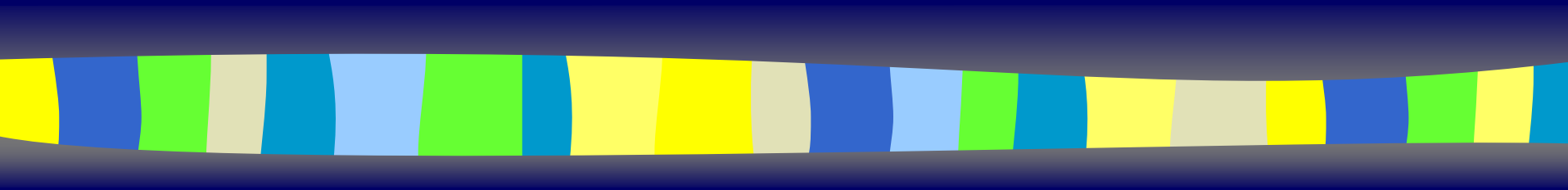
Méthode d'implémentation efficace des modèles PAC et PAC-Amodeus à l'aide de patrons de conception



Thierry Duval
IRISA – Siames
Thierry.Duval@irisa.fr

Contexte

- Rendre interactif un noyau orienté objet existant...
  - C++, Eiffel, Java, C#, …
- L'architecture de ce noyau a été déterminée par le domaine d'application
- La conception a pu être réalisée par des experts de ce domaine
- Les interactions entre les objets de ce noyau et les utilisateurs des applications peuvent ne pas avoir été prévues

Réalité du contexte

- C’est souvent le cas quand des spécialistes d’un domaine conçoivent un logiciel :
  - il est souhaitable de les laisser modéliser leur concepts, c’est la meilleure façon d’utiliser leur expertise
  - on ne peut pas leur imposer une double compétence incluant les IHM
- La collaboration avec un expert IHM n’est pas toujours possible au bon moment

Objectif : rendre interactives de telles applications

- Sans être obligé de tout refaire
- En minimisant les coûts :
  - de conception
  - de réalisation logicielle
- Sans casser l'architecture initiale
- En faisant clairement la distinction entre :
  - l'application initiale
  - l'interface utilisateur

En résumé : le point de départ

- Possibilité de disposer d'un noyau applicatif orienté objet :
  - sans visualisation
  - sans interaction
  - susceptible de fonctionner de façon autonome
- Volonté de lui ajouter une IHM :
  - devant manipuler les concepts applicatifs
  - devant être indépendante de :
    - l'implémentation du noyau applicatif
    - l'architecture du noyau applicatif
  - ne devant pas remettre en cause l'architecture initiale

Les problèmes qui se posent alors...

- Comment lier efficacement l'application initiale et l'interface utilisateur ?

- Comment profiter de la (bonne) conception (UML) de l'application initiale ?
  - comment ne pas la remettre en cause…

Les solutions…

- Utiliser un modèle d'architecture pour les IHM :
  - à choisir judicieusement
  - à mettre en œuvre à l'aide de patrons de conception
- Inciter une bonne conception (UML) de l'application initiale :
  - à l'aide de patrons de conception adaptés

Choix du modèle d'architecture

- Doit séparer clairement le noyau applicatif de l'interface utilisateur :
  - prendre plutôt un modèle apparenté au modèle Seeheim
- Doit être adapté au contexte objet :
  - prendre plutôt un modèle multi-agents

