CHAPITRE 7

Assignation des responsabilités

Sommaire

- De l'analyse à la conception,
- Assignation des responsabilités,
- Les patrons d'analyse GRASP
  - Expert (*expert*),
  - Créateur (*creator*),
  - Couplage faible (*low coupling*),
  - Cohésion élevée (*high cohesion*),
  - Contrôleur (*controller*),
  - Polymorphisme (*polymorphism*)
  - Pure fabrication
  - Indirection
  - Variation protégée (*protected variation*)

De l'analyse à la conception

La phase d'analyse fournit une bonne compréhension des requis, des concepts et du comportement d'un système. Un ensemble minimal de documents fournissant un modèle d'analyse d'un système comprend par exemple:

- Un document de description des points de fonction,

ou

- Un document de description des cas d'utilisation,
- Un document de description du modèle conceptuel,
- Les diagrammes de séquence du système,
- Un document de haut niveau décrivant l'architecture.

Une fois ces documents assemblés, on passe immédiatement à une première phase de conception.

La conception

Passage à l'étape de conception:

→ Construire les diagrammes qui permettront de décrire les communications entre les objets, et

→ Leurs responsabilités respectives afin de remplir les requis.

Deux principaux types de diagrammes sont produits durant cette phase:

- Les diagrammes de classes,
- Les diagrammes d'interaction (séquence et/ou collaboration).

Cette phase nécessite principalement la connaissance des:

- Principes d'assignation de responsabilités,
- Patrons de conception.

La conception

Une façon très répandue d'aborder la conception d'objets, et même de composants logiciels de plus grande taille, consiste à considérer la conception en termes de:

- Responsabilités,
- Rôles, et
- Collaborations.

C'est l'approche de la « conception guidée par les responsabilités » (*responsability-driven design* ou RDD).

Identifier les responsabilités des classes

- Une responsabilité est une obligation ou un contrat assumé par la classe

En raffinant les responsabilités, on obtient les attributs et les opérations de la classe

On peut utiliser une 4ième boîte dans la classe pour décrire les Responsabilités, ou associer la responsabilité à la classe sous la forme d'une note.

Identifier les responsabilités des classes

Modèle (**M**)	Contrôle (**C**)	Vue (**V**)
Responsabilités: gérer l'état du modèle	Responsabilités: - Synchroniser les changements du modèle et de la vue	Responsabilités: -permettre la visualisation du modèle, -gérer le mouvement et le redimensionnement de la vue, -intercepter les évènements de l'usager.

Identifier les responsabilités des classes

Les responsabilités d'un objet sont reliées au comportement de cet objet; on distingue deux types principaux de responsabilités:

- Faire:
  - L'objet accomplit quelque chose lui-même,
  - L'objet initie une action d'un autre objet,
  - L'objet contrôle ou coordonne les activités d'autres objets,
- Savoir:
  - L'objet connaît les données privées qu'il encapsule,
  - L'objet connaît les objets auxquels il est lié,
  - L'objet connaît des données qu'il peut dériver ou calculer.

GRASP: des patrons pour l'assignation de responsabilités

- Assigner correctement les responsabilités aux classes est extrêmement important dans une conception orientée objet,
- La découverte des responsabilités se fait souvent durant la construction des diagrammes d'interaction,
- Les patrons fournissent des solutions à des problèmes courant de conception.

Les patrons GRASP (*General Responsability Assignment Software Patterns*) décrivent des principes fondamentaux d'assignation des responsabilités à des objets. Ils sont utilisés comme guide pour assigner correctement les responsabilités aux différentes classes lors du passage vers la conception.

GRASP: des patrons pour l'assignation de responsabilités

Neuf patrons d'assignation de responsabilités:

- **Expert**: donner la responsabilité à l'objet le mieux en mesure d'y satisfaire,
- **Créateur**: donner la responsabilité de créer un objet à un autre objet directement impliqué dans une relation avec l'objet créé,
- **Couplage faible**: assigner les responsabilités de façon à ce que le couplage reste faible,
- **Cohésion élevée**: assigner les responsabilités de façon à ce que la cohésion reste élevée,
- **Contrôleur**: assigner la responsabilité de gérer les messages d'évènements du système à une classe spécialisée du modèle,
- **Polymorphisme**: utiliser des interfaces pour traiter des alternatives qui dépendent du type des objets,
- **Indirection:** assigner des responsabilités à un objet intermédiaire pour éviter que des classes soient couplées,
- **Pure Fabrication:** assigner un ensemble de responsabilités à une classe créée artificiellement,
- **Variation protégée:** identifier des points d'instabilité ou de variation et développer des interfaces stables autour de ces points.

Le patron Expert

Intention

Assigner une responsabilité à l'expert, c'est-à-dire celui qui possède l'information nécessaire pour se charger de la responsabilité.

Applicabilité

Certainement le patron le plus utilisé en conception orientée objet. Il s'agit d'un principe fondamental d'assignation de responsabilités qui se retrouve dans tous les systèmes quels qu'ils soient. Dans bien des cas, la prise en charge d'une responsabilité requiert de l'information distribuée parmi plusieurs objets ou classes, ce qui donne lieu à plusieurs « experts partiels » qui collaborent pour remplir complètement la responsabilité.

