECLIPSE

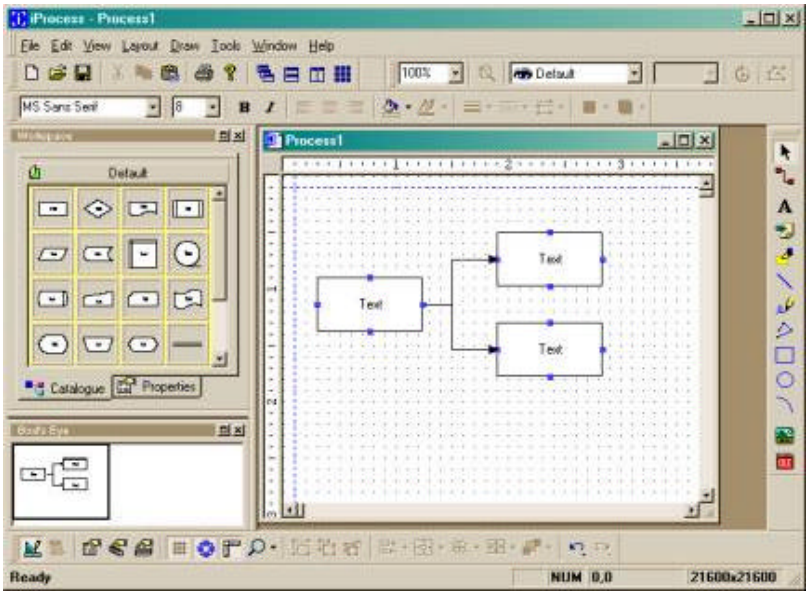


Figure 3. Représentation de la réalisation d'un diagramme UML à l'aide du logiciel Microsoft Visio

Les premières notions que nous avons acquises des patrons de conception ont été lors du cours IFT2251 (INTRODUCTION AU GÉNIE LOGICIEL) où les différents groupes des patrons de conception ont été illustrés en approfondissant ceux jugés importants par le professeur tel que :

- Le patron *ABSTRACT FACTORY*
- Le patron *VISITOR*
- Le patron *OBSERVER*

Les plus complexes tels que les patrons Proxy, Flyweight, Interpreter ou Bridge ont été sommairement survolés ou uniquement mentionnés.

C'est donc avec une faible expérience et comme outil de travail le livre « *Design Patterns Element of Reusable Object Oriented Software* » de GAMA et AL que nous avons procédé à l'identification des patrons de conception.

Méthodologie

1 - Première méthode:

Vu notre expérience de débutant dans la détection des patrons de conceptions notre première méthode de travail n'était pas efficace.

Le but de la méthode était de réaliser des diagrammes UML à comparer par la suite avec ceux du livre de référence, dans le but d'identifier le patron de conception associé.

Ainsi nous associons une classe prise aléatoirement avec les autres classes de son package, en fonction des liens d'héritage, d'agrégation ou d'utilisation rencontrés.

Mais nous nous sommes rendus compte de l'inefficacité de cette méthode lorsque les diagrammes UML ne correspondaient à aucun diagramme de classe de patrons de conception contenu dans le livre de référence. A moins d'avoir découvert des patrons jusque là inexistants ou de partir avec l'hypothèse que les développeurs du Code n'avaient pas utilisé des patrons de conception, dans le développement du logiciel il a été plus sage d'avouer notre tort et de revoir notre méthode d'identification.

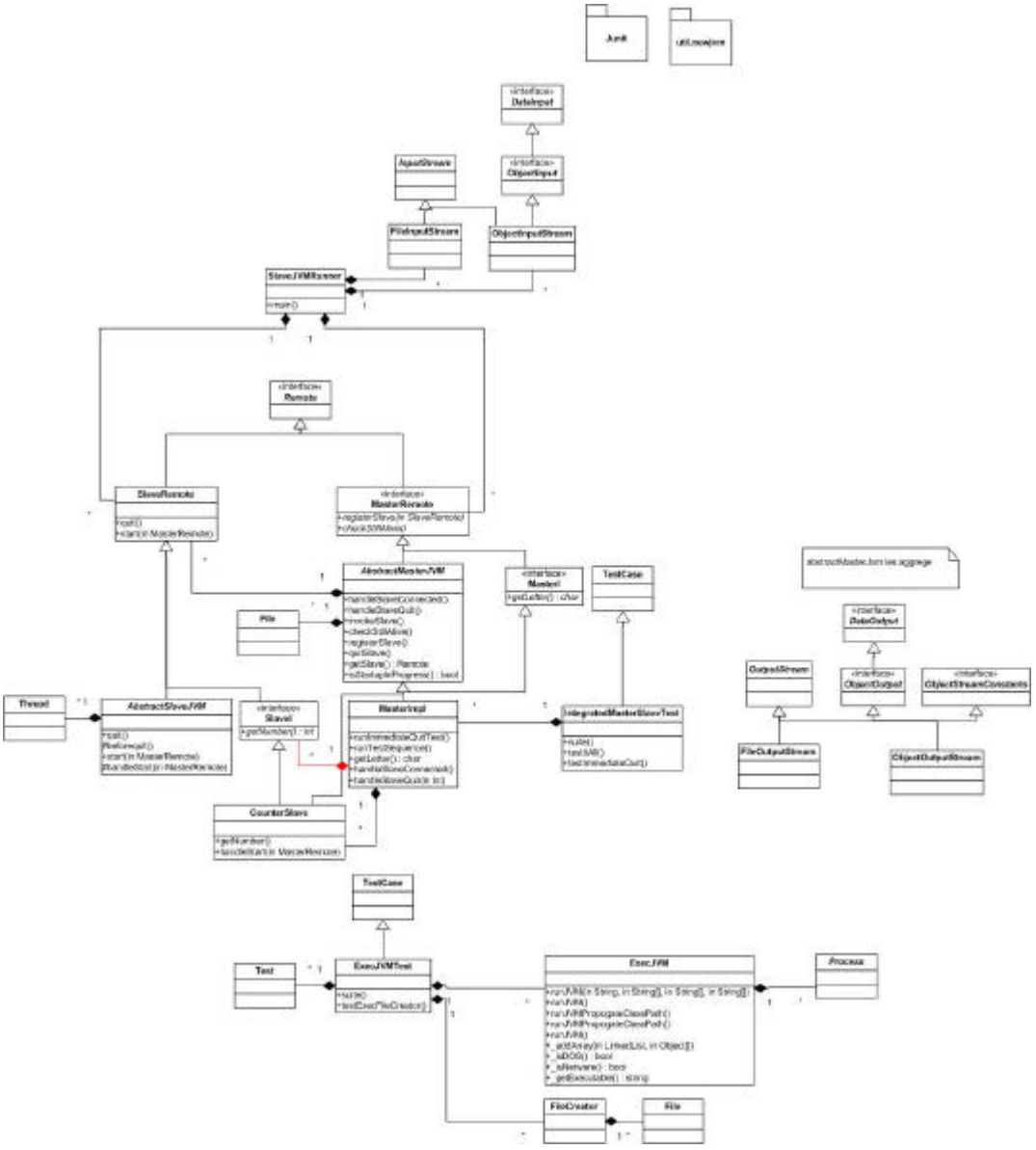


Figure 4. Représentation incomplète d'un « pseudo patron de conception » à partir de classes appartenant à un même package.

2 - Deuxième méthode:

La première méthode s'étant avérée défaillante, notre collaborateur Duc déjà expert dans la recherche nous a légué la méthode de détection qui lui a servi dans le développement de son projet antérieur.
Cette méthode peut se décomposer en plusieurs étapes.

Étape 1:

Après avoir sélectionné le package dans lequel nous voulions identifier les patrons de conception, nous lancions une recherche du mot “interface” dans ECLIPSE.
Les classes qui contiennent le mot sont retournées comme résultat de la recherche et nous déterminions les relations d'héritage ou d'agrégation existant entre ces classes prises individuellement et d’autres classes.

Étape 2 :

Au fur et à mesure que des relations sont identifiées, le diagramme UML de l'ensemble des relations est constitué à l’aide du logiciel Microsoft Visio.

Étape 3:

Dès que le squelette d’un patron de conception prend forme, nous consultions le livre de référence dans le but d'identifier les différents acteurs mais aussi de mieux axer notre recherche, voire même de limiter cette dernière aux liens déjà trouvés.

Cette méthode a eu donc l'avantage de ne pas limiter la détection à un package en particulier mais à plusieurs en même temps. Elle nous a permis de découvrir nos

premiers patrons et aussi d'avoir espoir pour la suite.

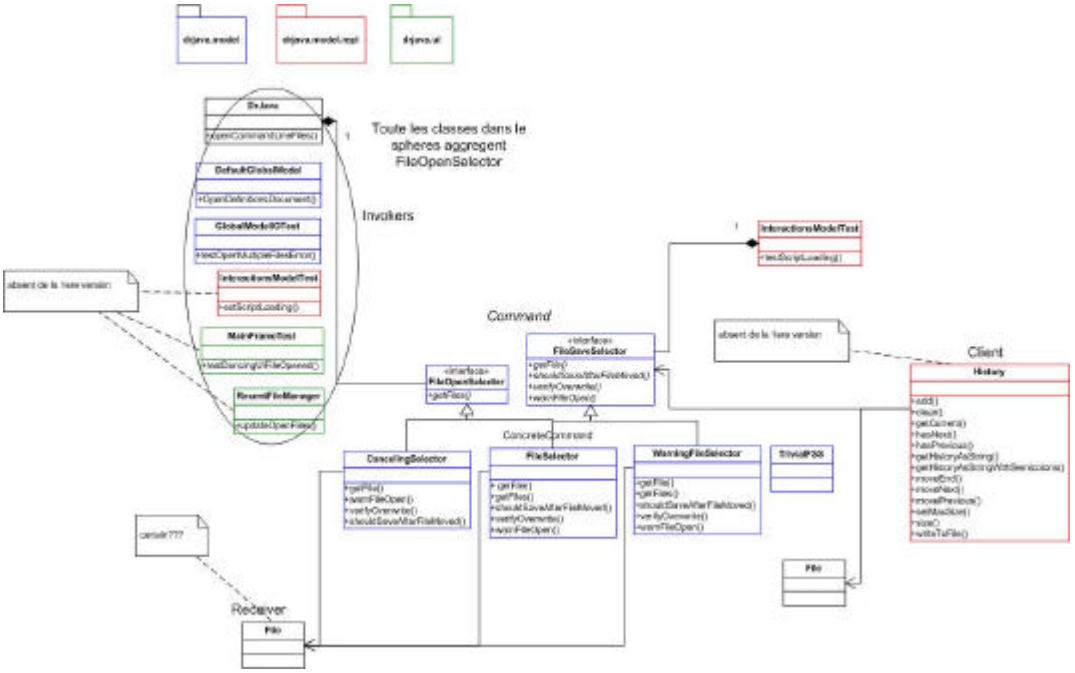


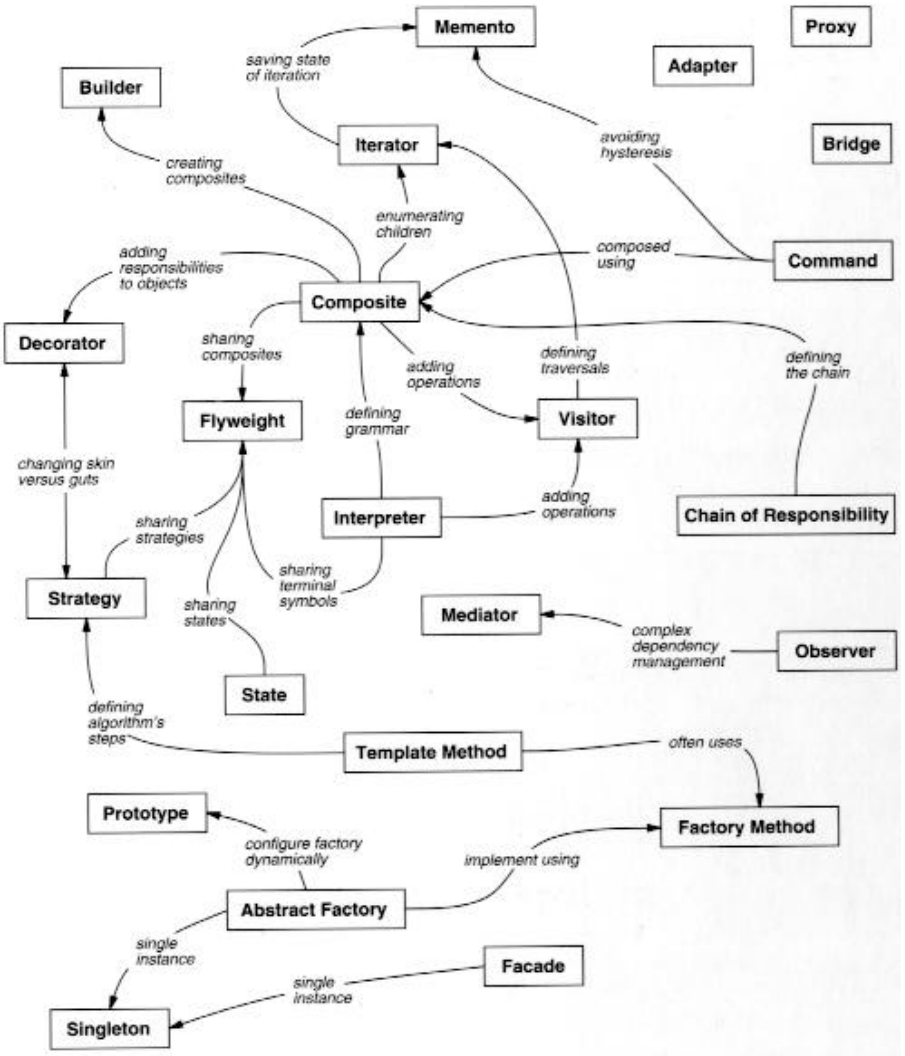
Figure 5. Représentation d'un patron de conception à partir de classes appartenant à des packages différents.

3 - Troisième méthode:

Cette méthode a été la plus efficace qu'on ait trouvé jusqu'à maintenant.

Elle repose sur le même principe que la deuxième méthode.