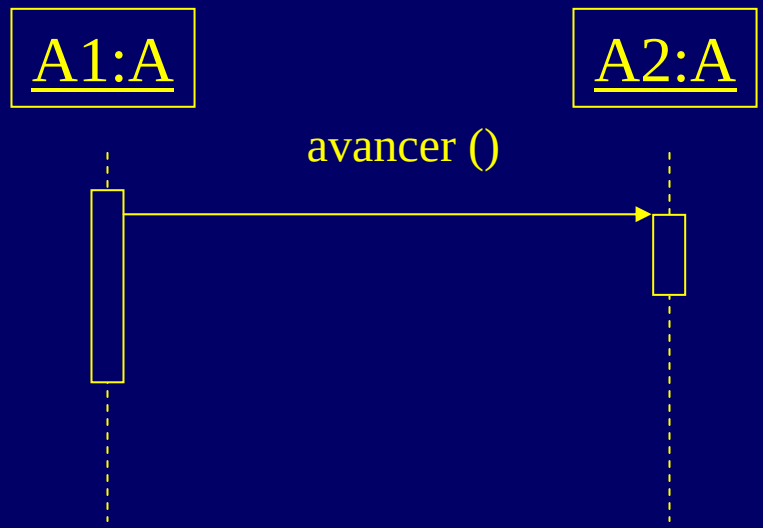
⇨ PAC ou PAC-Amodeus…

⇨ Comment les mettre en œuvre efficacement ?

