# Un cadre pour la traçabilité des motifs de conception

Soutenance de thèse de doctorat

Yann-Gaël Guéhéneuc

École des Mines
de Nantes, France

Object Technology
International, Inc., Canada

# Plan

- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l’architecture d’un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Plan

- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l'architecture d'un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Contexte (1/2)

- Maintenance des programmes à objets
  - Rétro-conception
  - Compréhension
  - Traçabilité
  - Modification

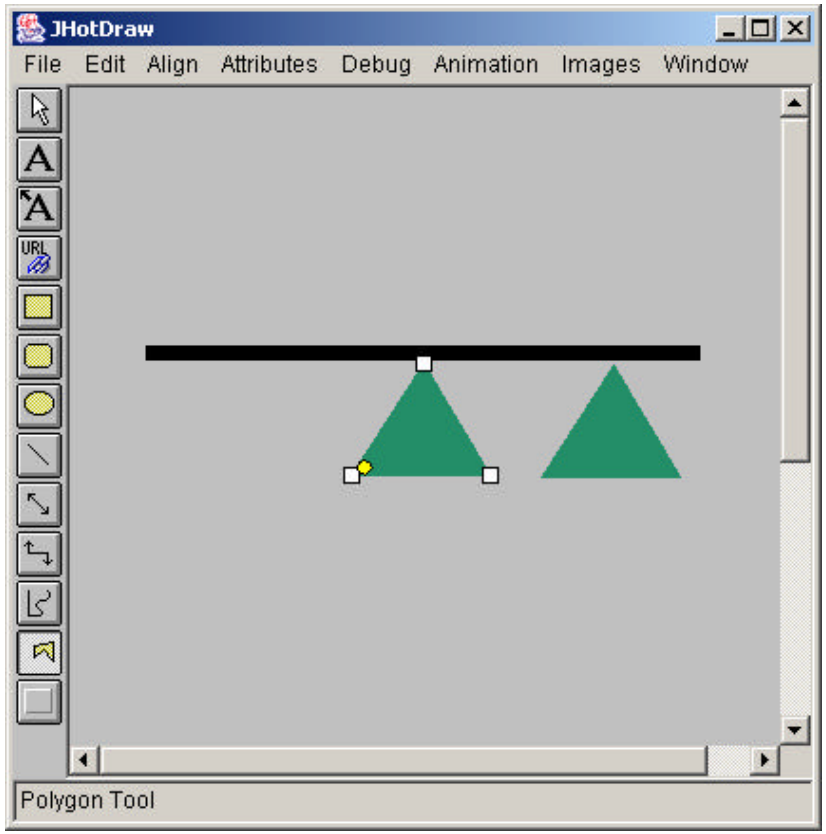
- Coûts humains, temporels, financiers prépondérants [Sharon96, Takang96, Pressman01]

# Contexte (2/2)

- Absence des développeurs originels
  - Roulement
- Absence de documentation
  - Implémentation
  - Architecture
  - Choix de conception

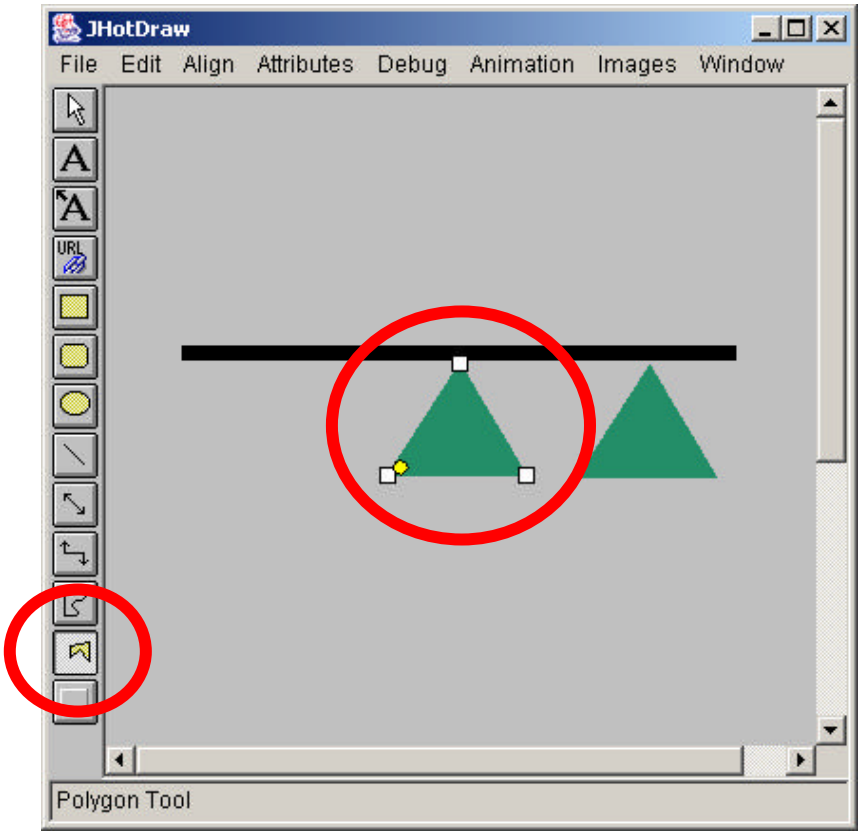
# Scénario (1/8)

- *JHotDraw* [Gamma98]
  - Programme de dessin vectoriel
  - 14 578 lignes de code Java
  - Figures
    - Carré, arrondi
    - Cercle, ellipse
    - Ligne, polygone
    - Connexion, dessin à main levée



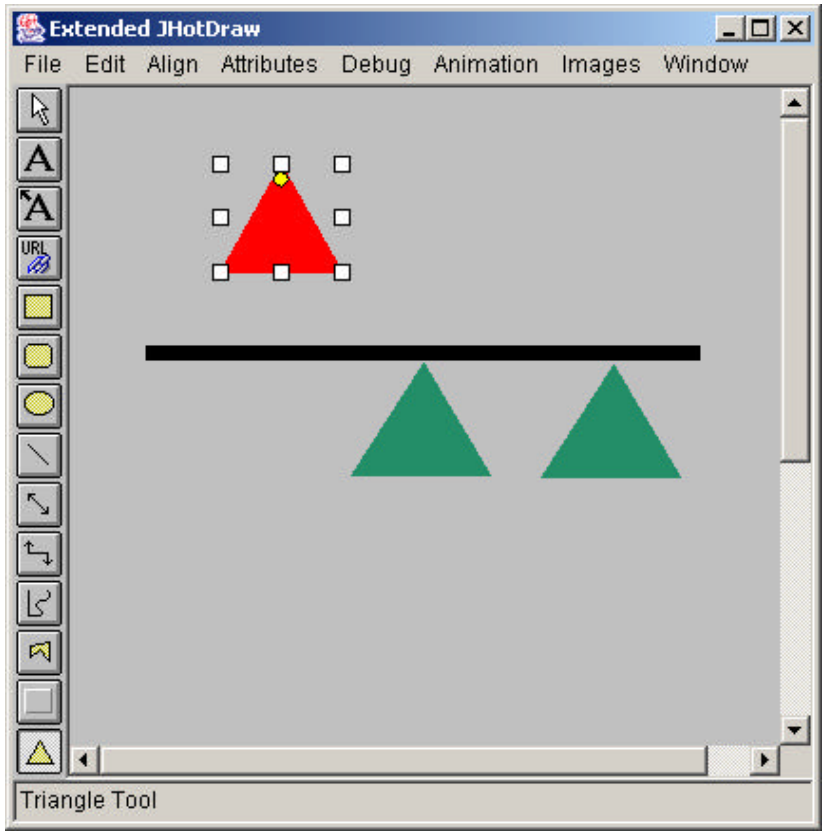
# Scénario (1/8)

- *JHotDraw* [Gamma98]
  - Programme de dessin vectoriel
  - 14 578 lignes de code Java
  - Figures
    - Carré, arrondi
    - Cercle, ellipse
    - Ligne, polygone
    - Connexion, dessin à main levée



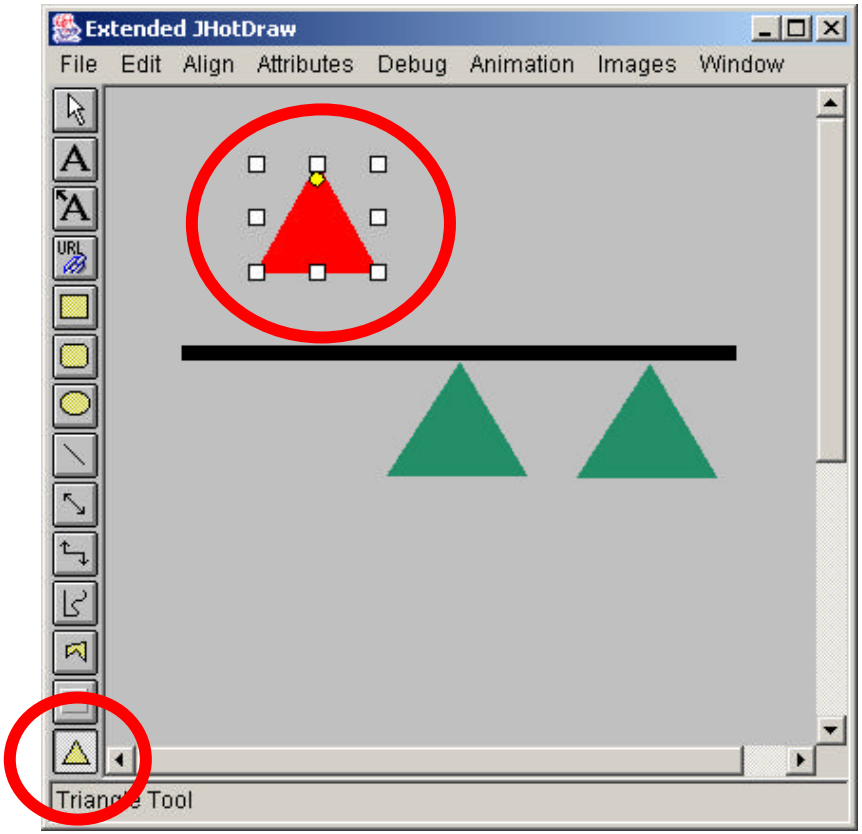
# Scénario (2/8)

- Ajout de fonctionnalité
  - La forme triangle
  - Manipulation identique aux autres formes



# Scénario (2/8)

- Ajout de fonctionnalité
  - La forme triangle
  - Manipulation identique aux autres formes



# Scénario (3/8)

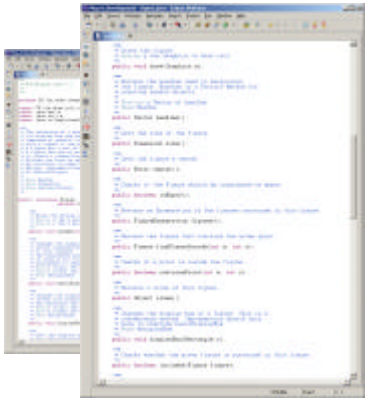
- Implémentation
- Architecture
- Choix de conception

# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception

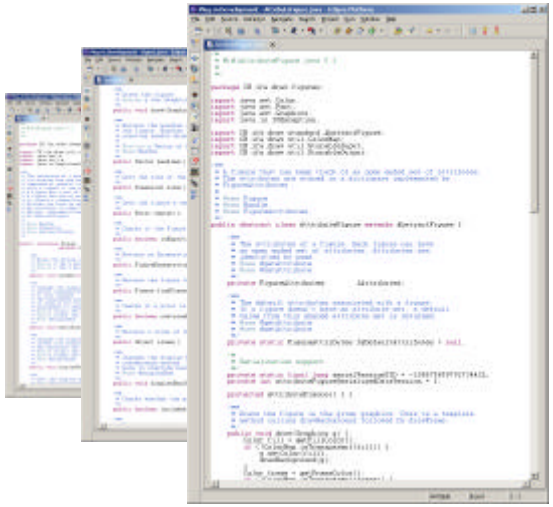
# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception



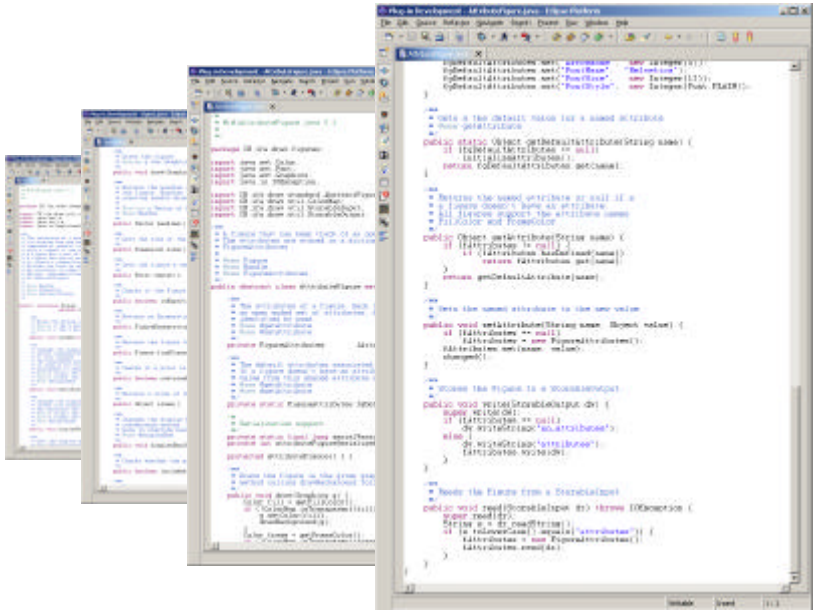
# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception



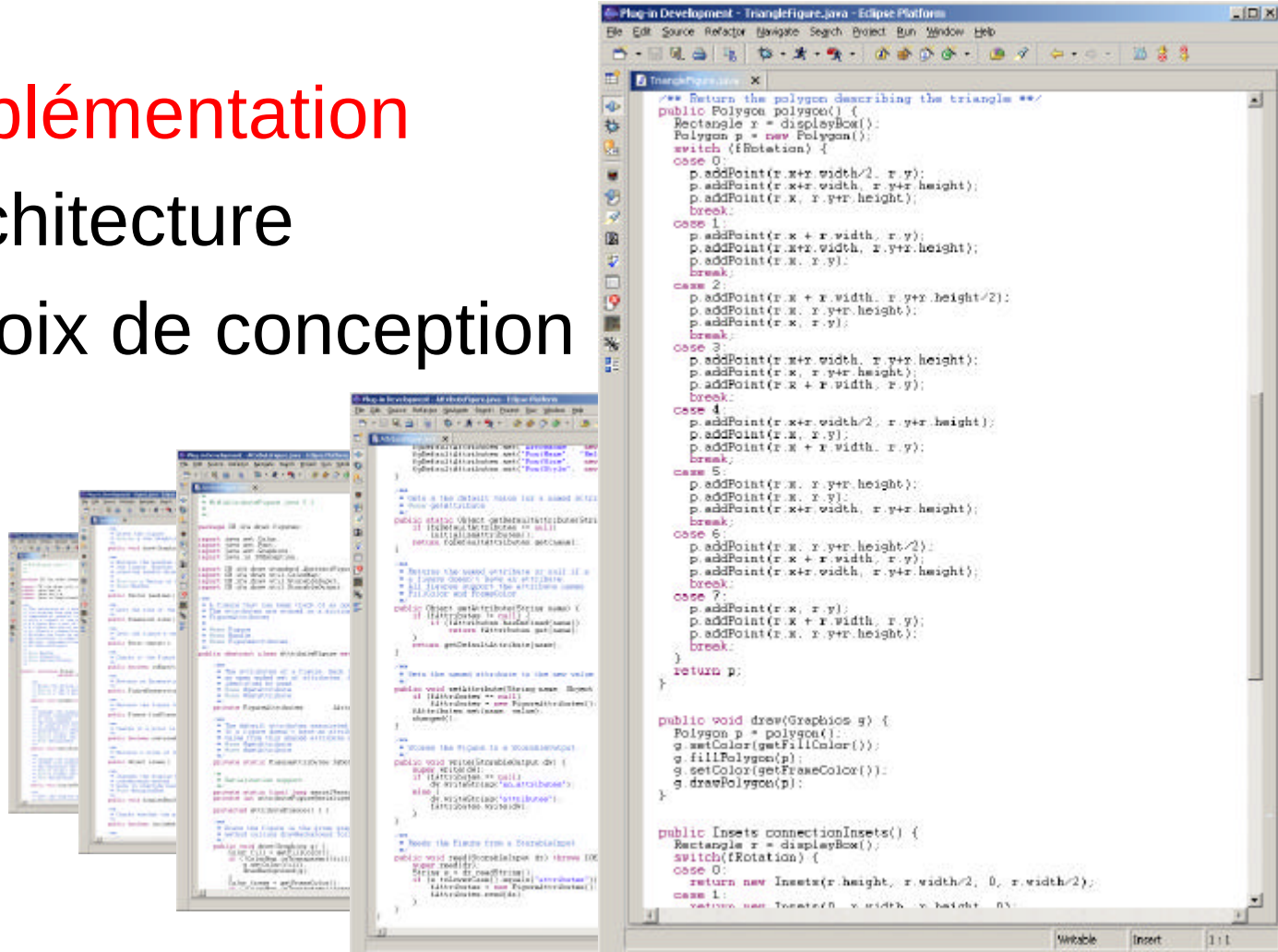
# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception



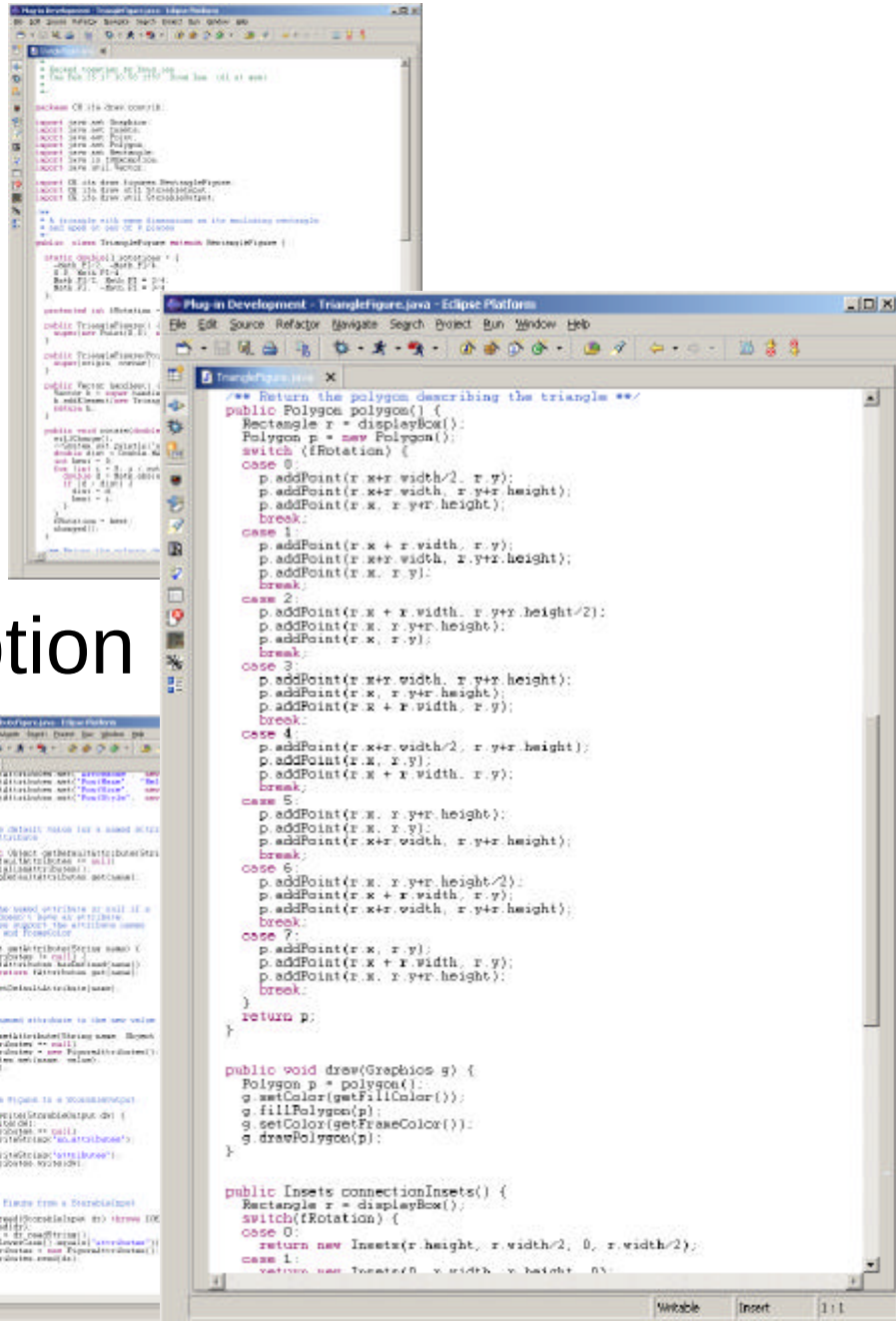
# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception



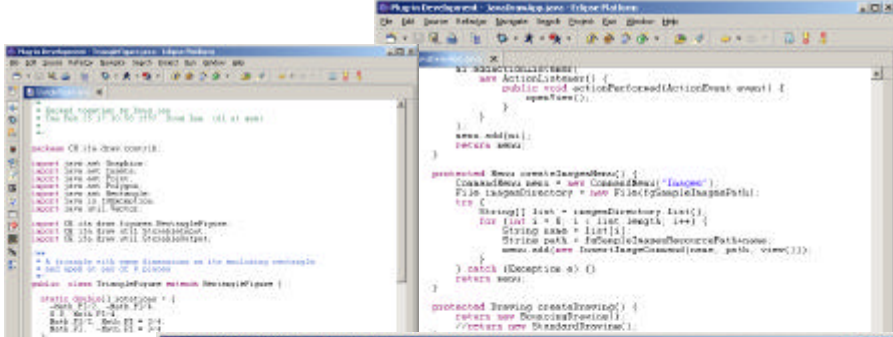
# Scénario (3/8)

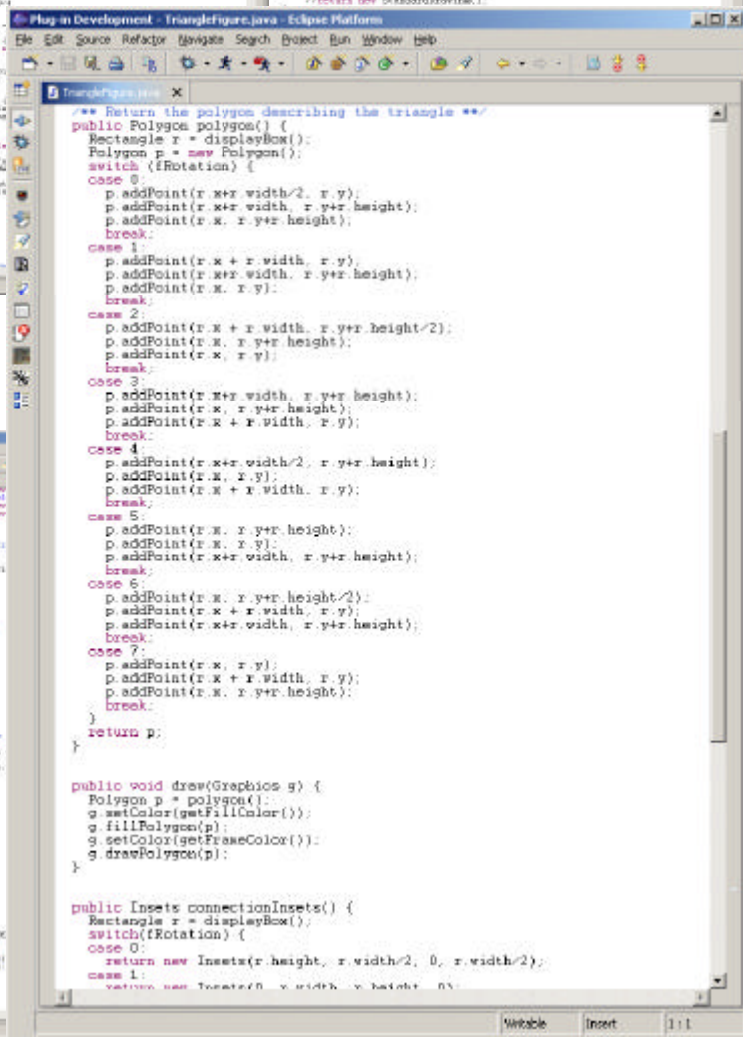
- Implémentation
- Architecture
- Choix de conception