Elle est entièrement basée sur le schéma des liens du livre de référence



Design Pattern Relationships

Figure 6. Lien du « *Design Patterns Element of Reusable Object Oriented Software*

En effet pour un patron donné, le schéma du livre de référence nous permettait de prédire en fonction des liens rencontrés le patron de conception susceptible d'être découvert dans le code.
Si par exemple nous avions un patron ABSTRACT FACTORY, nous pouvions prédire que le prochain patron à identifier serait oit un PROTOTYPE ou un SINGLETON.

Des techniques additionnelles nous ont également permis d'accélérer notre identification de patrons, elles ont été les suivantes :

- le nom des méthodes et dans certains cas des classes.

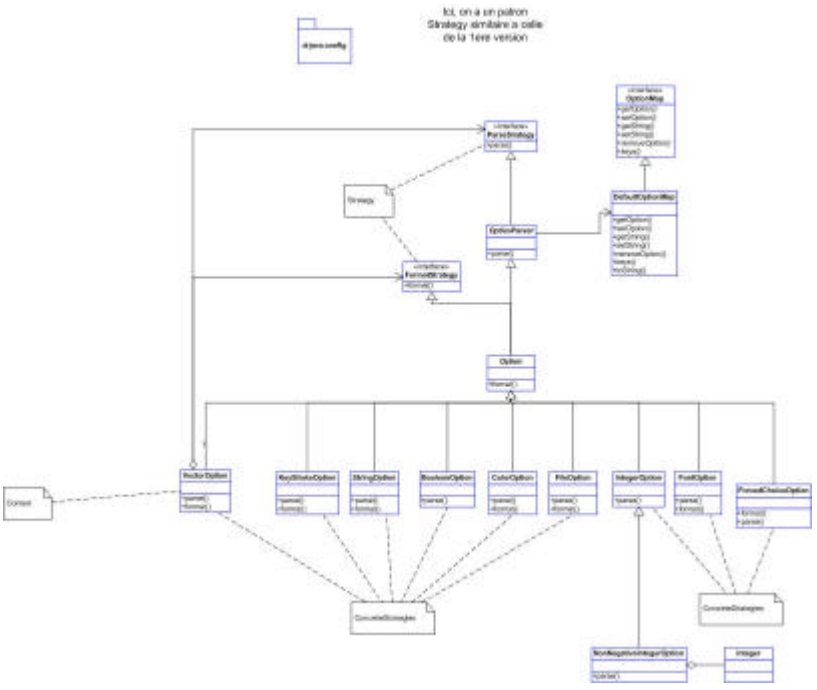


Figure 7. Représentation d'un patron de conception similaire au patron de conception STRATEGY identifié par le nom des classes.

- Aussi nous voyions des noms de méthodes contenant le verbe créer nous pouvions affirmer avec une grande certitude qu'elles référaient à un patron de conception de type créationnel.

- Lorsque la structure physique n'était pas facile à définir ou que celle trouvée n'était pas conventionnelle (selon les prototypes du livre de référence) il s'est avéré important de vérifier la structure logique des diagrammes UML obtenus et cela par certaines actions comme :

- le fait que la classe se crée elle-même dans une de ses méthodes (cas des patrons SINGLETON)
- le fait que la classe, dans une de ses méthodes, crée l'instance d'une autre classe.
- le fait que des classes se partagent des méthodes.

Étant donné la méthodologie rigoureuse d'analyse employée nous pouvons affirmer que les résultats produits sont assez fiables.

Résultats

De la première version du logiciel DrJava v20020619 nous avons obtenus des résultats.

Afin de savoir si lors de la modification des classes dans une nouvelle version les développeurs avaient utilisé de nouveaux patrons de conception, la plus récente version de DrJava v20040326 a été analysée.

Pour voir les ajouts progressifs qui ont été faits entre les deux versions éloignées, la version v20030203 intermédiaire a été analysée.

Version 20020619	*Version 20030203*	*Version 20040326*
ITERATOR	idem	idem
STRATEGY (2)	modifié	idem
COMMAND (2)	une modification	idem
PROXY	idem	idem
ADAPTER (2)	1 patron absent	1 patron absent
MEMENTO	idem	idem
SINGLETON (8)	-	ABSTRACT FACTORY
-	-	ITERATOR
-	-	VISITOR
-	-	SINGLETON (12) ADAPTER
		STATE
INTERPRETER(incomplet)	INTERPRETER(incomplet)	INTERPRETER(complet)

Tableau des résultats

Commentaires personnels

Gbétibouo Denise

Au début du projet, mes connaissances théoriques sur les patrons de conception se résumaient aux notions acquises durant les cours IFT2251et IFT3901. Ces dernières n'étant pas très exhaustives, pour arriver à bien identifier les patrons utilisés dans le code, il aurait fallu ingurgiter le livre de référence mais surtout apprendre par cœur les vingt trois différentes structures.

Réaliser les diagrammes de classes à la « main » et étapes par étapes a été très constructif. Contrairement à ce que certaines personnes pourraient penser, ce travail est fastidieux mais il est d'une extrême précision , dans ce sens que lorsque des diagrammes de classes complexes étaient générés, il y avait la possibilité de les réorganiser afin d'y voir un peu plus clair. Pour ma part cette méthode est beaucoup plus efficace que celle qui est de générer les diagrammes de classe à l'aide d'un logiciel qui dans certains cas sont erronés.

Dans mon cas, les différents risques que j'ai rencontré durant le projet sont les suivants :

- l'échéancier : durant cette session, le département a été secoué par une grève de trois semaines et entre nous cela a complètement changé mon rythme de travail. À la fin ce celle-ci, la rentrée des classes coïncidait bizarrement avec la remise des trente six mille travaux et la préparation des examens, c'était le rush. Il a fallu s'organiser de façon plus stratégique : penser aux examens étant donné que le projet s'est déroulé durant une session normale. Finalement le pire est passé et j'ai pu remettre à la bonne date mes travaux.

- le fait que nous ne disposions pas de tous les logiciels, nécessaires au projet, dans tous les laboratoires était assez embêtant, pour ma part cela revenait à installer ces derniers chez moi, mais étant donné que mon ordinateur ne marchait pas il a fallu faire des heures supplémentaires à l'école aux endroits appropriés ou même me retrouver chez mon collègue Ives pour travailler.

En conclusion je pourrai dire que l'identification des patrons de conception demande beaucoup de techniques et qu'elle ne se fait pas à partir de règles empiriques.
Il peut arriver que les diagrammes de classes qu'on obtient ne répondent pas aux définitions théoriques, dans ce cas il a été plus judicieux de porter une attention particulière à la structure logique : les comportements des méthodes ou des classes pour l'identification des rôles.

La chose qui a été très bénéfique et qui m'a beaucoup touchée est le fait que notre responsable de projet Yann-Gaël Guéhéneuc et notre collaborateur Duc-Loc Huynh étaient constamment disponibles pour nous mettre sur la bonne voie et nous éclairer dans face à certains problèmes coriaces.
Personnellement le projet a été l'occasion de me surpasser, j'avais quatre cours cette session et je travaillais tous les jours, mais faire ce projet avec toutes ces obligations, m'a permis de m' organiser et d'optimiser mon temps de libre.
La gestion des projets n'est plus un concept abstrait pour moi; j'ai pu enfin la mettre en pratique, et l'avantage est qu'il n'y aura pas de grande différence entre le déroulement de ce projet et le monde du travail.

Toe Ives

Bonjour à la future équipe, voici quelques conseils qui, je pense, vous aideront dans la recherche de patrons de conception.

Premièrement la compréhension des patrons de conception est très importante. A nos débuts, quand nous trouvions un patron de conception, nous avions beaucoup de mal à attribuer les rôles aux classes. Ceci était dû à notre incompréhension des patrons. En effet il a fallu 3 semaines avant que l'on trouve notre premier patron qui était malheureusement un SINGLETON ☹ .

Au fur et à mesure que nous avancions dans nos recherches, il était plus facile de trouver les patrons car on commençait à bien comprendre dans quel contexte et pour quelle raison ils étaient utilisés. Le livre de référence *"Design Patterns Element of Reusable Object Oriented Software"* a été bien utile car il montre en exemple les classes et leur relation pour chaque patron de conception. Si vous avez le temps, lisez chaque description avec son exemple afin d'avoir des bases solides pour vos recherches.

Une des choses les plus importantes à faire est de bien identifier les packages dans lesquels se trouvent les classes quand vous dessinez les diagrammes UML. Une technique utilisée par notre groupe a été de colorier chaque classe de la couleur attribuée au package correspondant. Cela facilite le transfert des diagrammes UML vers les fichiers XML (où il faut donner les noms des packages dans lesquels se trouve un acteur en particulier). A nos débuts, nous perdions beaucoup de temps à essayer de retrouver les packages des diagrammes.

Pour les dessins UML, si vous avez le choix utilisez VISIO !!!!!!!!!!!!!!!! et non UMONDO (trop lent, bizarre et le pire ça PLANTE !!!!!!!!!)

En ce qui concerne la recherche, une technique très efficace mise au point par notre collaborateur Duc - Loc Huynh et personnalisée par notre équipe fut de trouver d'autres patrons existants à partir des liens décrits dans le livre de référence *"Design Patterns*

Element of Reusable Object Oriented Software". Un exemple a été la rencontre des singletons qui nous menaient d'après notre livre de référence au patron ABSTRACT FACTORY. (Finalement les SINGLETON ont été UTILES contrairement à ce que disait DUC ☺).

Pour plus d'aide de la part de Duc achetez-lui le SUPER LAIT CONCENTRE JUMBO de Mcdonalds à la saveur ROLO ☺.

Pour plus de conseils, écrivez-moi à YvesToe@hotmail.com

BONNE CHANCE

P.S : Je voudrais remercier Yann-Gaël Guéhéneuc, Houari A. Sahraoui et Duc - Loc Huynh pour leur support et une expérience enrichissante.

Orcel Emmanuelle

La recette pour dénicher le plus de patrons de conception que possible? La voici : attacher Duc à un poteau pour ensuite le menacer de lui faire manger de la bouffe santé s'il ne vous remet pas ses résultats!

Je blague, je blague!

En fait, dans notre cas, le début a été extrêmement pénible car, malgré les conseils de Duke et le rapport de son ancien projet, nous demeurions toujours hésitants quant à la manière de procéder. Par contre, tout au long de ce chemin ardu, j'ai remarqué que quelques petits trucs facilitaient grandement la tâche. Les voici :

Première méthode

- Recherchez les singletons!! En effet, si vous regardez sur la couverture arrière du bouquin qu'on vous a sans doute fourni, des liens entre différents patrons de conception sont démontrés. De cette façon, vous êtes sûrs de pouvoir dénicher un Abstract Factory ou un Factory Method. Ces derniers vous mèneront sûrement à d'autres patrons (en suivant les liens…!).

Deuxième méthode

- Cliquez une fois sur la version à analyser.
- Tout en haut de l'écran, sélectionnez Search, puis File pour rechercher le mot 'interface'.
- Grâce aux résultats trouvés, commencez à dessiner les liens entre les interfaces et les classes pour, au fur et à mesure, les comparer aux structures décrites dans le livre. Vous trouverez bien assez vite des patrons de conception.

Troisième méthode

- Si vous avez de la chance, le code qu'on vous aura donné sera bien écrit et documenté (sinon, dommage et bonne chance!). Alors, il se pourrait bien que les noms de certaines classes soient révélateurs comme par exemple, parseStrategy, DocumentAdapter, etc. Dans ce cas, vous n'avez qu'à sauter sur l'occasion et rechercher les liens manquants.

Voilà, je pense, les trois recettes pour trouver efficacement des patrons de conception. Faites-en bon usage, et bonne chance!!!

El Moutaouakkil Mehdi

Dans le cadre du Projet 3051, nous avons dû rechercher des patrons de conception afin de comparer différentes versions. Cette expérience a beaucoup amélioré mes aptitudes en programmation orientée objet en ce qui concerne l'analyse et la conception de programmes. En effet les patrons de conception offrent des solutions très pratiques à des problèmes courants qui autrement seraient très durs à résoudre. Maintenant quand j'aurai à coder un programme je vais essayer d'insérer les patrons de conception afin d'augmenter l'efficacité du code.

Le seul conseil que je peux vous donner est ceci : étudiez bien les différents patrons avant de commencer. Cela vous facilitera énormément la tache. Une autre chose qui nous a beaucoup aidé est le diagramme montrant les liens entre les patrons (à la fin de notre livre de référence). De cette façon nous avons pu dériver de nouveaux patrons de ceux qu'on avait trouvé.

Bonne chance à la future équipe!

Annexes

Dans cette partie sont mis les diagrammes de classe qui ont été réalisés durant le projet.