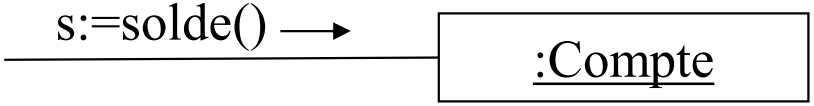
Le patron Expert

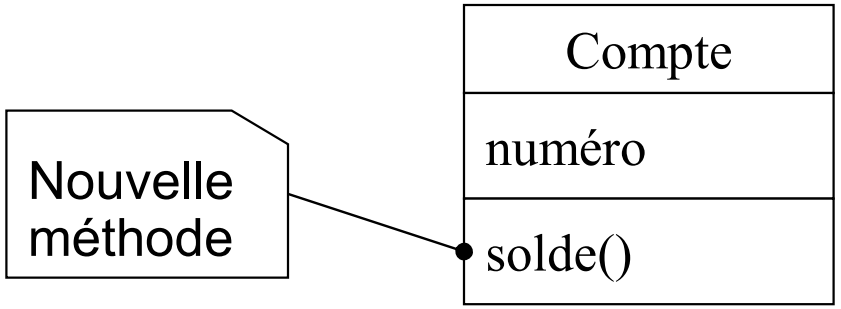
Conséquences

+ Favorise l'encapsulation, puisque les objets utilisent leur propre information pour satisfaire la responsabilité,
+ Supporte un **couplage faible** entre les classes, ce qui mène à des systèmes plus robustes et plus faciles à maintenir,
+ Distribue le comportement parmi les classes qui possèdent l'information nécessaire: ce qui encourage des définitions de classes plus cohérentes et plus faciles à comprendre.

Le patron Expert

Application du patron Expert pour déterminer la classe à laquelle on doit assigner la responsabilité de calculer le solde d'un compte: le patron Expert indique que la classe Compte serait l'expert.





Mais ce n'est pas fini…

Le patron Expert

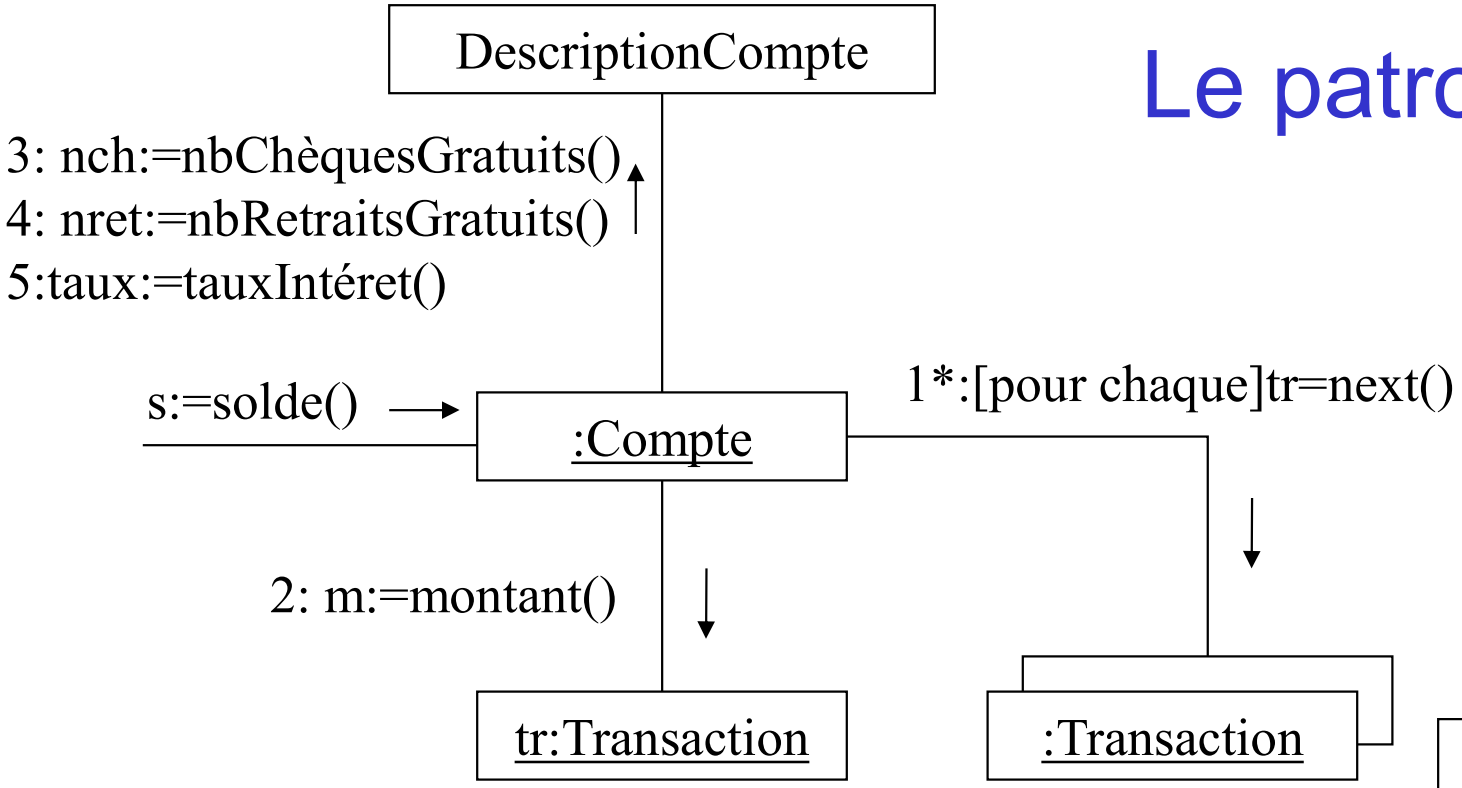
Le compte doit:

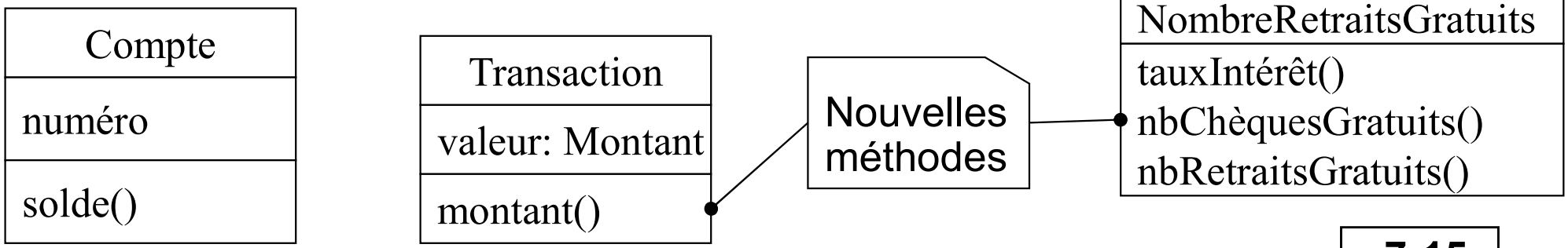
- récupérer les montants de toutes les transactions du mois courant,
- calculer les frais de gestion et l'intérêt à partir de la description du compte.