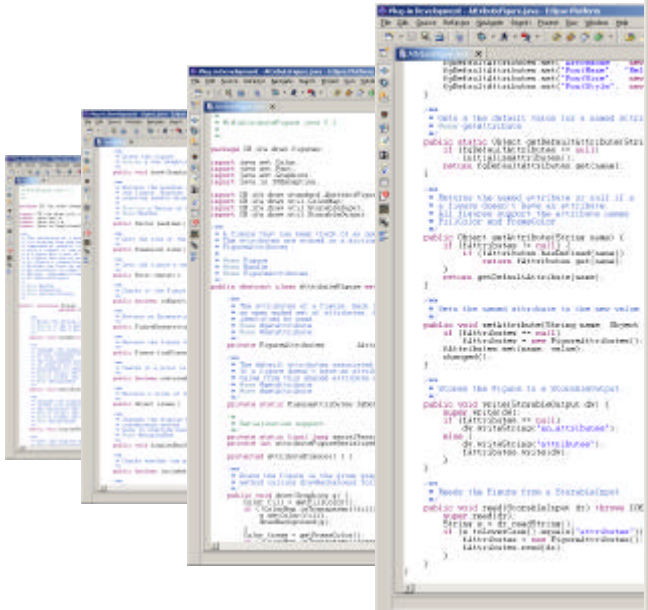


# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception

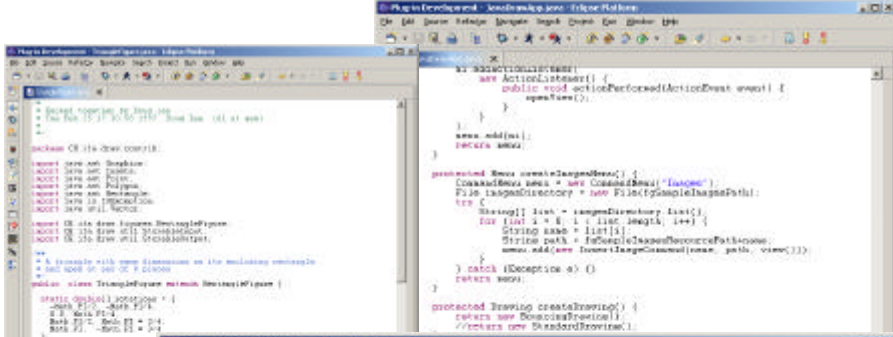


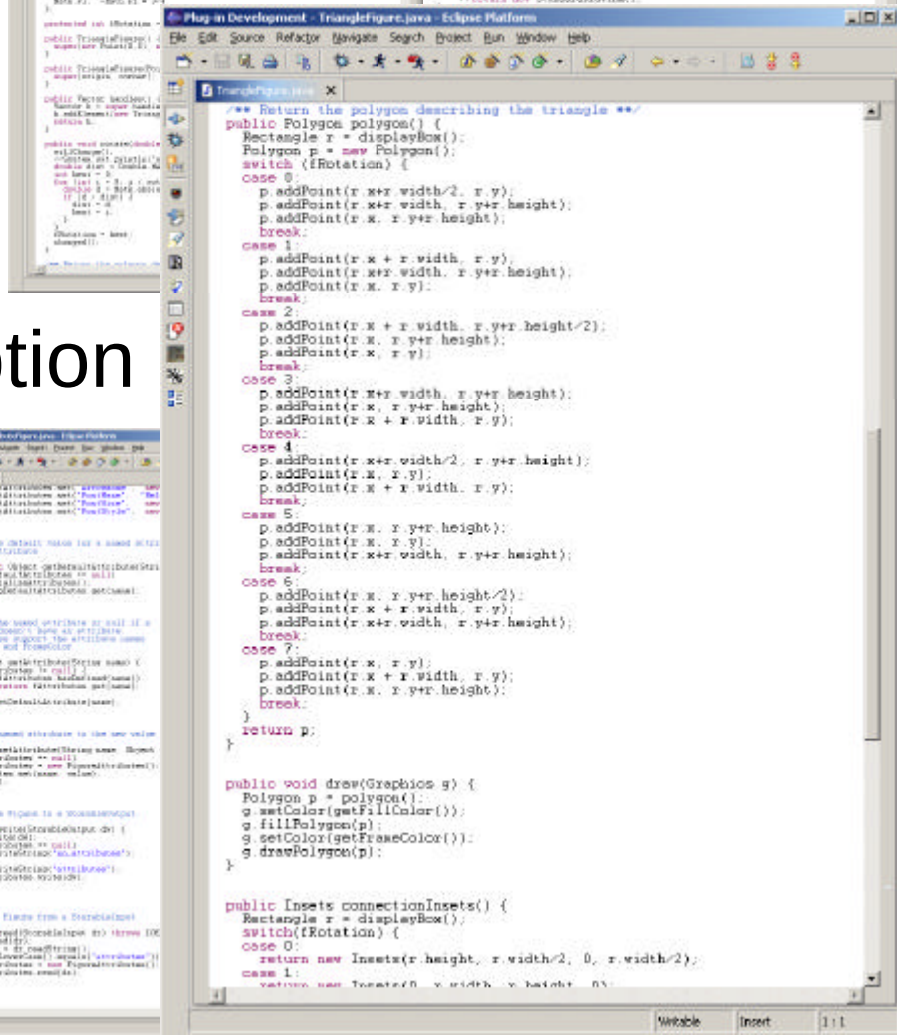


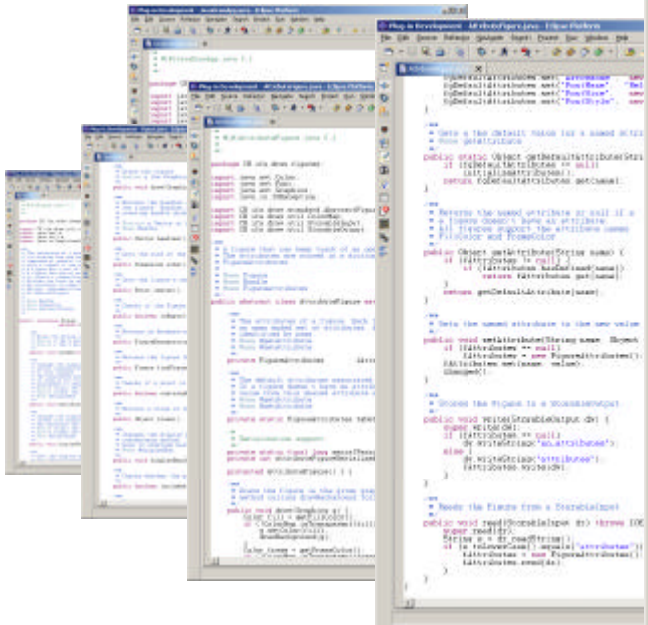



# Scénario (3/8)

- Implémentation
- Architecture
- Choix de conception

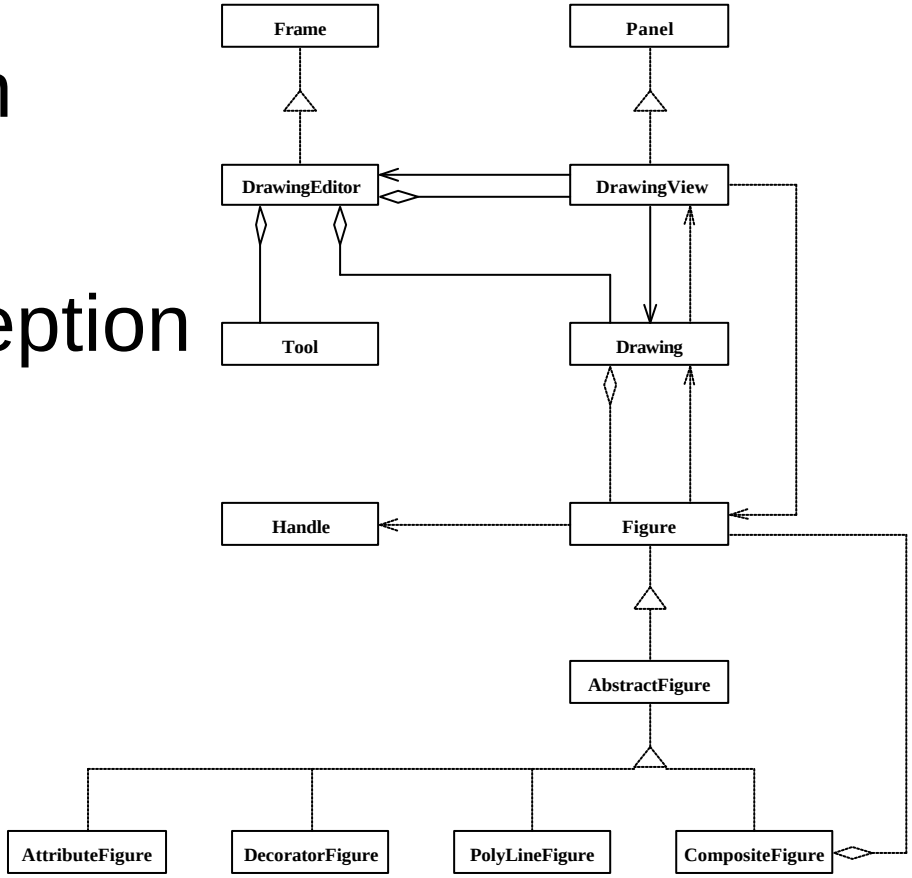




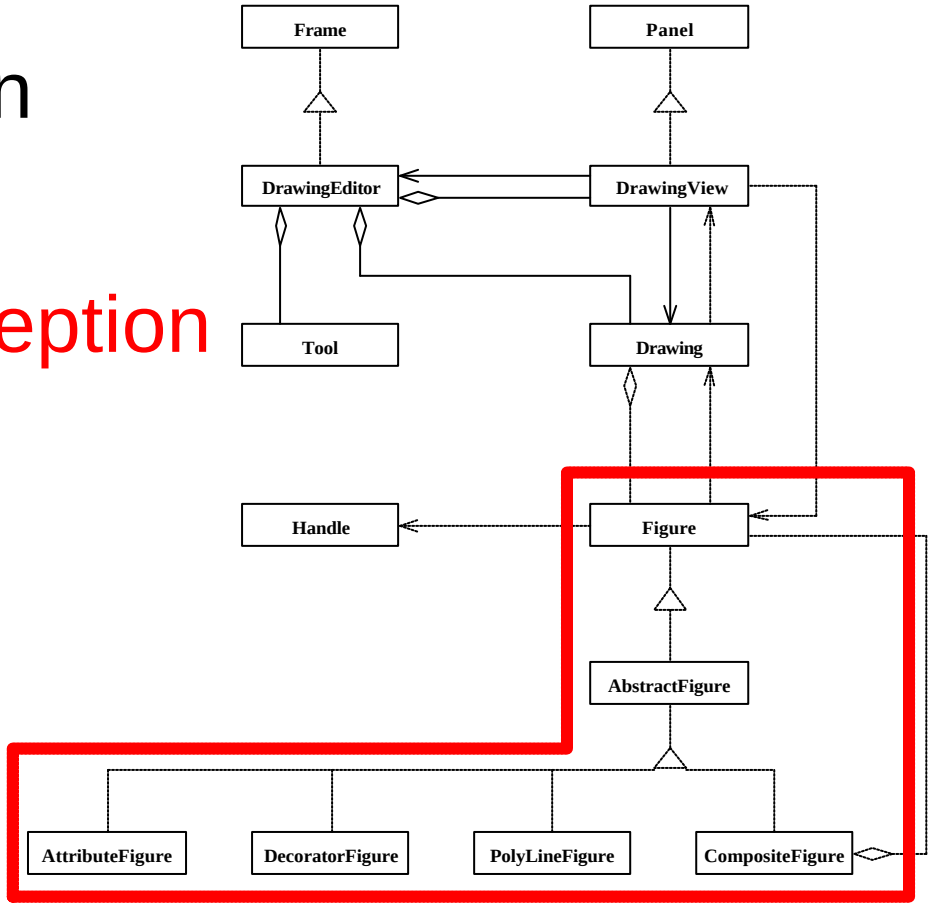
# Scénario (4/8)

- Implémentation
- Architecture
- Choix de conception



# Scénario (5/8)

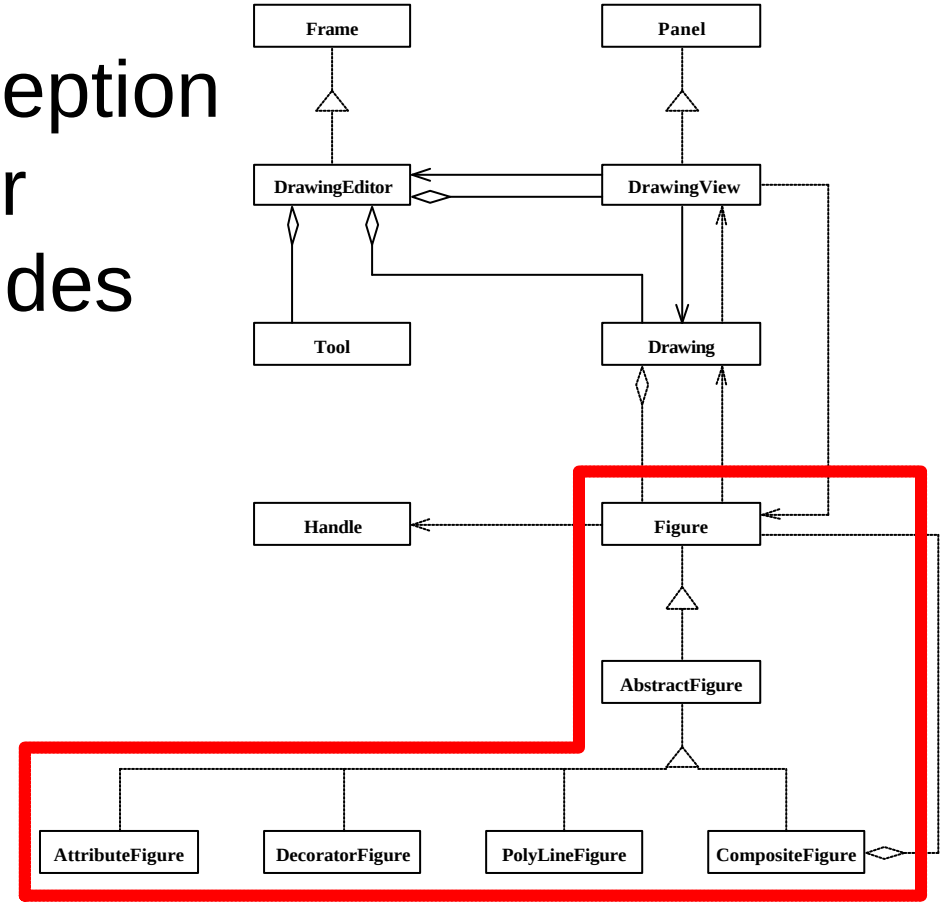
- Implémentation
- Architecture
- Choix de conception




Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

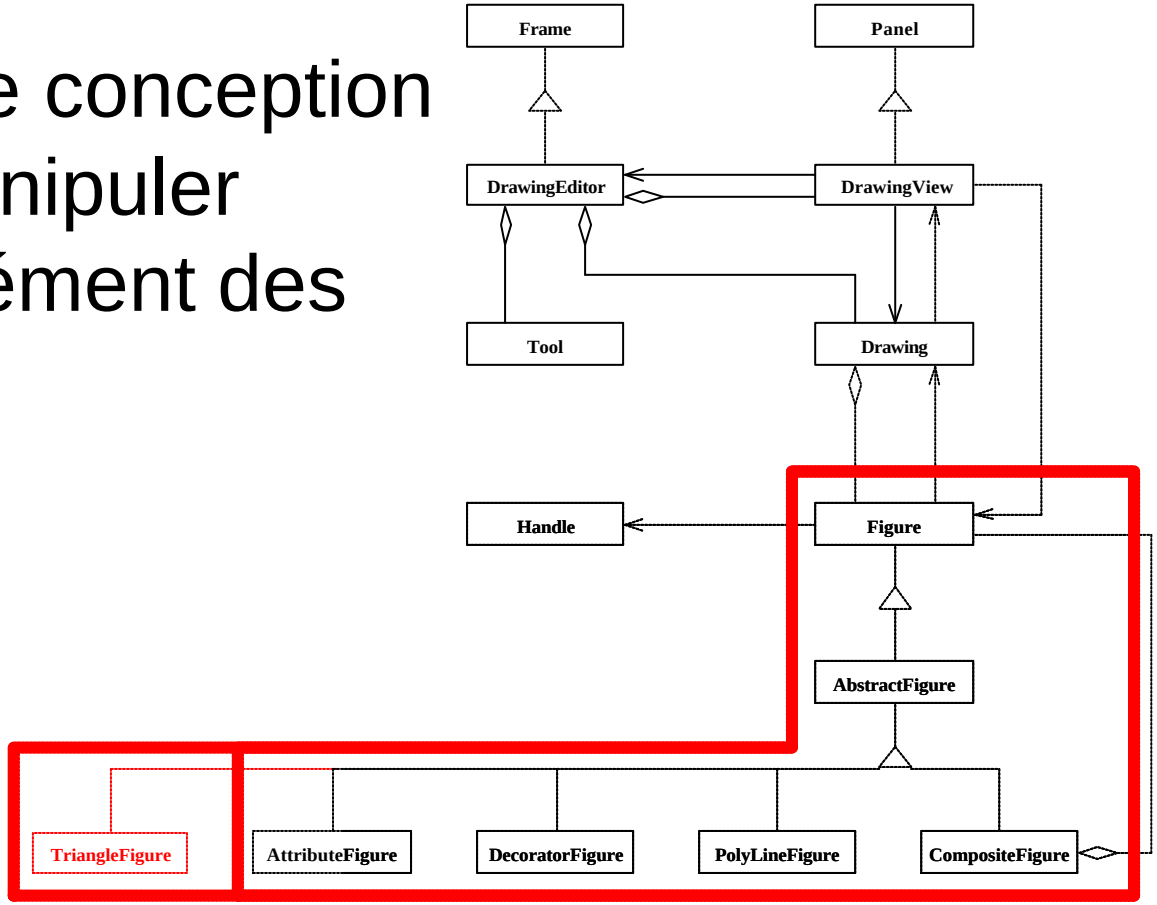
# Scénario (6/8)

- Choix de conception pour manipuler uniformément des objets



# Scénario (6/8)

- Choix de conception pour manipuler uniformément des objets

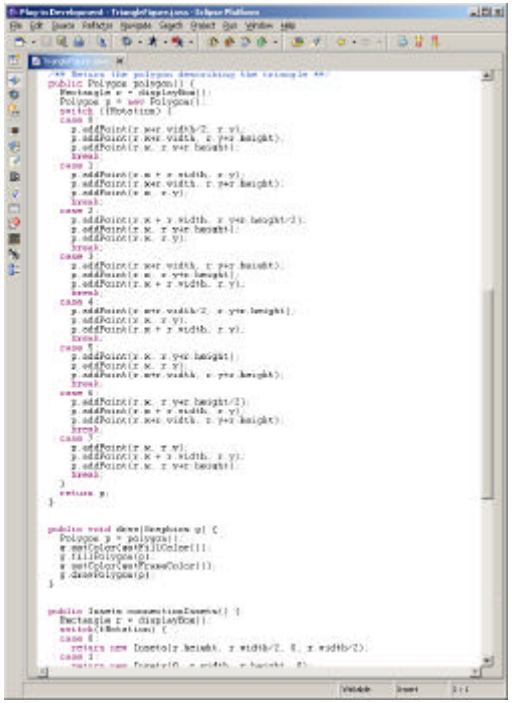


# Scénario (7/8)

- Modification
  - Plus pertinente
  - Plus rapide

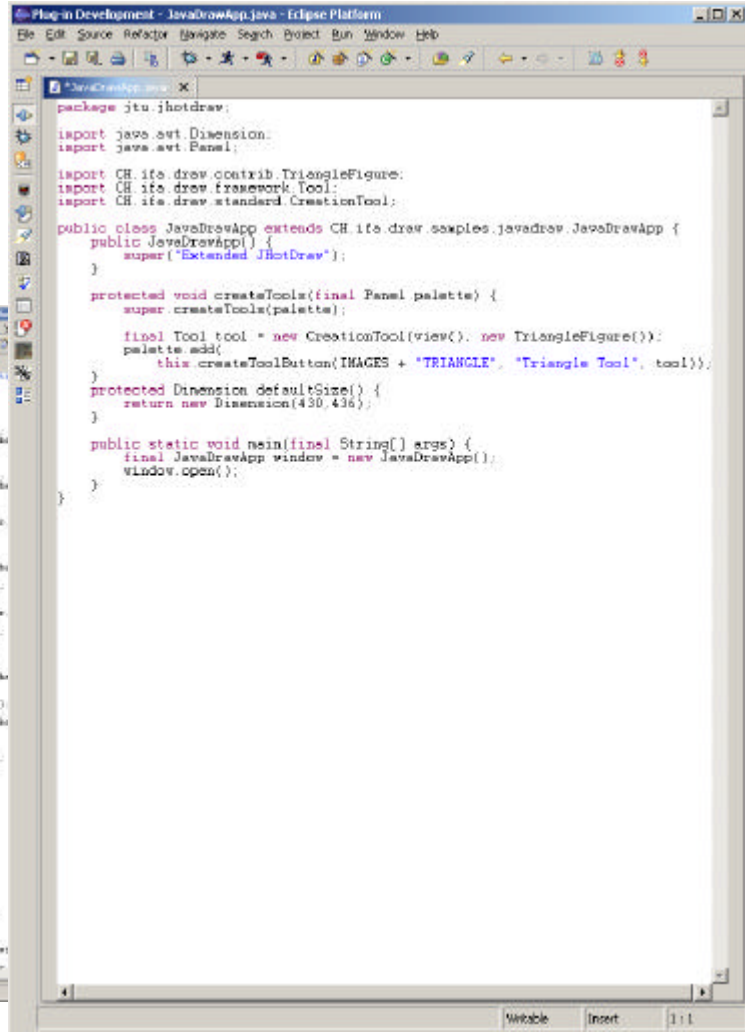
# Scénario (7/8)

- Modification
  - Plus pertinente
  - Plus rapide



# Scénario (7/8)

- Modification
  - Plus pertinente
  - Plus rapide



# Scénario (8/8)

- Choix de conception
  - Facilitent la maintenance
    - Rétro-conception
    - Compréhension
    - Traçabilité
    - Modification
  - Diminuent les coûts

# Constat (1/2)

- Développement des programmes : techniques et outils bien intégrés
  - Implémentation
    - *Eclipse*
  - Architecture
    - *Rational Rose*
  - Choix de conception
    - Patrons de conception [Gamma94]

# Constat (2/2)

- Maintenance des programmes : techniques et outils déconnectés
  - Implémentation
  - Architecture
  - Choix de conception

# Objectifs (1/2)

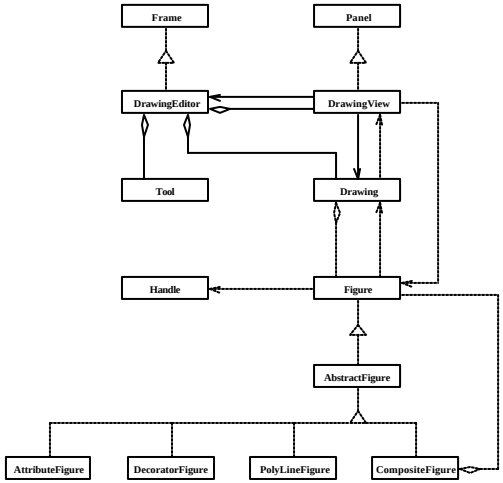
- Maintenance des programmes à objets
  - Rétro-conception
  - Compréhension
  - Traçabilité
  - Modification

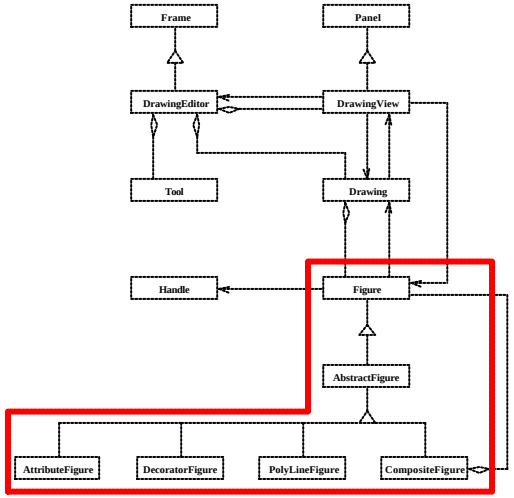
- Assistance semi-automatique
  - Obtention de l'architecture d'un programme
  - Identification des choix de conception