Version 2002 -06 -19

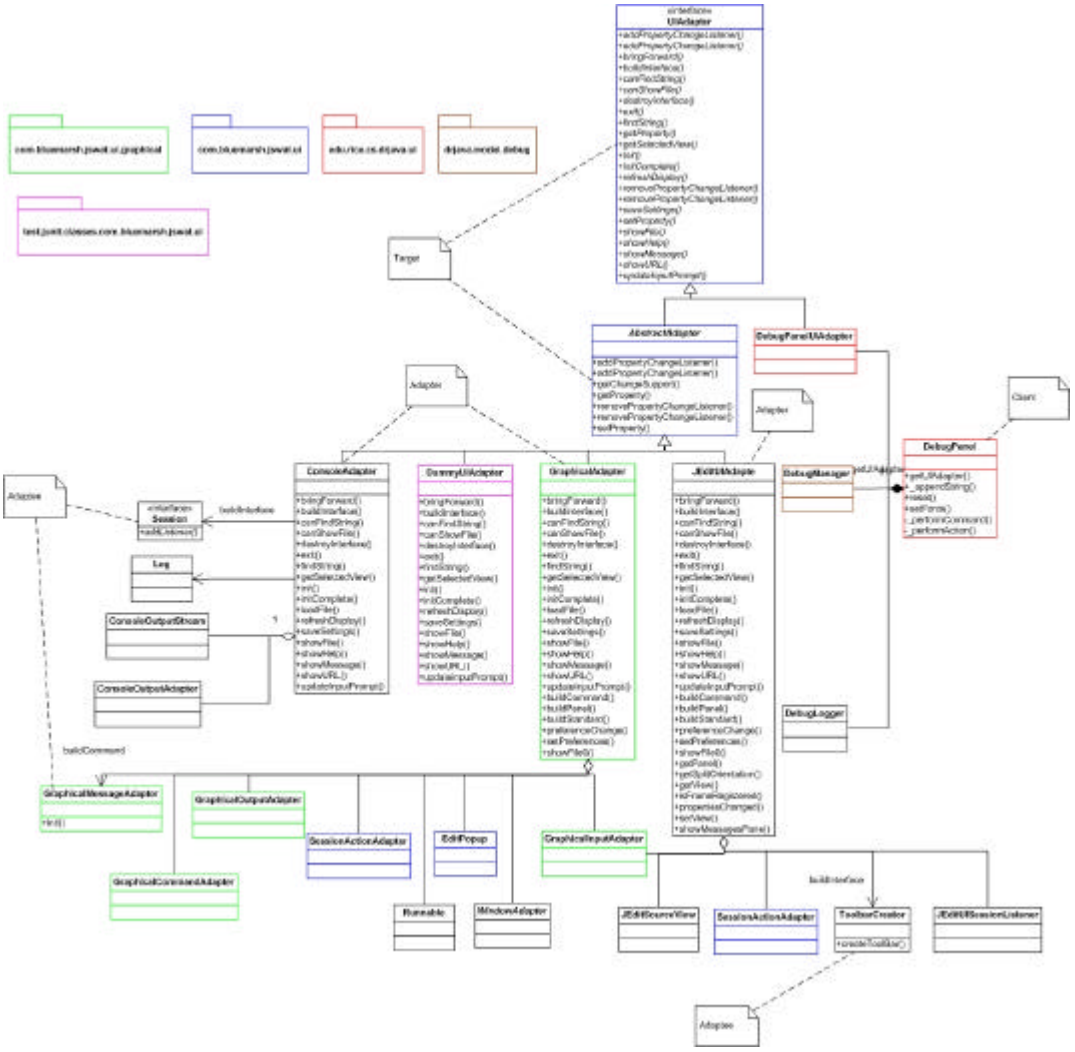


Illustration d'un diagramme de classes similaire au motif « Adpter ».

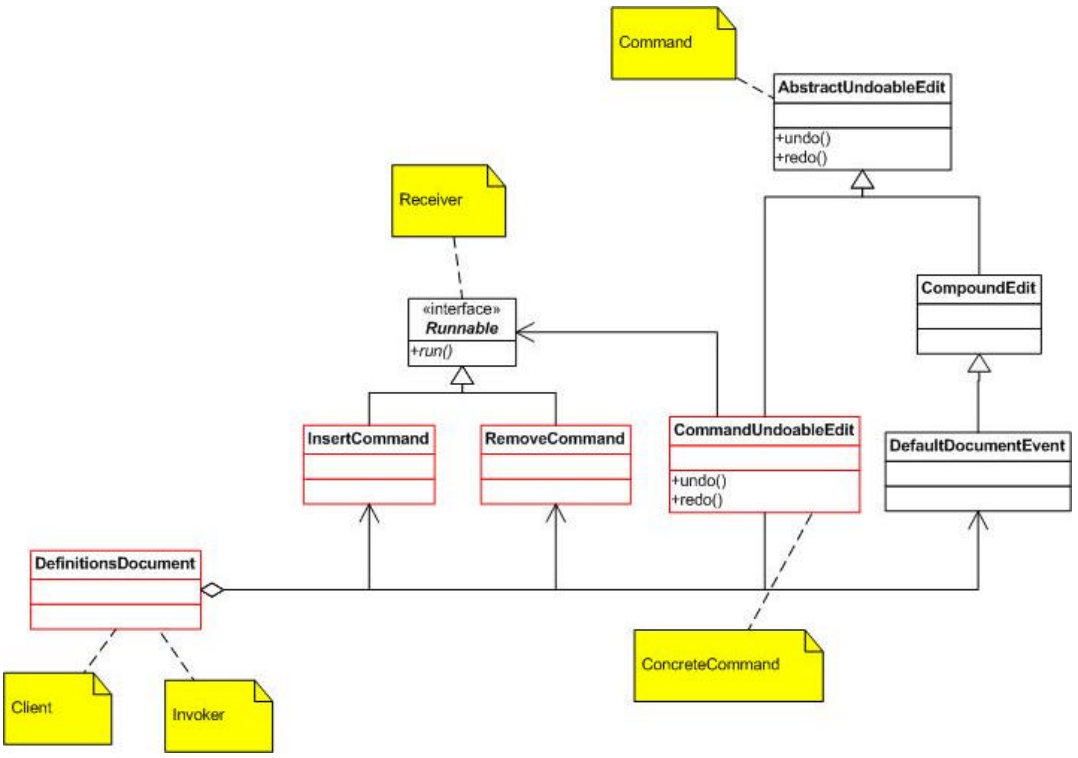


Illustration d'un diagramme de classes similaire au motif «Command».

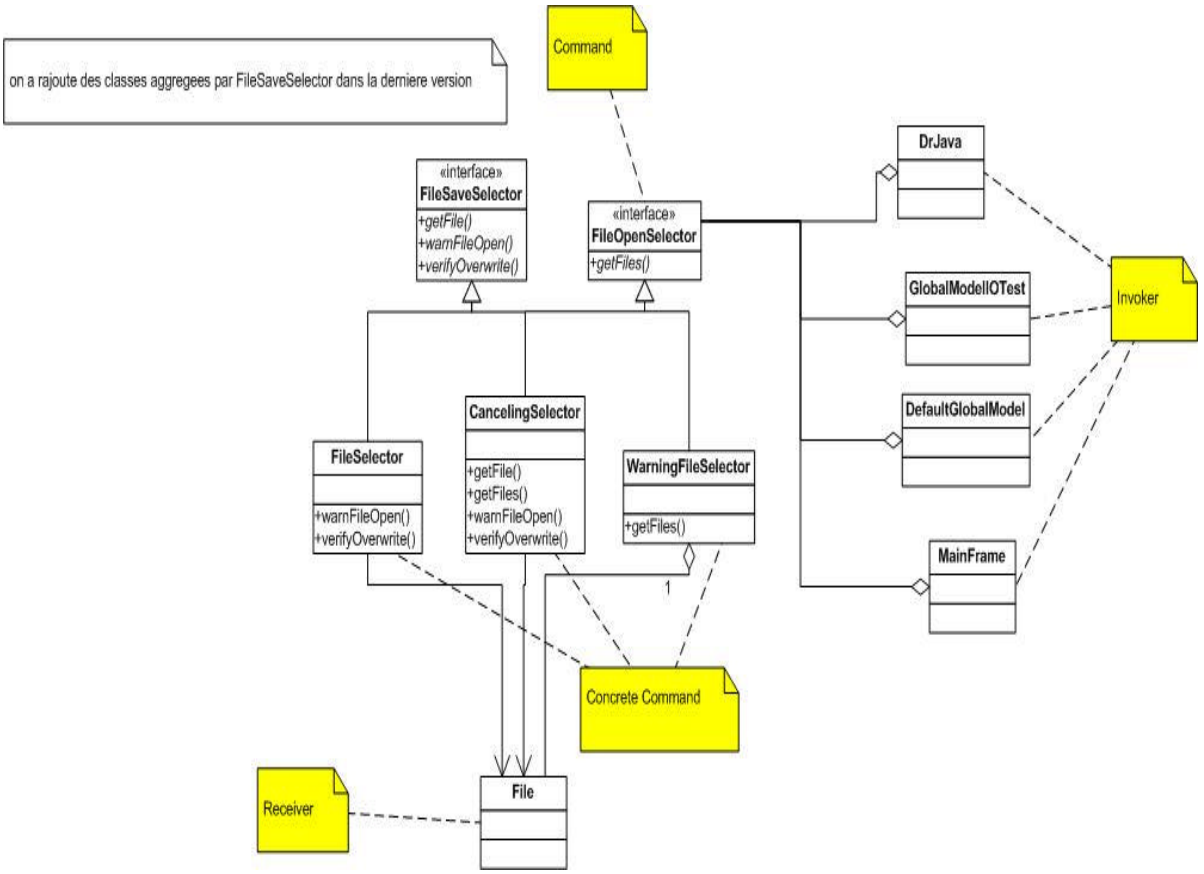


Illustration d'un diagramme de classes similaire au motif «Command».

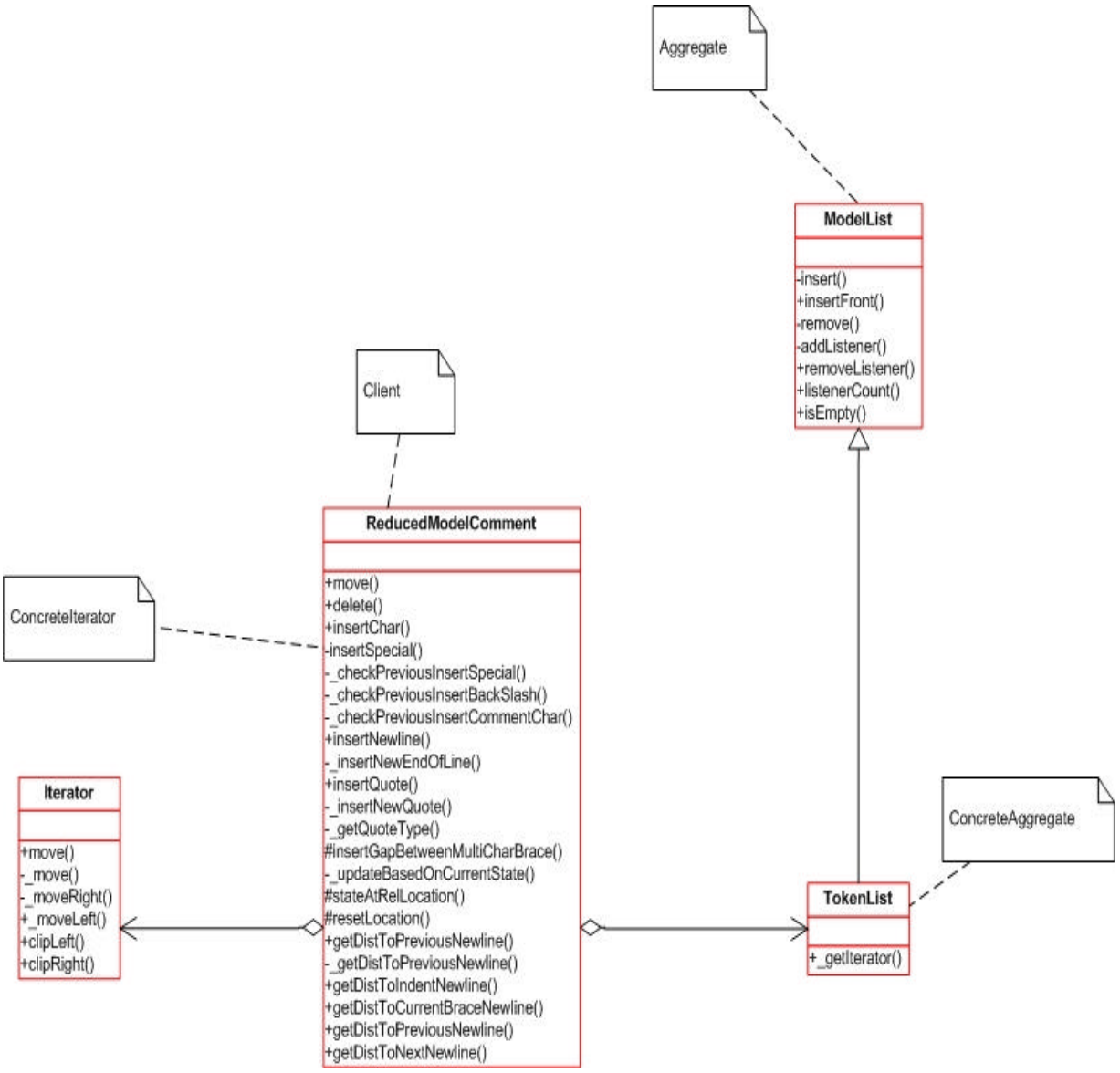


Illustration d'un diagramme de classes similaire au motif «Iterator ».

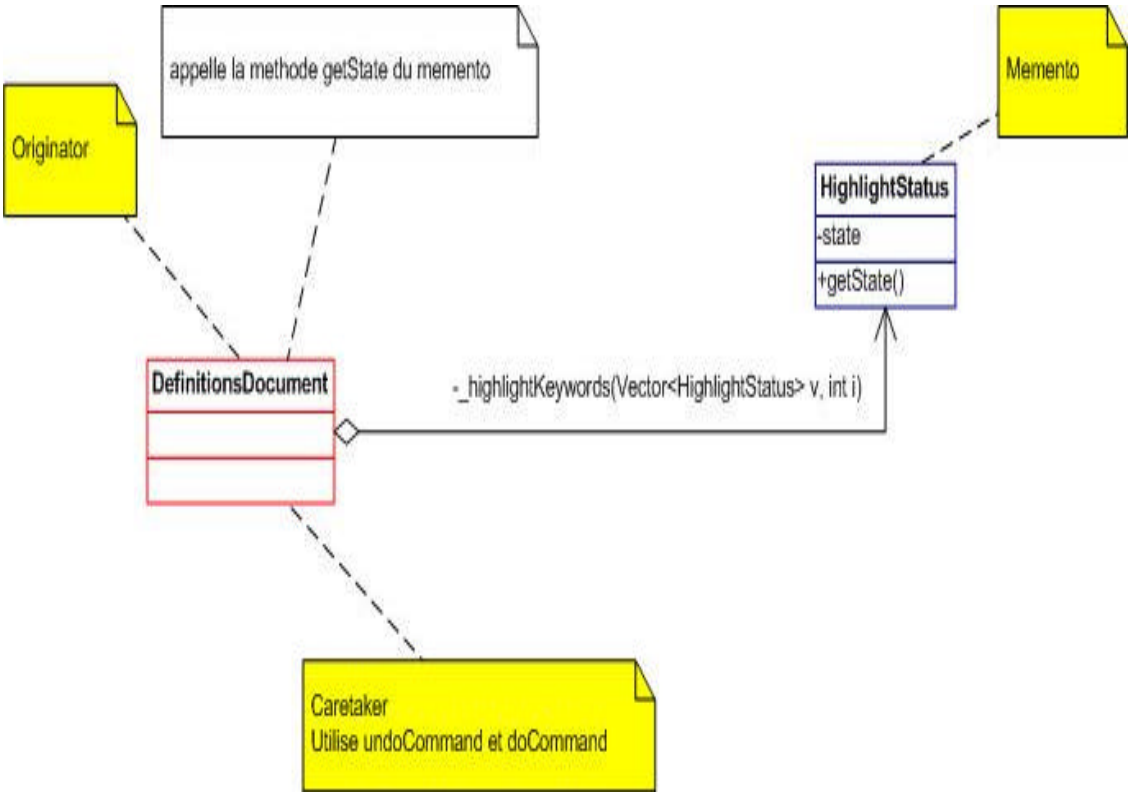


Illustration d'un diagramme de classes similaire au motif « Memento ».