Puisque les transactions connaissent leur montant, le patron Expert implique que ce sont les transactions qui doivent fournir les montants des transactions.

De plus, la description du compte fournira les informations sur les frais de gestion et le taux d'intérêt.

Le patron Expert





Le patron Expert

Pour satisfaire la responsabilité de connaître et de retourner le solde d'un compte, il faut assigner les responsabilités suivantes:

Classe	Responsabilité
Compte	Connaît son solde
Transaction	Connaît son montant
DescriptionCompte	Connaît la structure tarifaire pour les frais de gestion, Connaît le taux d'intérêt

Le patron Créateur

Intention

Assigner la responsabilité de créer une ou des instances d'objets d'une classe A à une classe B si B contient, enregistre, utilise directement, ou a les données pour initialiser A; ou si A est un agrégat de B.

Applicabilité

Le patron Créateur guide dans l'assignation de la responsabilité de création d'objets, une tâche très commune dans les systèmes orientés-objets. L'idée de base du patron Créateur consiste à trouver une classe qui a déjà d'excellentes raisons d'être associée à la classe à créer.

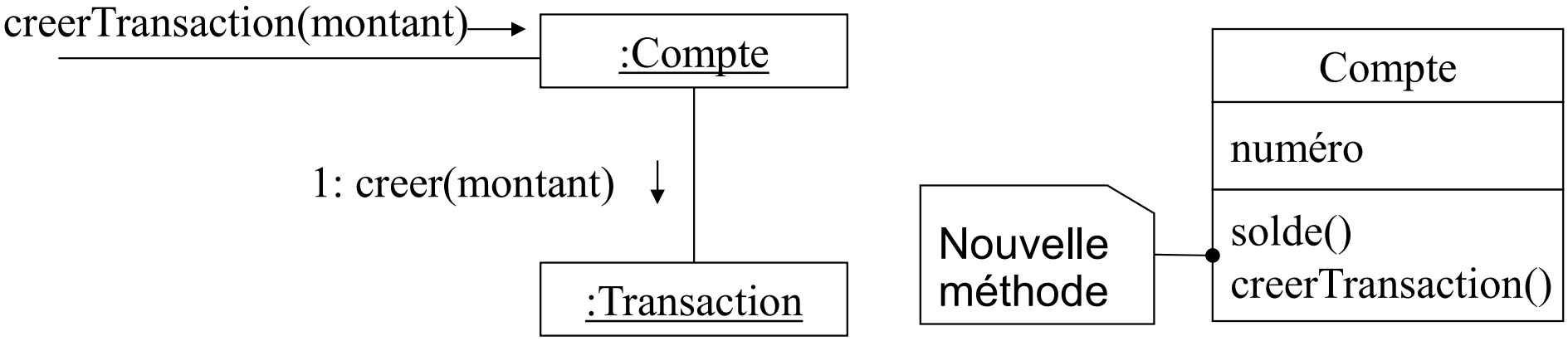
Le patron Créateur

Conséquences

+ Supporte un **couplage faible** entre les classes, ce qui mène à moins de dépendances et plus de possibilités de réutilisation. Le couplage n'est probablement pas augmenté, puisque le Créateur a déjà d'excellentes raisons d'être connecté à la classe des objets qu'il crée.

Le patron Créateur

Application du patron Créateur pour déterminer la classe à laquelle on doit assigner la responsabilité de créer les objets Transaction: le patron Créateur suggère de chercher une classe qui contient, enregistre, etc. des transactions. Puisque Compte enregistre des transactions, Compte pourrait être un bon candidat:



Le patron Faible couplage

Intention

Assigner les responsabilités de façon à s'assurer que le couplage entre les classes, c'est-à-dire le nombre total d'associations reliant les classes entre elles, reste faible.

Applicabilité

Le couplage est une mesure du **niveau de connexion d'une classe avec les autres classes du système**. Une classe ayant un Faible couplage n'est pas connectée à un « trop » grand nombre d'autres classes. Le Faible couplage est un principe de conception qui doit toujours être présent à l'esprit des concepteurs, il s'agit d'un patron d'évaluation d'une conception.