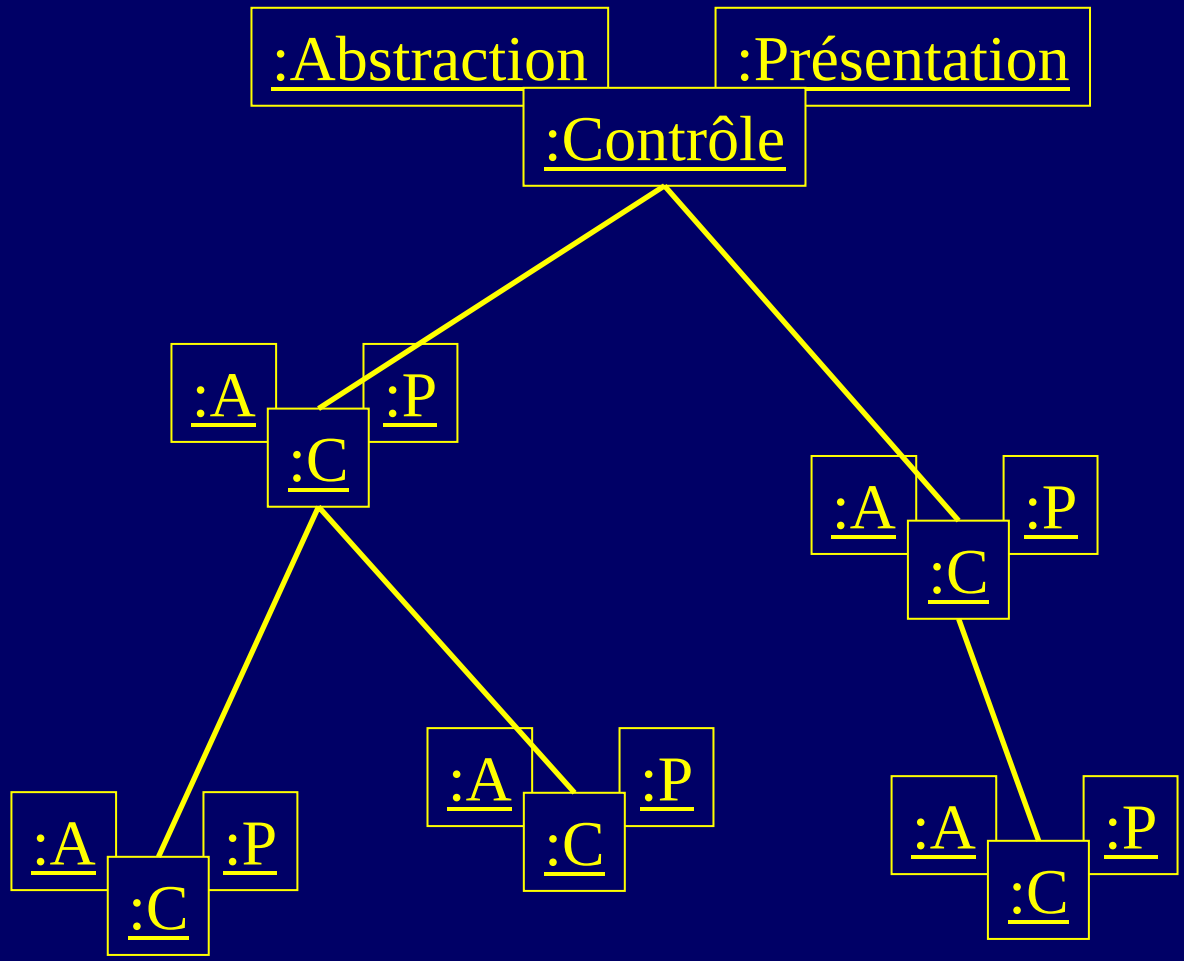
Plan

- Problématique
- Rappel des modèles PAC et PAC-Amodeus
- Mauvaises utilisations de ces modèles
- Mise en œuvre de PAC et PAC-Amodeus
  - les patrons de conception à appliquer
  - trois mises en œuvres différentes
  - l'exemple : les tours de Hanoï en java avec Swing

Exemple typique en simulation



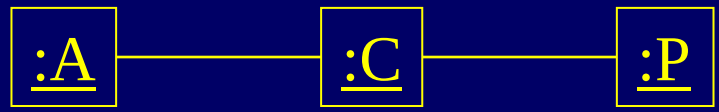
Le modèle PAC (J. Coutaz, 1987)



