

Quelques explications pour les patrons : l'outil PTIDEJ

Yann-Gaël Guéhéneuc et Narendra Jussien
École des Mines de Nantes
Département Informatique



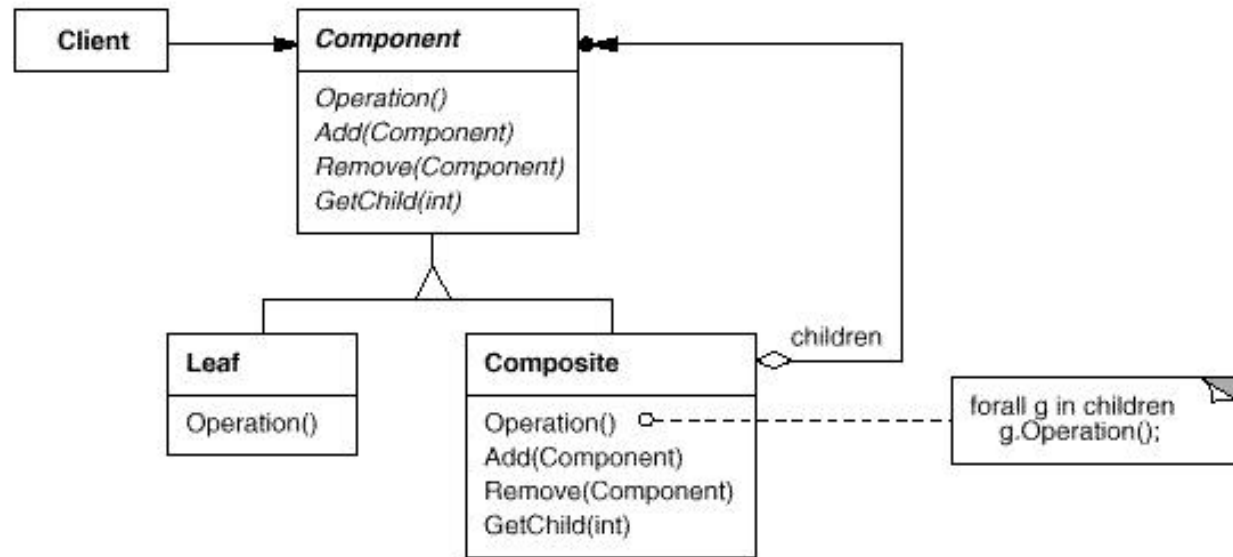
Plan

- But du travail
- Les patrons de conceptions (design-patterns)
- Le CSP à résoudre
- L'apport des explications
- Conclusion

Objectif

- Améliorer la qualité des logiciels
 - *uniquement aspects architecturaux*
 - *amélioration a posteriori*
- Outil privilégié :
patrons de conception (*Design patterns*)
 - Compilation de l'expérience des développeurs
 - Problèmes récurrents d'architecture
 - Une **bonne** solution

Un exemple : composite

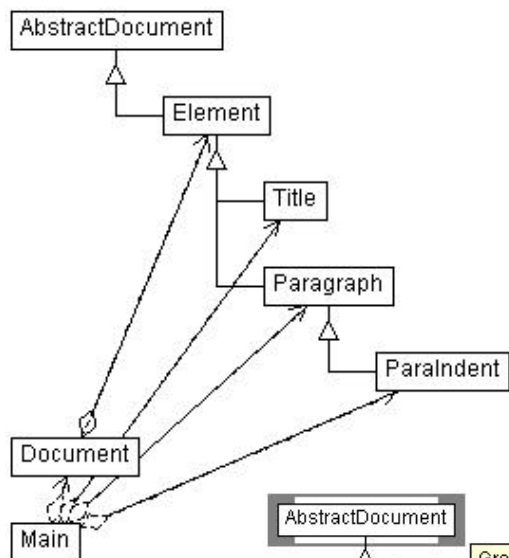


- Composition d'objets dans une structure arborescente pour un traitement uniforme des objets et des compositions