Le patron Faible couplage

Conséquences

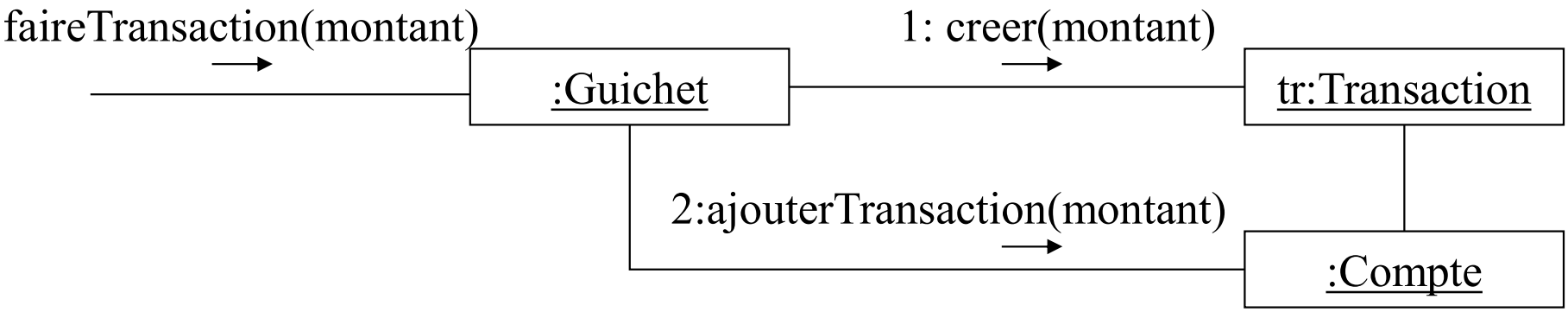
+ Le Faible couplage mène à la conception de classes qui sont plus indépendantes, ce qui réduit l'impact de changements, et qui sont plus réutilisables, ce qui améliore les possibilités de gains de productivité.
– Le couplage ne peut pas être considéré de façon isolée, mais doit être évalué en même temps que d'autres principes comme Expert et Forte Cohésion dans l'évaluation d'une conception,
– Le couplage n'est peut-être pas important si la réutilisation n'est pas un objectif.
– Le cas extrême du Faible couplage, où une classe n'est connectée à pratiquement aucune classe n'est pas désirable, puisqu'il mène à des classes ayant une très faible cohésion, qui assume toutes les responsabilités à l'interne.

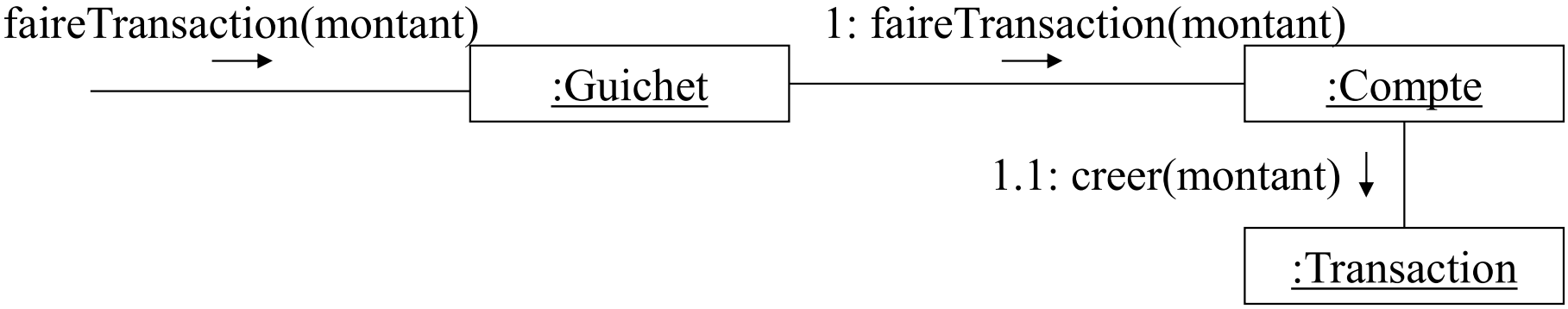
Le patron Faible couplage

Application du patron Faible couplage pour décider laquelle des classes Guichet ou Compte devrait créer les objets Transaction:

- si l'on suppose qu'un Compte devra, d'une manière ou d'une autre, être connecté à une Transaction, ça ne sert à rien d'augmenter le couplage entre les classes en assignant la responsabilité de créer une Transaction à la classe Guichet .

Le patron Faible couplage





La 2ième solution a un couplage plus faible.

Le patron Cohésion élevée

Intention

Assigner les responsabilités de façon à s'assurer que la cohésion de chaque classe reste élevée.

Applicabilité

La cohésion est une mesure du **niveau de lien** qui existe entre les **différentes fonctions d'une classe**. Si toutes les responsabilités d'une classe sont reliées les unes aux autres, la cohésion de la classe est forte. La Cohésion élevée est un principe de conception qui doit toujours être présent à l'esprit des concepteurs, il s'agit d'un patron d'évaluation d'une conception.

Le patron Cohésion élevée

Conséquences

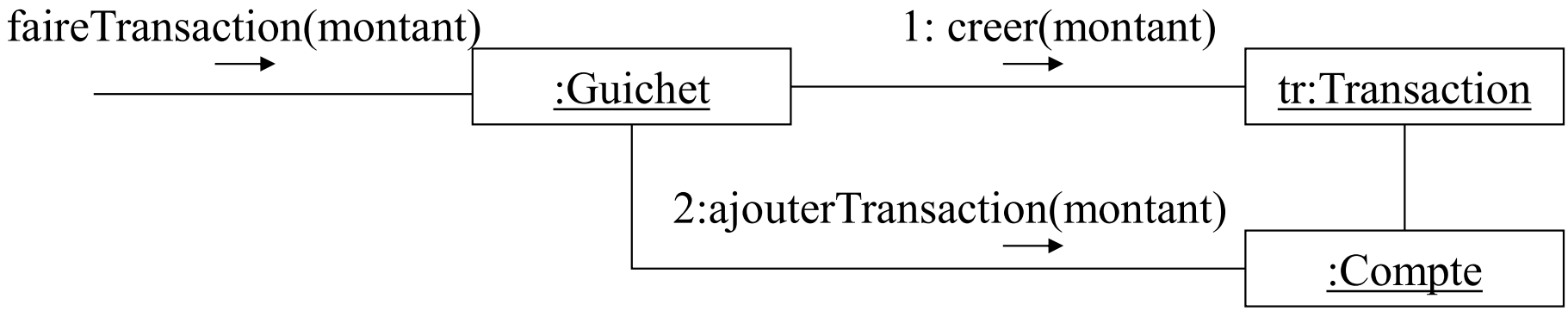
+ La cohésion élevée d'une classe augmente la clarté et la facilité de compréhension de la classe,
+ La cohésion élevée facilite la maintenance et les améliorations,
+ Aide à maintenir le couplage bas,
+ La granularité fine de l'assignation des responsabilités augmente la possibilité de réutilisation, étant donné qu'une classe ayant un haut niveau de cohésion peut-être utilisée dans un but bien précis.