# Objectifs (2/2)

- Implémentation
- Architecture
- Choix de conception

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

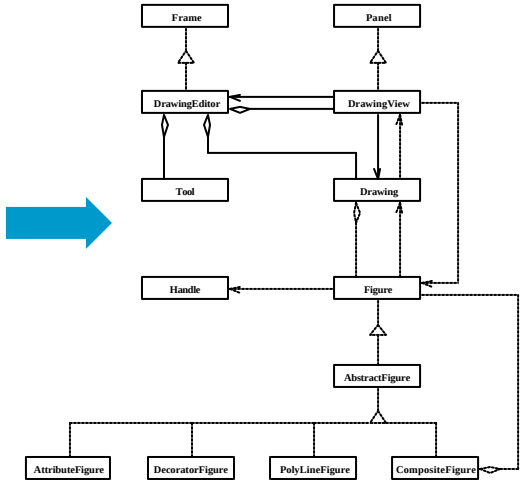


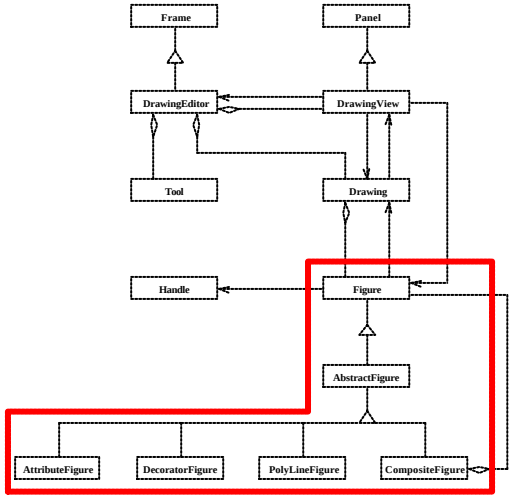



# Objectifs (2/2)

- Implémentation
- Architecture
- Choix de conception

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

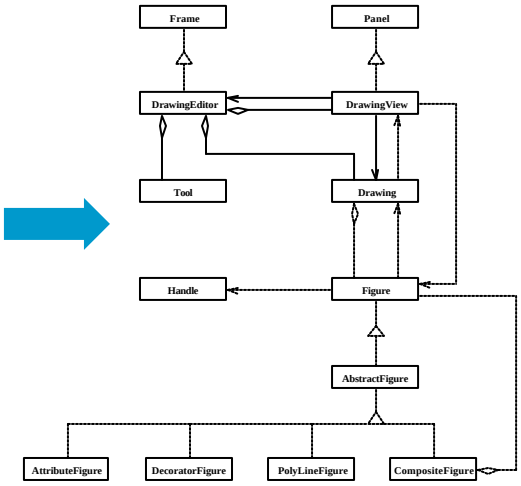


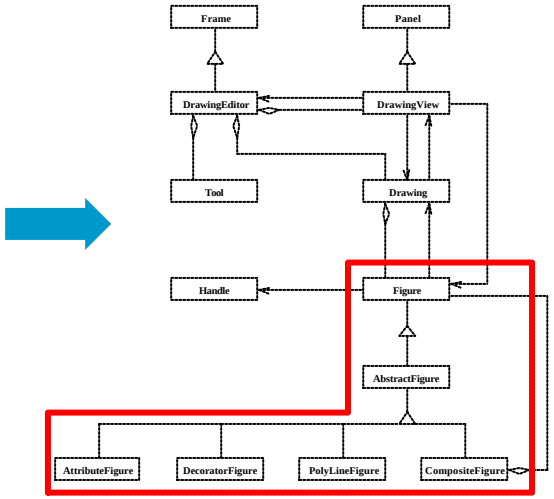



# Objectifs (2/2)

- Implémentation
- Architecture
- Choix de conception

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

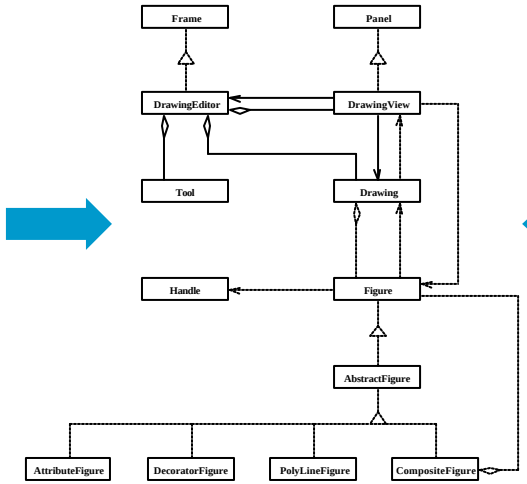


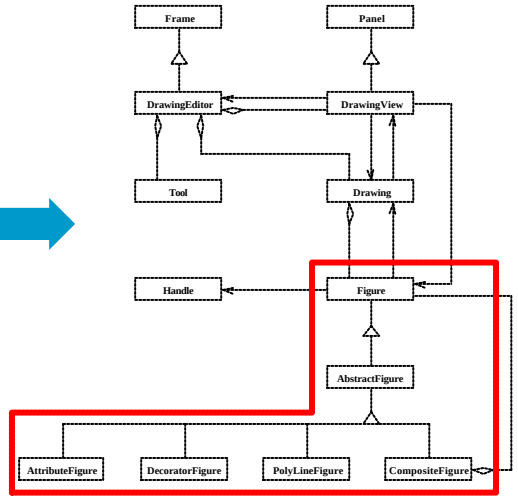



# Objectifs (2/2)

- Implémentation
- Architecture
- Choix de conception

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

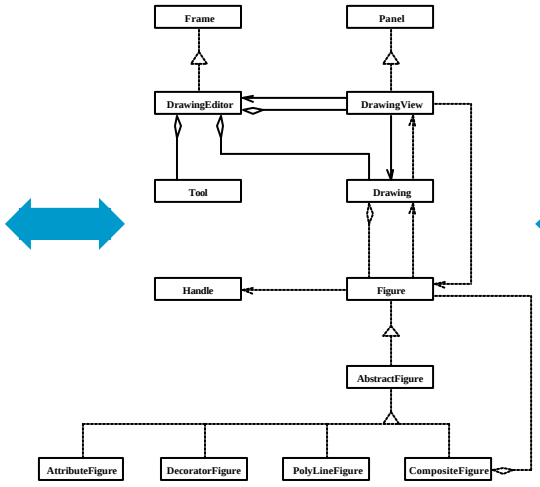


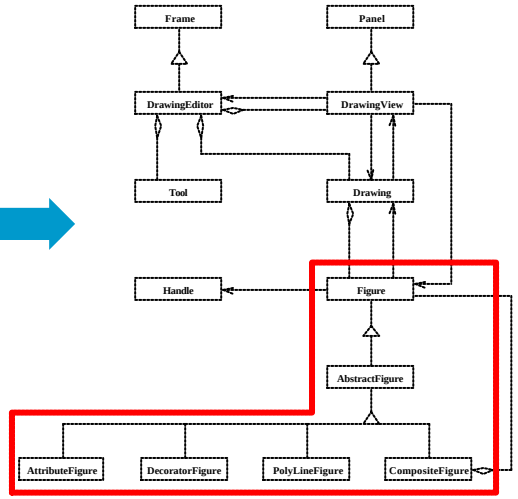



# Objectifs (2/2)

- Implémentation
- Architecture
- Choix de conception

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

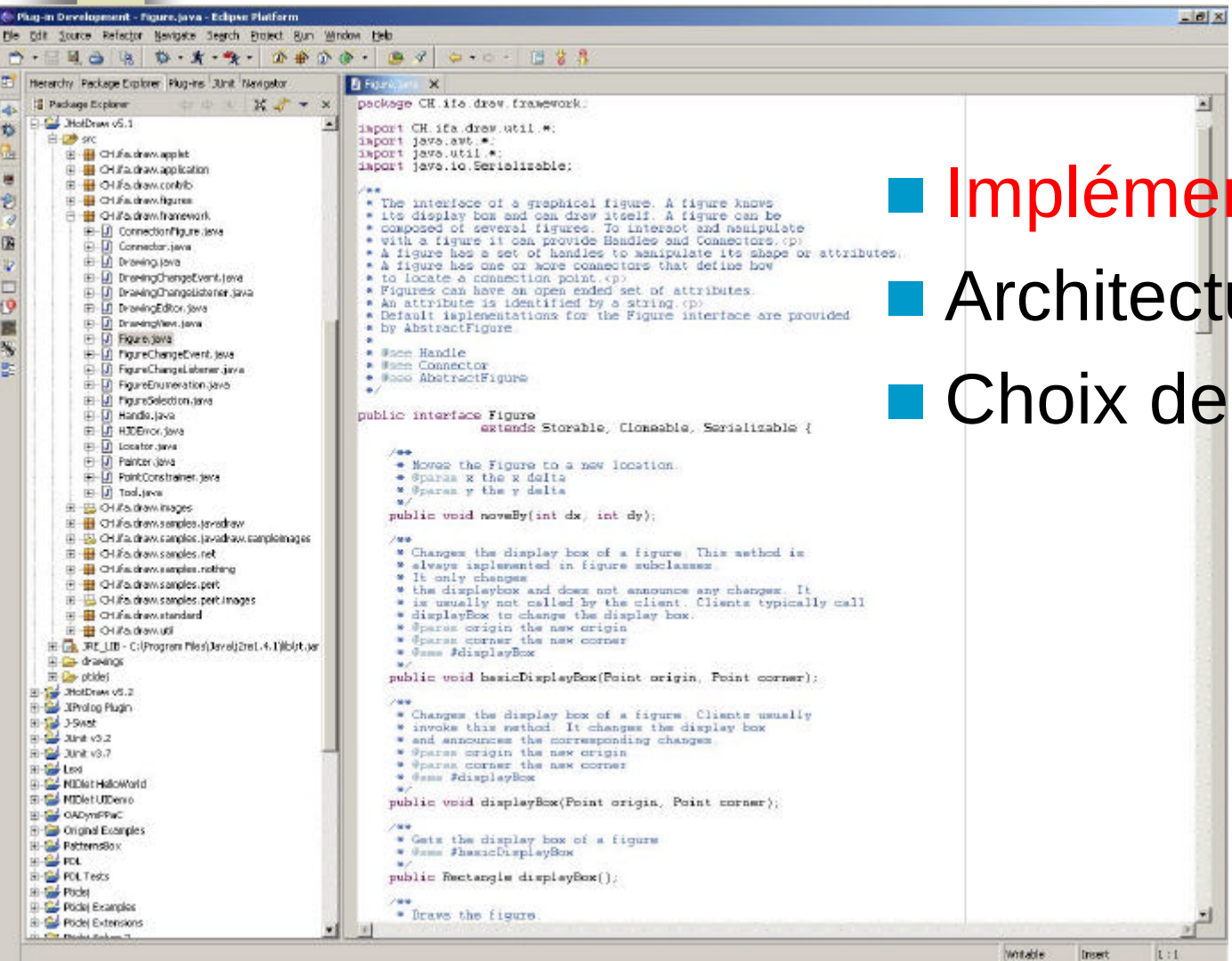


# Contributions (1/4)

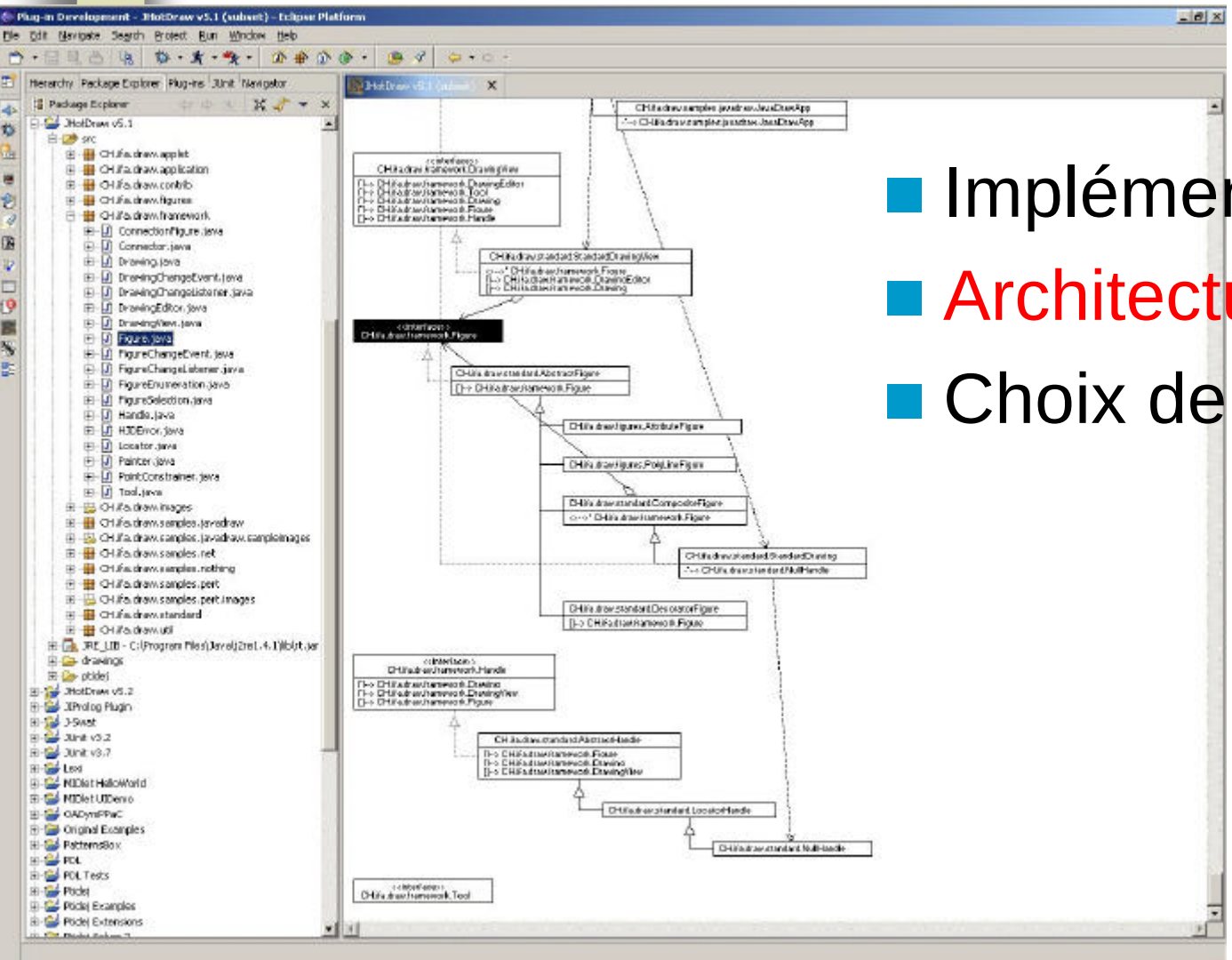
- Une suite d'outils intégrés, *Ptidej*
  - Techniques
    - Analyses statiques et dynamiques
    - Programmation par contraintes avec explications
  - Outils
    - *Introspector*, *Caffeine*
    - *PtidejSolver*, *PtidejLibrary*

# Contributions (2/4)



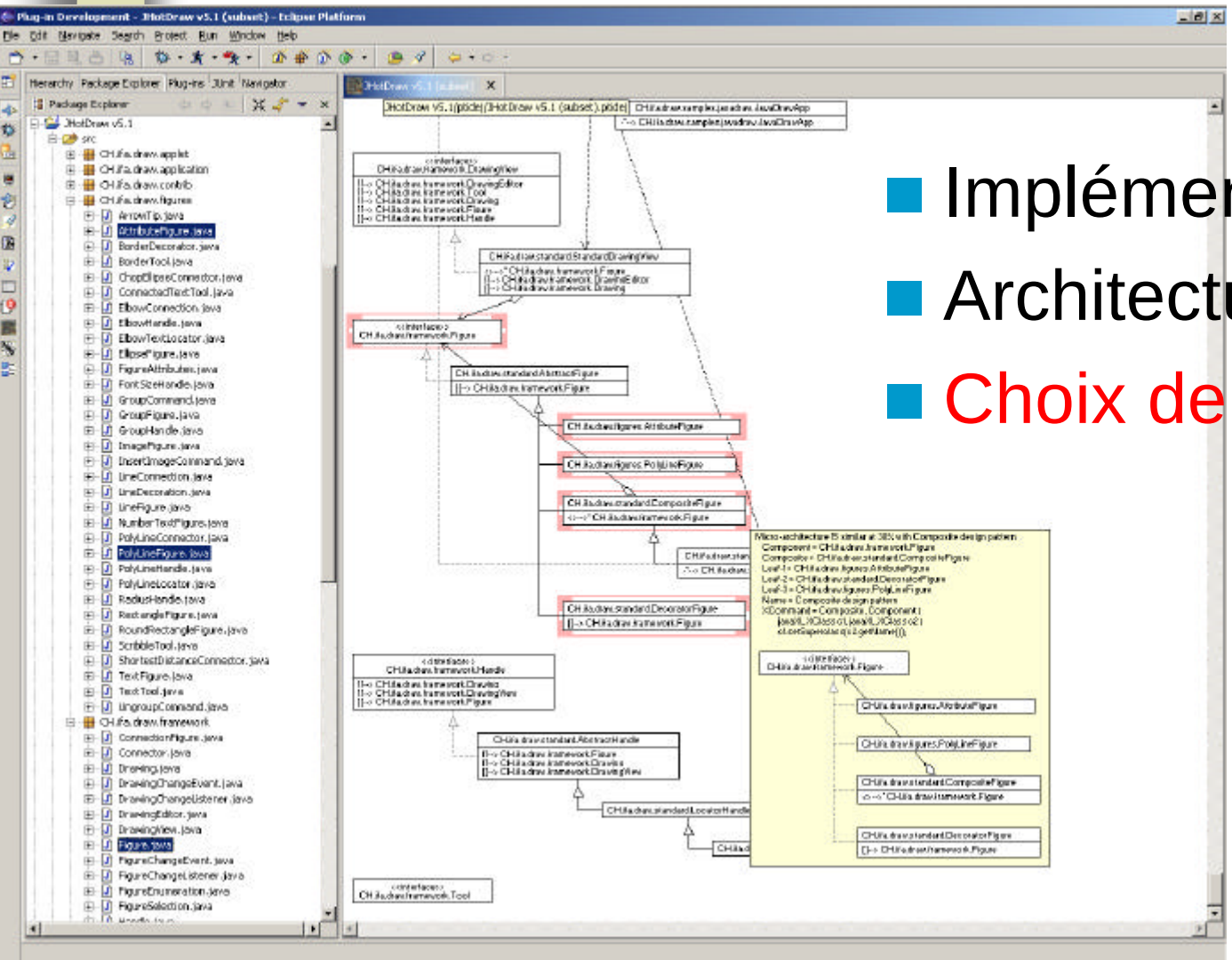

- Implémentation
- Architecture
- Choix de conception

# Contributions (3/4)




- Implémentation
- Architecture
- Choix de conception

# Contributions (4/4)



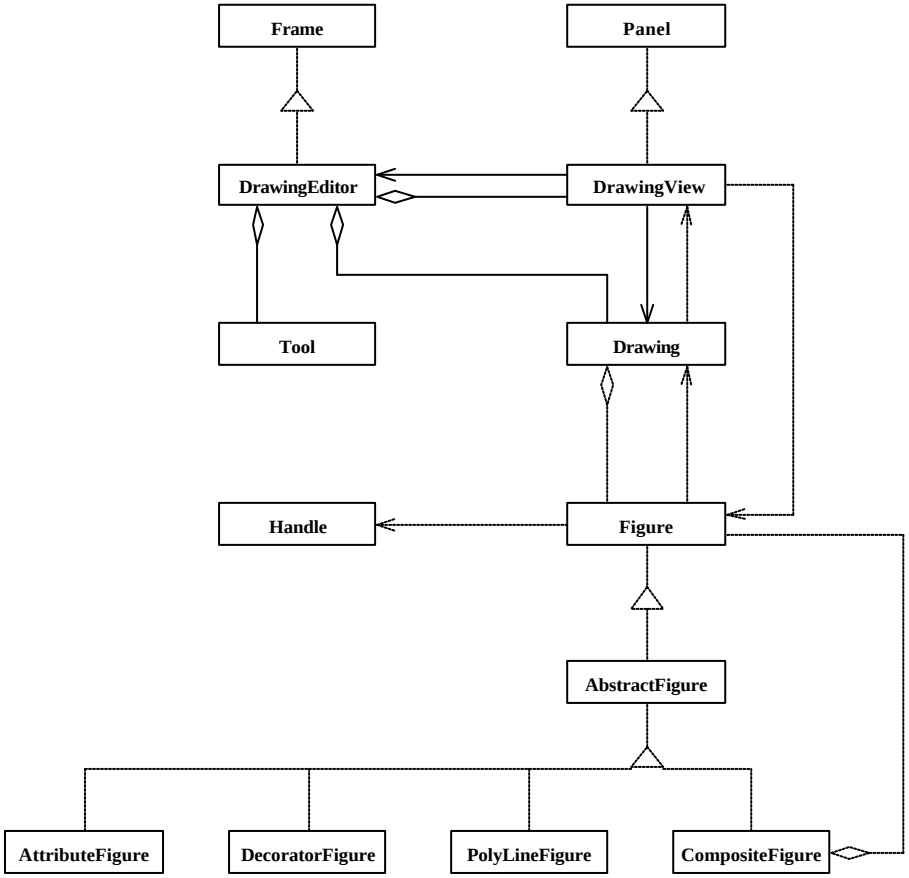

- Implémentation
- Architecture
- Choix de conception

# Plan

- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l'architecture d'un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Architecture d'un programme

- *UML* [OMG03]
  - Classe
  - Interface
  - Héritage
  - Instanciation
  - Association
  - Agrégation
  - Composition



# État de l'art (1/2)

- Programmation logique floue [Niere01]
  - Variantes d'implémentation
- Analyses statiques
  - Académiques [Korn99, Jackson99]
    - Efficacité spatiale et temporelle
  - Industrielles [ArgoUML, Rational]
    - Intégration aux outils de développement

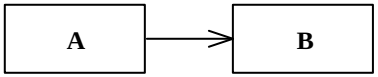
# État de l’art (2/2)

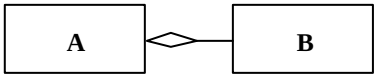
- Classe, interface, héritage, instanciation
  - Définitions *UML* « précises »
- Association, agrégation, composition
  - Définitions *UML* « ambiguës »
  - Littérature : au moins 25 définitions
    - Peu de définitions constructives
    - Définitions constructives incomplètes

# Objectifs

- Définitions consensuelles et constructives
  - Association, agrégation, composition
- Algorithmes d'analyses
  - Statiques
  - Dynamiques

# Définitions

- 

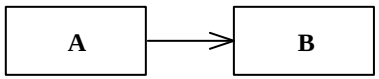
Association
  - Indique que les instances de A envoient des messages aux instances de B
- 

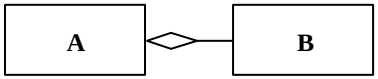
Agrégation
  - Association définissant un tout (A) et une partie (B) (champ appartenant au tout)
- 

Composition
  - Agrégation dans laquelle la partie (B) appartient exclusivement au tout (A)

# Propriétés (1/2)

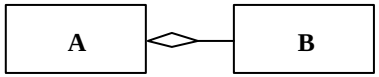
- Multiplicité : *MU*
  - Nombre d'instances min. et max.





- Site d'invocation : *SI*
  - Site d'invocation des envois de message




# Propriétés (2/2)

- Durée de vie : *DV*
  - Dépendance entre les durées de vie des instances de la partie et du tout


- Exclusivité : *EX*
  - Exclusivité des instances de la partie au tout

# Redéfinitions

- Association
  - $DV = ?\ \wedge\ EX = \textit{indifférent} \wedge MU = [0, +\infty] \wedge SI \in \{\textit{variable}, \textit{paramètre}, \textit{champ}, \ldots\}$
- Agrégation
  - $DV = ?\ \wedge\ EX = \textit{indifférent} \wedge MU = [1, +\infty] \wedge SI \in \{\textit{champ}, \ldots\}$
- Composition
  - $DV = + \wedge EX = \textit{vrai} \wedge MU = [1, +\infty] \wedge SI \in \{\textit{champ}, \ldots\}$

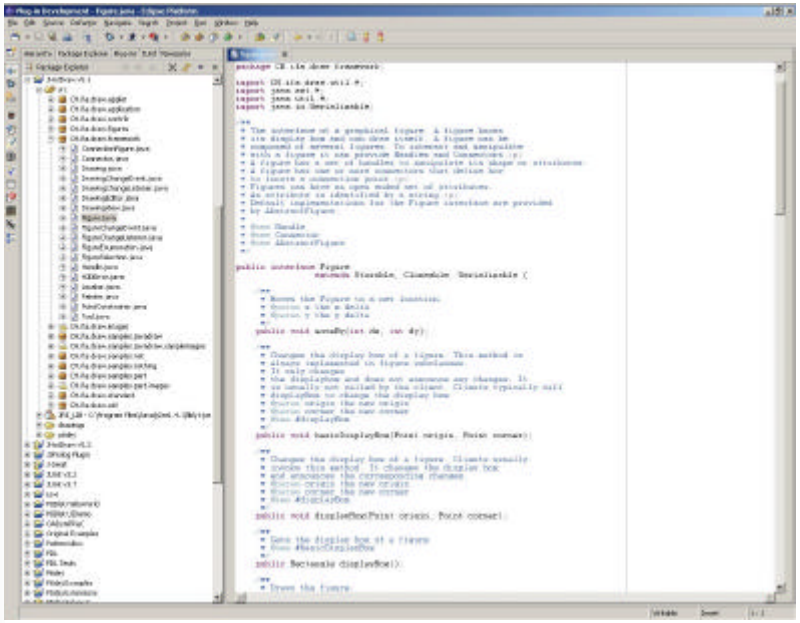
# Techniques, outils

- Analyses statiques : *Introspector*
  - Multiplicité, *MU*
  - Site d'invocation, *SI*
- Analyses dynamiques : *Caffeine*
  - Durée de vie, *DV*
  - Exclusivité, *EX*

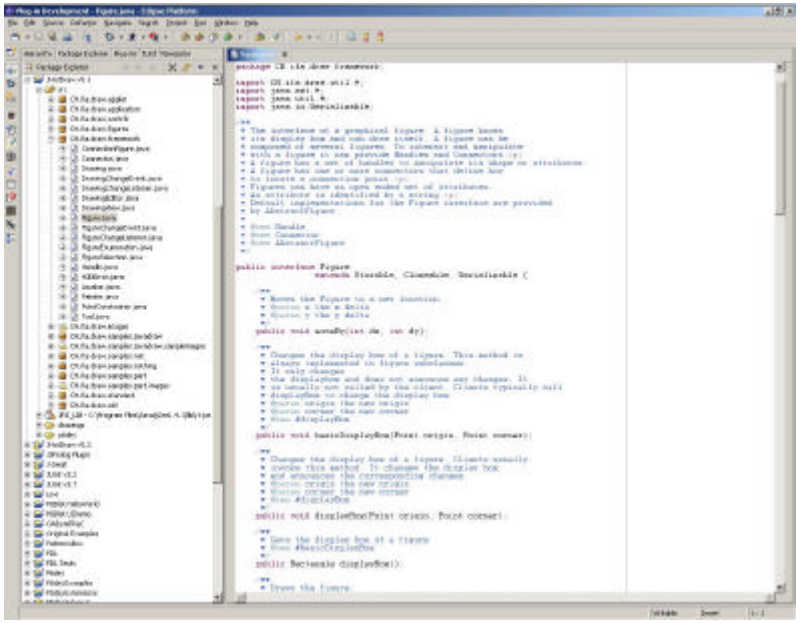
# Architecture (1/2)

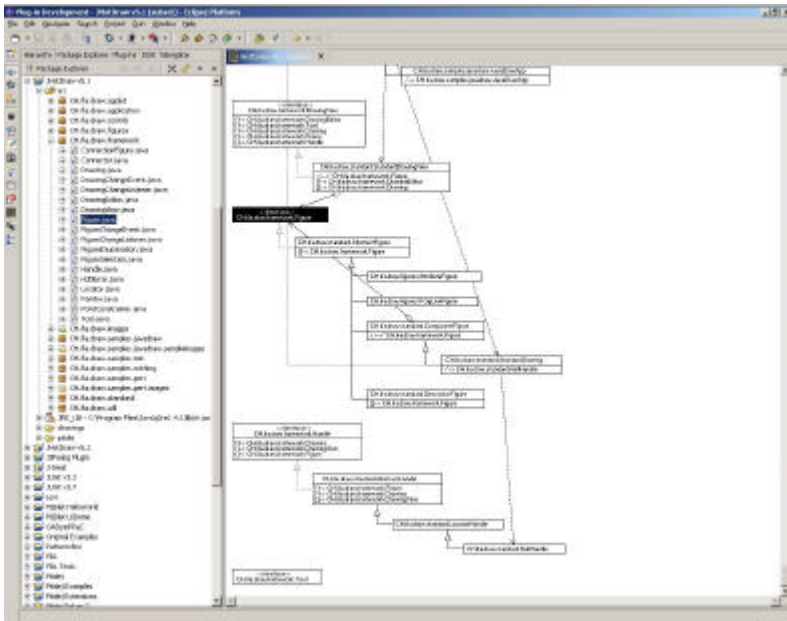
- *Introspector*
  - Classes, interfaces
  - Héritage, instanciation
  - Association, agrégation
- *Caffeine*
  - Composition

# Architecture (2/2)



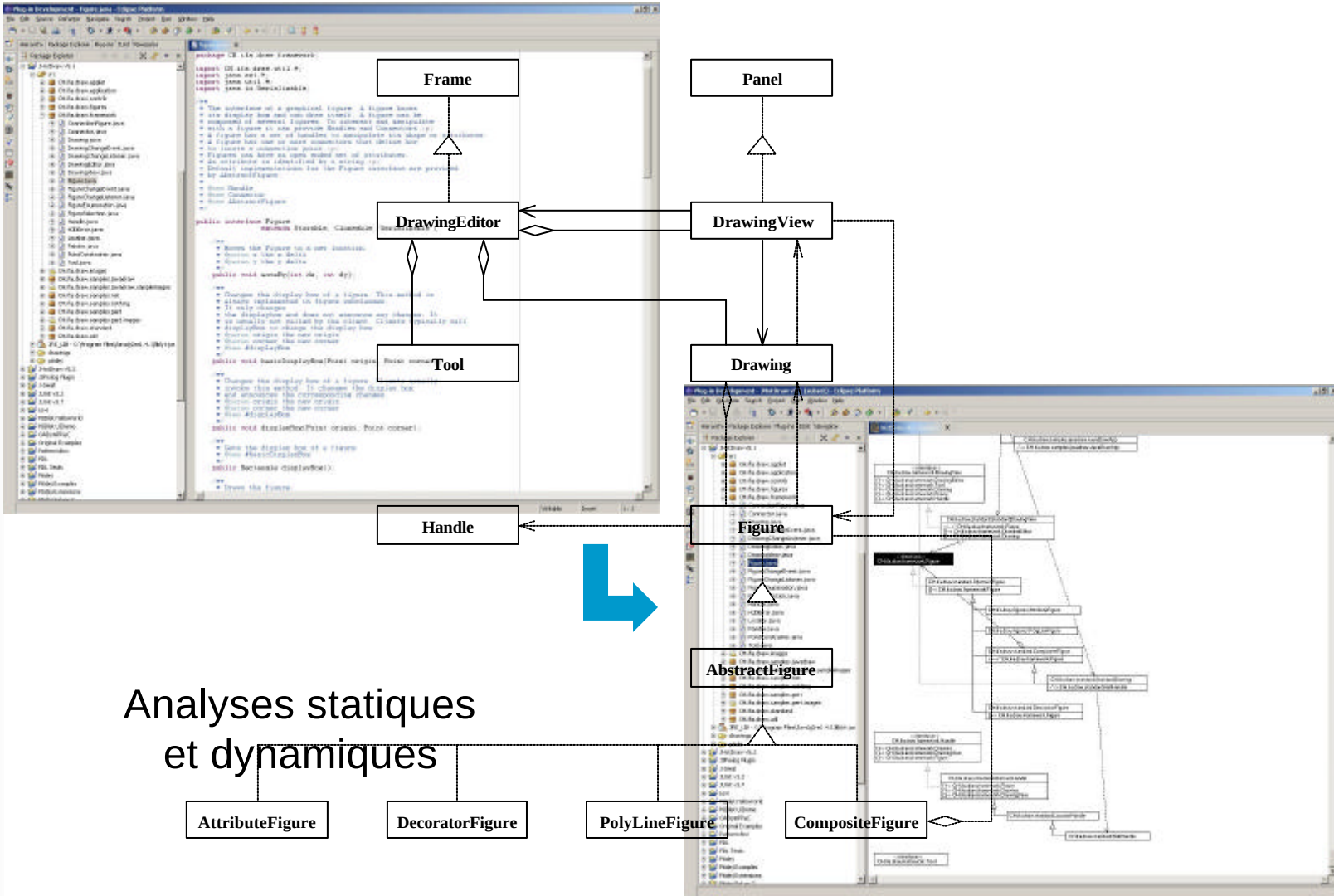
# Architecture (2/2)





Analyses statiques
et dynamiques

# Architecture (2/2)



# Plan

- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l'architecture d'un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Choix de conception

- Découverte des choix de conception
  - Vaste problème [Shull96]
  - Méthodologie spécifique à l'I.A. [Tonella99]

- Catalogue de choix de conception

# Patrons de conception

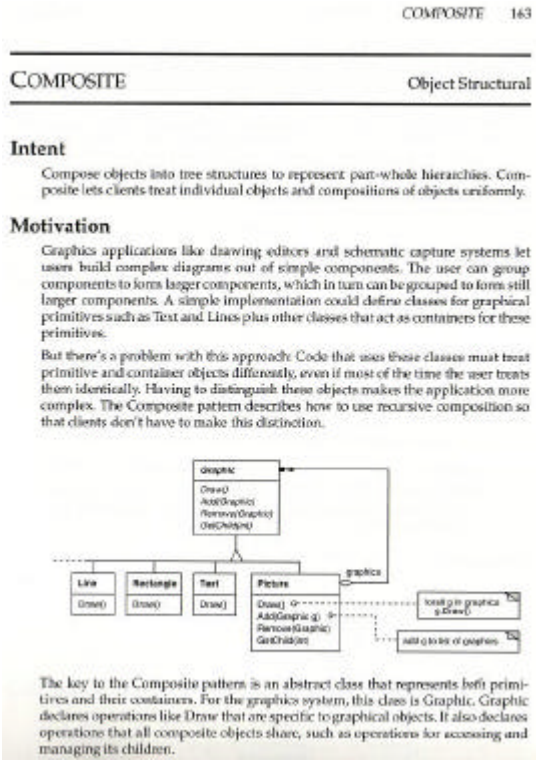
- Erich Gamma *et al. ; Patrons de conception* ; Addison-Wesley, 1994 [Gamma94]

- Ensemble [Alexander77]
  - Nom
  - Problème
  - Solution
  - Compromis

# Patrons de conception

- Erich Gamma *et al. ; Patrons de conception* ; Addison-Wesley, 1994 [Gamma94]

- Ensemble [Alexander77]
  - Nom
  - Problème
  - Solution
  - Compromis



COMPOSITE 163

COMPOSITE Object Structural

**Intent**

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

**Motivation**

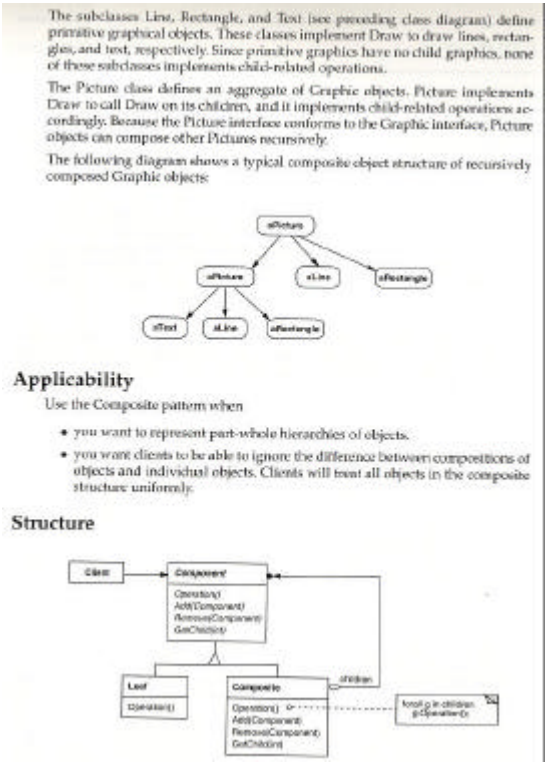
Graphics applications like drawing editors and schematic capture systems let users build complex diagrams out of simple components. The user can group components to form larger components, which in turn can be grouped to form still larger components. A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives.