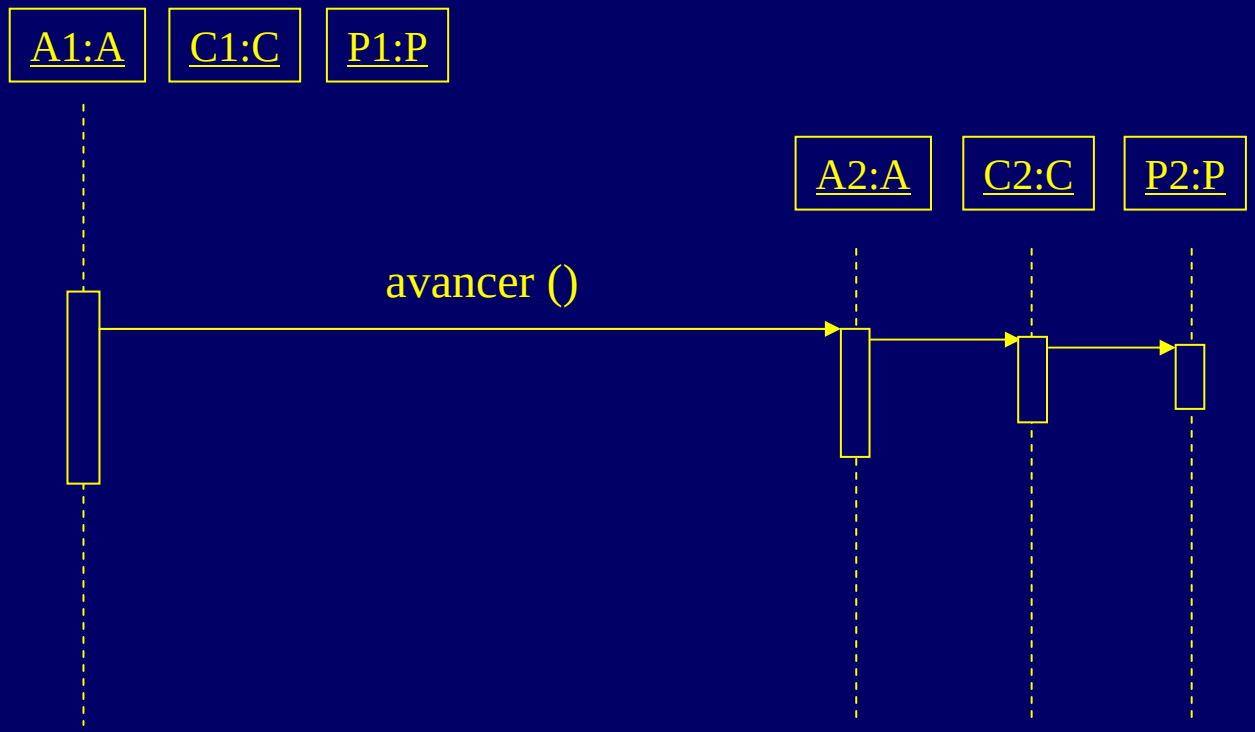
Application directe de PAC

- Créer un agent PAC pour chaque objet initial :
  - ajout d'un composant contrôle et d'un composant présentation par objet existant
  - chaque objet initial sera l'abstraction d'un agent PAC
- Problème :
  - les objets initiaux communiquaient entre eux…
  - …or avec PAC les différents agents ne peuvent communiquer que via leurs composants de contrôle !

Application directe de PAC

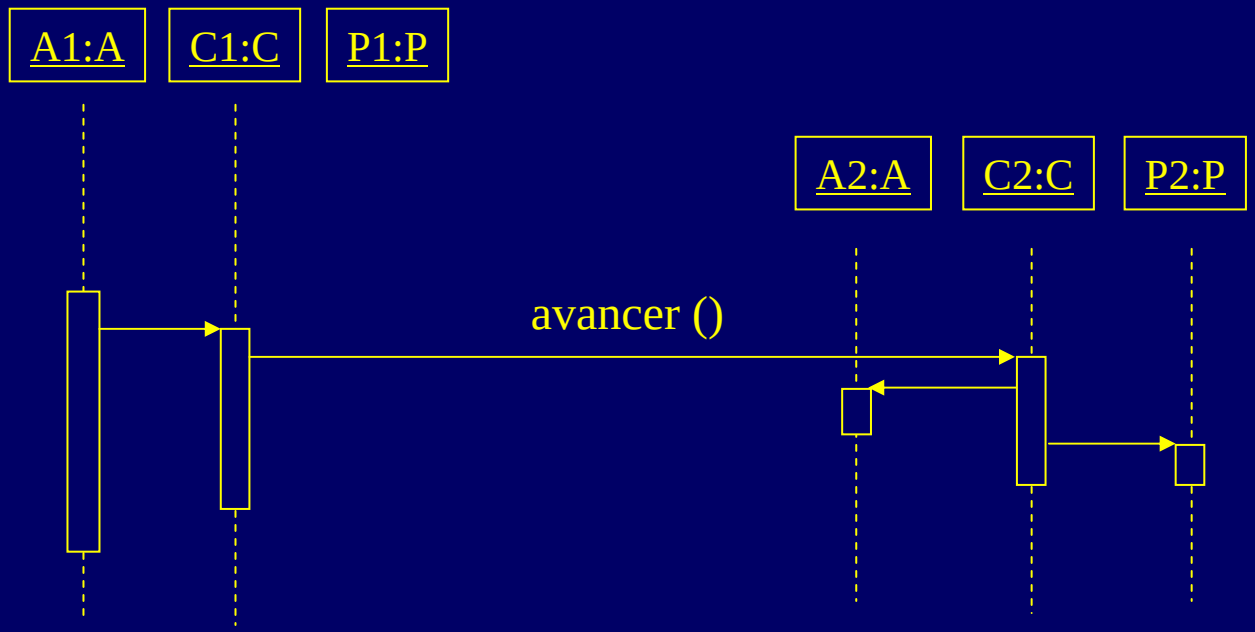


Application directe de PAC v1



- A1 communique directement avec A2
- Modification de A2 pour communiquer avec C2

Application directe de PAC v2



- La connaissance de A2 n'est plus utile à A1
- C'est à C1 d'envoyer un message à C2
- Duplication dans C1 de la connaissance de A1
- Modification de A1 pour communiquer avec C1