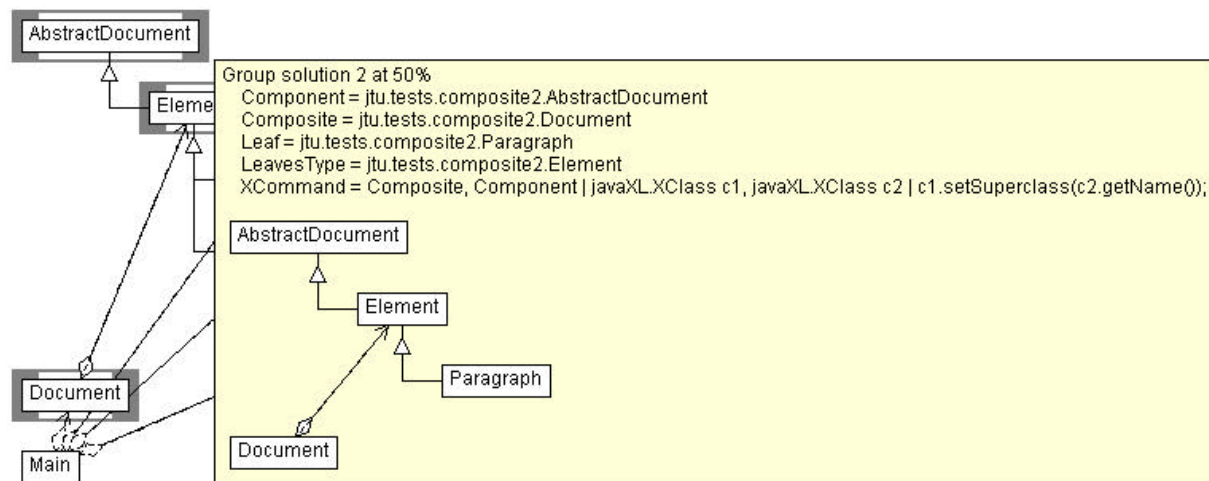
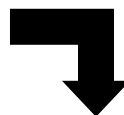
Le problème

- Identifier les utilisations d'un patron
 - *montrer que le code est bien écrit*
 - *notion de solution classique*
- Identifier les quasi-utilisations d'un patron
 - *proposer des améliorations du code analysé*
 - *notion de **solution dégradée** d'un problème donné*
- Modéliser le patron sous forme de CSP
- Trouver solutions et solutions dégradées
 - *interaction avec l'utilisateur*

Exemple



Recherche de solution



Caractéristiques du problème

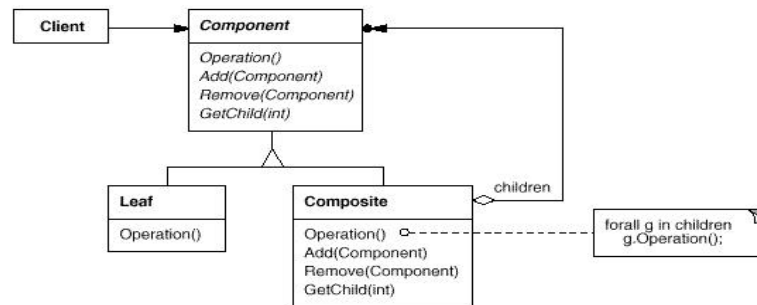
- Pas de description *a priori* des « dégradations »
 - *on ne veut/peut pas penser à tout*
- Capacité de justifications
 - *programmation : activité « artistique »*
 - *délicat de modifier du code sans expliquer pourquoi*
- Interaction forte avec l'utilisateur
 - *description dynamique des dégradations*
- Une solution : utilisation de la ePPC

Le CSP résolu

- Le CSP est déduit de deux informations
 - *Le patron de conception*
 - Les **variables** sont les classes actrices
 - Les **contraintes** les relations voulues entre les classes
 - *Le code à analyser*
 - Les **domaines** des variables sont les classes du logiciel
 - La **sémantique des contraintes** dérive des relations effectives entre les classes du logiciel