But there's a problem with this approach: Code that uses these classes must treat primitive and container objects differently, even if most of the time the user treats them identically. Having to distinguish these objects makes the application more complex. The Composite pattern describes how to use recursive composition so that clients don't have to make this distinction.

The key to the Composite pattern is an abstract class that represents both primitives and their containers. For the graphics system, this class is Graphic. Graphic declares operations like Draw that are specific to graphical objects. It also declares operations that all composite objects share, such as operations for accessing and managing its children.

# Patrons de conception

- Erich Gamma *et al. ; Patrons de conception* ; Addison-Wesley, 1994 [Gamma94]



The subclasses Line, Rectangle, and Text (see preceding class diagram) define primitive graphical objects. These classes implement Draw to draw lines, rectangles, and text, respectively. Since primitive graphics have no child graphics, none of these subclasses implements child-related operations.

The Picture class defines an aggregate of Graphic objects. Picture implements Draw to call Draw on its children, and it implements child-related operations accordingly. Because the Picture interface conforms to the Graphic interface, Picture objects can compose other Pictures recursively.

The following diagram shows a typical composite object structure of recursively composed Graphic objects:

**Applicability**

Use the Composite pattern when

- you want to represent part-whole hierarchies of objects.
- you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

**Structure**

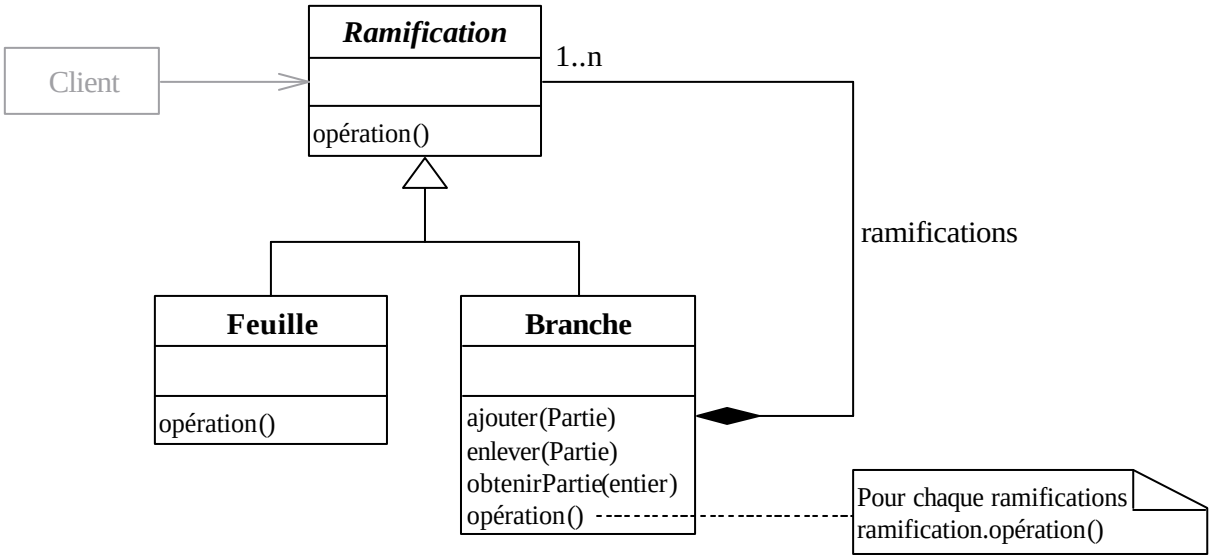
- Ensemble [Alexander77]
  - Nom
  - Problème
  - Solution
  - Compromis

# Le patron de conception
## Composite – problème

- Composer des objets en une structure d'arbre pour représenter des hiérarchies tout–partie
- Permettre au client de manipuler uniformément des objets et des compositions d'objets

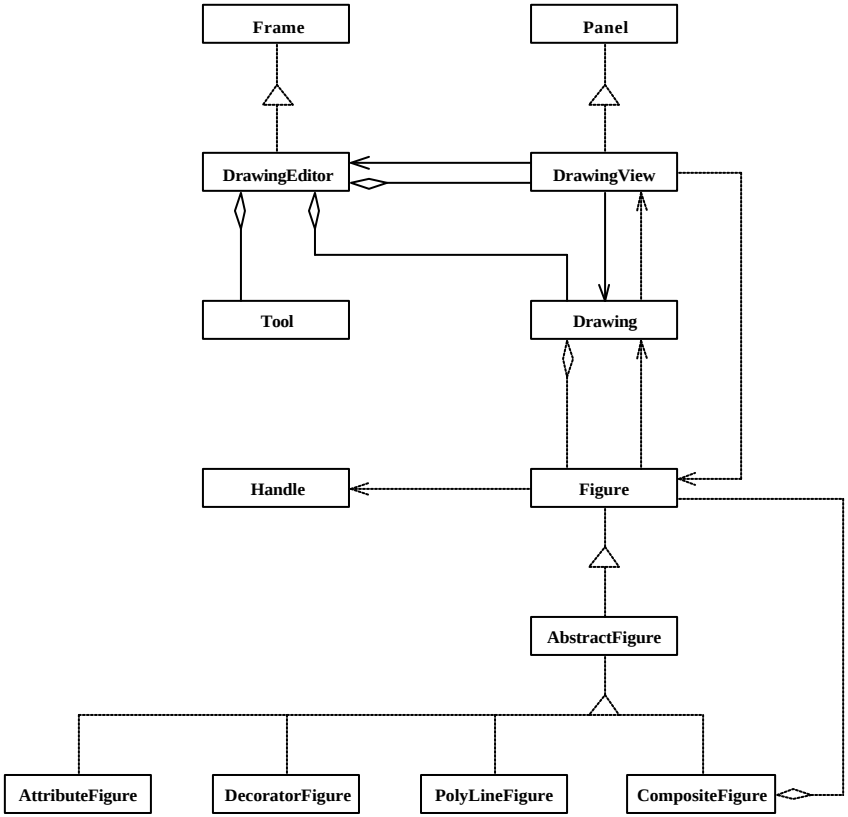
# Le patron de conception Composite – solution

- Solution d'un patron de conception = motif de conception qui décrit des micro-architectures récurrentes



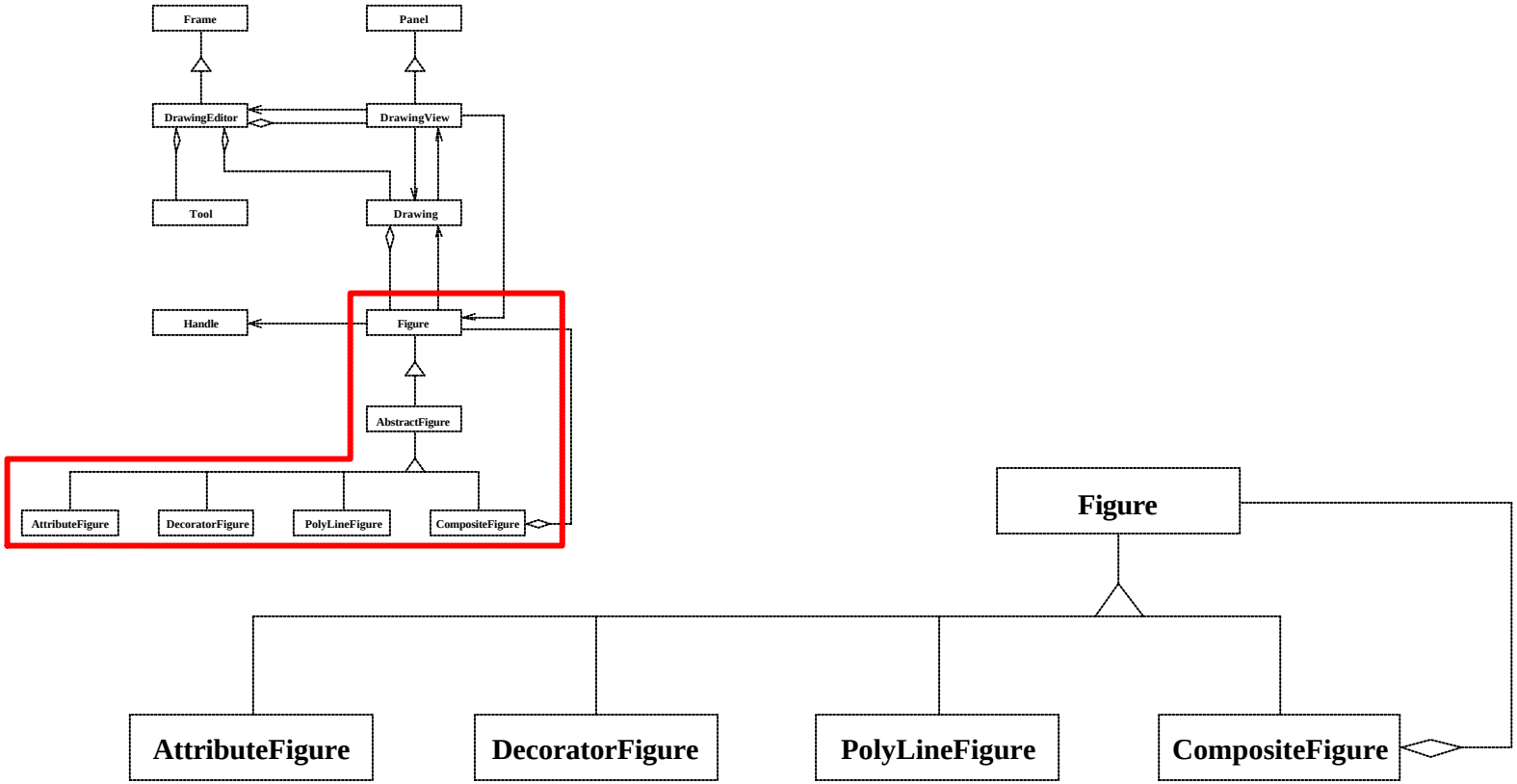
# Le patron de conception Composite – exemple

- Micro-architecture similaire au motif



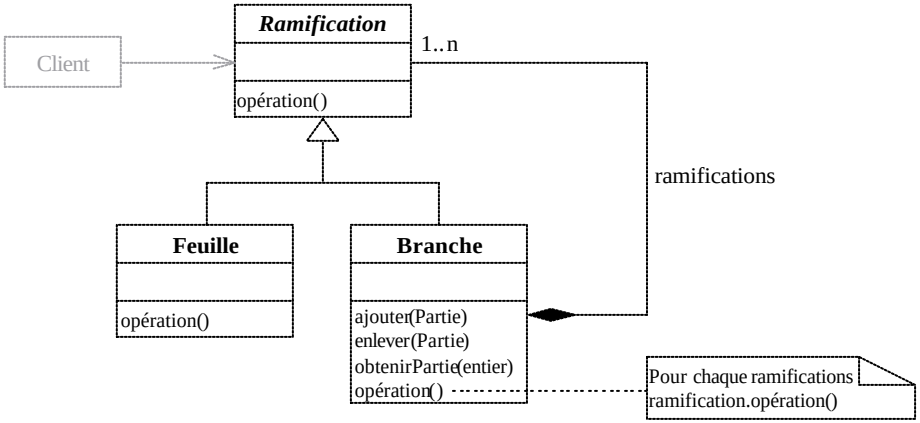
# Le patron de conception Composite – exemple

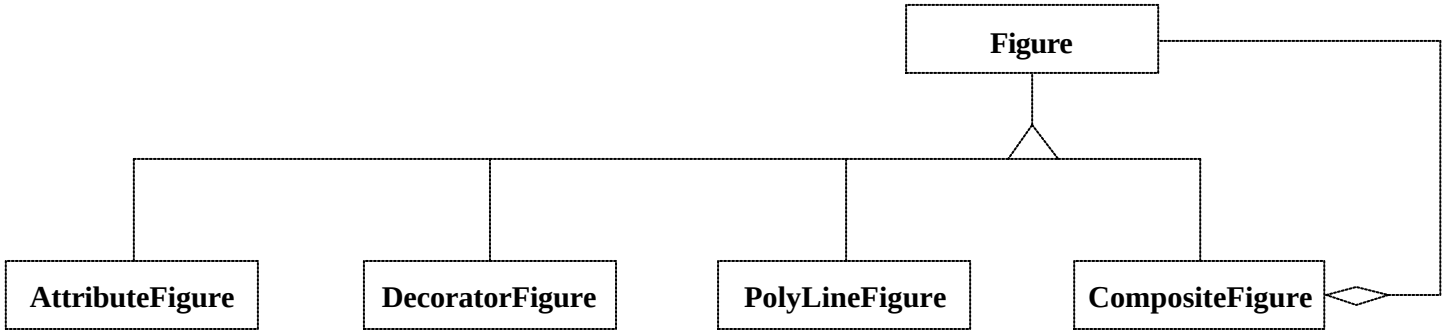
- Micro-architecture similaire au motif



# Le patron de conception Composite – exemple

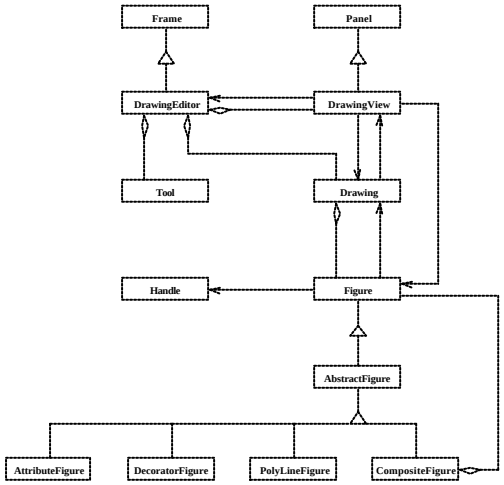
- Micro-architecture similaire au motif

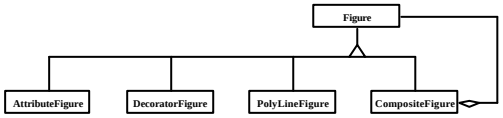


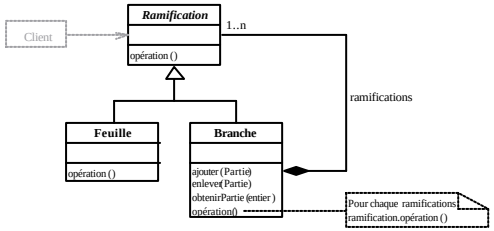



# Problème

Comment identifier dans l'architecture d'un programme des micro-architectures similaires à des motifs de conception pour expliquer les choix de conception ?



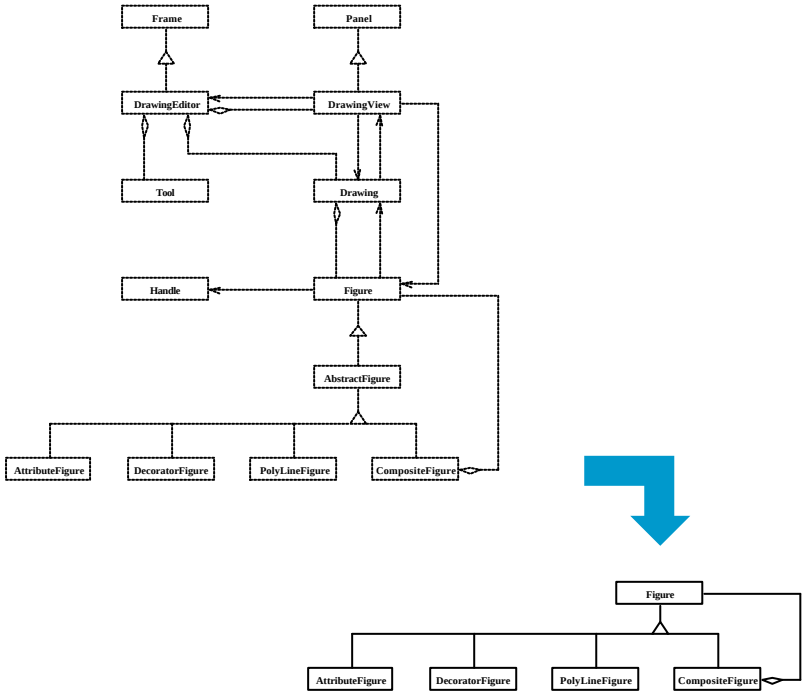



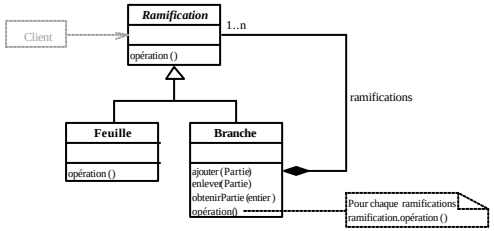




Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

# Problème

Comment identifier dans l'architecture d'un programme des micro-architectures similaires à des motifs de conception pour expliquer les choix de conception ?

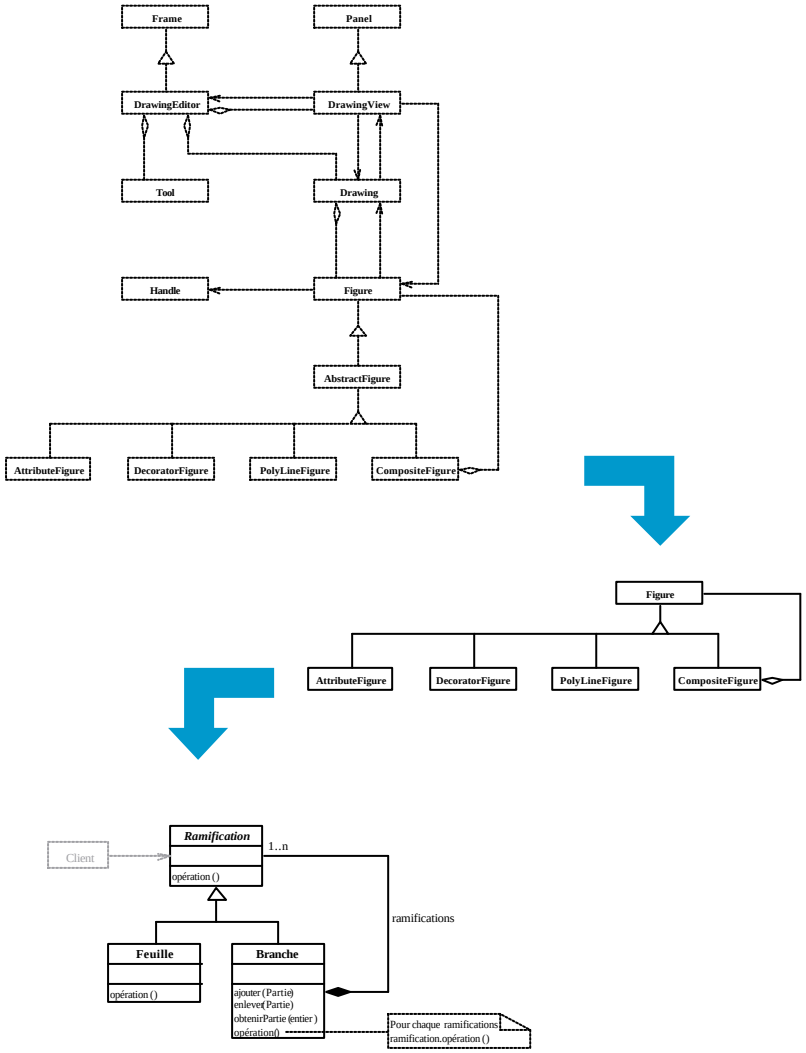



Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

# Problème

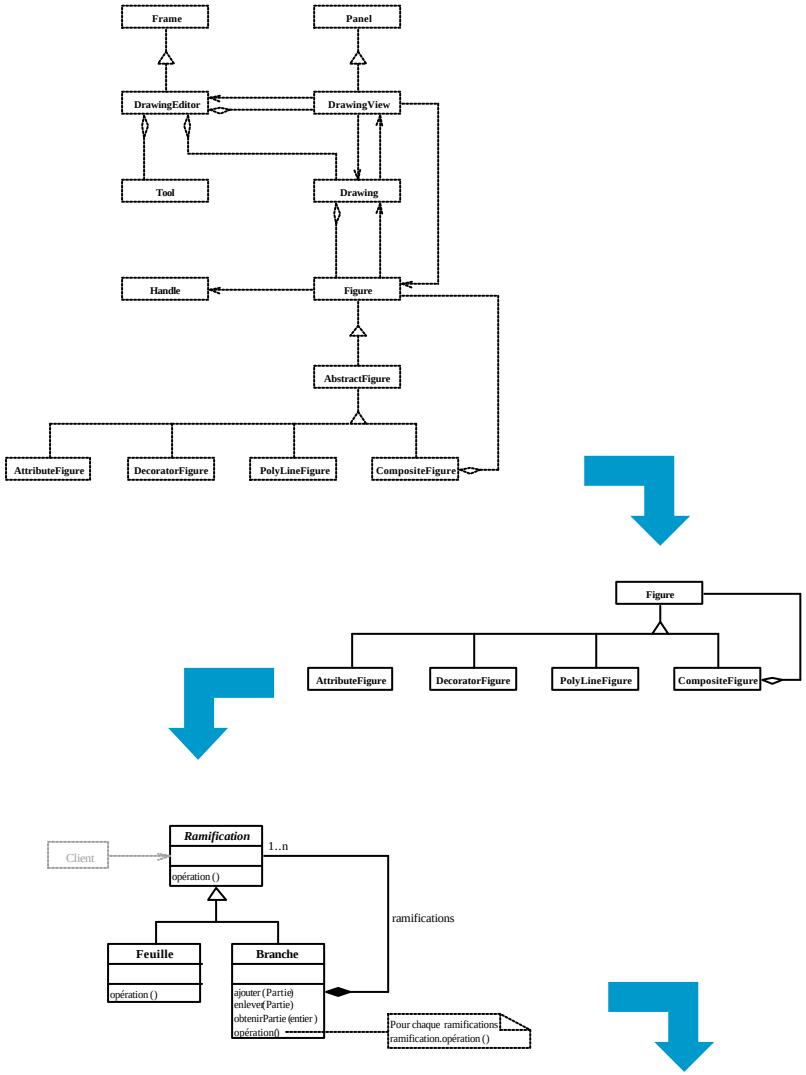
Comment identifier dans l'architecture d'un programme des micro-architectures similaires à des motifs de conception pour expliquer les choix de conception ?




Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

# Problème

Comment identifier dans l'architecture d'un programme des micro-architectures similaires à des motifs de conception pour expliquer les choix de conception ?




Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

# État de l'art (1/2)

- Programmation logique [Wuyts98]
  - Simplicité de mise en œuvre
  - Intégration à *Visual Works*
- Programmation par contraintes [Rich90, Quilici97]
  - Représentation des motifs
  - Passage à l'échelle
- Analyses syntaxiques [Alencar95, Brown96, Hedin97, Albin03]
  - Efficacité spatiale et temporelle

# État de l’art (2/2)

- Identification des micro-architectures
  - Exactes
  - Similaires
- Interactions avec les mainteneurs

# Objectifs

- Représentation des motifs
- Pas de descriptions a priori des similarités recherchées
  - Rigoureuse → catalogue
  - Lointaine → justifications
- Interaction avec le mainteneur
- Passage à l'échelle
- Solution uniforme

# Proposition

- Programmation par contraintes [Tsang93]
  - Représentation des motifs
  - Passage à l'échelle
  - Solution uniforme

  avec explications [Jussien01]
  - Pas de description a priori des similarités
  - Identification guidée par le mainteneur

# Problème de satisfaction de contraintes (PSC) [Montanari74] (1/6)

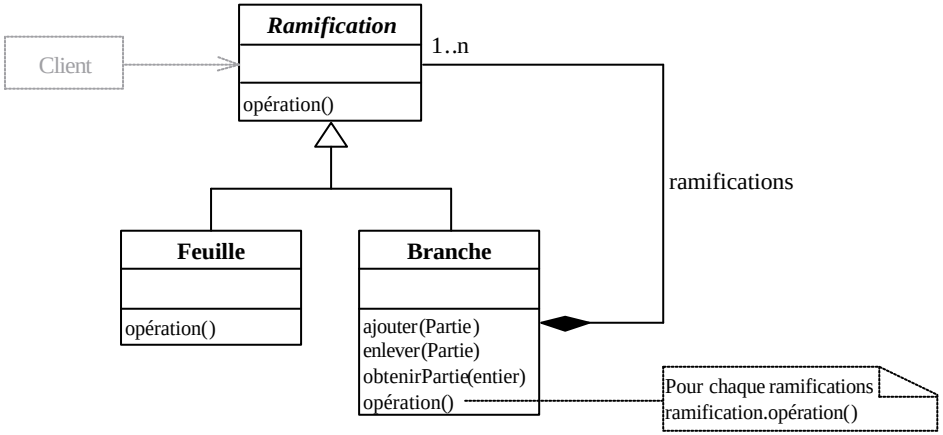
- Ensemble $\langle V, C, D \rangle$
  - Variables $V = \{v_1, \ldots, v_n\}$
  - Contraintes $C = \{C_1, \ldots, C_e\}$
  - Domaines des variables $D = \{D_1, \ldots, D_n\}$

- Solution
  - Résolution par propagation des contraintes

# PSC (2/6)

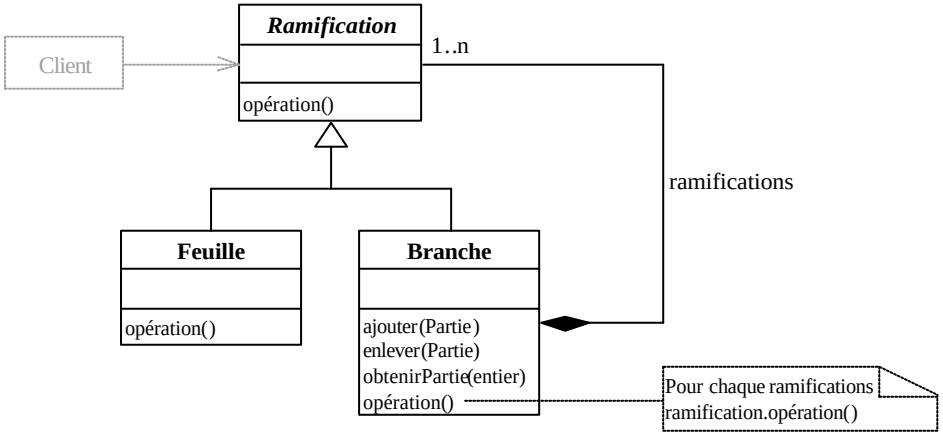
- PSC déduit de
  - Modèle du motif de conception
    - Participants → variables
    - Relations entre participants → contraintes
  - Modèle de l'architecture du programme
    - Classes du programme → domaine
    - Relations entre les classes du programme → sémantique effective des contraintes

# PSC (3/6)



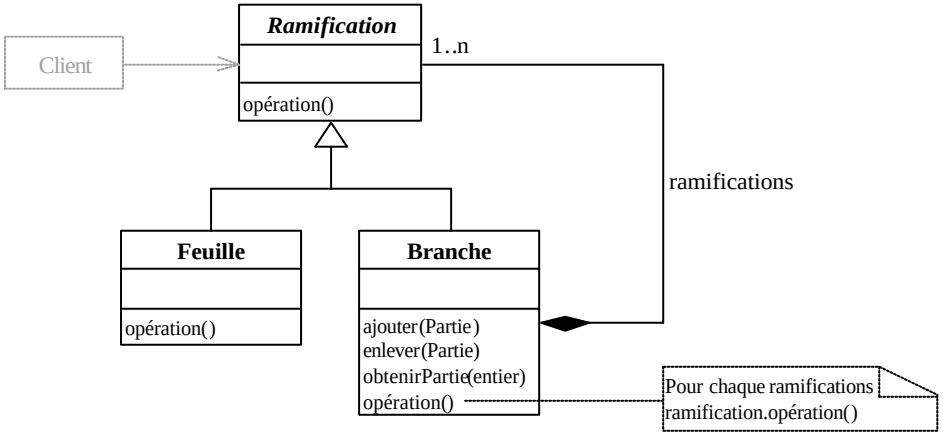

- Motif de conception Composite
  - Trois participants → trois variables
    - *ramification*
    - *feuille*
    - *branche*

# PSC (4/6)



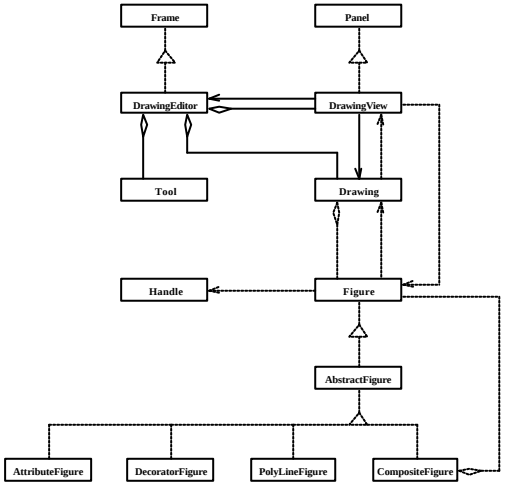

- Motif de conception Composite
  - Relations entre participants → contraintes
    - *branche* < *ramification*
    - *feuille* < *ramification*
    - *branche* ▶ *ramification*
    - Différences deux-à-deux

# PSC (4/6)




- Motif de conception Composite
  - Relations entre participants → contraintes
    - *branche < ramification*
    - *feuille < ramification*
    - *branche ▶ ramification*
    - Différences deux-à-deux

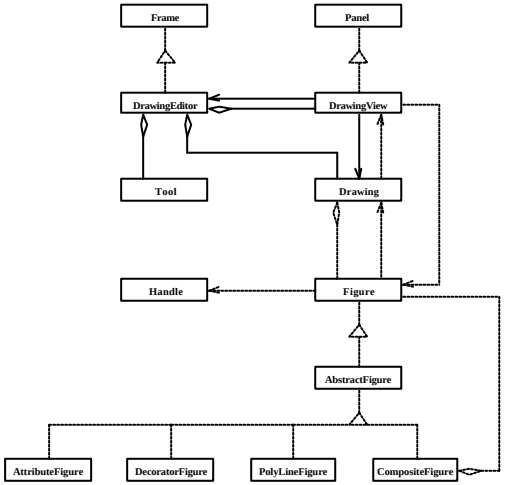
# PSC (5/6)




- Architecture du programme *JHotDraw*
  - Classes du programme → domaine
    - `DrawingEditor`
    - `DrawingView`
    - `Tool`
    - `Drawing`
    - …

    } + Relations entre les classes, attributs

# PSC (5/6)




- Architecture du programme *JHotDraw*
  - Classes du programme → domaine
    - `DrawingEditor`
    - `DrawingView`
    - `Tool`
    - `Drawing`
    - …

} + Relations entre les classes, attributs

# PSC (6/6)

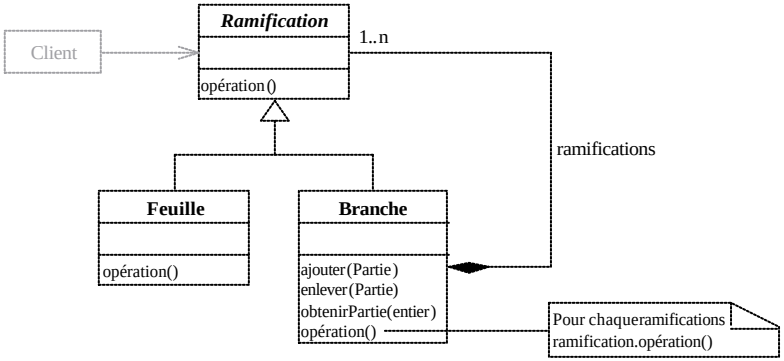
- Identification des micro-architectures similaires au motif de conception Composite

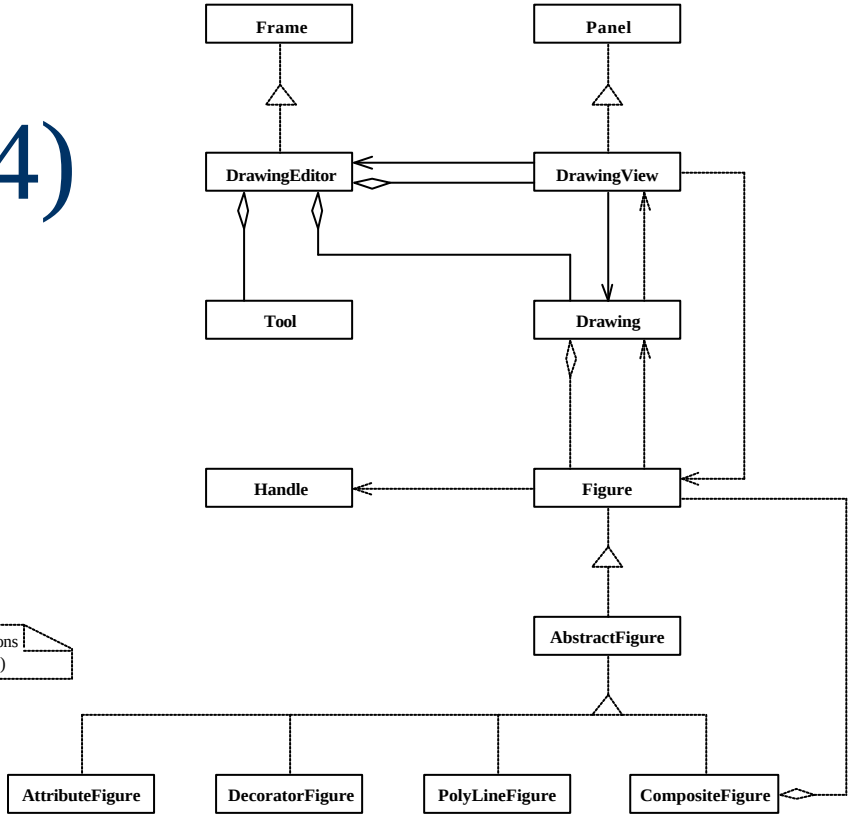
$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Solution (1/4)



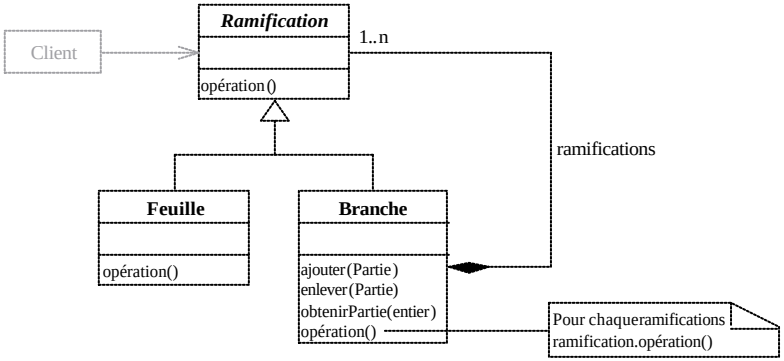



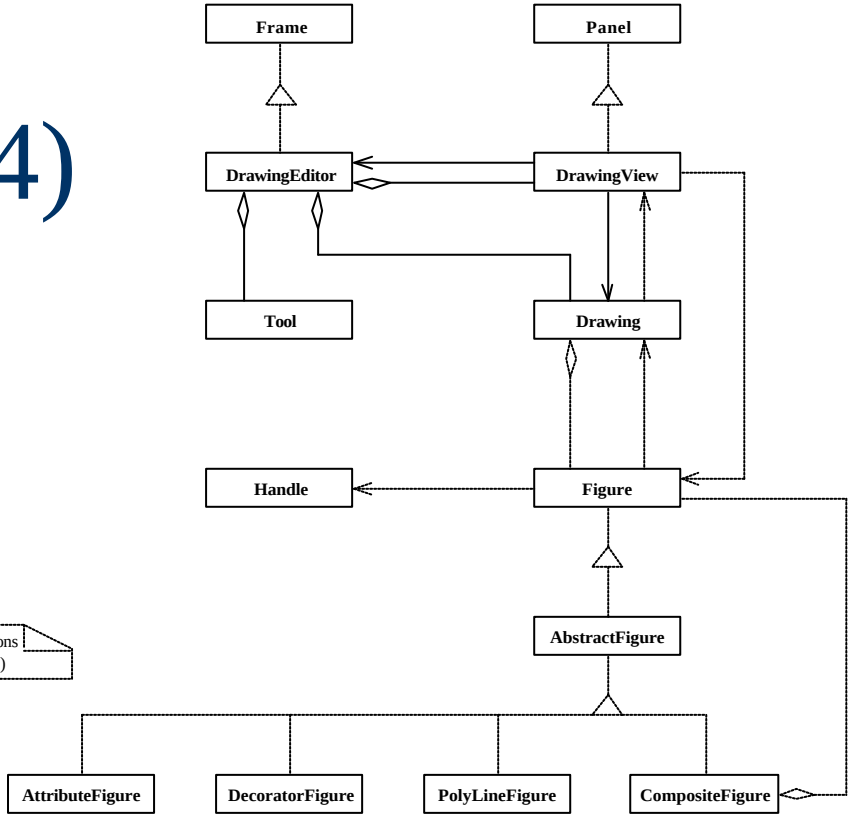

$V = \{ramification, branche, feuille\}$

$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Solution (1/4)



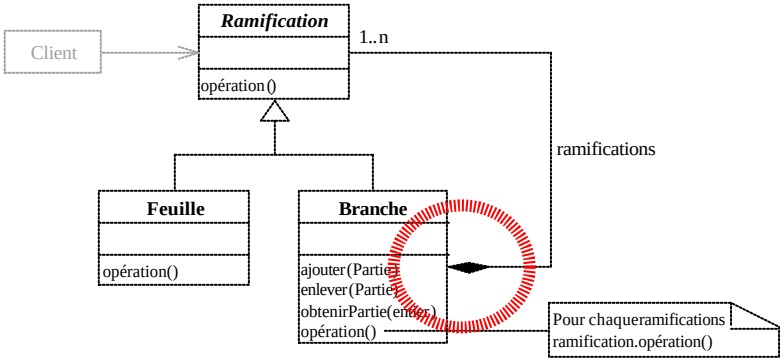



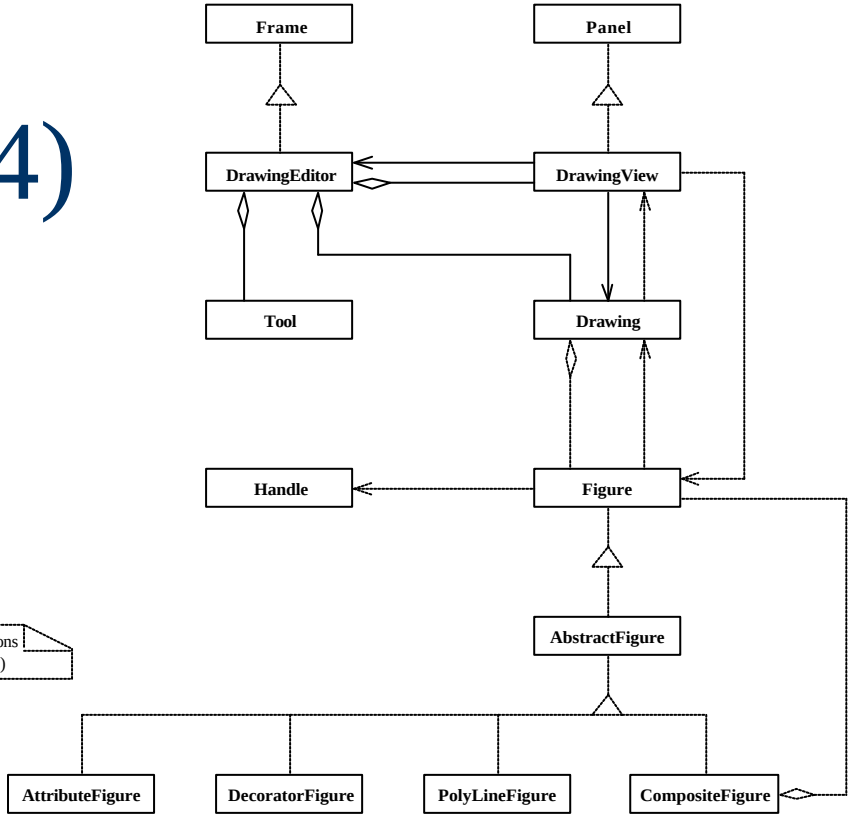

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Solution (1/4)



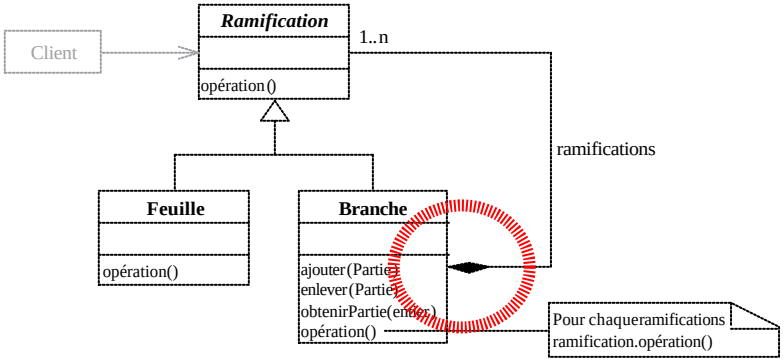



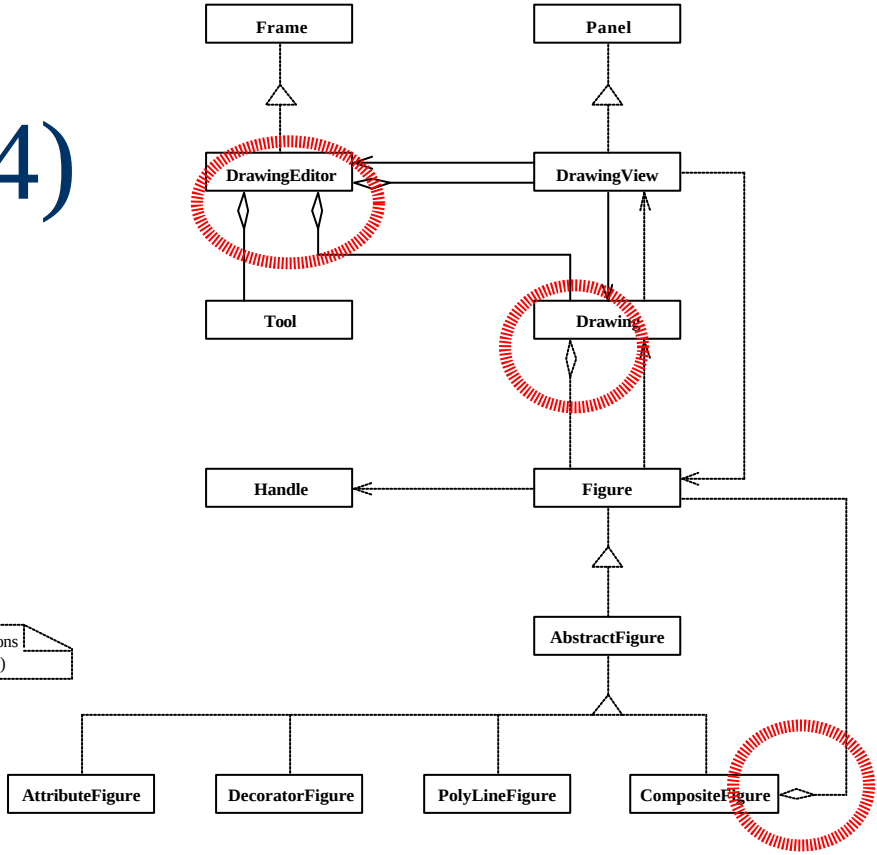

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Solution (1/4)




$V = \{ramification, branche, feuille\}$

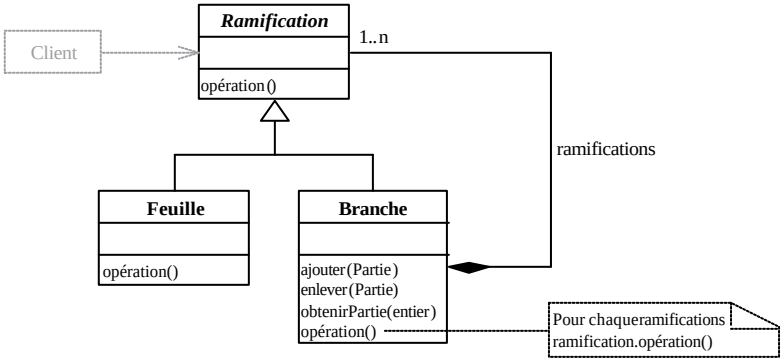
$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

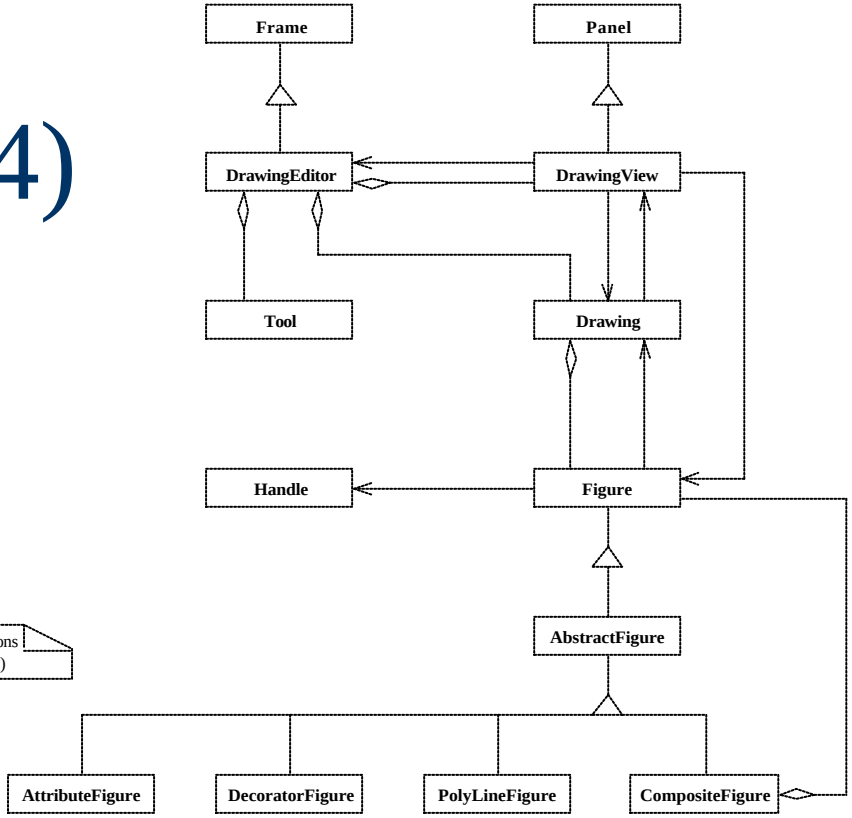
$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Solution (2/4)

- Explication de contradiction
  - *branche* ▸ *ramification*
- Pas de solution avec cette contrainte

# Solution (3/4)



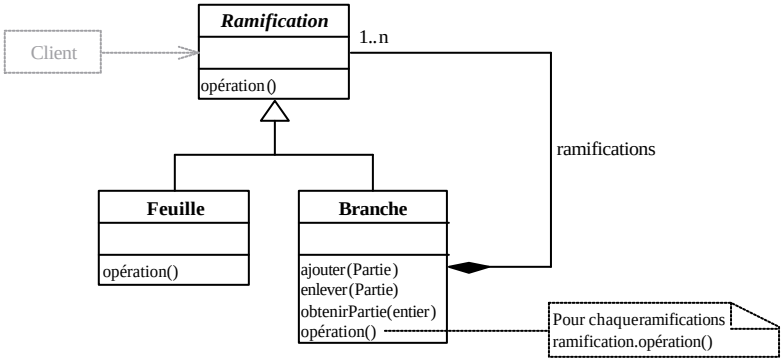



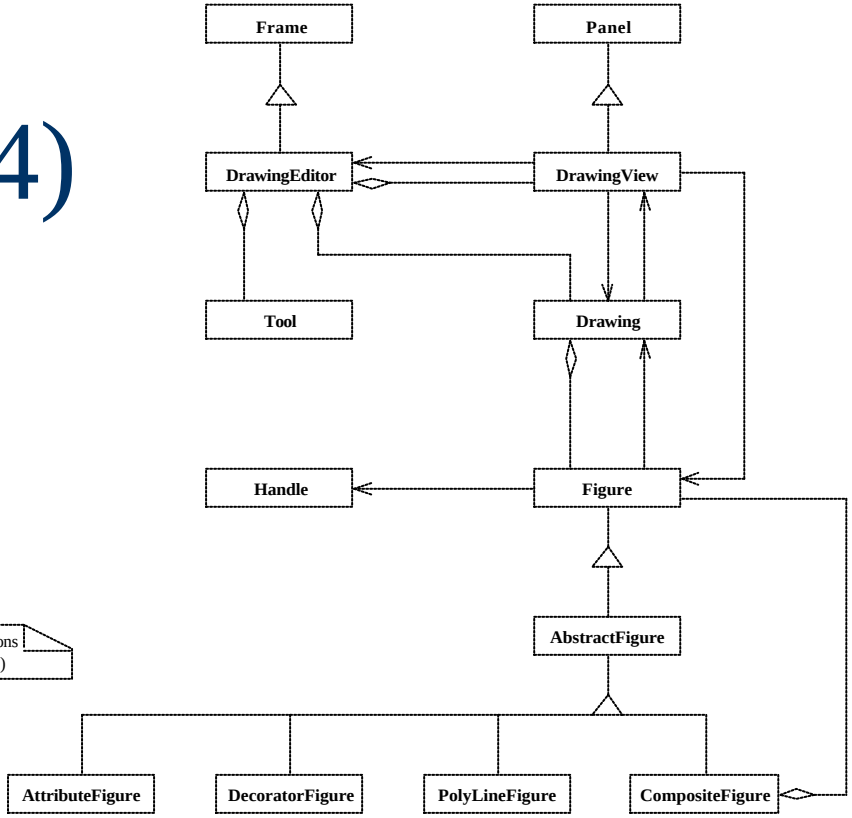

$V = \{ramification, branche, feuille\}$

$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Solution (3/4)



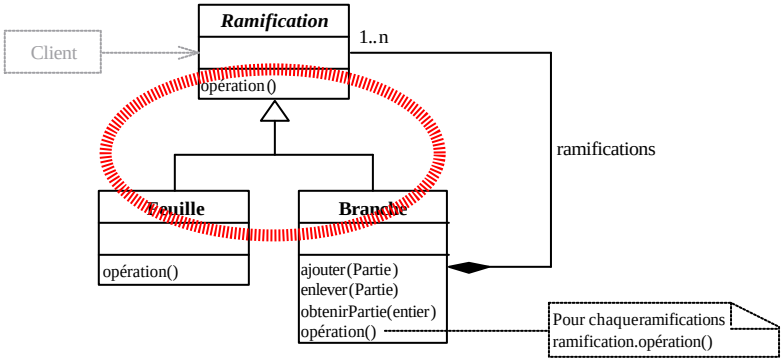



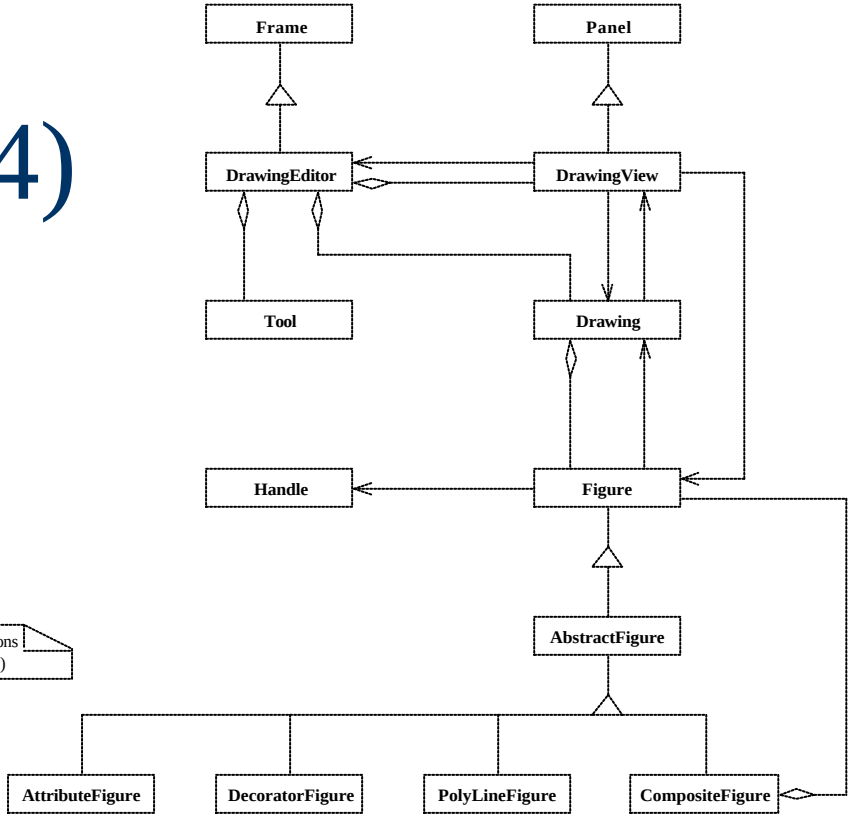

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Solution (3/4)