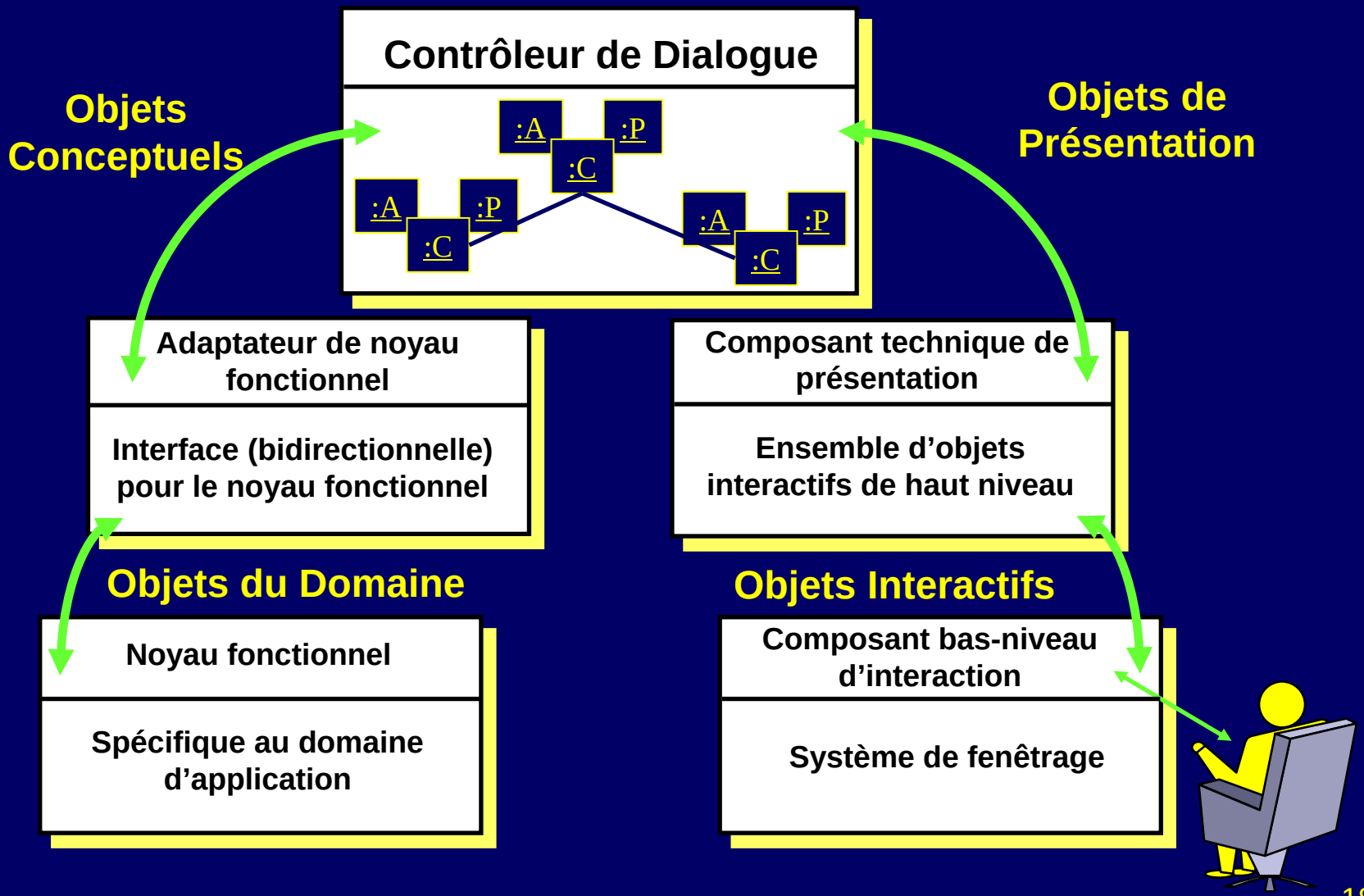
Premier bilan sur PAC

- Résultat non satisfaisant en termes de :
  - coût de développement (modification du code...)
  - facilité de maintenance et d'évolution (duplication de code)
  - hiérarchie obtenue : c'est la hiérarchie initiale et non pas une hiérarchie dictée par le dialogue avec l'utilisateur

Pourquoi PAC est-il inadapté ici ?