Exemple : contraintes

■ Le patron de conception



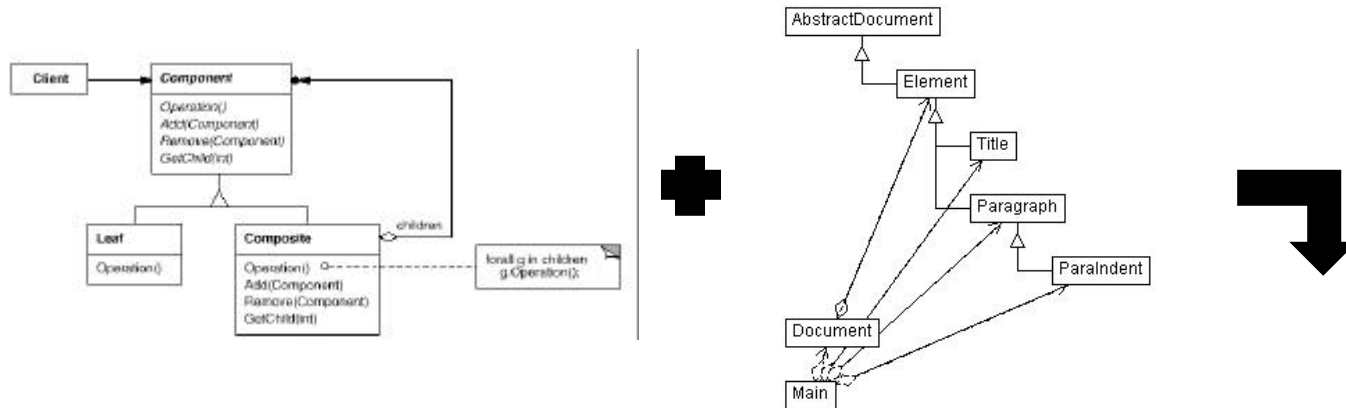
■ Les contraintes

`composite < component`

`leaf < component`

`composite \supset component`

Exemple : résolution



```
CPTidej 0.5
Setting constraints for naive Composite pattern...

There is no more solution because of:
  1. "Composite <>-- Component"
    (weight 90)
  2. "Leaf -!>- Component"
    (weight 90)
  3. "Composite -!>- Component"
    (weight 90)
Which one do you want to relax? (0 -> none) 3
Solution :
  (Leaf:"jtu.tests.composite2.Paragraph", Composite:"jtu.tests.composite2.Document", Component:"jtu
.tests.composite2.Element")
Do you want another solution? (y/n) y
Solution :
  (Leaf:"jtu.tests.composite2.Title", Composite:"jtu.tests.composite2.Document", Component:"jtu.tes
ts.composite2.Element")
Do you want another solution? (y/n) y
Solution :
  (Leaf:"jtu.tests.composite2.ParaIndent", Composite:"jtu.tests.composite2.Document", Component:"jt
u.tests.composite2.Paragraph")
Do you want another solution? (y/n)
```

Implémentation : le système PTIDEJ

- Développé en PaLM
- Bibliothèque de contraintes
 - *héritage, création, composition, connaissance, ...*
 - *méta-contraintes : écriture de nouvelles contraintes*
- Bibliothèque de patrons de conception
 - *Composite, Facade, Mediator, AbstractFactory, Observer, Singleton, ...*
- Expérimentations
 - *Petits exemples de code*
 - *framework JHotDraw (~100 classes)*
 - *PTITDEJ lui-même !!! (en cours)*



Conclusion et perspectives

- Une application des explications
 - *application de type débogage/aide à la décision et problèmes dynamiques*
- Perspectives à court terme
 - *ajout de nouveaux patrons*
 - *ajout de nouvelles contraintes (globales)*
 - *autres tests*
- Perspectives à moyen/long terme
 - *prise en compte d'informations à l'exécution*
 - *travail sur la sémantique des liens entre classes*
 - certaines relations sont implicites