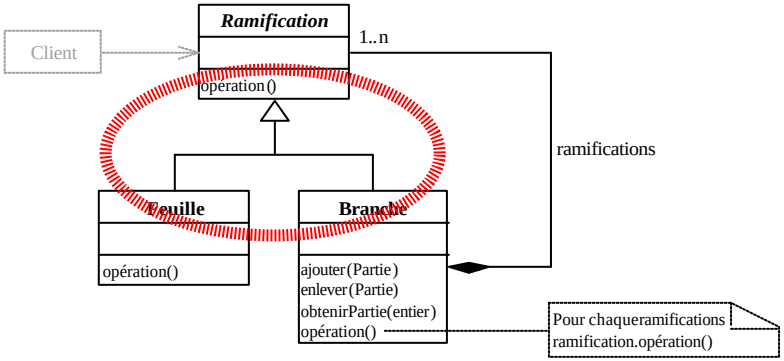



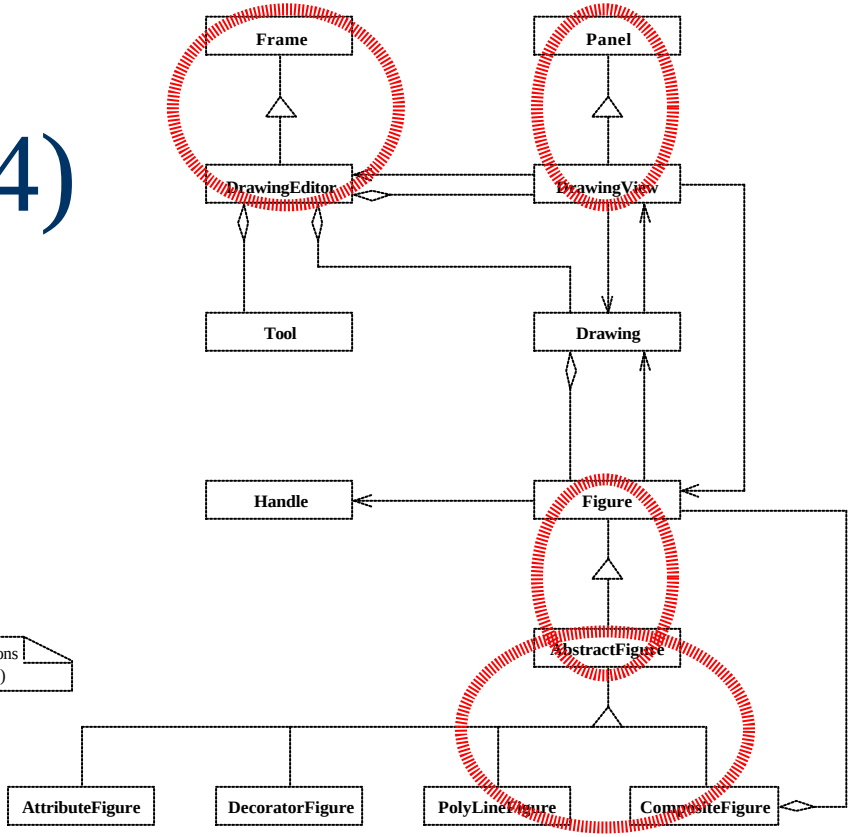

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Solution (3/4)



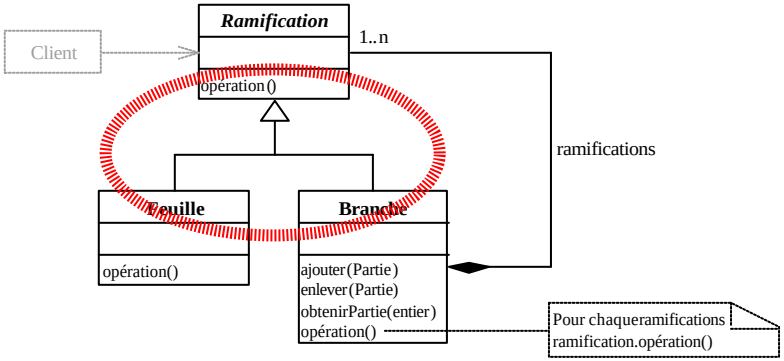



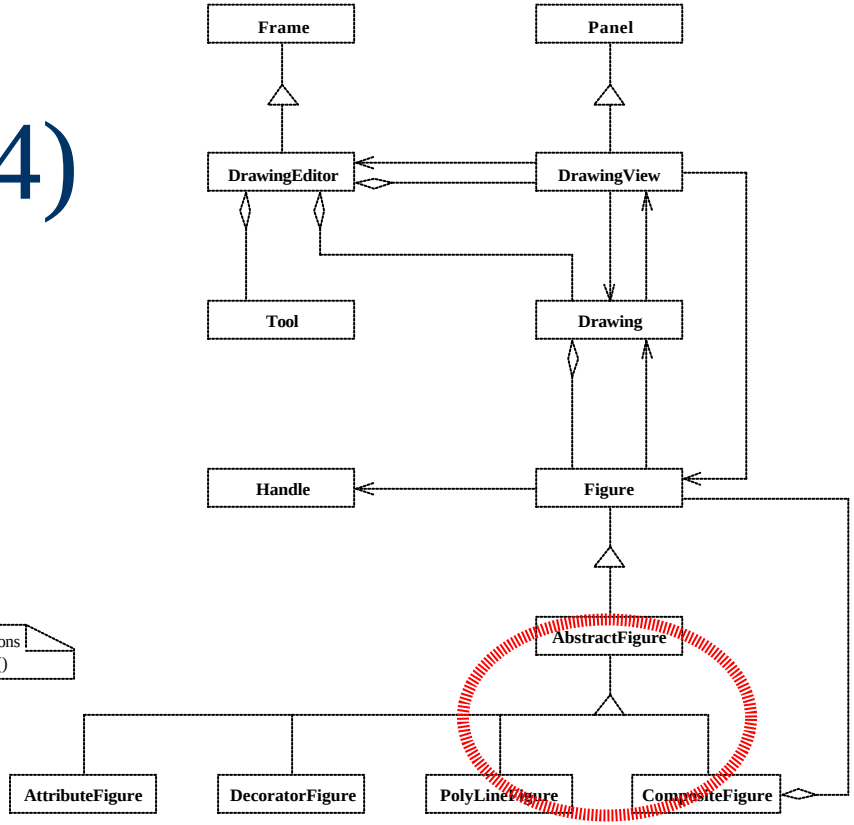

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Solution (3/4)



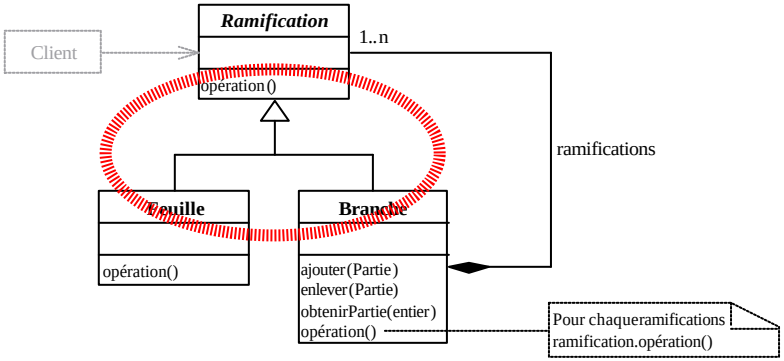



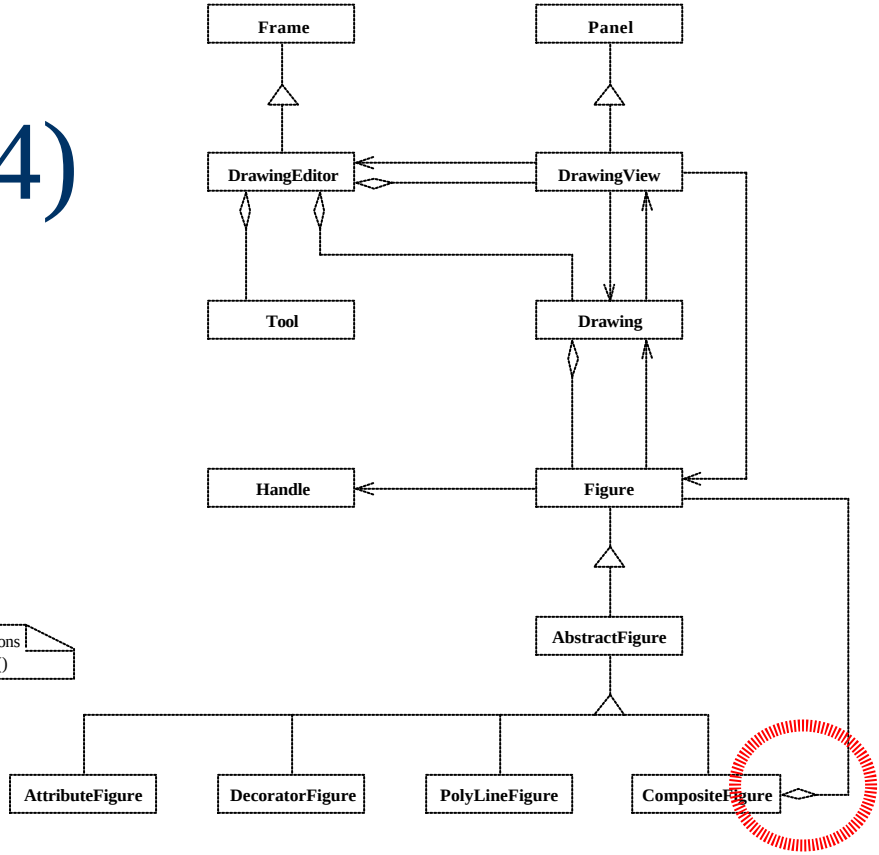

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Solution (3/4)




$V = \{ramification, branche, feuille\}$

$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Solution (4/4)

- Explication de contradiction
  - *branche* < *ramification*
  - *feuille* < *ramification*
  - *branche* ▶ *ramification*

- Pas de solution avec ces contraintes

# Programmation par contraintes avec explications (PPCE) [Jussien01] (1/3)

- Programmation par contraintes
  - Opérations de retraits ou de restauration de valeurs des domaines
- Explications
  - Sous-ensemble des opérations qui mène à une solution ou à une contradiction
  - Dynamicité de la résolution

# PPCE (2/3)

- Solution

  $\exists$ opérations de retraits et de restauration des valeur des domaines | toutes les contraintes soient satisfaites

- Contradiction
  - Sous-ensemble des opérations qui mène à une contradiction
  - Ensemble des explications

# PPCE (3/3)

- Relaxation du problème
- Relaxation des contraintes

# Relaxation du problème [Petit02] (1/2)

- Retirer la contrainte entraînant une contradiction
  - *branche* ▶ *ramification*

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation du problème [Petit02] (1/2)

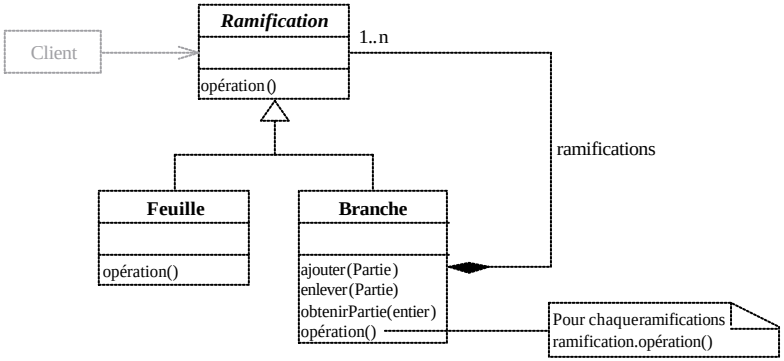
- Retirer la contrainte entraînant une contradiction
  - *branche* ▸ *ramification*

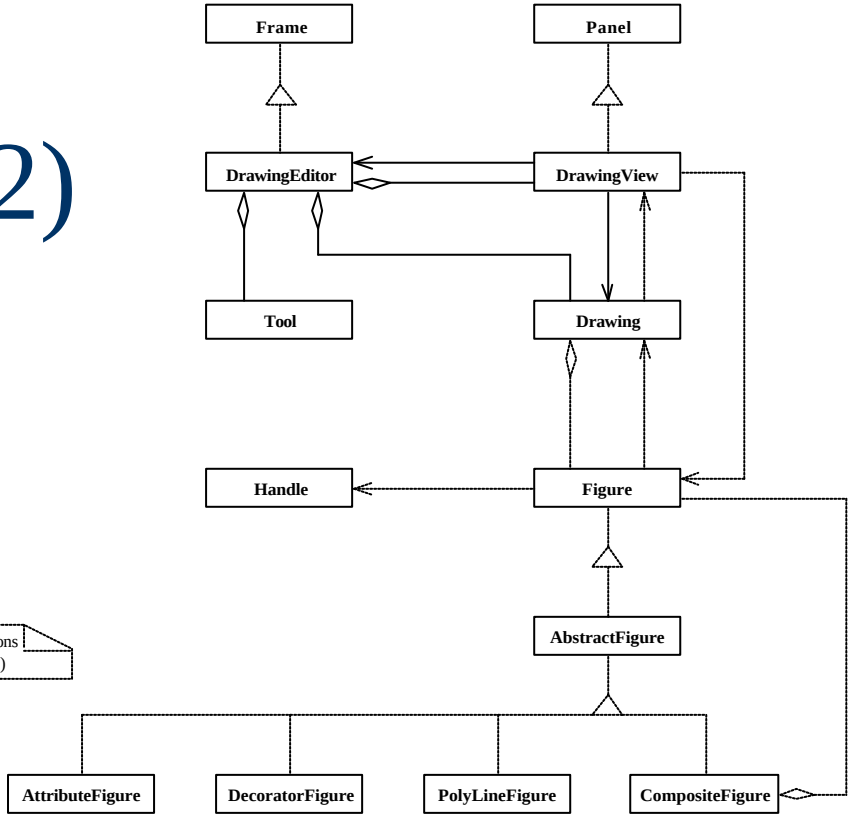
$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, ~~*branche* ▸ *ramification*~~}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation du problème (2/2)



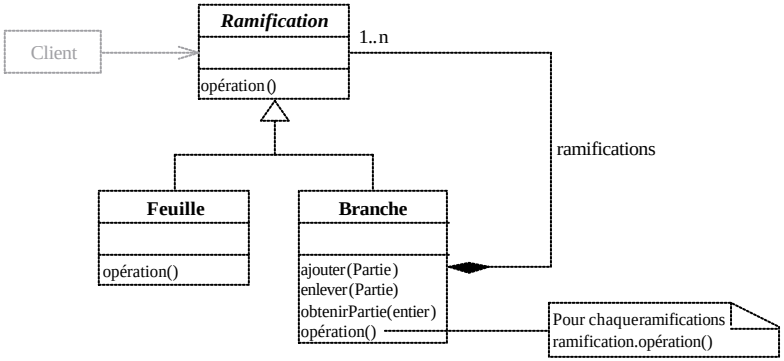



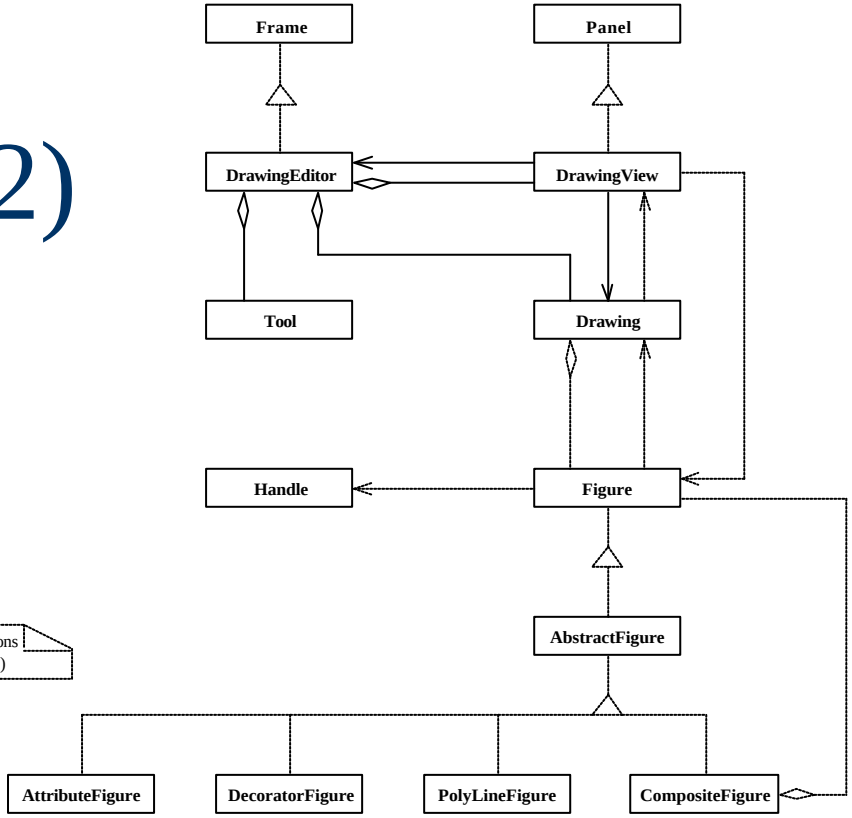

$V = \{ramification, branche, feuille\}$

$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Relaxation du problème (2/2)



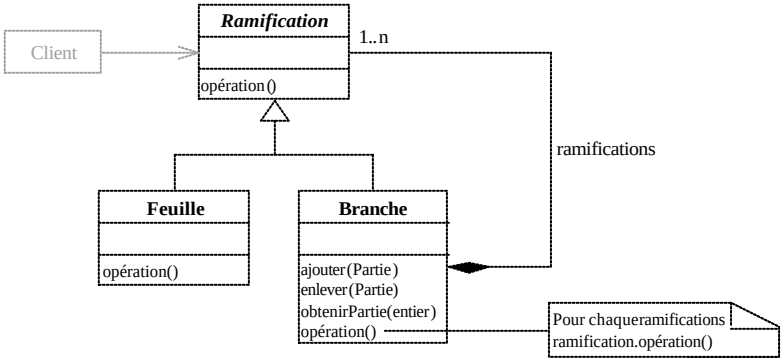



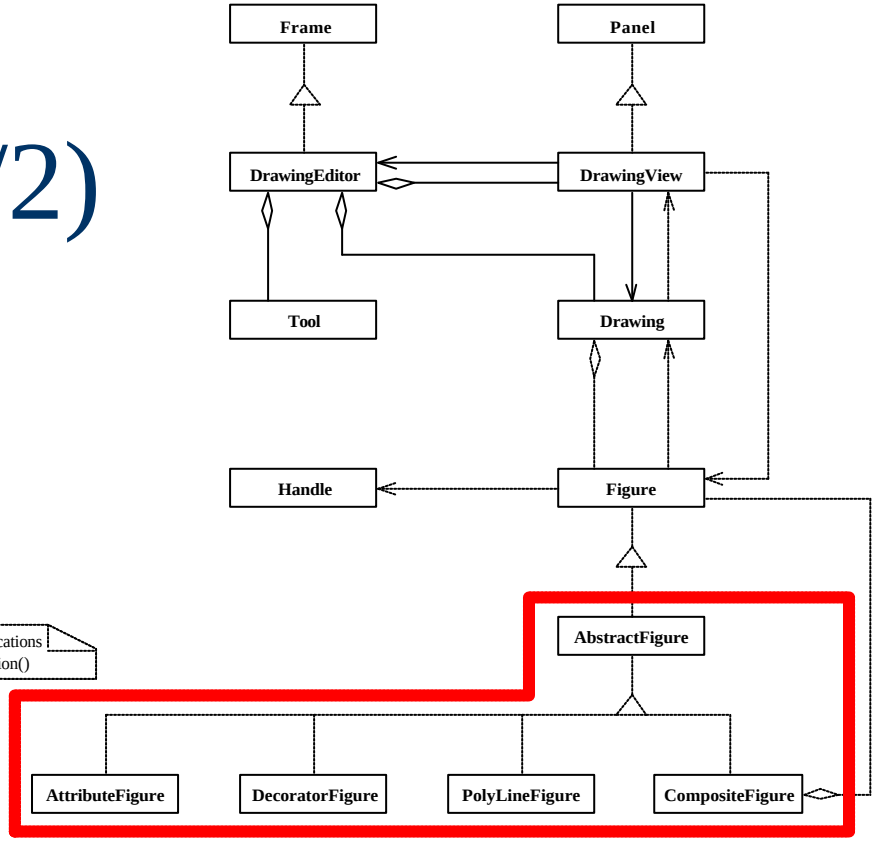

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, ~~*branche* ▶ *ramification*~~}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Relaxation du problème (2/2)




$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, ~~*branche* ▶ *ramification*~~}

$D$ = {⟨`DrawingEditor`, `Drawing`...⟩}

# Relaxation des contraintes (1/3)

- Remplacer une contrainte par un autre *sémantiquement* moins forte
  - *branche* ▶ *ramification*
  - *branche* ▷ *ramification*

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (1/3)

- Remplacer une contrainte par un autre *sémantiquement* moins forte
  - *branche* ▶ *ramification* Composition
  - *branche* ▷ *ramification*

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (1/3)

- Remplacer une contrainte par un autre *sémantiquement* moins forte
  - *branche* ▶ *ramification* Composition
  - *branche* ▷ *ramification* Agrégation

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (1/3)

- Remplacer une contrainte par un autre *sémantiquement* moins forte
  - *branche* ▶ *ramification* Composition
  - *branche* ▷ *ramification* Agrégation

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▷ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (2/3)

- Remplacer une contrainte par un autre *sémantiquement* différente
  - *branche* < *ramification*
  - *branche* ≪ *ramification*

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (2/3)

- Remplacer une contrainte par un autre *sémantiquement* différente
  - *branche* < *ramification* Héritage direct
  - *branche* ≪ *ramification*

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (2/3)

- Remplacer une contrainte par un autre *sémantiquement* différente
  - *branche* < *ramification* Héritage direct
  - *branche* ≪ *ramification* Héritage indirect

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* < *ramification*, *feuille* < *ramification*, *branche* ▶ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (2/3)

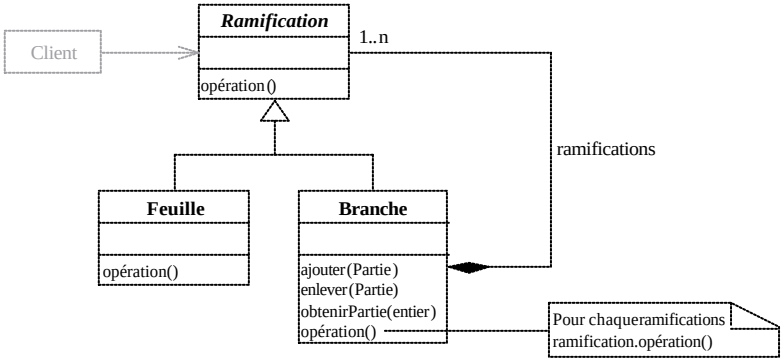
- Remplacer une contrainte par un autre *sémantiquement* différente
  - *branche* < *ramification* Héritage direct
  - *branche* ≪ *ramification* Héritage indirect

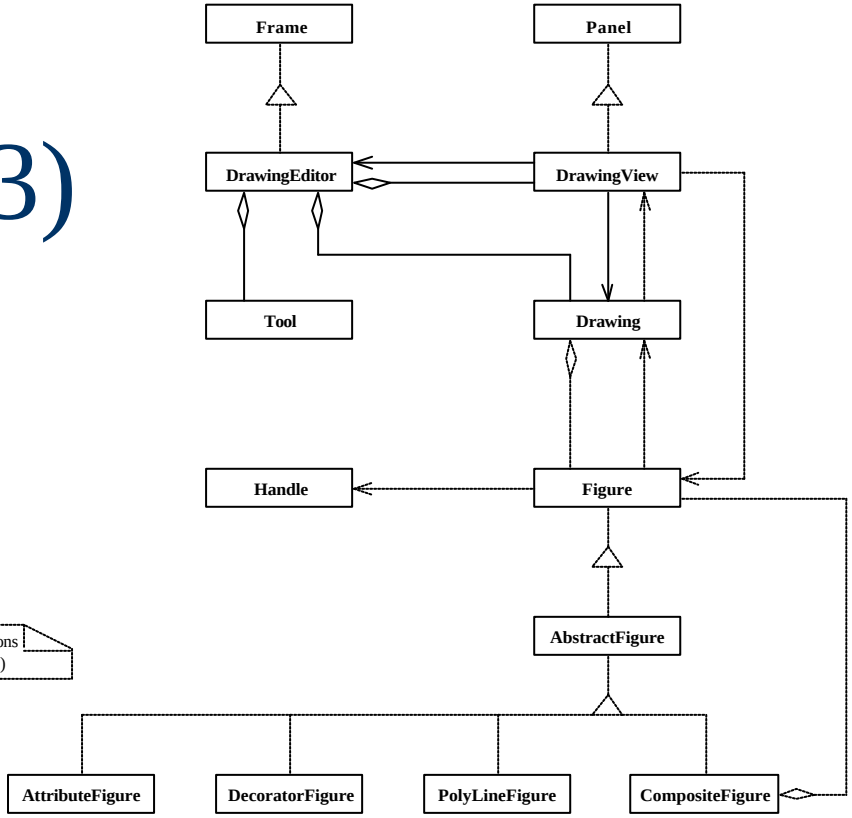
$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* ≪*ramification*, *feuille* ≪ *ramification*, *branche* ▸ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (3/3)



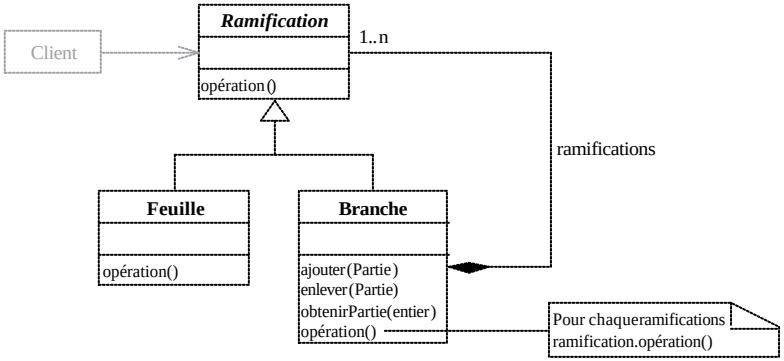



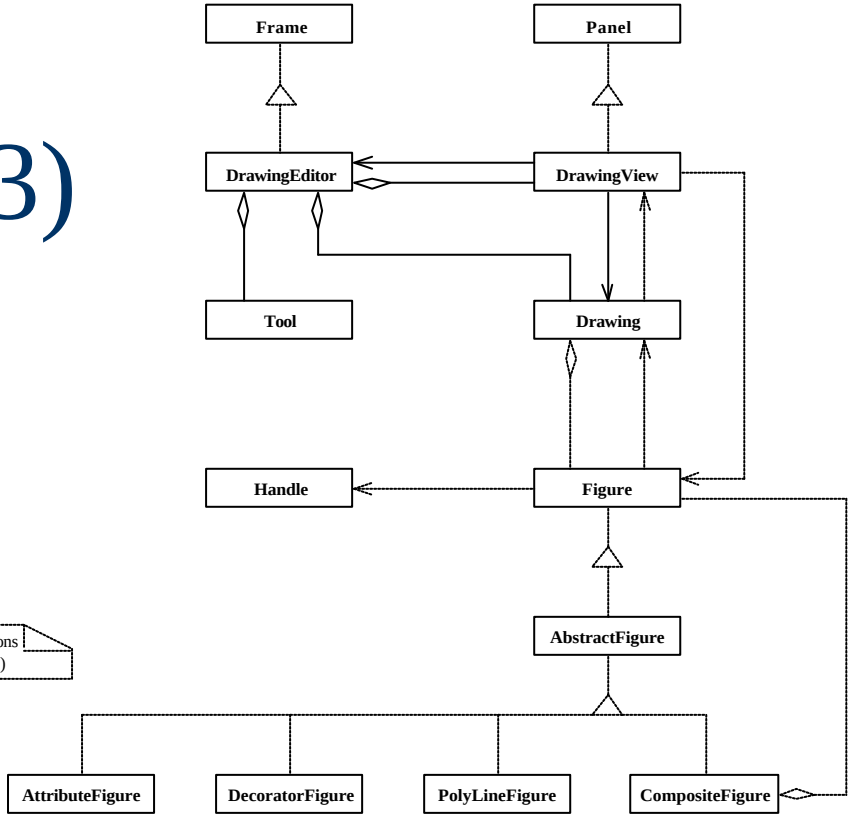

$V = \{ramification, branche, feuille\}$

$C = \{branche < ramification, feuille < ramification, branche \blacktriangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Relaxation des contraintes (3/3)