- PAC est un modèle purement multi-agents :
  - il faut plaquer la structure initiale en objets de simulation sur une autre structure induite par le modèle…
  - la hiérarchie obtenue correspond à l'organisation du noyau initial et non à celle du dialogue avec l'utilisateur
- Cette transformation ne peut donc être effectuée :
  - à moindre coût
  - sans modifier la structure initiale
- Il faut donc un modèle hybride...

Le modèle PAC-Amodeus (L. Nigay, 1993)



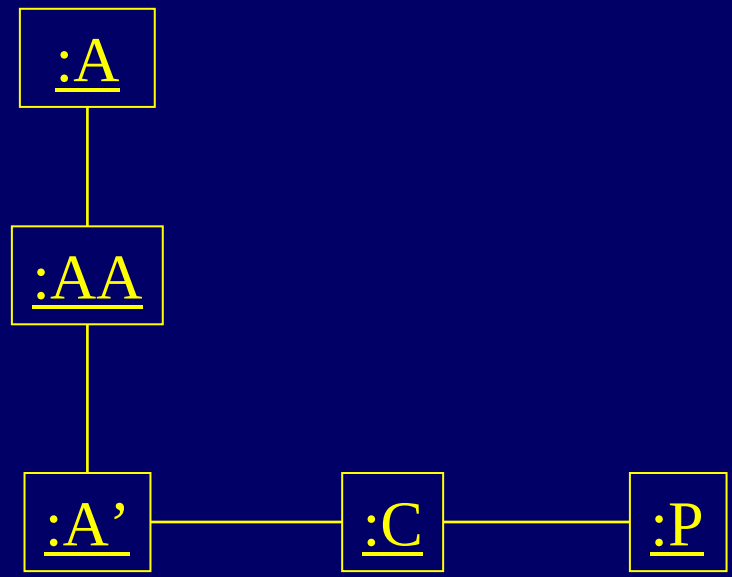
Application directe de PAC-Amodeus

- Les objets initiaux forment le noyau fonctionnel
  - ils peuvent être adaptés au niveau de l'ANF
  - ils peuvent communiquer avec une Abstraction d'un agent PAC du contrôleur de dialogue