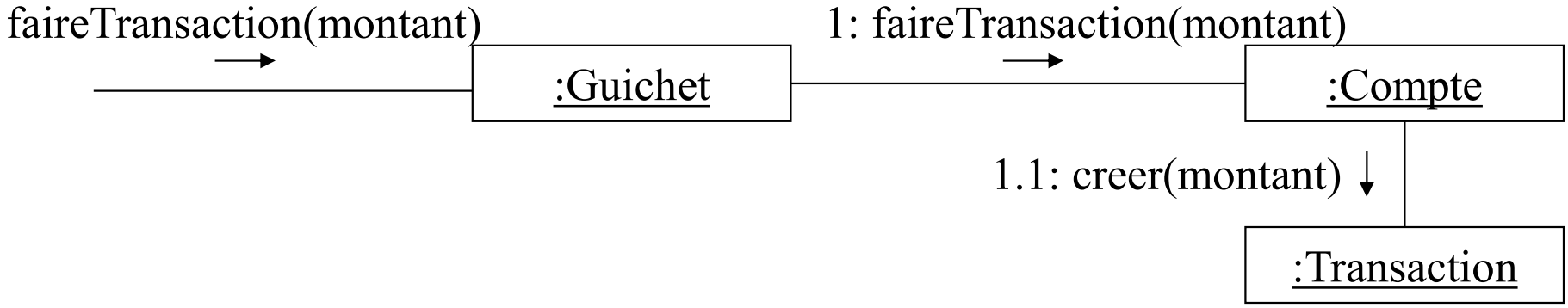
Le patron Cohésion élevée

Application du patron Cohésion élevée pour décider laquelle des classes Guichet ou Compte devrait créer les objets Transaction:

- La même analyse, faite dans le contexte du Faible couplage peut aussi s'appliquer à la Cohésion élevée. Le fait d'impliquer la classe Guichet dans le traitement des transactions diminue la cohésion de la classe Guichet. Isolément, cette responsabilité peut être acceptable, mais si la classe Guichet doit assumer de plus en plus de responsabilités pour faire de plus en plus de choses, elle risque de s'écraser sous le poids des responsabilités.

Le patron Cohésion élevée





La 2ième solution augmente la cohésion de la classe Guichet

Le patron Contrôleur

Intention

Assigner la responsabilité de traiter les événements à une classe spécialisée.

Applicabilité

Le patron Contrôleur s'applique dans pratiquement tous les systèmes orientés objet qui doivent traiter des événements externes impliquant par exemple un usager ou des systèmes logiciels ou physiques externes (senseurs, commutateurs, etc.). Dans tous les cas où une conception orientée objet est utilisée, une classe Contrôleur doit être choisie pour traiter les événements. Le patron Contrôleur constitue un guide pour choisir une classe acceptable.

Le patron Contrôleur

Conséquences

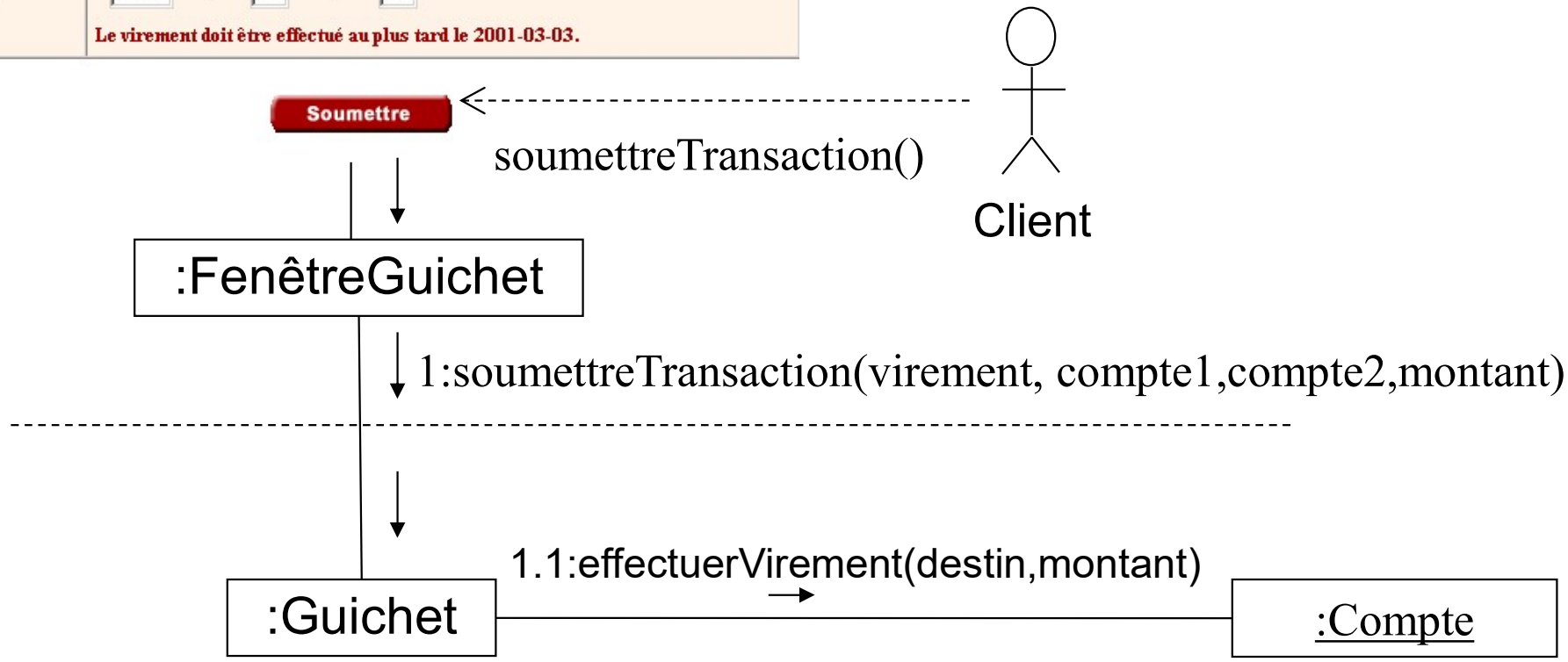
+ Centralise la gestion des événements dans une classe bien identifiée,
+ Augmente les possibilités de réutilisation en s'assurant que les processus liés au domaine d'application sont traités au niveau de la couche du modèle et non de la couche de présentation,
+ Permet de s'assurer de la cohérence séquentielle des événements en fournissant une classe unique pour traiter les événements, qui peut s'assurer qu'ils se produisent dans le bon ordre.
– Il est facile d'assigner trop de responsabilités au Contrôleur, ce qui mène à une classe ayant peu de cohésion, et difficile à comprendre,
– Lorsqu'un Contrôleur assume trop de responsabilités, on peut le décharger en créant des Contrôleurs spécialisés, ou en s'assurant que le Contrôleur délègue la gestion de chaque événement à un objet spécialisé.