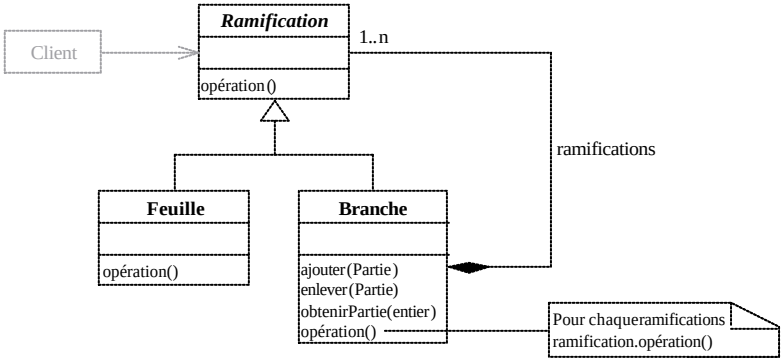



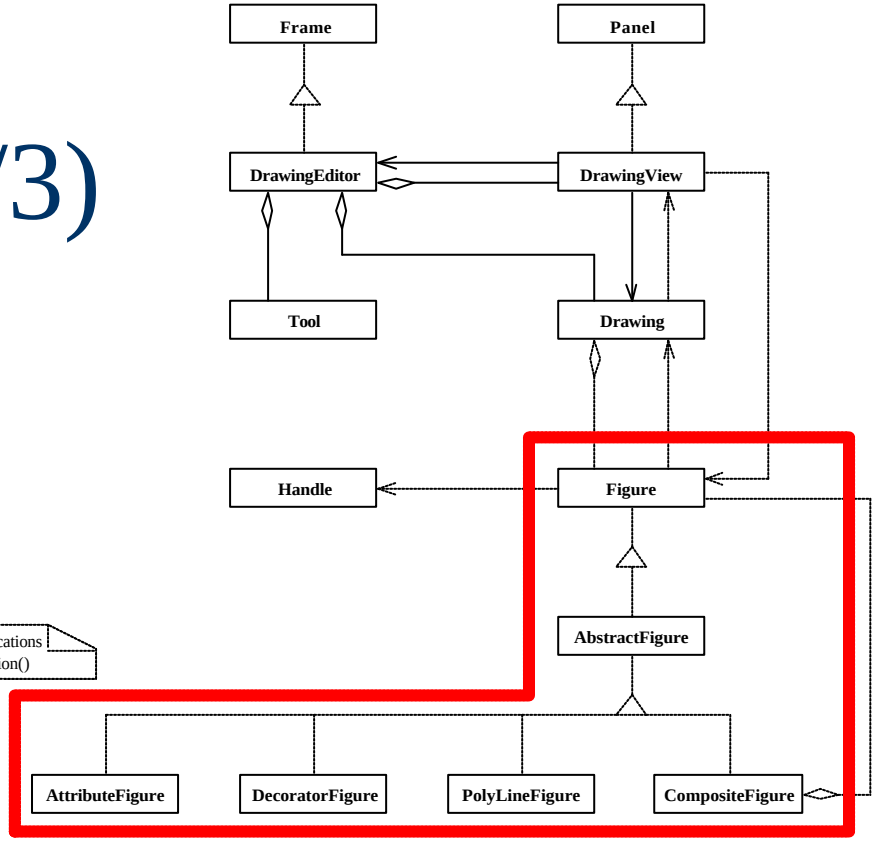

$V$ = {*ramification*, *branche*, *feuille*}

$C$ = {*branche* ≪ *ramification*, *feuille* ≪ *ramification*, *branche* ▷ *ramification*}

$D$ = {⟨`DrawingEditor`, `Drawing`…⟩}

# Relaxation des contraintes (3/3)




$V = \{ramification, branche, feuille\}$

$C = \{branche \ll ramification, feuille \ll ramification, branche \triangleright ramification\}$

$D = \{\langle \texttt{DrawingEditor}, \texttt{Drawing} \ldots \rangle\}$

# Technique, outils

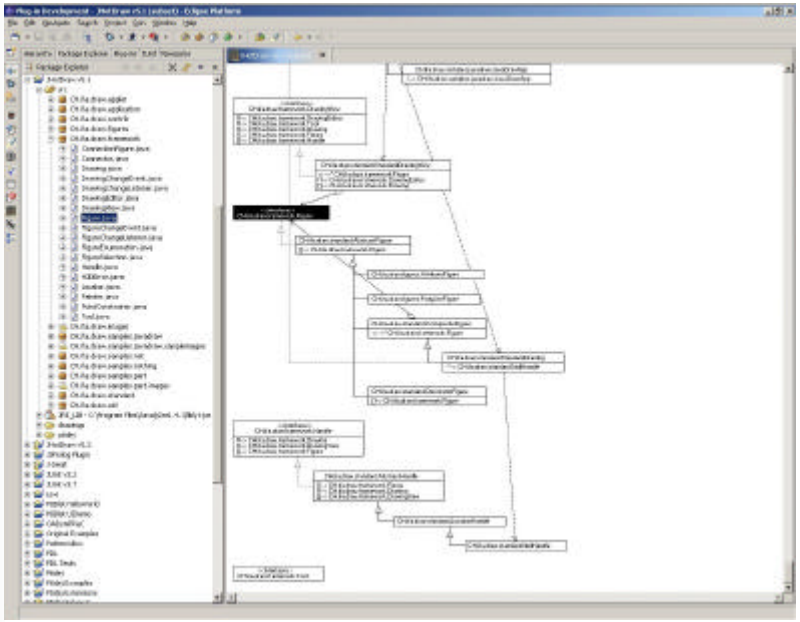
- Solveur de contraintes : *PtidejSolver*
  - Interactif
  - Automatique (⚠ complexité)

- Bibliothèque de contraintes : *PtidejLibrary*
  - Héritage direct, indirect
  - Association, agrégation, composition
  - Utilisation, …

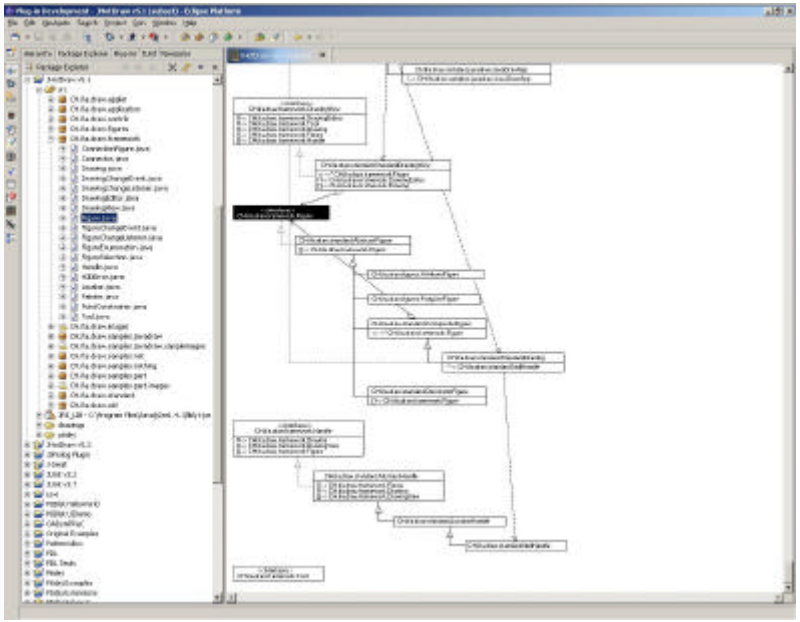
# Choix de conception (1/2)

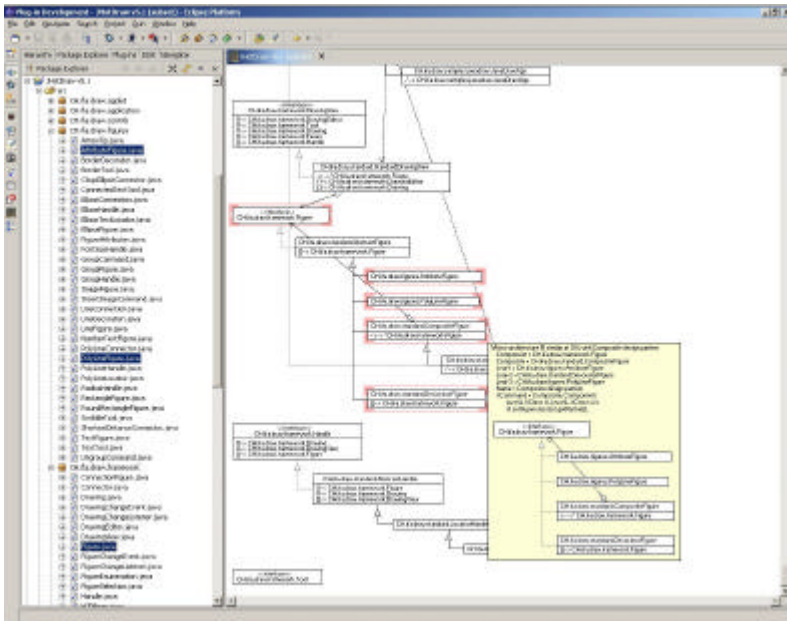
- *PtidejSolver*, *PtidejLibrary*
  - Identification des micro-architectures similaires à un motif de conception
  - Justifications des micro-architectures identifiées
  - Interaction avec les mainteneurs

# Choix de conception (2/2)



# Choix de conception (2/2)

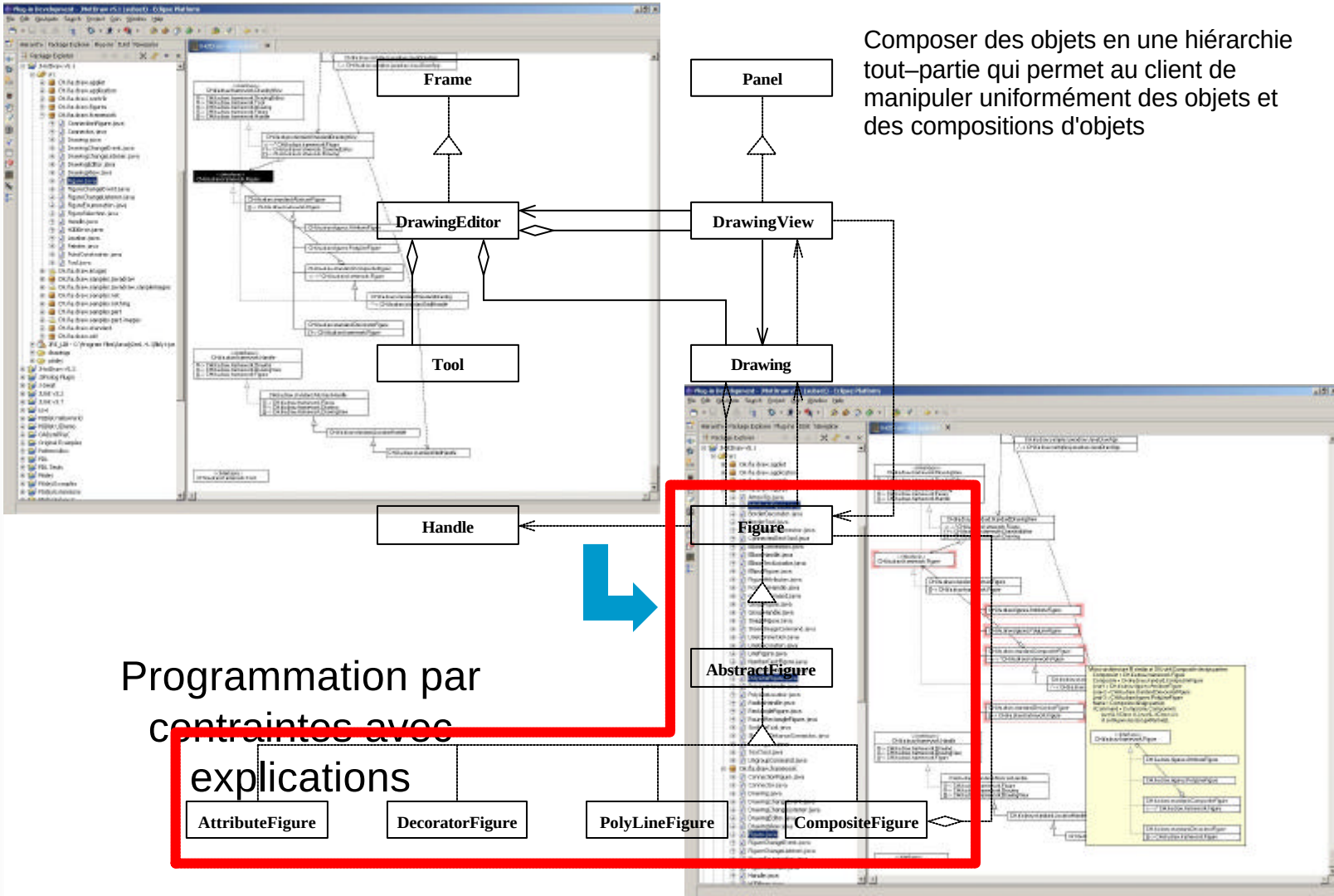




Programmation par contraintes avec explications

# Choix de conception (2/2)

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets




Programmation par contraintes avec explications

# Plan

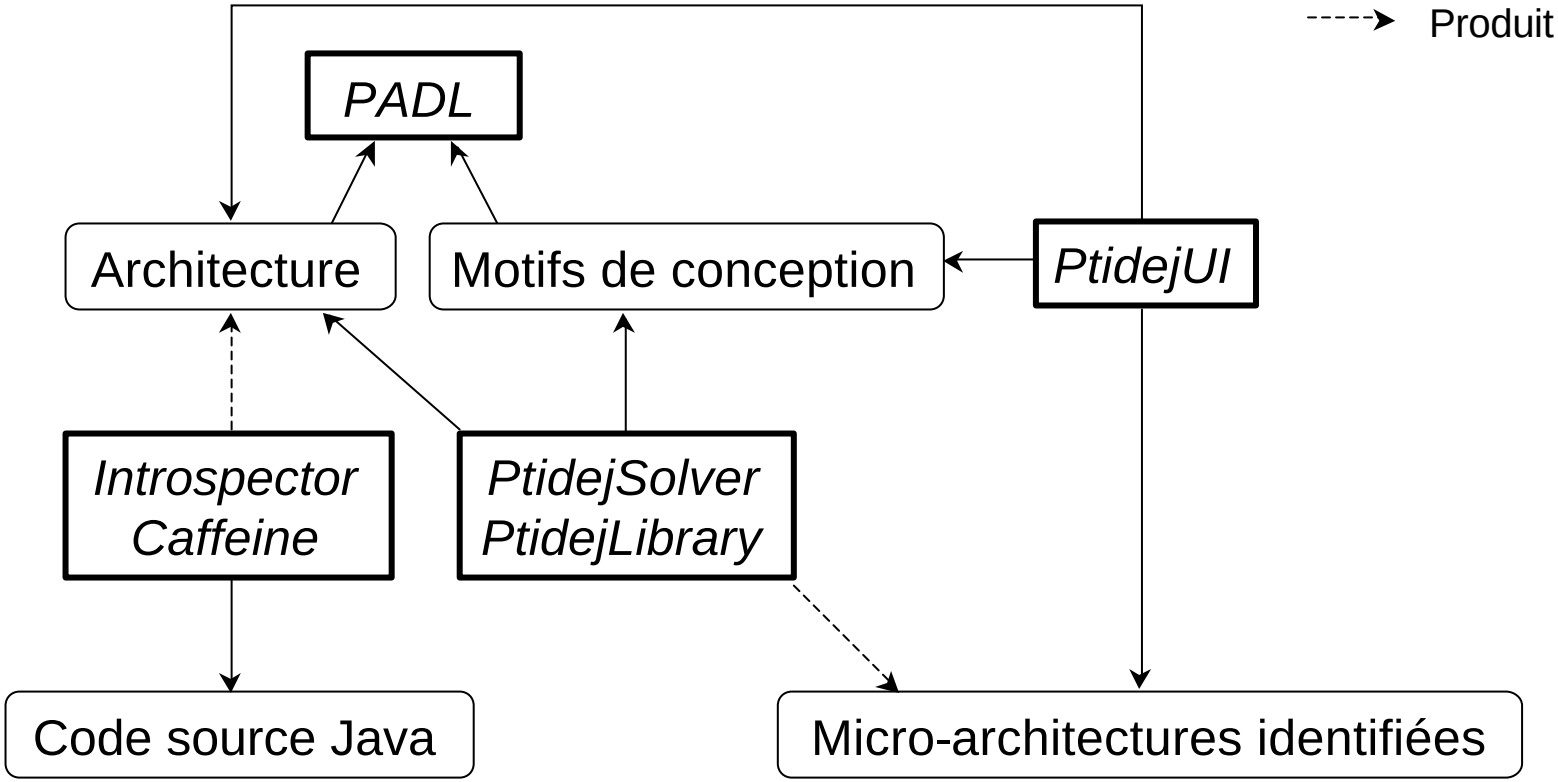
- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l'architecture d'un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Un cadre pour la traçabilité des motifs de conception (1/4)

- Métamodélisation
  - *PADL*
- Analyse de programme
  - *Introspector*, *Caffeine*
- Programmation par contraintes
  - *PtidejSolver*, *PtidejLibrary*
- Visualisation
  - *PtidejUI*

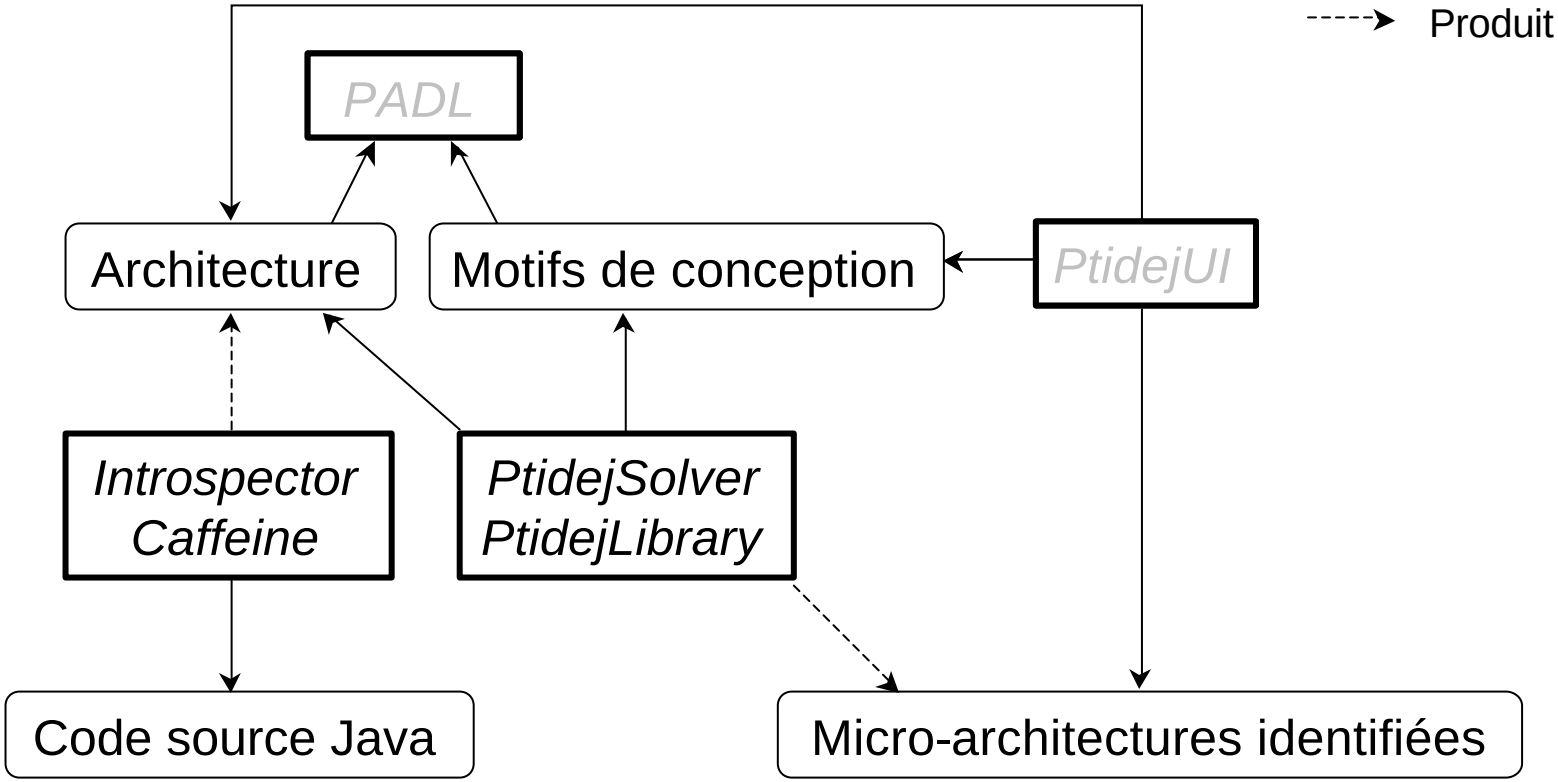
# Un cadre pour la traçabilité des motifs de conception (2/4)

- *Ptidej* (*Pattern Trace Identification, Detection, and Enhancement in Java*)



# Un cadre pour la traçabilité des motifs de conception (2/4)

- *Ptidej* (*Pattern Trace Identification, Detection, and Enhancement in Java*)

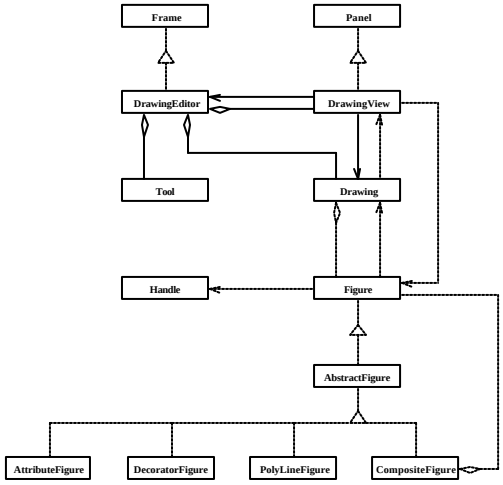


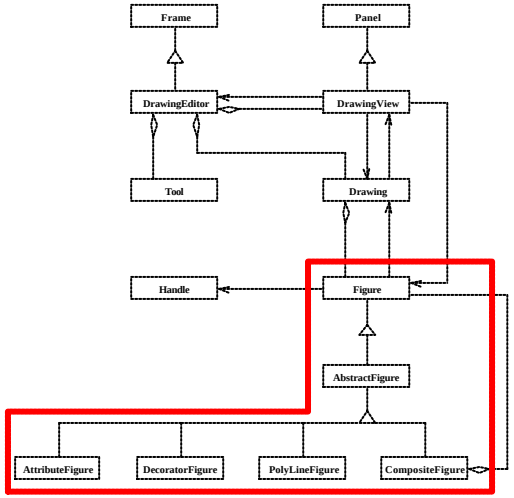
# Un cadre pour la traçabilité des motifs de conception (3/4)

- *Ptidej*, ~700 heures de développement
  - Java
    - 20 projets, 132 paquetages, 669 classes, 72 interfaces, 79 330 lignes
  - Claire
    - 41 fichiers, 45 classes, 235 méthodes, 7 835 lignes
  - Prolog
    - 6 fichiers, 76 prédicats, 524 lignes