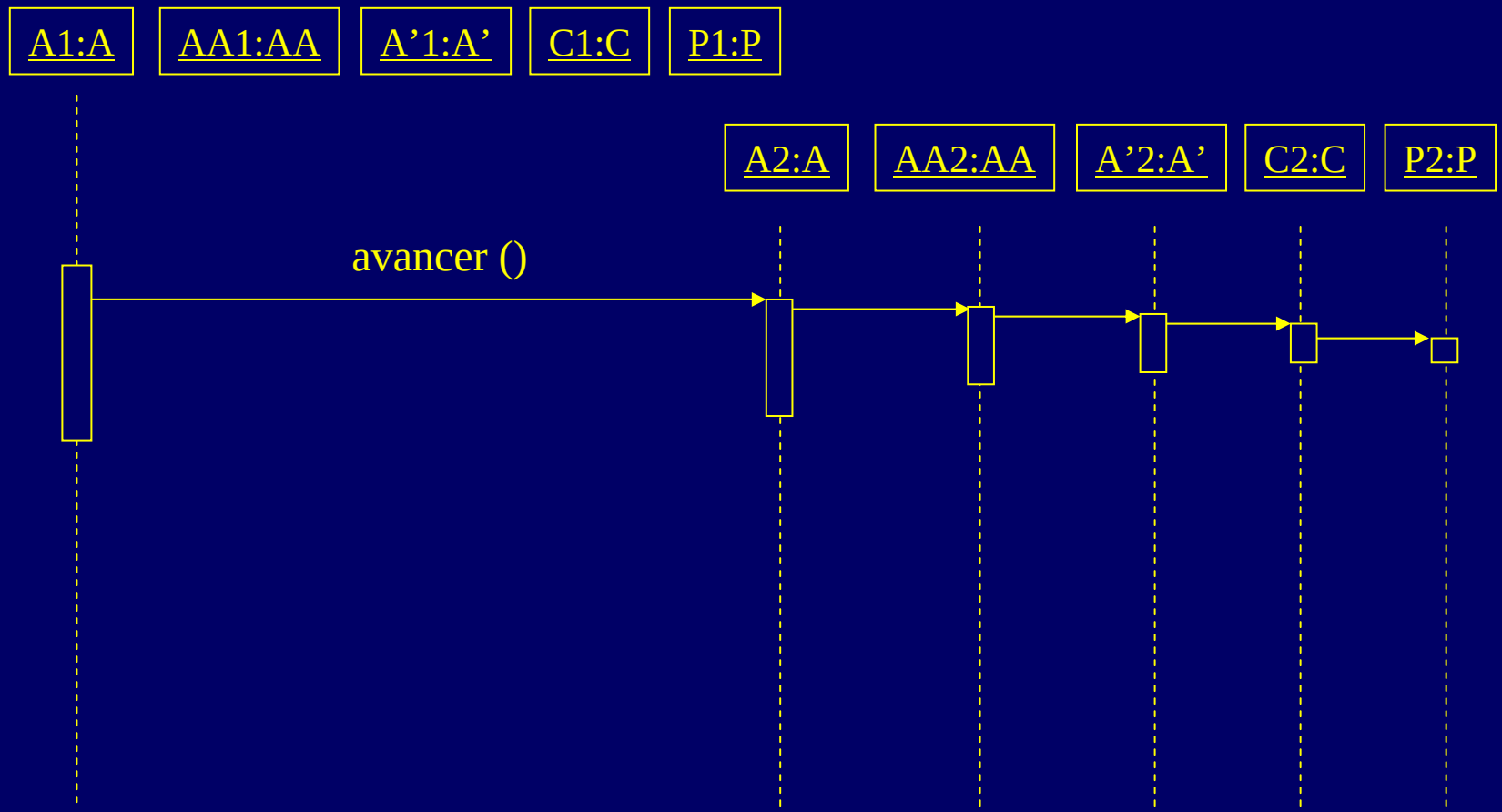
Application directe de PAC-Amodeus

- Avantages :
  - les objets initiaux restent des composants logiciels indépendants
  - la hiérarchie PAC est indépendante de la structure des objets initiaux
  - un agent PAC du Contrôleur de Dialogue est à l'écoute :
    - de l'utilisateur par sa facette présentation
    - du Noyau Fonctionnel par sa facette abstraction

Application directe de PAC-Amodeus



Application directe de PAC-Amodeus



Application directe de PAC-Amodeus

- Inconvénient majeur (et rédhibitoire) :
  - comme avec PAC, il faut modifier les classes existantes pour leur permettre de communiquer avec les agents PAC-Amodeus
- Autres inconvénients :
  - beaucoup (4) de classes doivent être créées pour rendre interactif un objet existant
  - un risque d'inefficacité à l'exécution à cause des nombreux relais lors des invocations de méthodes vers les objets initiaux

Plan

- Problématique
- Rappel des modèles PAC et PAC-Amodeus
- Mauvaises utilisations de ces modèles
- Mise en œuvre efficace de PAC et PAC-Amodeus
  - les patrons de conception à appliquer :
    - Proxy
    - Abstract Factory (et Template Method ou Interface)
  - trois mises en œuvres différentes
  - l'exemple : les tours de Hanoï en java avec Swing