Le patron Contrôleur

Effectuez un virement

Du compte	COMPTE ACCELERATION - *****00*6388
Au compte	VISA CLASSIQUE - 4537*****8100064
Montant $	
Date	aaaa 2001 - mm 01 - jj 22 Le virement doit être effectué au plus tard le 2001-03-03.

La classe Guichet agit comme Contrôleur au niveau du domaine.



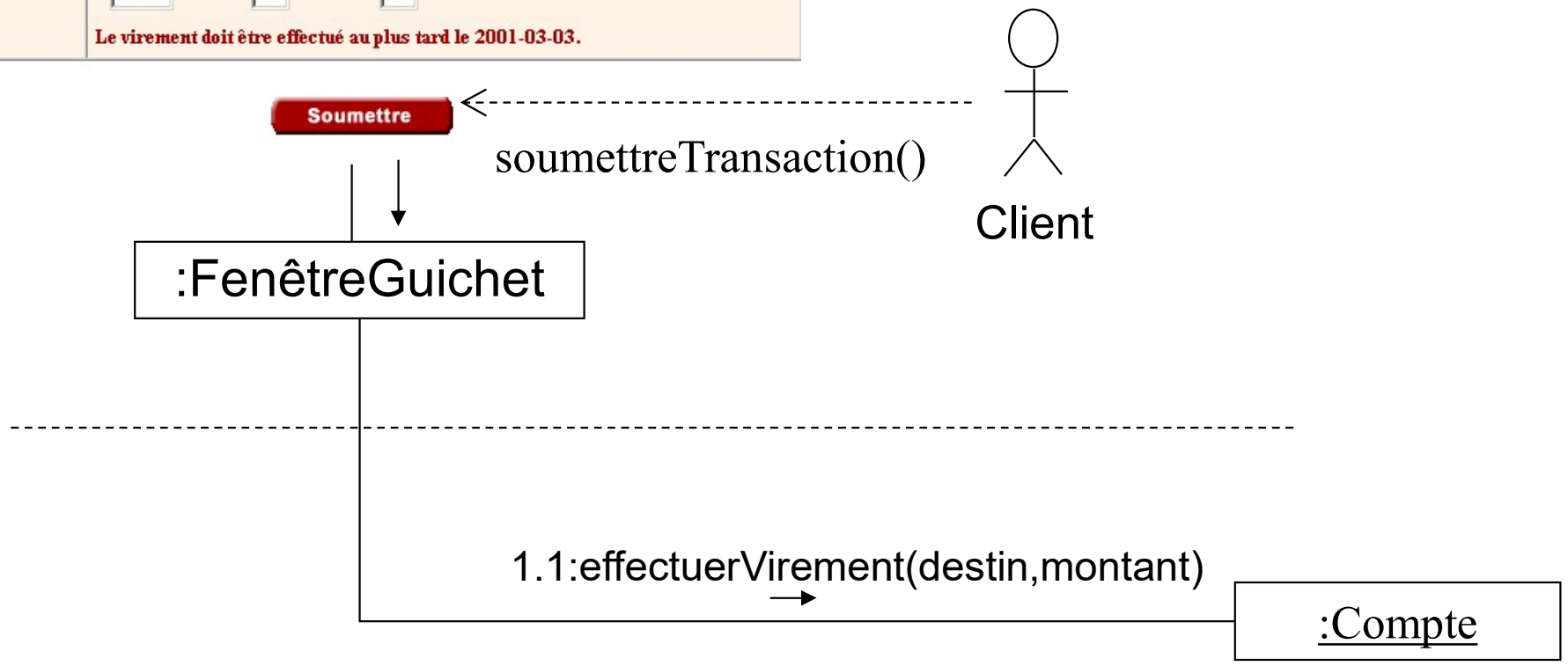
Le patron Contrôleur

Effectuez un virement

Du compte	COMPTE ACCELERATION - *****00*6388
Au compte	VISA CLASSIQUE - 4537*****8100064
Montant $	
Date	aaaa 2001 - mm 01 - jj 22

Le virement doit être effectué au plus tard le 2001-03-03.

La classe FenêtreGuichet ne devrait pas traiter directement un processus lié au domaine d'application.



Le patron Contrôleur

Dans des cas plus complexes, la définition d'un seul contrôleur pour tout le système peut mener à une classe Contrôleur très complexe est ayant une faible cohésion.