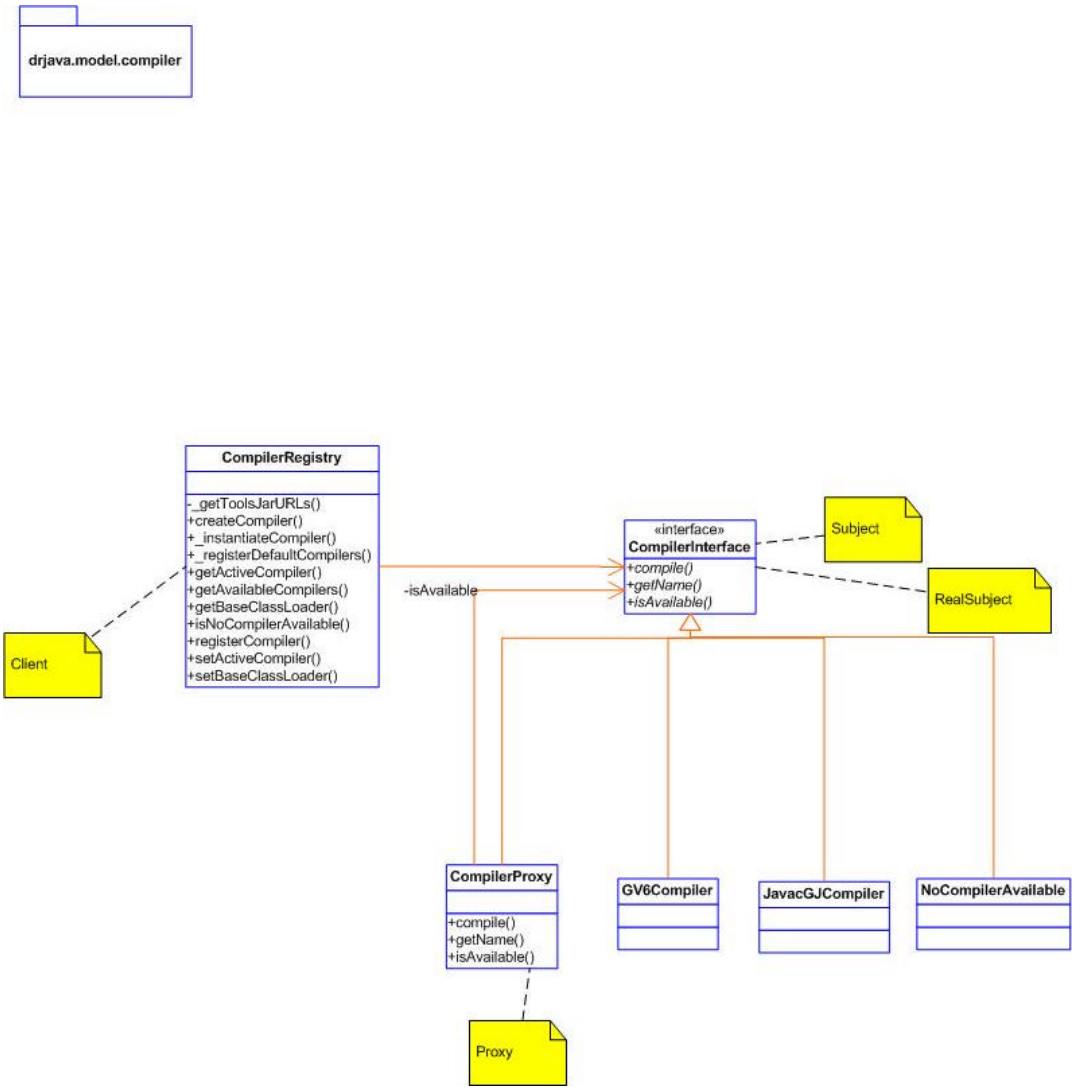


Illustration d'un diagramme de classes similaire au motif «Proxy».

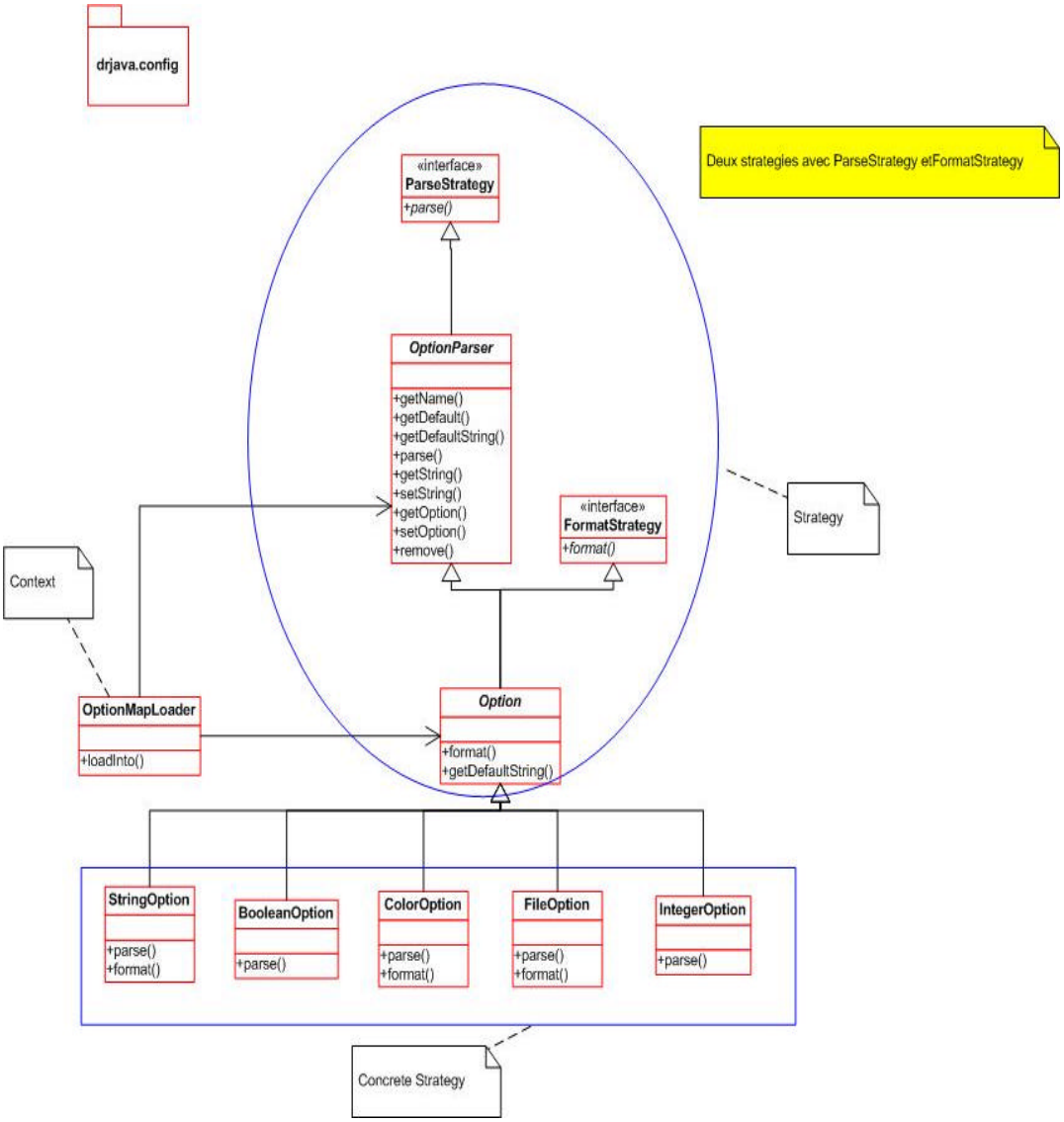


Illustration d'un diagramme de classes similaire au motif «Strategy».

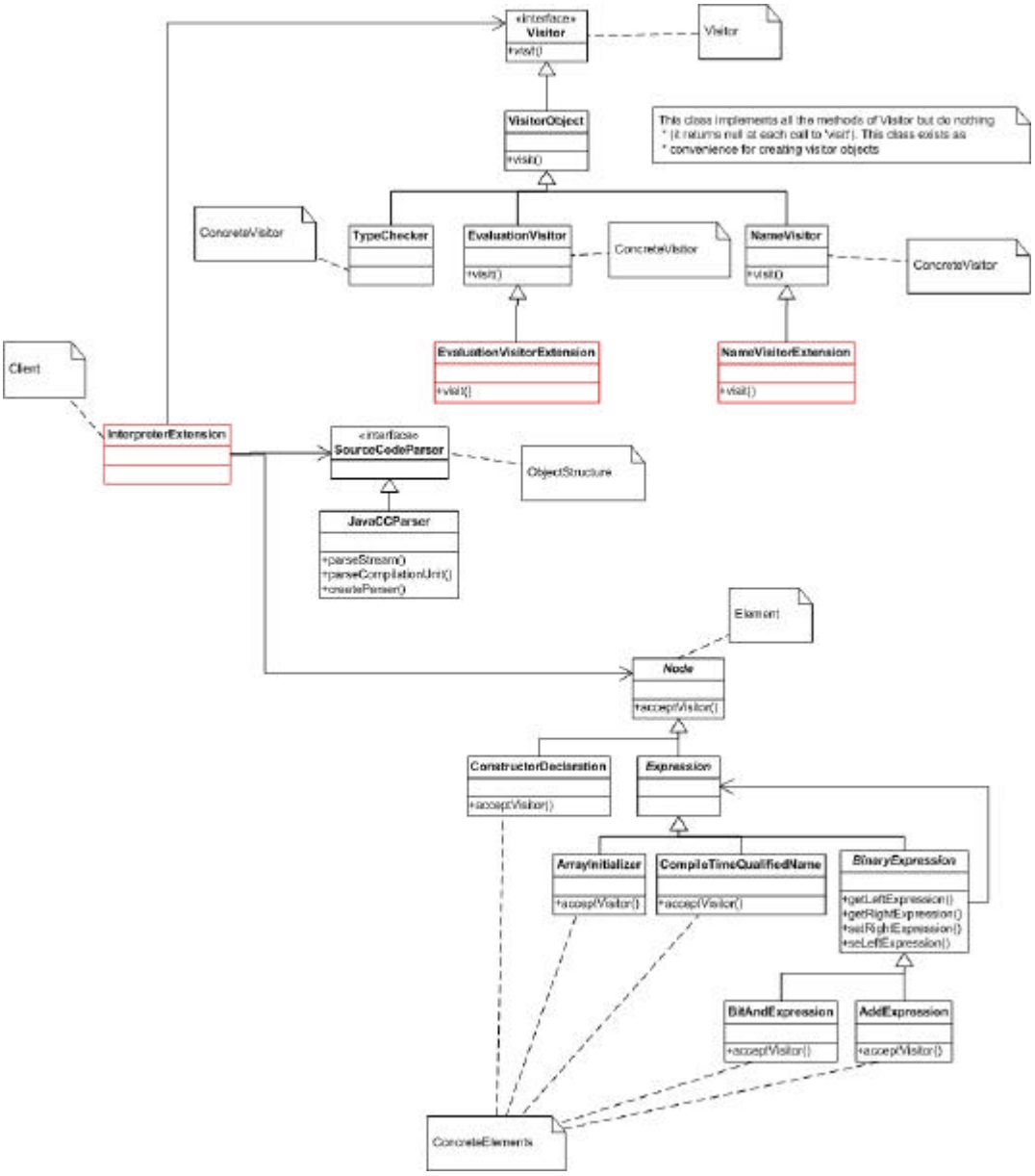


Illustration d'un diagramme de classes similaire au motif « Visitor ».

Version 2003 -02 -03

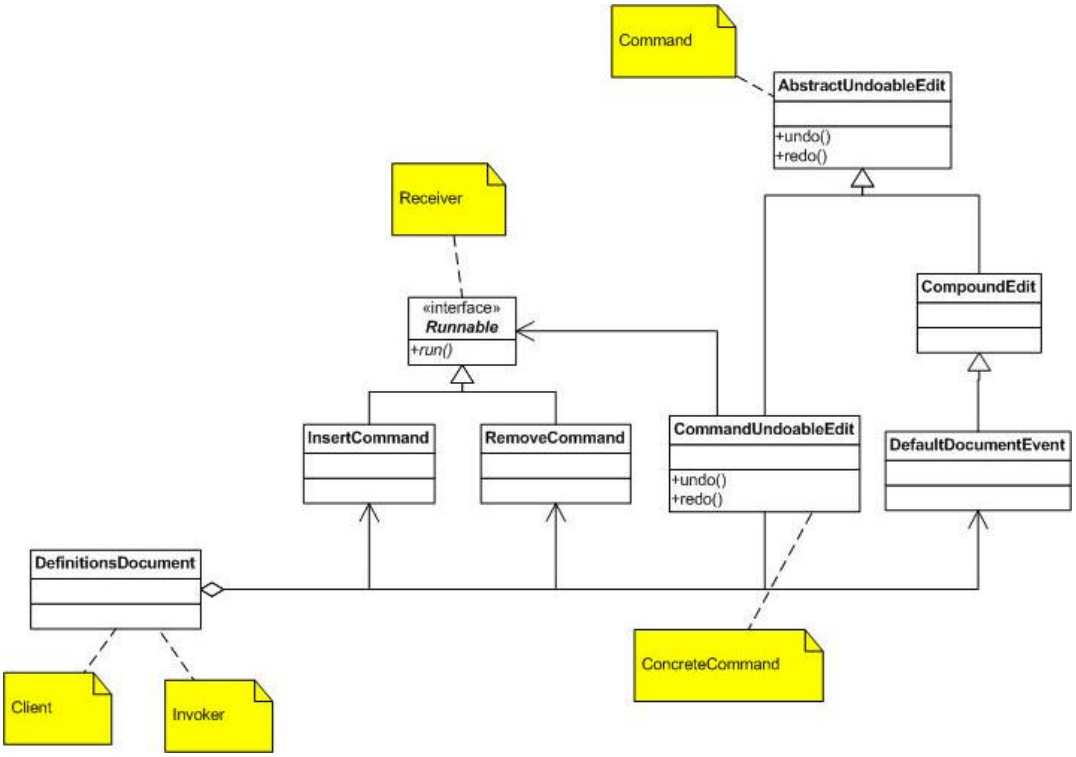


Illustration d'un diagramme de classes similaire au motif «Command».