# Un cadre pour la traçabilité des motifs de conception (4/4)

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

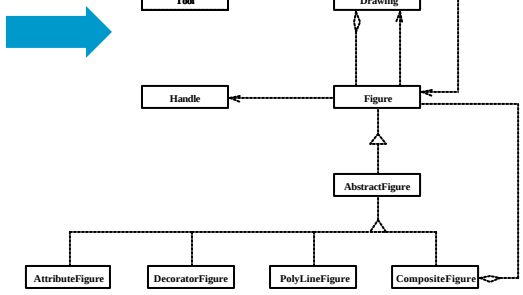


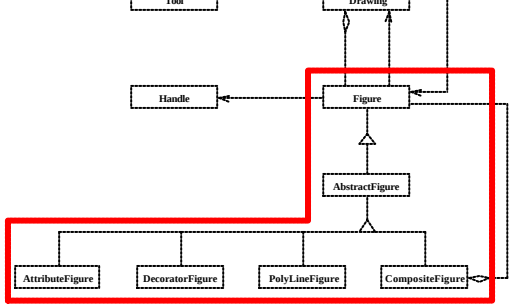



# Un cadre pour la traçabilité des motifs de conception (4/4)

*Définitions*
*Introspector*
*Caffeine*

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

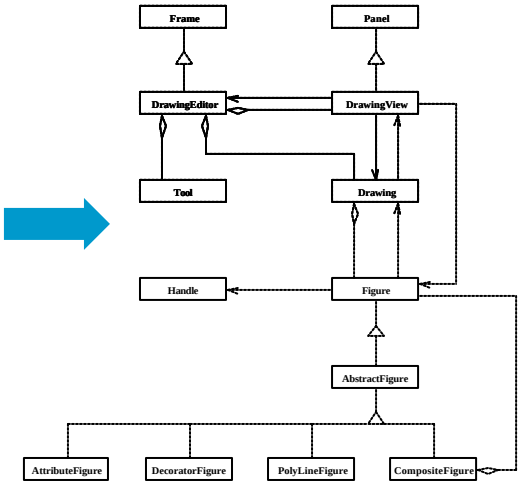


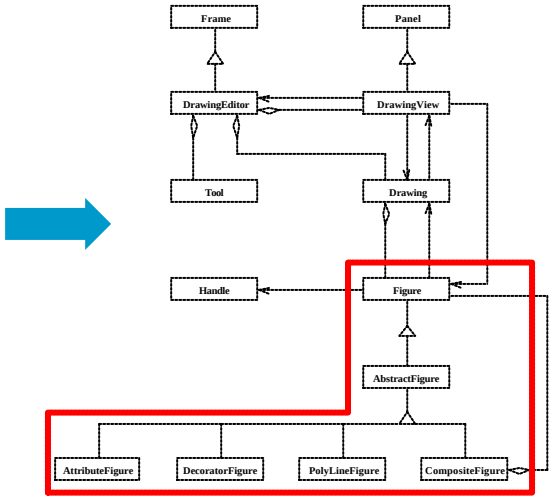



# Un cadre pour la traçabilité des motifs de conception (4/4)

*PtidejSolver*
*PtidejLibrary*
*PtidejUI*

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

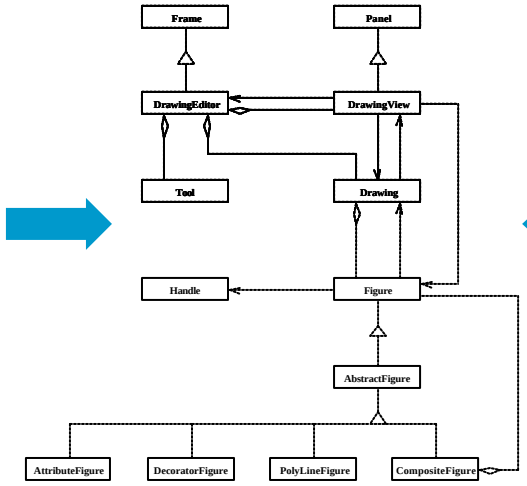


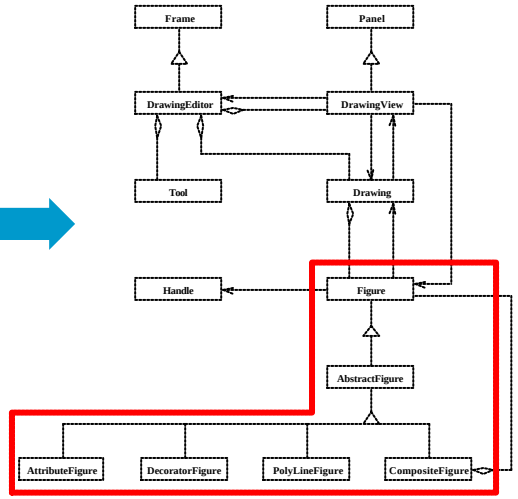



# Un cadre pour la traçabilité des motifs de conception (4/4)

*Eclipse*

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets

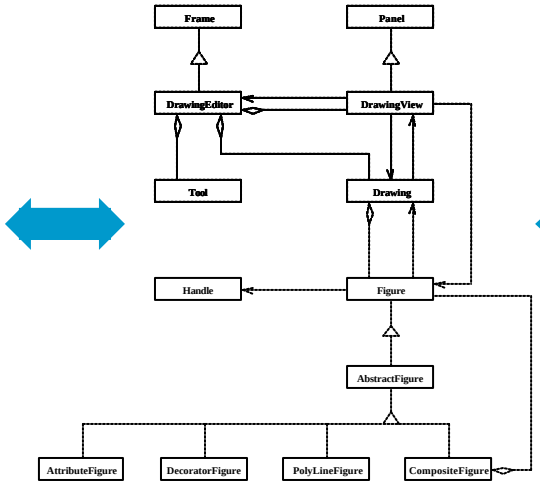


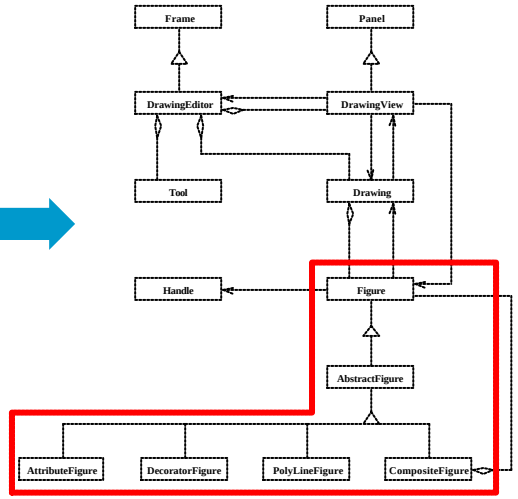



# Un cadre pour la traçabilité des motifs de conception (4/4)

*Eclipse*

Composer des objets en une hiérarchie tout–partie qui permet au client de manipuler uniformément des objets et des compositions d'objets



# Plan

- Contexte
  - Identification des choix de conception
- Problèmes
  - Obtention de l’architecture d’un programme
  - Identification des choix de conception
- Contributions
- Évaluation, perspectives

# Évaluation (1/4)

- Nombre de micro-architectures identiques à un motif de conception
- Nombre et pertinence des micro-architectures similaire à un motif de conception
- Utilisabilité (temps de calcul)

# Évaluation (2/4)

| Patron de conception | Programmes | NLC | Existants | Documentation (PatternsBox) | | | | Compréhension (Ptidej) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Hits | Manqués | Erronés | t (sec.) | Complets | | | Affaiblis | t (sec.) |
| | | | | | | | | Hits | Manqués | Erronés | Hits | |
| Composite | java.awt.* | 121 | 1 | 1 | | | 1 | 1 | | | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 1 | | | 0,3 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | | 1 | | 0,4 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 1,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| | | 610 | 3 | 2 | 1 | 0 | 0,7 | 3 | 0 | 0 | 20 | 48,2 |

# Évaluation (2/4)

| Patron de conception | Programmes | NLC | Existants | Documentation (PatternsBox) | | | | Compréhension (Ptidej) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Hits | Manqués | Erronés | t (sec.) | Complets | | | Affaiblis | t (sec.) |
| | | | | | | | | Hits | Manqués | Erronés | Hits | |
| Composite | java.awt.* | 121 | 1 | 1 | | | 1 | 1 | | | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 1 | | | 0,3 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | | 1 | | 0,4 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 1,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| | | 610 | 3 | 2 | 1 | 0 | 0,7 | 3 | 0 | 0 | 20 | 48,2 |

# Évaluation (2/4)

| Patron de conception | Programmes | NLC | Existants | Documentation (PatternsBox) | | | | Compréhension (Ptidej) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Hits | Manqués | Erronés | t (sec.) | Complets | | | Affaiblis | t (sec.) |
| | | | | | | | | Hits | Manqués | Erronés | Hits | |
| Composite | java.awt.* | 121 | 1 | 1 | | | 1 | 1 | | | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 1 | | | 0,3 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | | 1 | | 0,4 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 1,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| | | 610 | 3 | 2 | 1 | 0 | 0,7 | 3 | 0 | 0 | 20 | 48,2 |

# Évaluation (3/4)

| Patrons de conception | Programmes | NLC | Existants | Documentation (PatternsBox) | | | | Compréhension (Ptidej) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Hits | Manqués | Erronés | t (sec.) | Complets | | | Affaiblis | t (sec.) |
| | | | | | | | | Hits | Manqués | Erronés | Hits | |
| Composite | java.awt.* | 121 | 1 | 1 | | | 1 | 1 | | | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 1 | | | 0,3 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | | 1 | | 0,4 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 1,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| Décorateur | java.awt.* | 121 | | | | | 0,2 | | | 1 | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 3 | | 2 | 0,4 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | 1 | | | 0,2 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 0,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| Méthode usine | java.awt.* | 121 | 3 | | | | 1,1 | 3 | | 8 | 1 | 7 |
| | JHotDraw | 155 | 2 | 2 | 1 | 1 | 0,5 | 2 | | 37 | 1 | 103,7 |
| | JUnit | 34 | | | | | | | | 1 | 1 | 23 |
| | JEdit | 248 | | | | | 0,1 | | | | 6 | 29,7 |
| | PatternsBox | 52 | | | | | 0,2 | | | 7 | 1 | 13,5 |
| Itérateur | java.awt.* | 121 | | | | | | | | 12 | | 75,6 |
| | JHotDraw | 155 | 3 | 3 | | | 0,1 | | 3 | 100 | | 231,1 |
| | JUnit | 34 | | | | | | | | 8 | | 22,7 |
| | JEdit | 248 | 1 | 1 | | | 0,1 | | | 1 | | 28,5 |
| | PatternsBox | 52 | | | | | | | | 79 | | 36,5 |
| Observateur | java.awt.* | 121 | | | | | 3,4 | 4 | | 4 | | 73,5 |
| | JHotDraw | 155 | 2 | 2 | | | 3,2 | 2 | 2 | 2 | | 61,4 |
| | JUnit | 34 | 4 | 4 | | | 2,5 | | | | | 19,9 |
| | JEdit | 248 | 3 | 3 | | | 13 | | 3 | | | 27,8 |
| | PatternsBox | 52 | 1 | 1 | | | 2,8 | 2 | 1 | 2 | | 31,5 |
| | | 610 | 24 | 22 | 2 | 3 | 1,5 | 18 | 9 | 262 | 50 | 50,7 |

# Évaluation (3/4)

| Patrons de conception | Programmes | NLC | Existants | Documentation (PatternsBox) | | | | Compréhension (Ptidej) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Complets | | | Affaiblis | |
| | | | | Hits | Manqués | Erronés | t (sec.) | Hits | Manqués | Erronés | Hits | t (sec.) |
| Composite | java.awt.* | 121 | 1 | 1 | | | 1 | 1 | | | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 1 | | | 0,3 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | | 1 | | 0,4 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 1,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| Décorateur | java.awt.* | 121 | | | | | 0,2 | | | 1 | 7 | 82,6 |
| | JHotDraw | 155 | 1 | 3 | | 2 | 0,4 | 1 | | | 3 | 65,5 |
| | JUnit | 34 | 1 | 1 | | | 0,2 | 1 | | | 7 | 22,9 |
| | JEdit | 248 | | | | | 0,4 | | | | | 29,5 |
| | PatternsBox | 52 | | | | | 0,3 | | | | 3 | 40,3 |
| Méthode usine | java.awt.* | 121 | 3 | | | | 1,1 | 3 | | 8 | 1 | 7 |
| | JHotDraw | 155 | 2 | 2 | 1 | 1 | 0,5 | 2 | | 37 | 1 | 103,7 |
| | JUnit | 34 | | | | | | | | 1 | 1 | 23 |
| | JEdit | 248 | | | | | 0,1 | | | | 6 | 29,7 |
| | PatternsBox | 52 | | | | | 0,2 | | | 7 | 1 | 13,5 |
| Itérateur | java.awt.* | 121 | | | | | | | | 12 | | 75,6 |
| | JHotDraw | 155 | 3 | 3 | | | 0,1 | | 3 | 100 | | 231,1 |
| | JUnit | 34 | | | | | | | | 8 | | 22,7 |
| | JEdit | 248 | 1 | 1 | | | 0,1 | | | 1 | | 28,5 |
| | PatternsBox | 52 | | | | | | | | 79 | | 36,5 |
| Observateur | java.awt.* | 121 | | | | | 3,4 | 4 | | 4 | | 73,5 |
| | JHotDraw | 155 | 2 | 2 | | | 3,2 | 2 | 2 | 2 | | 61,4 |
| | JUnit | 34 | 4 | 4 | | | 2,5 | | | | | 19,9 |
| | JEdit | 248 | 3 | 3 | | | 13 | | 3 | | | 27,8 |
| | PatternsBox | 52 | 1 | 1 | | | 2,8 | 2 | 1 | 2 | | 31,5 |
| | | 610 | 24 | 22 | 2 | 3 | 1,5 | 18 | 9 | 262 | 50 | 50,7 |

# Évaluation (4/4)

- Difficile
  - Pas d'autres outils existants
  - Différences sémantiques
- Besoin d'une méthodologie [Albin-Amiot03, chapitre 6]
  - Postulat
  - Hypothèses, interprétation du patron
  - Portée de l'identification du motif

# Limitations, perspectives (1/3)

- Modélisation des motifs
  - Motif de conception structuraux, comportementaux ?, générateurs ?
- Analyses dynamiques
  - Analyses d'alias

# Limitations, perspectives (2/3)

- Résolution des PSC
  - Algorithmes spécialisés
  - Automatisation
  - Interactions
  - Passage à l’échelle

# Limitations, perspectives (3/3)

- Résultats des identifications
  - Micro-architectures
    - Formes affaiblies d’un motif de conception
    - Pas un motif de conception (découverte ?)
  - Visualisation
    - Modèle de l’architecture
    - Modèles des motifs de conception
    - Micro-architectures identifiées
    - Problèmes résolus

# Perspectives

- Défauts de conception
  - Transformation des programmes
- Motifs spécialisés
  - Caractéristique de qualité
- Intégration de l’outil dans le cycle de développement

Yann-Gaël Guéhéneuc, Rémi Douence, and Narendra Jussien ; *No Java without Caffeine – A Tool for Dynamic Analysis of Java Programs* ; Proceedings of ASE, 2002.

Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien ; *Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together* ; Proceedings of ASE, 2001.

Yann-Gaël Guéhéneuc and Hervé Albin-Amiot ; *Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects* ; Proceedings of TOOLS USA, 2001.

Hervé Albin-Amiot, Pierre Cointe, and Yann-Gaël Guéhéneuc ; *Un méta-modèle pour coupler application et détection des design patterns* ; Actes de LMO (8ème Colloque Langages et Modèles à Objets), 2002.

Yann-Gaël Guéhéneuc et Narendra Jussien ; *Quelques Explications Pour Les Patrons – Une Utilisation de la PPC Avec Explications pour l'Identification de Patrons de Conception* ; Actes des 7èmes JNPC (Journées Nationales sur la Résolution Pratique de Problèmes NP-complets), 2001.

Andrés Farías and Yann-Gaël Guéhéneuc ; *On the coherence of component protocols* ; Proceedings of the ETAPS Workshop on Software Composition, 2003.

Yann-Gaël Guéhéneuc ; *Three Musketeers to the Rescue – Meta-modelling, Logic Programming, and Explanation-based Constraint Programming for Pattern Description and Detection* ; Proceedings of ASE Workshop on Declarative Meta-Programming, 2002.

Andrés Farías, Yann-Gaël Guéhéneuc, and Mario Südholt ; *Integrating Behavioral Protocols in Enterprise Java Beans* ; Proceedings of the OOPSLA Workshop on Behavioral Semantics, 2002.

Hervé Albin-Amiot and Yann-Gaël Guéhéneuc ; *Design Patterns Application: Pure-Generative Approach vs. Conservative-Generative Approach* ; Proceedings of OOPSLA Workshop on Generative Programming, 2001.

Yann-Gaël Guéhéneuc and Narendra Jussien ; *Using Explanations for Design-Patterns Identification* ; Proceedings of IJCAI Workshop on Modelling and Solving Problems with Constraints, 2001.

Hervé Albin-Amiot and Yann-Gaël Guéhéneuc ; *Meta-Modelling Design Patterns: Application to Pattern Detection and Code Synthesis* ; Proceedings of ECOOP Workshop on Automating Object-Oriented Software Development Methods, 2001.

Hervé Albin-Amiot and Yann-Gaël Guéhéneuc ; *Design Patterns: A Round-Trip* ; Proceedings of ECOOP 11th Workshop for PhD Students in Object-Oriented Systems, 2001.

[Albin-Amiot03] Hervé Albin-Amiot ; *Idiomes et patterns Java : application à la synthèse de code et à la détection* ; Thèse de doctorat de l'université de Nantes, février 2003

[Alexander77] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angel ; *A Pattern Language* ; Oxford University Press, 1977, ISBN 0-19-501919-9.

[Caseau96] Yves Caseau and François Laburthe ; *Claire: Combining Objects and Rules for Problem Solving* ; Proceedings of JICSLP, workshop on Multi-Paradigm Logic Programming, pages 105–114, TU Berlin, September 1996.

[Compagnon02] Antoine Compagnon ; *La notion de genre – Introduction : forme, style et genre littéraire* ; Décembre 2002, disponible à `www.fabula.org/compagnon/genre1.php`.

[Gamma94] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides ; *Design Patterns – Elements of Reusable Object-Oriented Software* ; Addison-Wesley, 1994, ISBN 0-201-63361-2.

[Gamma98] Erich Gamma et Thomas Eggenschwiler ; *JHotDraw* ; Disponible à `members.pingnet.ch/gamma/JHD-5.1.zip` et sur `sourceforge.net`.

[Guéhéneuc01] Yann-Gaël Guéhéneuc and Hervé Albin-Amiot ; *Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects* ; Proceedings of the 39th TOOLS USA conference, pages 296—305, IEEE Computer Society Press, July 2001.

[Jussien00] Narendra Jussien and Vincent Barichard ; *The PaLM System: Explanation-Based Constraint Programming* ; Proceedings of TRICS, pages 118–133, National University of Singapore, September, 2000.

[Jussien01] Narendra Jussien ; *Programmation par contraintes avec explications* ; actes des 7e JNPC, pages 147–158, ONERA, juin 2001.

[Laburthe00] François Laburthe et le projet OCRE ; *Choco : implémentation du noyau d'un système de contraintes* ; actes des 6e JNPC, pages 151–165, ONERA, juin 2000.

[Montanari74] Ugo Montanari ; *Networks of constraints fundamental properties and applications to picture processing* ; Information Science, volume 7, number 2, pages 95–132, Elsevier Science, 1974.

[Petit02] Thierry Petit ; *Modélisation et Algorithmes de Résolution de Problèmes Sur-Contraints* ; Thèse de doctorat de l'université du Languedoc, novembre 2002.

[OTI-IBM01] Object Technology International, Inc. / IBM ; *Éclipse – Un plate-forme d'outillage universelle* ; Disponible à `www.eclipse.org`.

[Tsang93] Edward Tsang ; *Foundations of Constraint Satisfaction* ; Academic Press, 1993, ISBN 0-127-01610-4.

# Cycle de vie du logiciel

- Développement incrémental / itératif
- Maintenance
  - Rétro-conception
  - Re-conception
- Documentation

# PPCE

- Applications
  - Assistance en cas de contradiction
  - Algorithmes de résolution interactifs
  - Nouveaux algorithmes de résolution
    - Path-repair [Jussien02]
    - Mac-DBT [Jussien00]

[Jussien02] Narendra Jussien and Olivier Lhomme ; *Local search with constraint propagation and conflict-based heuristics* ; Journal of Artificial Intelligence, volume 139, number 1, pages 21–45, Elsevier Science, July 2002.

[Jussien00] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault ; *Maintaining Arc-Consistency within Dynamic Backtracking* ; Proceedings of CP, pages 249–261, Springer-Verlag, September 2000.

# Travaux apparentés (1/2)

- Modélisation
  - Logique du première ordre [Eden00]
  - Modèle à fragments [Florijn97]
- Application
  - Scripts de génération [Budinsky96]
  - Programmation méta-logique [Eden97]
- Identification
  - Programmation logique [Wuyts98]
  - Étapes de filtrages (métriques) [Antoniol98]

# Travaux apparentés (2/2)


[Antoniol98] Giuliano Antoniol, Roberto Fiutem, and L. Cristoforetti ; *Design Pattern Recovery in Object-Oriented Software* ; Proceedings of the 6th workshop on Program Comprehension, pages 153–160, IEEE Computer Society Press, June 1998.

[Budinsky96] Frank J. Budinsky, Marilyn A. Finnie, John M. Vlissides, and Patsy S. Yu ; *Automatic Code Generation from Design Patterns* ; IBM Systems Journal 35 (2), pages 151–171, February 1996.

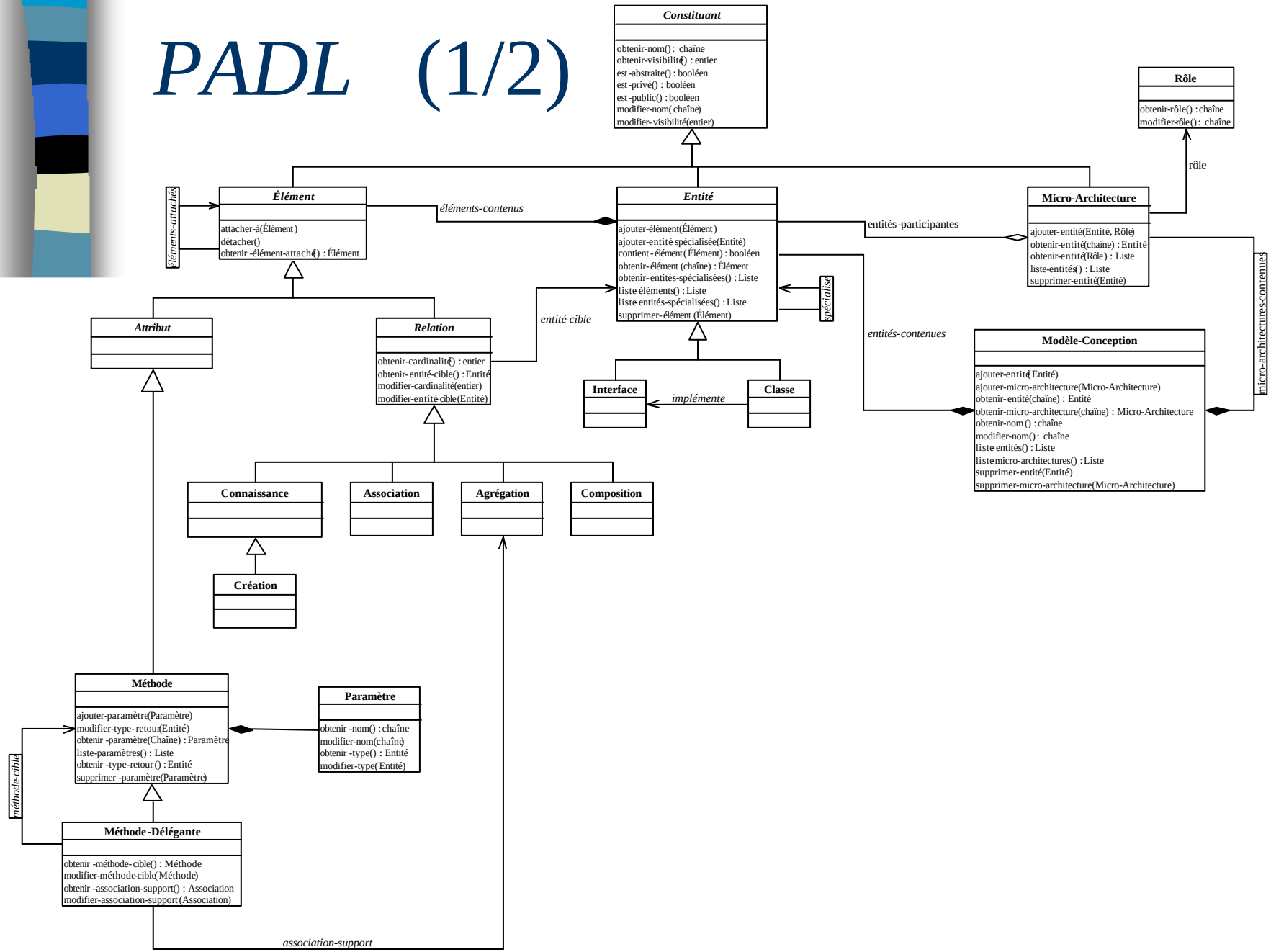
[Eden97] Amnon H. Eden, Amiram Yehudai, and Joseph (Yossi) Gil ; *Precise Specification and Automatic Application of Design Patterns* ; Proceedings of the 12th ASE conference, pages 143–152, IEEE Computer Society Press, November 1997.

[Eden00] Amnon H. Eden ; *Precise Specification of Design Patterns and Tool Support in their Application* ; Ph.D. thesis, Tel Aviv University, 2000.

[Florijn97] Gert Florijn, Marco Meijers, and Pieter Van Winsen ; Tool Support for Object-Oriented Patterns ; Proceedings of the 11th ECOOP conference, Springer-Verlag, June 1997.

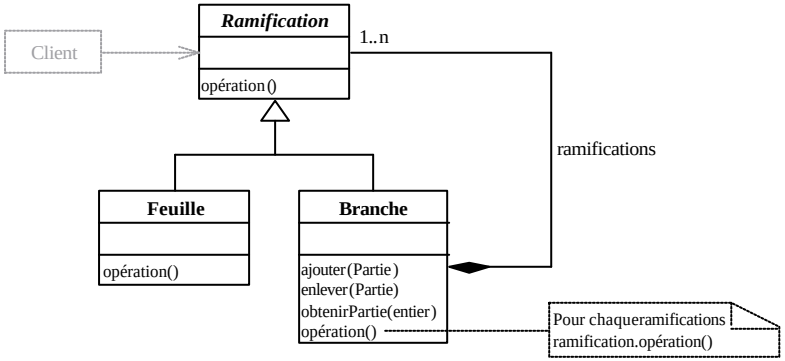
[Wuyts98] Roel Wuyts ; Declarative Reasoning About the Structure of Object-Oriented Systems ; Proceedings of the 26th TOOLS USA conference, pages 112–124, IEEE Computer Society Press, August 1998.

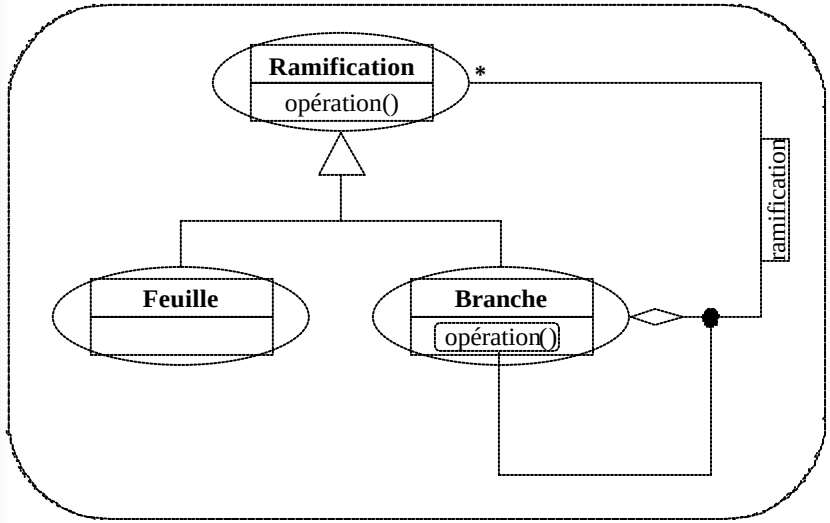
# PADL (1/2)


Constituant
obtenir-nom() : chaîne
obtenir-visibilité() : entier
est-abstraite() : booléen
est-privé() : booléen
est-public() : booléen
modifier-nom(chaîne)
modifier-visibilité(entier)
Rôle
obtenir-rôle() : chaîne
modifier-rôle() : chaîne
rôle
Élément
attacher-à(Élément)
détacher()
obtenir-élément-attaché() : Élément
éléments-attachés
éléments-contenus
Entité
ajouter-élément(Élément)
ajouter-entité-spécialisée(Entité)
contient-élément(Élément) : booléen
obtenir-élément(chaîne) : Élément
obtenir-entités-spécialisées() : Liste
liste-éléments() : Liste
liste-entités-spécialisées() : Liste
supprimer-élément(Élément)
entités-participantes
Micro-Architecture
ajouter-entité(Entité, Rôle)
obtenir-entité(chaîne) : Entité
obtenir-entité(Rôle) : Liste
liste-entités() : Liste
supprimer-entité(Entité)
micro-architectures-contenues
spécialisé
entités-contenues
Attribut
Relation
obtenir-cardinalité() : entier
obtenir-entité-cible() : Entité
modifier-cardinalité(entier)
modifier-entité-cible(Entité)
entité-cible
Interface
implémente
Classe
Modèle-Conception
ajouter-entité(Entité)
ajouter-micro-architecture(Micro-Architecture)
obtenir-entité(chaîne) : Entité
obtenir-micro-architecture(chaîne) : Micro-Architecture
obtenir-nom() : chaîne
modifier-nom() : chaîne
liste-entités() : Liste
liste-micro-architectures() : Liste
supprimer-entité(Entité)
supprimer-micro-architecture(Micro-Architecture)
Connaissance
Association
Agrégation
Composition
Création
Méthode
ajouter-paramètre(Paramètre)
modifier-type-retour(Entité)
obtenir-paramètre(Chaîne) : Paramètre
liste-paramètres() : Liste
obtenir-type-retour() : Entité
supprimer-paramètre(Paramètre)
Paramètre
obtenir-nom() : chaîne
modifier-nom(chaîne)
obtenir-type() : Entité
modifier-type(Entité)
méthode-cible
Méthode-Délégante
obtenir-méthode-cible() : Méthode
modifier-méthode-cible(Méthode)
obtenir-association-support() : Association
modifier-association-support(Association)
association-support

# PADL (2/2)

Description informelle
[Gamma94]




## Légende

- Instance de `Pattern`
- Instance de `Interface`
- Instance de `Class`
- Instance de `Association`
- Instance de `Delegation`
- *nom()* Instance de `Method`