On a un Contrôleur trop complexe si:

- Tous les événements du système sont traités par une seule classe, et qu'il y a un très grand nombre d'événements,
- Le Contrôleur fait lui-même un grand nombre des tâches liées au traitement des événements, sans déléguer ces tâches à d'autres objets,
- Le Contrôleur a un grand nombre d'attributs et conserve une portion importante des données qui pourraient être distribuées à d'autres objets.

Le patron Contrôleur

Deux approches sont envisageables pour solutionner le problème d'un Contrôleur trop complexe:

1. Déléguer le traitement de certains événements à d'autres contrôleurs: patron « Chain of responsability »,
2. Traiter chaque événement système comme un objet autonome: patron « Command »

Le patron Polymorphisme

Intention

Traiter des alternatives basées sur le type ou rendre des composants logiciels remplaçables en introduisant une interface qui permet de faire varier le comportement de façon polymorphique.

Applicabilité

Un autre principe fondamental de l'approche orientée-objet.

Dès que le comportement d'objets reliés les uns aux autres doit varier en fonction du type concret de l'objet.

Lorsqu'il serait nécessaire de tester le type concret de l'objet et d'utiliser des énoncés conditionnels ou des énoncés « switch/case ».

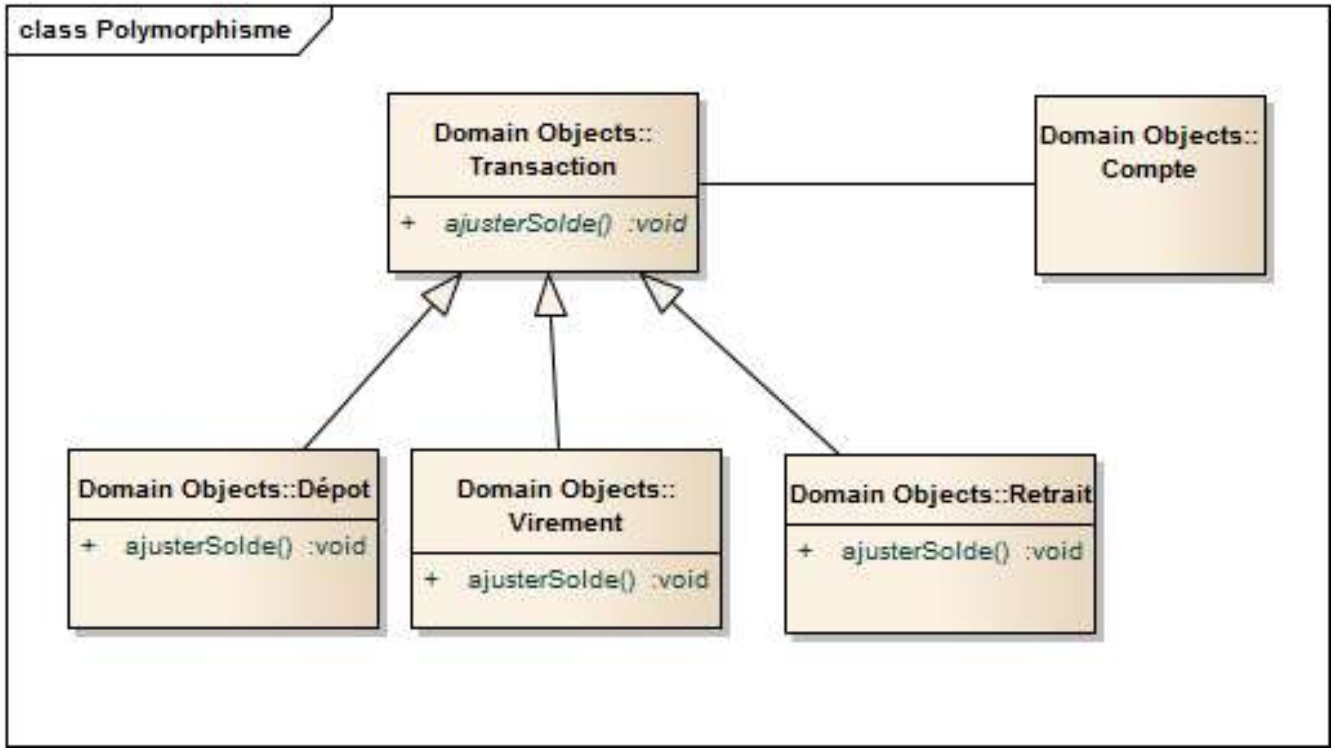
Le patron Polymorphisme

Conséquences

+ Permet d'ajouter facilement les extensions requises pour de nouvelles variations.
+ De nouvelles implémentations peuvent être ajoutées sans affecter les clients.
– Permet d'ajouter des interfaces ou des points de variation en prévision de développements futurs qui ne se réaliseront peut-être pas.
– Ajouter seulement les points de variation pour des alternatives qui sont confirmées par la conception du système et éviter d'ajouter des points de variation superflus.

Le patron Polymorphisme

Comment traiter l'ajustement du solde d'un compte suite à une transaction ?



Rendre la méthode d'ajustement du solde polymorphique dans la classe transaction.

Le patron Indirection

Intention

Assigner des responsabilités à un objet intermédiaire afin d'éviter que des classes soient directement couplées.