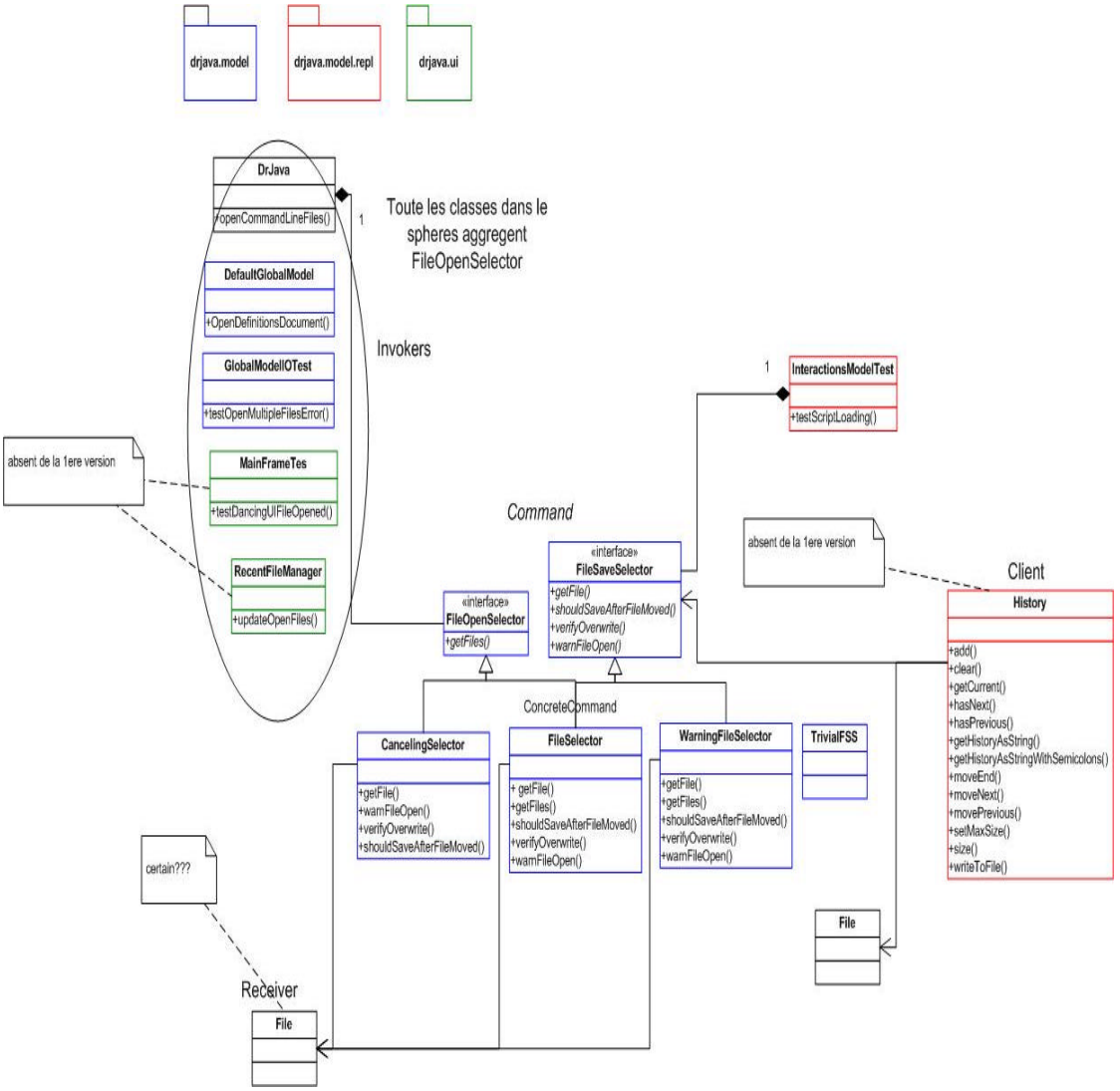


Illustration d'un diagramme de classes similaire au motif «Command».

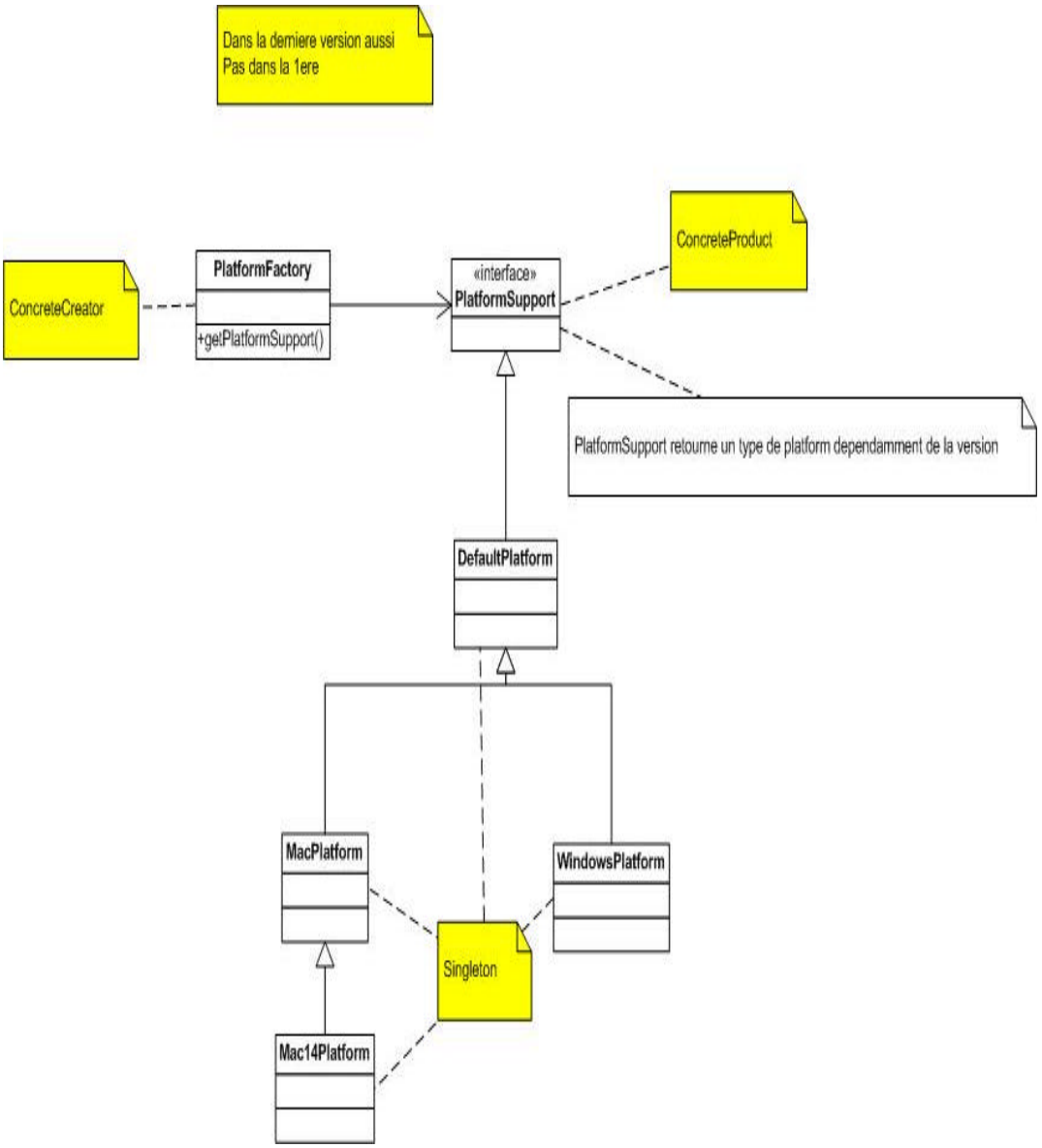


Illustration d'un diagramme de classes similaire au motif « Factory metthod».

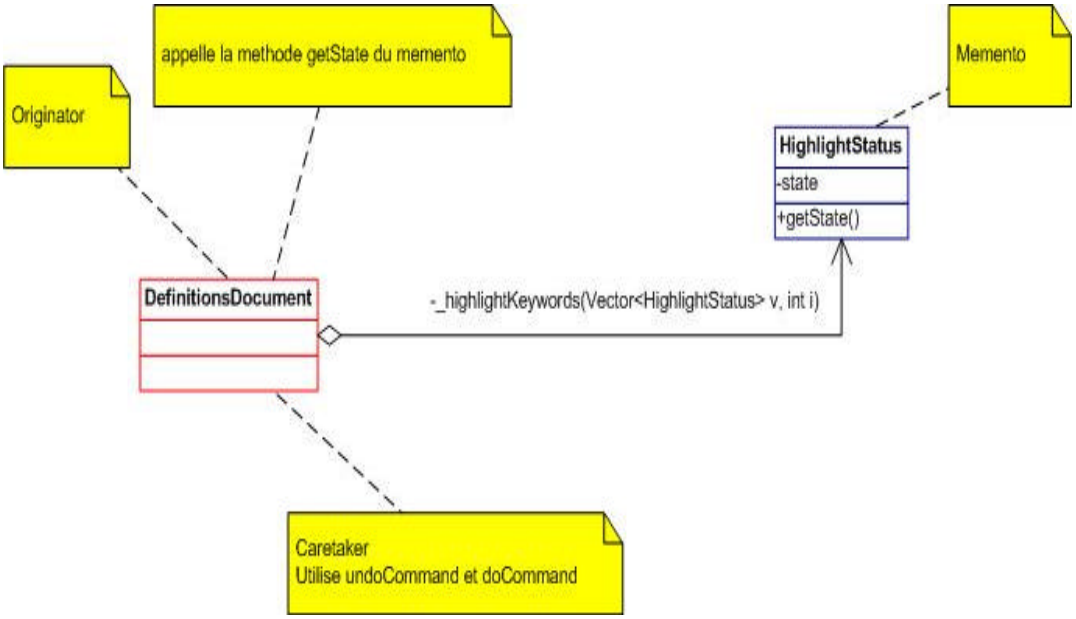


Illustration d'un diagramme de classes similaire au motif «Memento».

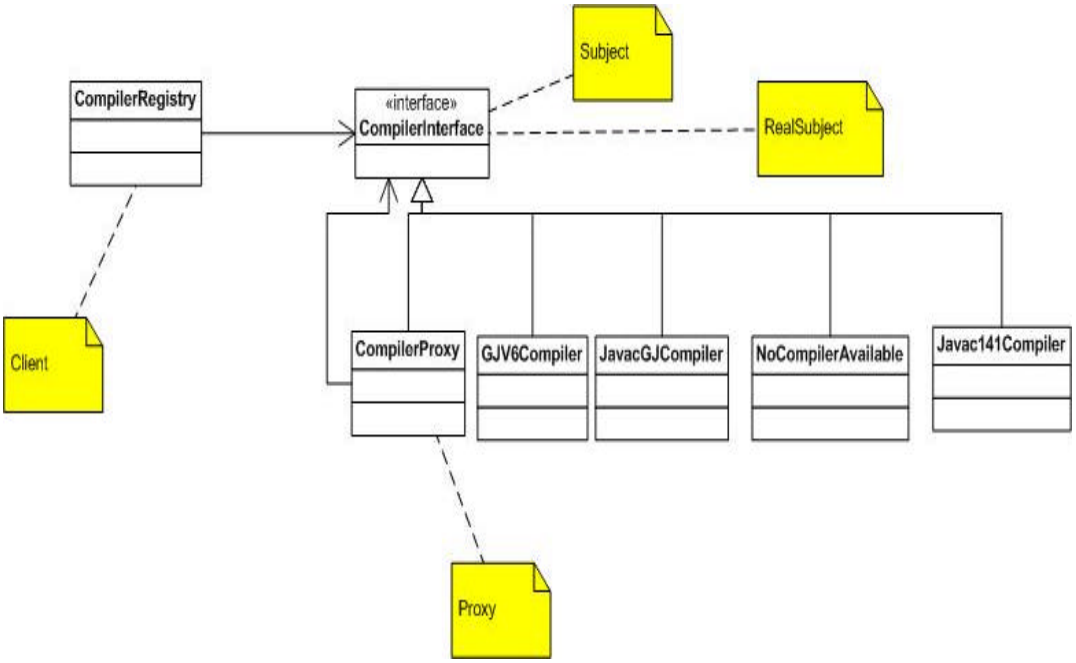


Illustration d'un diagramme de classes similaire au motif «Proxy ».

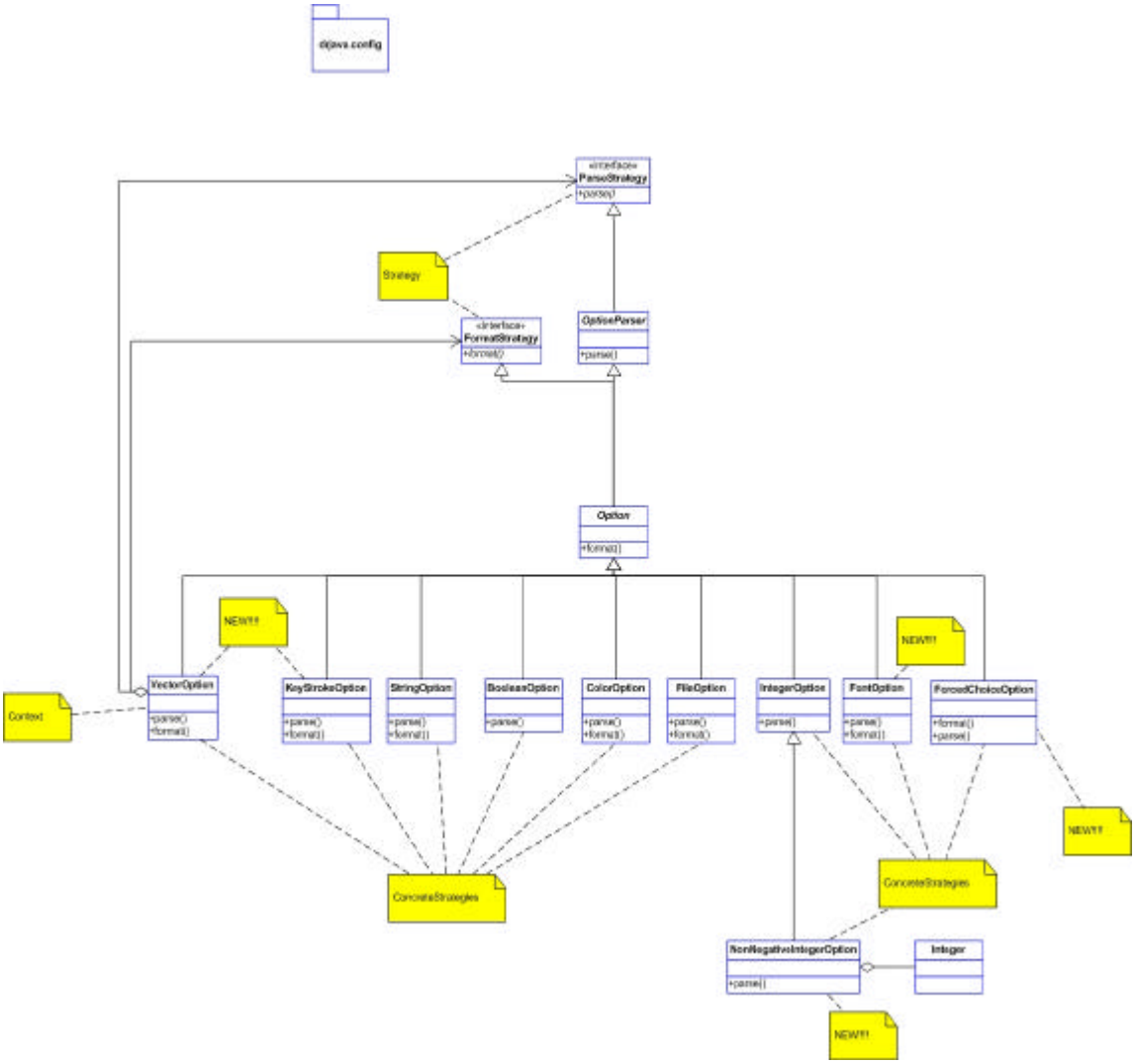


Illustration d'un diagramme de classes similaire au motif «Strategy ».

Version 2004 -03 -26

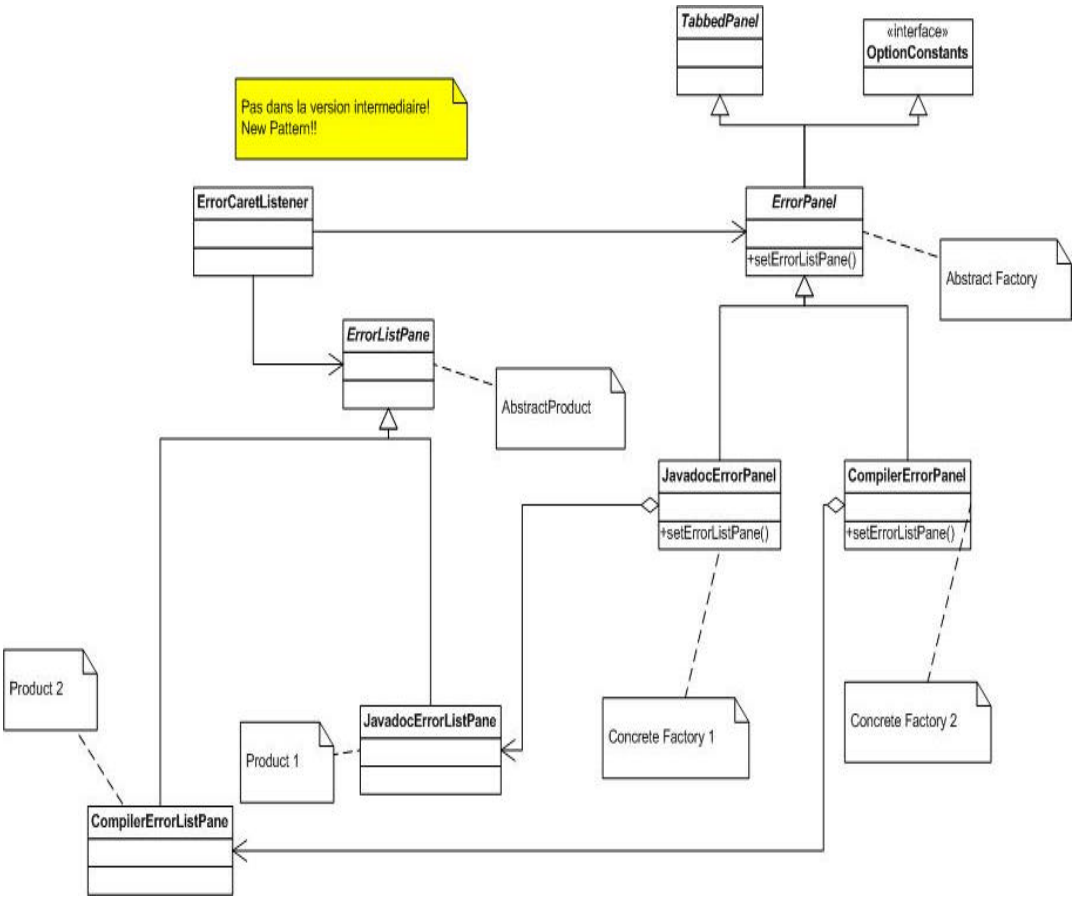


Illustration d'un diagramme de classes similaire au motif «Abstract Factory».

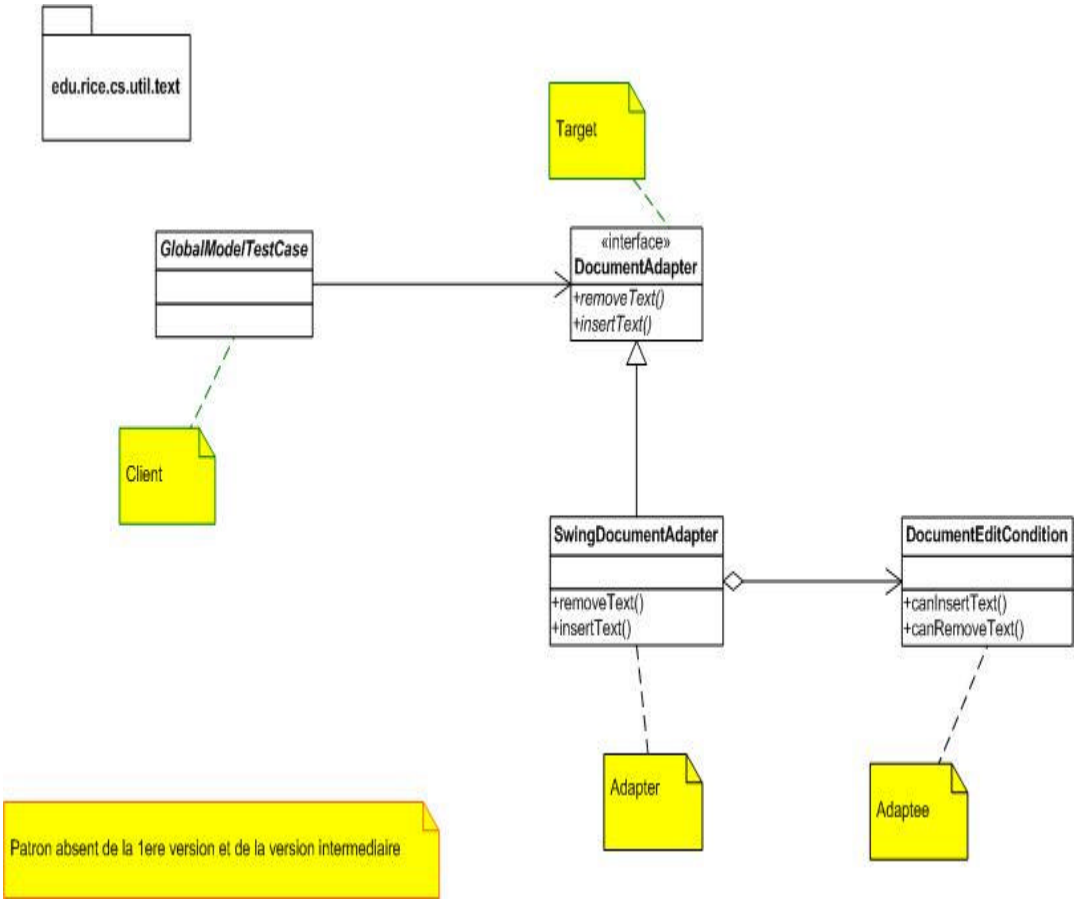


Illustration d'un diagramme de classes similaire au motif «Adapter».

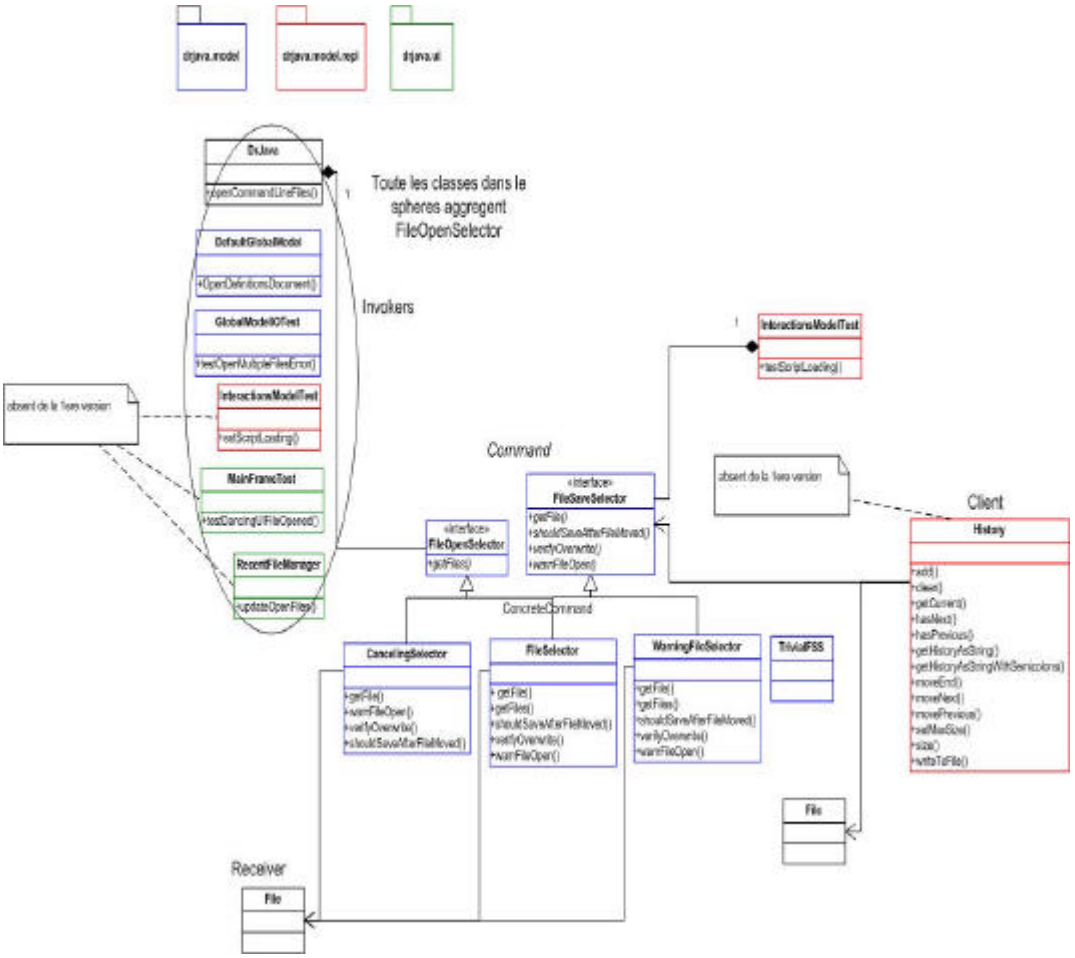


Illustration d'un diagramme de classes similaire au motif «Command».