Applicabilité

Vieil adage:

« La plupart des problèmes en informatique peuvent être solutionnés par un niveau supplémentaire d'indirection. »

Et son corolaire:

« La plupart des problèmes de performance peuvent être solutionnés en éliminant une couche d'indirection. »

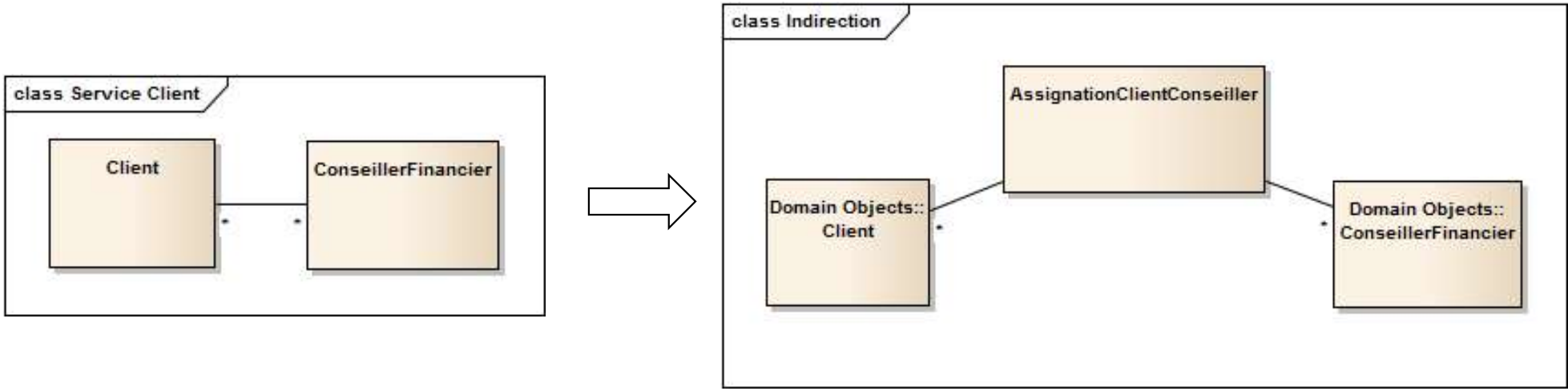
Le patron Indirection

Conséquences

+ Permet de réduire le couplage entre des classes ou des composants,
– Augmente le nombre de classes,
– Réduit la performance.

Le patron Indirection

Comment permettre à un ConseillerFinancier d'avoir plusieurs Clients et vice-versa ?



Introduire une nouvelle classe ajoutant une indirection.

Le patron Pure Fabrication

Intention

Assigner un ensemble de responsabilités hautement cohésives à une classe créée artificiellement, qui n'est pas tirée de l'ensemble des concepts du domaine.

Applicabilité

Lorsque l'on veut regrouper dans une classe un ensemble de méthodes qui sont liées entre-elles, mais qui ne correspondent pas directement à un concept du domaine d'application.

Lorsqu'un ensemble de comportements doivent être extrait d'une classe et regroupé afin de maintenir une cohésion élevée.

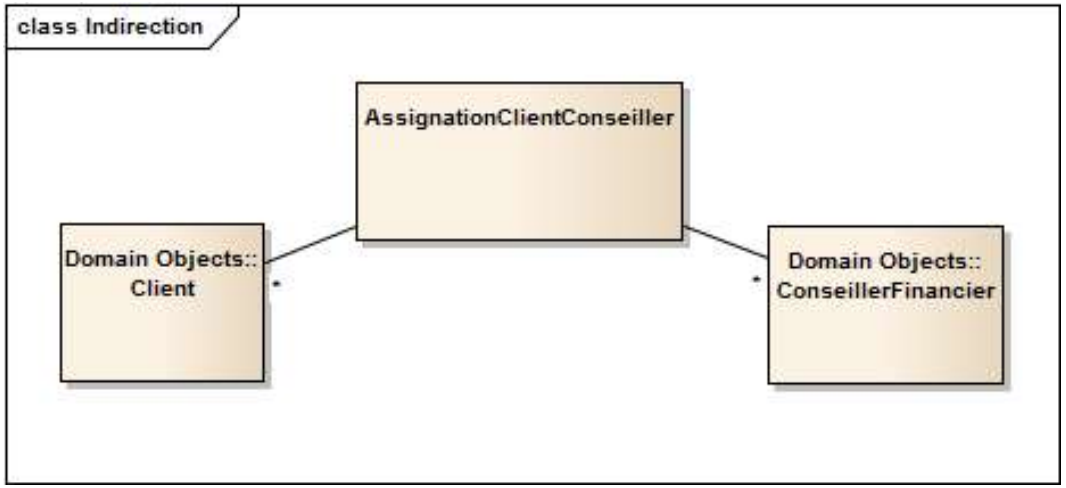
Le patron Pure Fabrication

Conséquences

+ Permet de factoriser dans une nouvelle classe un ensemble de comportements qui méritent d'être regroupés,
+ Permet de maintenir une cohésion élevée de la classe d'où sont extraits les comportements et de la nouvelle classe créée,
– Créer trop de classes strictement basées sur le comportement tend vers une décomposition plus fonctionnelle qu'orientée-objet,
– Une surabondance de classes fabriquées tend à éloigner les données et les comportements qui leur sont associés, ce qui contredit le patron Expert et affecte négativement le couplage.

Le patron Pure Fabrication

La classe AssignationClientConseiller ne fait pas partie du domaine du problème. C'est une pure fabrication.



Le patron Variation Protégée

Intention

Identifier des points d'instabilité ou de variation et développer des interfaces stables autour de ces points.

Applicabilité

Il s'agit d'un principe de conception fondamental, très important dans n'importe quel contexte de résolution de problème.

Principe de base derrière des principes tels que l'encapsulation des données, les interfaces, le polymorphisme et les indirections.

Le patron Variation Protégée

Conséquences

+ Simplifie l'ajout d'extensions nécessaires pour de nouvelles variations,
+ De nouvelles implémentations peuvent être ajoutées sans affecter les clients,
+ Réduit le couplage,
+ Permet de réduire l'impact ou le coût de changements,
– Favorise l'ajout de points d'évolution pour des alternatives qui ne se concrétiseront peut-être pas,
– Favorise l'ajout de flexibilité non requise.