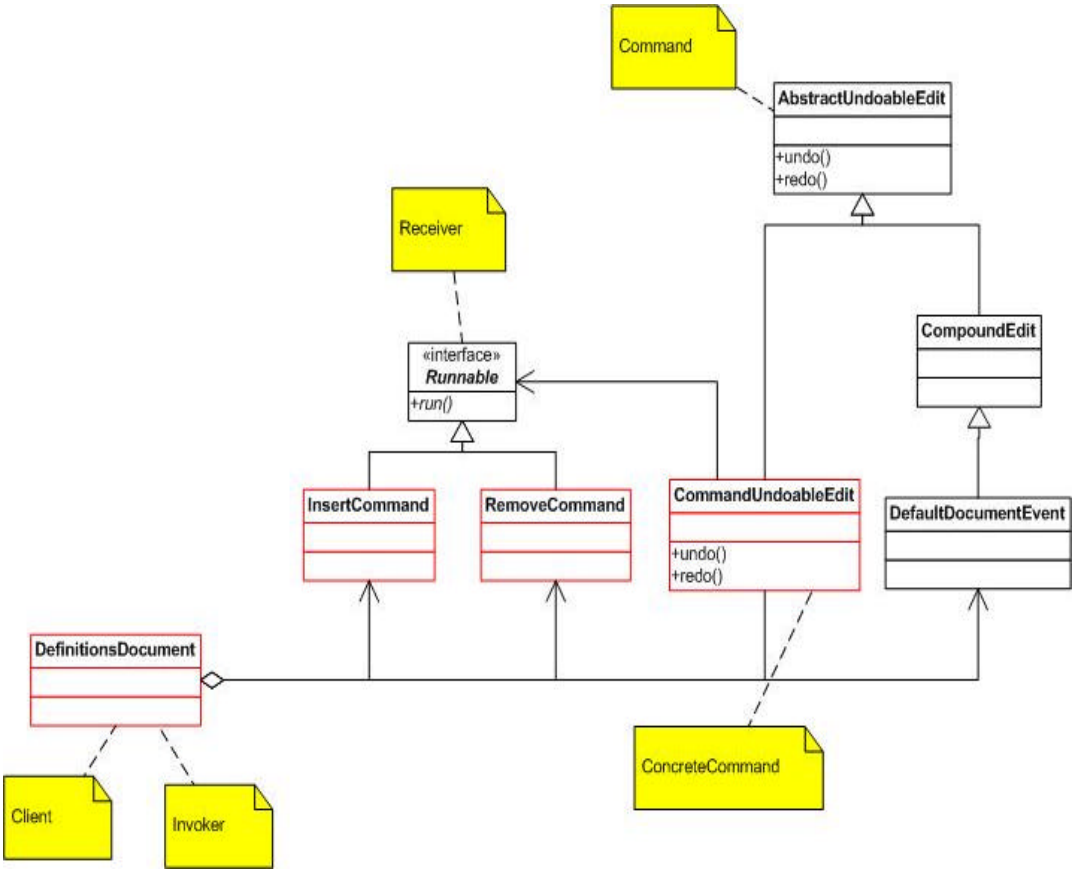


Illustration d'un diagramme de classes similaire au motif «Command».

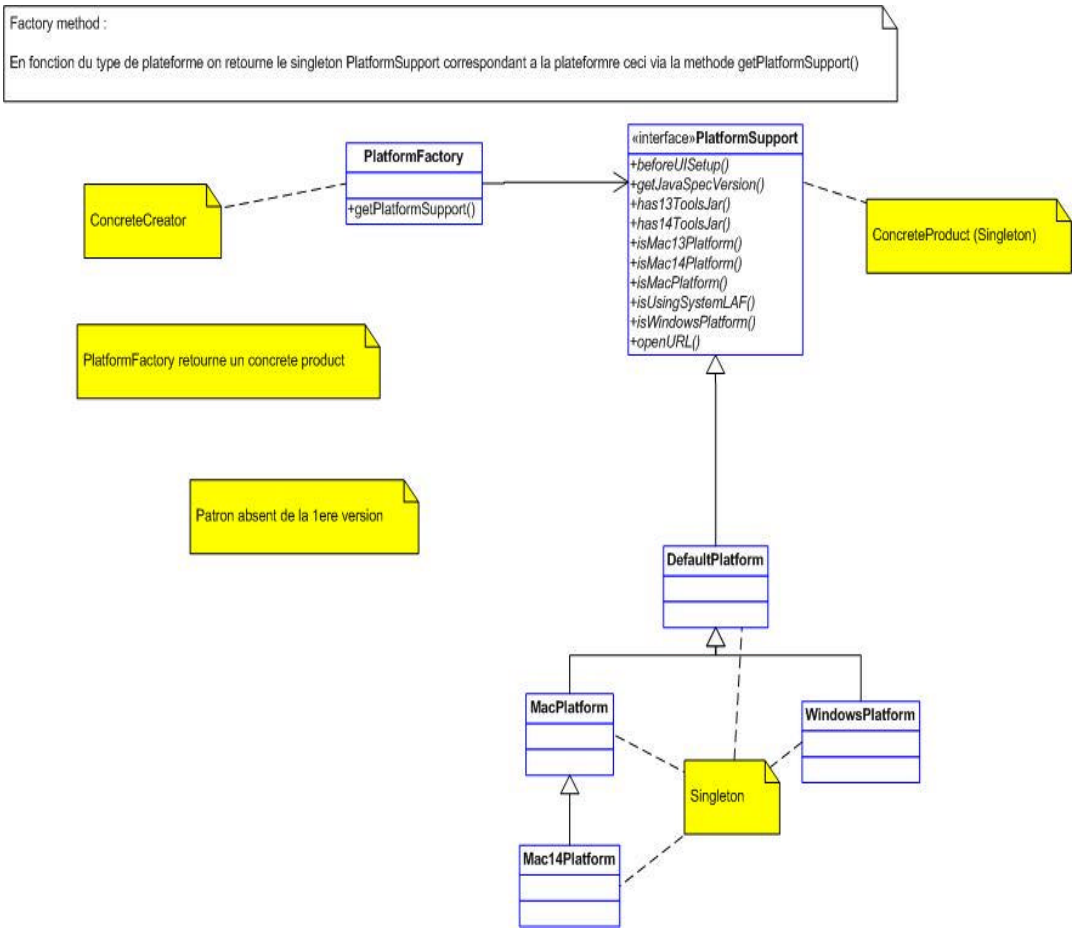


Illustration d'un diagramme de classes similaire au motif «Factory Method».

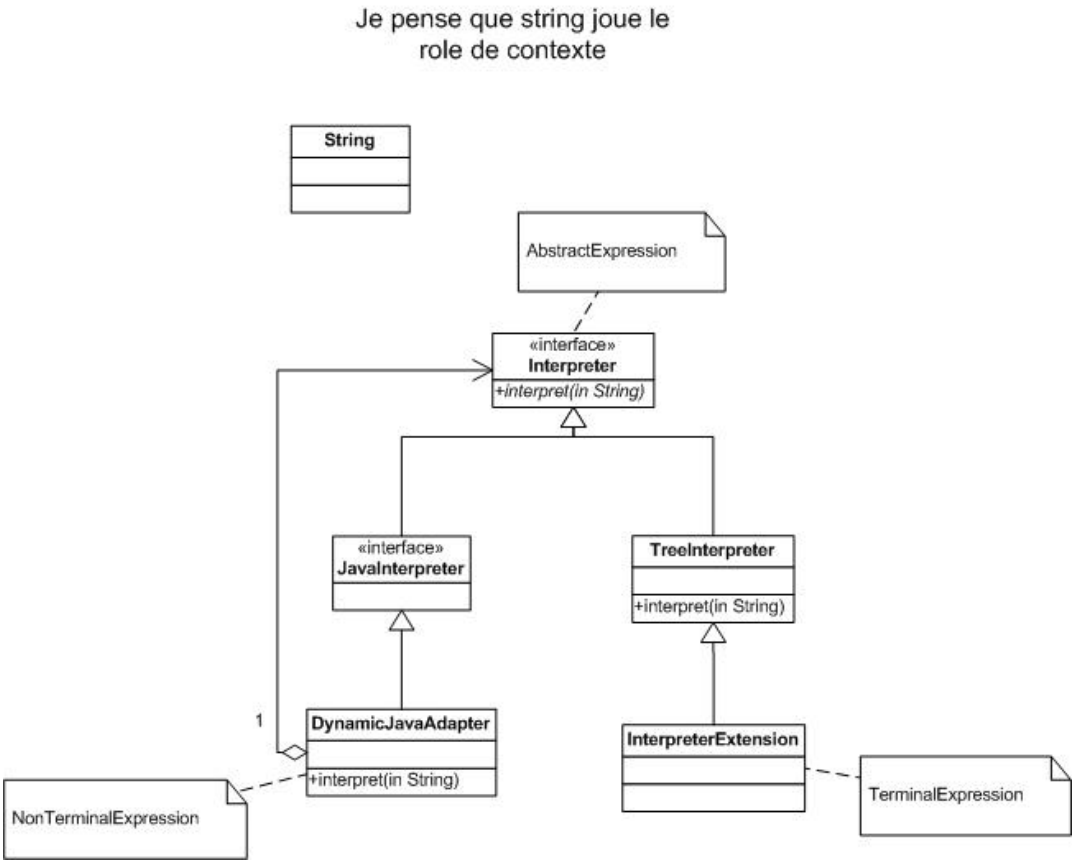


Illustration d'un diagramme de classes similaire au motif «Interpreter».

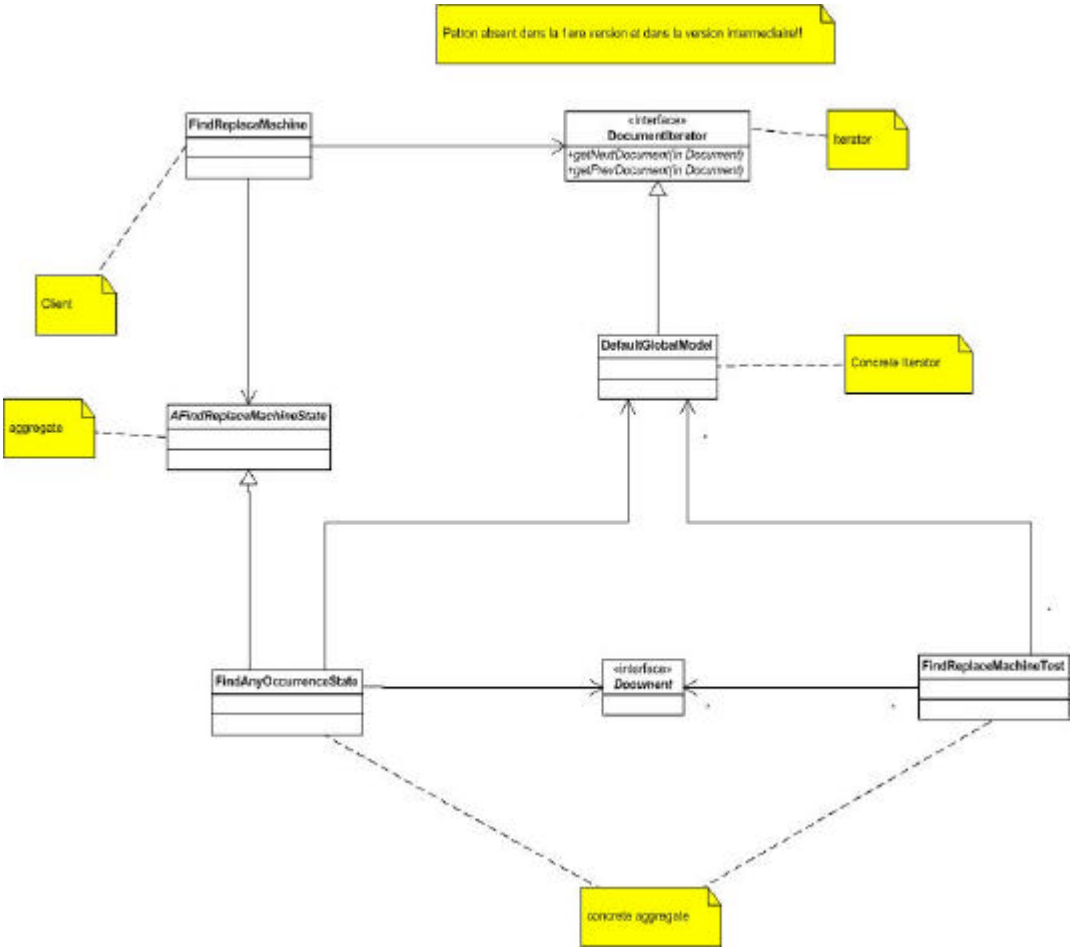


Illustration d'un diagramme de classes similaire au motif « Iterator ».

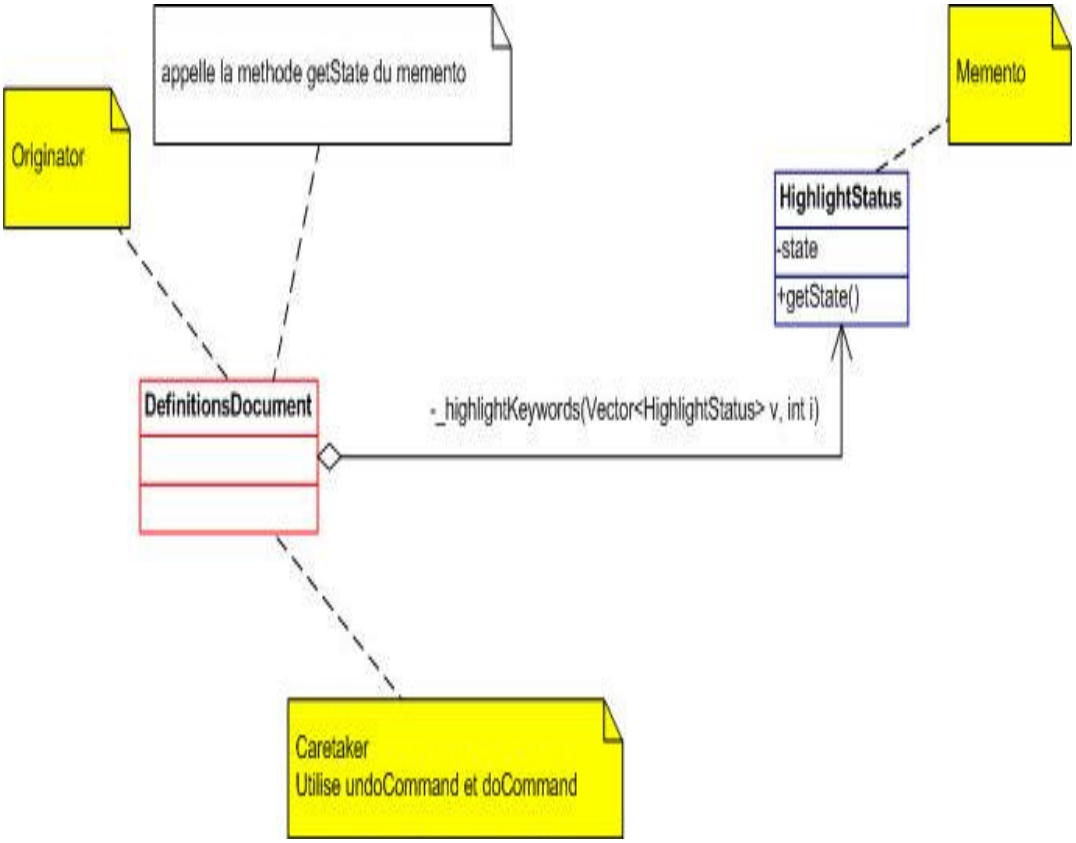


Illustration d'un diagramme de classes similaire au motif «Memento».

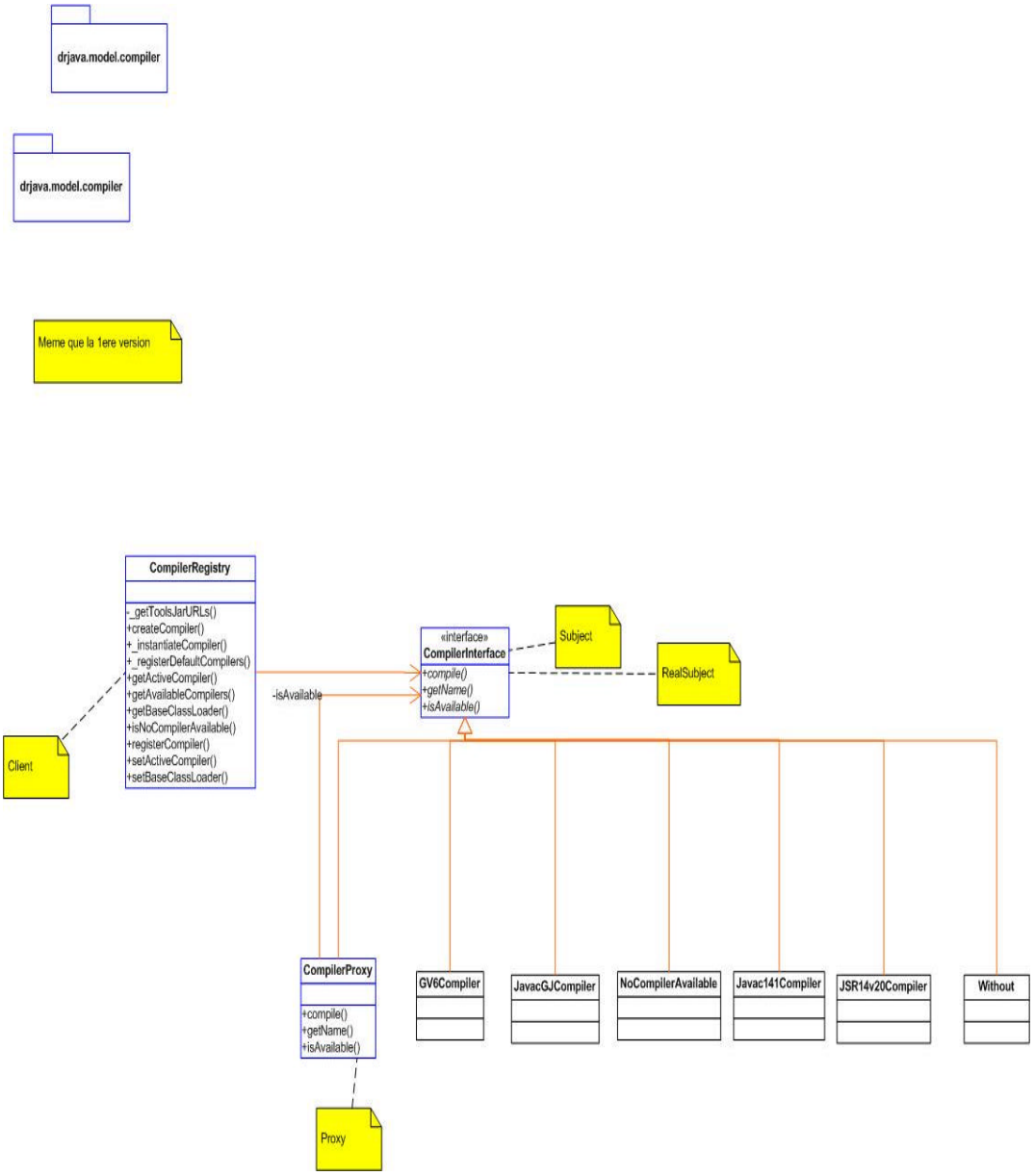


Illustration d'un diagramme de classes similaire au motif «Proxy ».

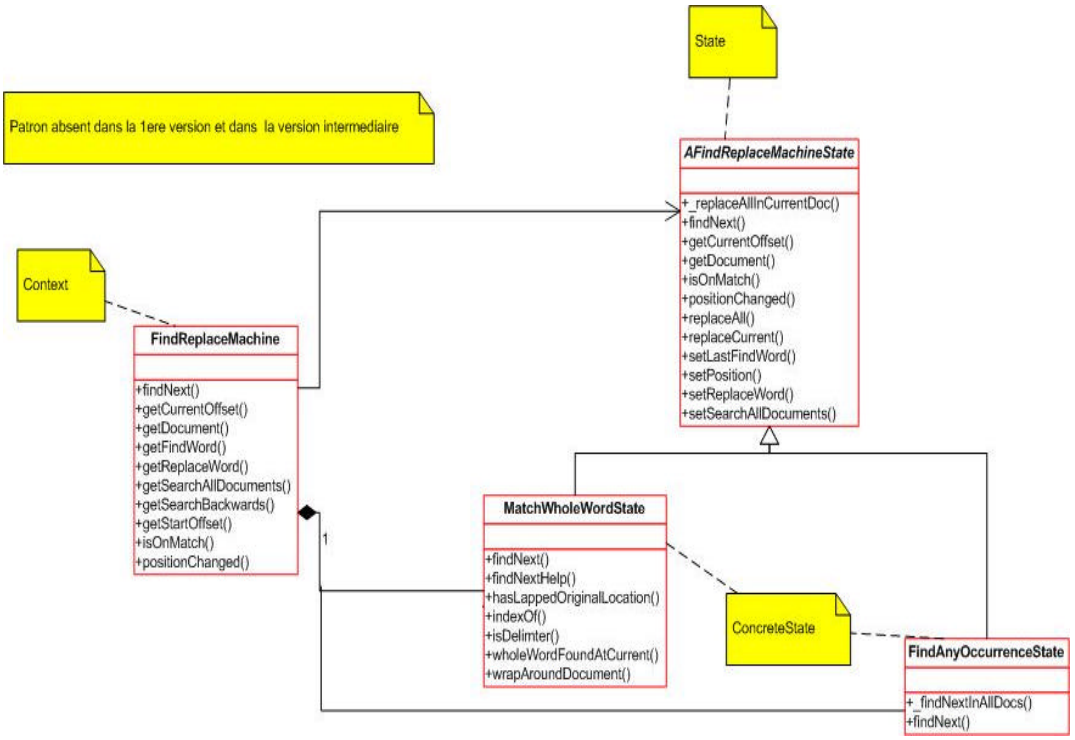


Illustration d'un diagramme de classes similaire au motif « State».

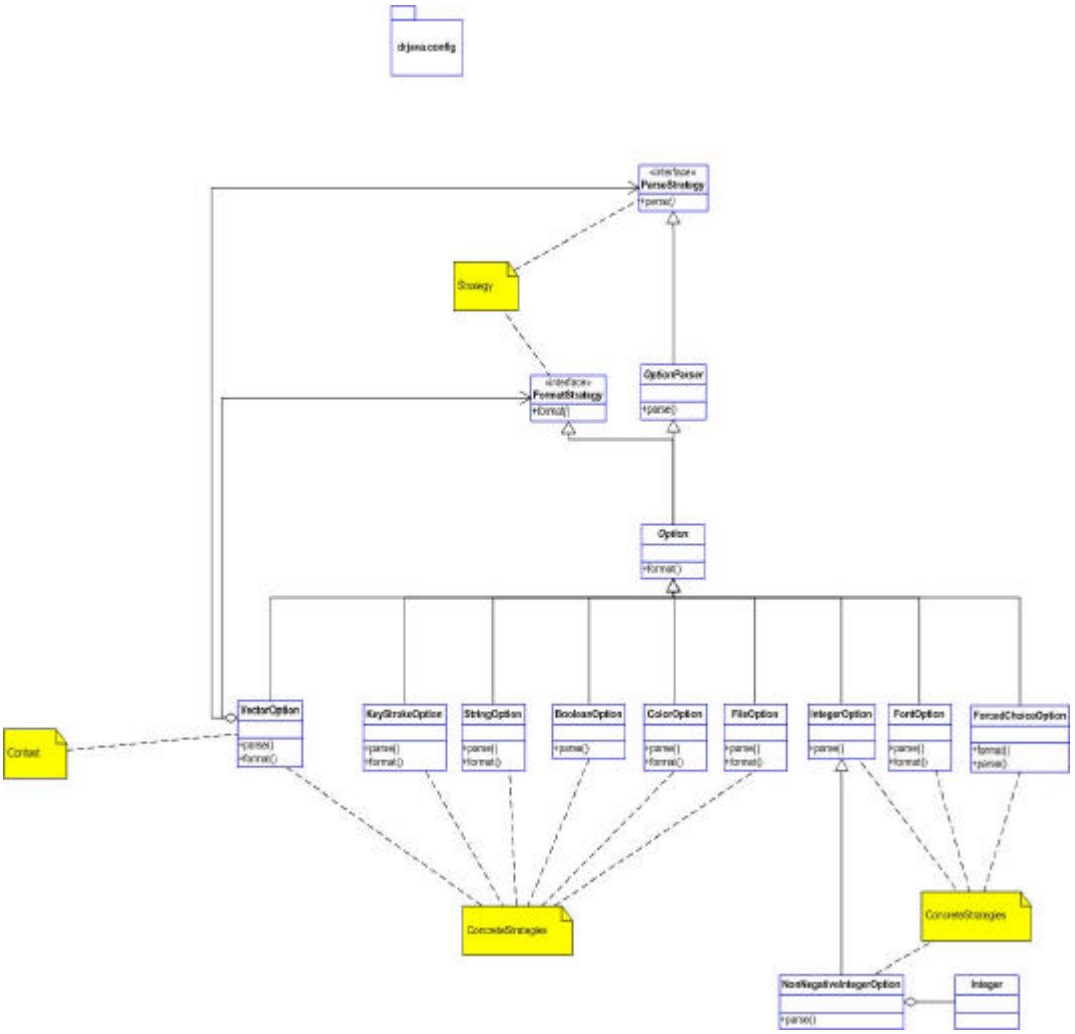


Illustration d'un diagramme de classes similaire au motif «Strategy».

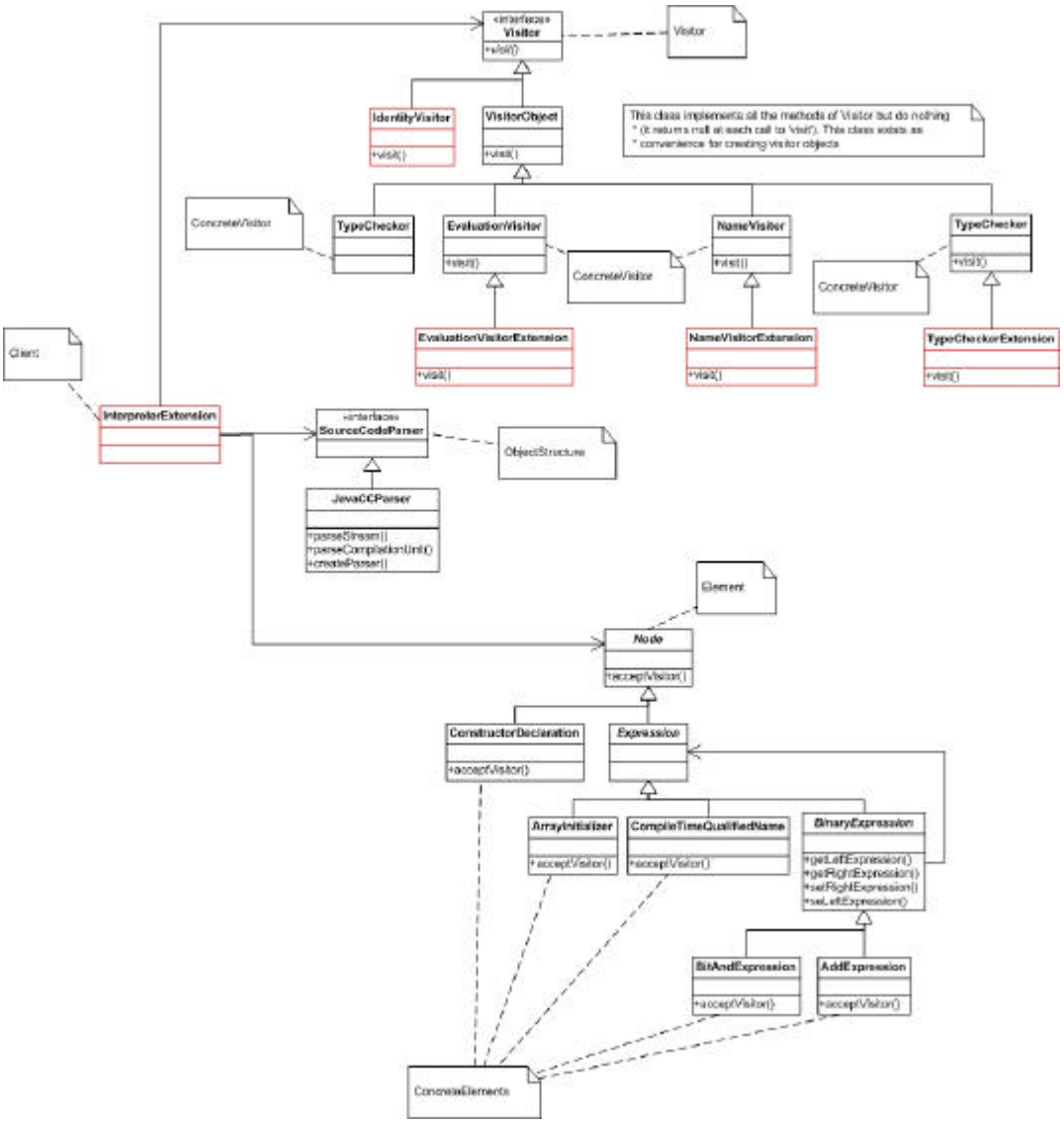


Illustration d'un diagramme de classes similaire au motif «Visitor ».

Bibliographie

- « *Design Patterns Element of Reusable Object Oriented Software* » de GAMA et AL.
- Éléments du cours IFT3903 : Qualité et métriques du logiciel du Département d'Informatique et de recherche Opérationnelle.