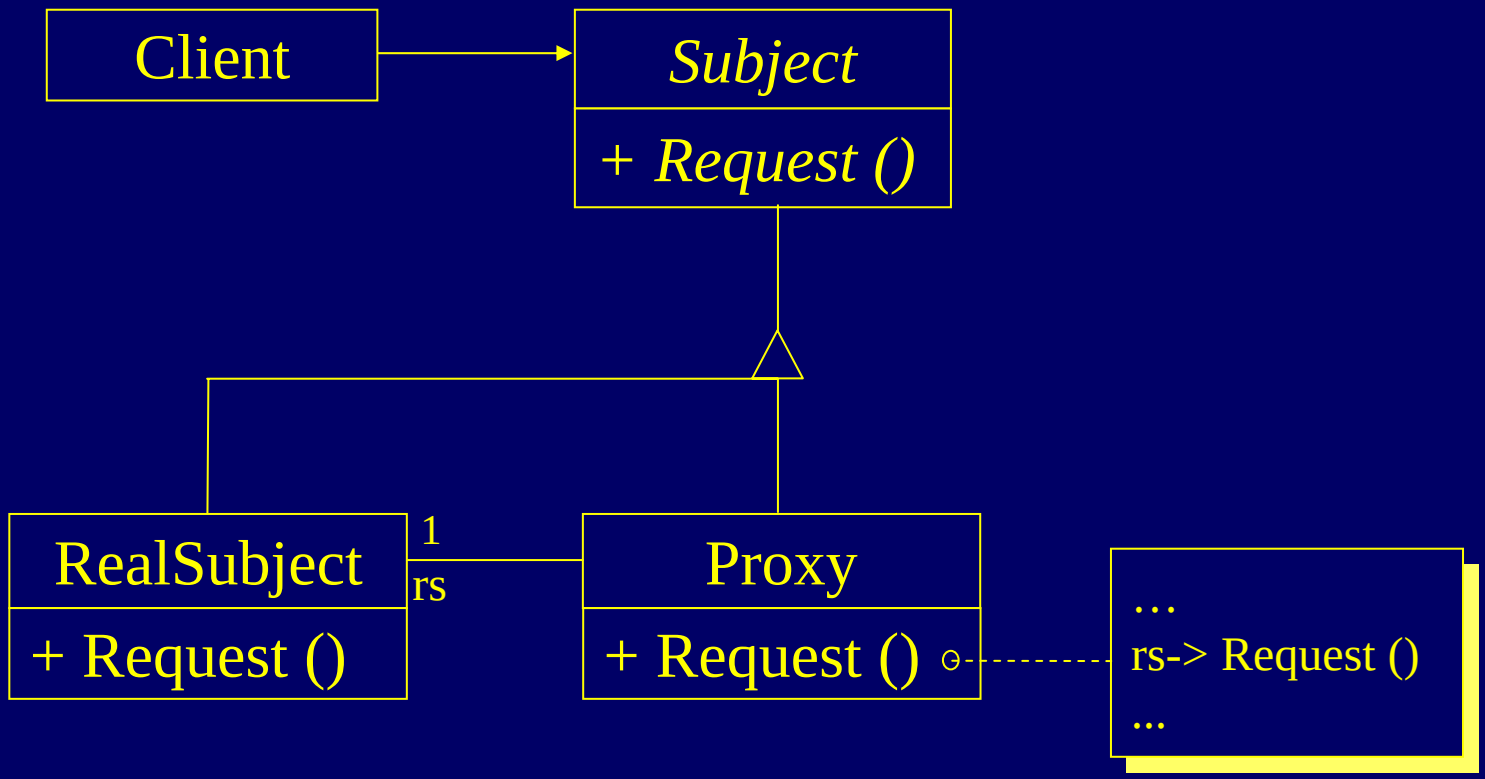
Principes de la méthode proposée

- Utiliser le modèle PAC ou PAC-Amodeus +
  - des interfaces pour chaque type de composant :
    - de façon à pouvoir utiliser le polymorphisme et la liaison dynamique
  - le design pattern « Proxy » :
    - les contrôles seront les proxys des abstractions
  - le design pattern « Abstract Factory » :
    - pour remplacer les abstractions par des contrôles
- Faire si besoin des optimisations pour réduire :
  - les coûts de développement (le nombre de classes)
  - le manque d'efficacité à l'exécution

Le « Proxy » (GoF 207)

- Pattern « Structural »
- But, intention :
  - fournir un représentant à un autre objet de façon à ce que ce représentant en contrôle l'accès

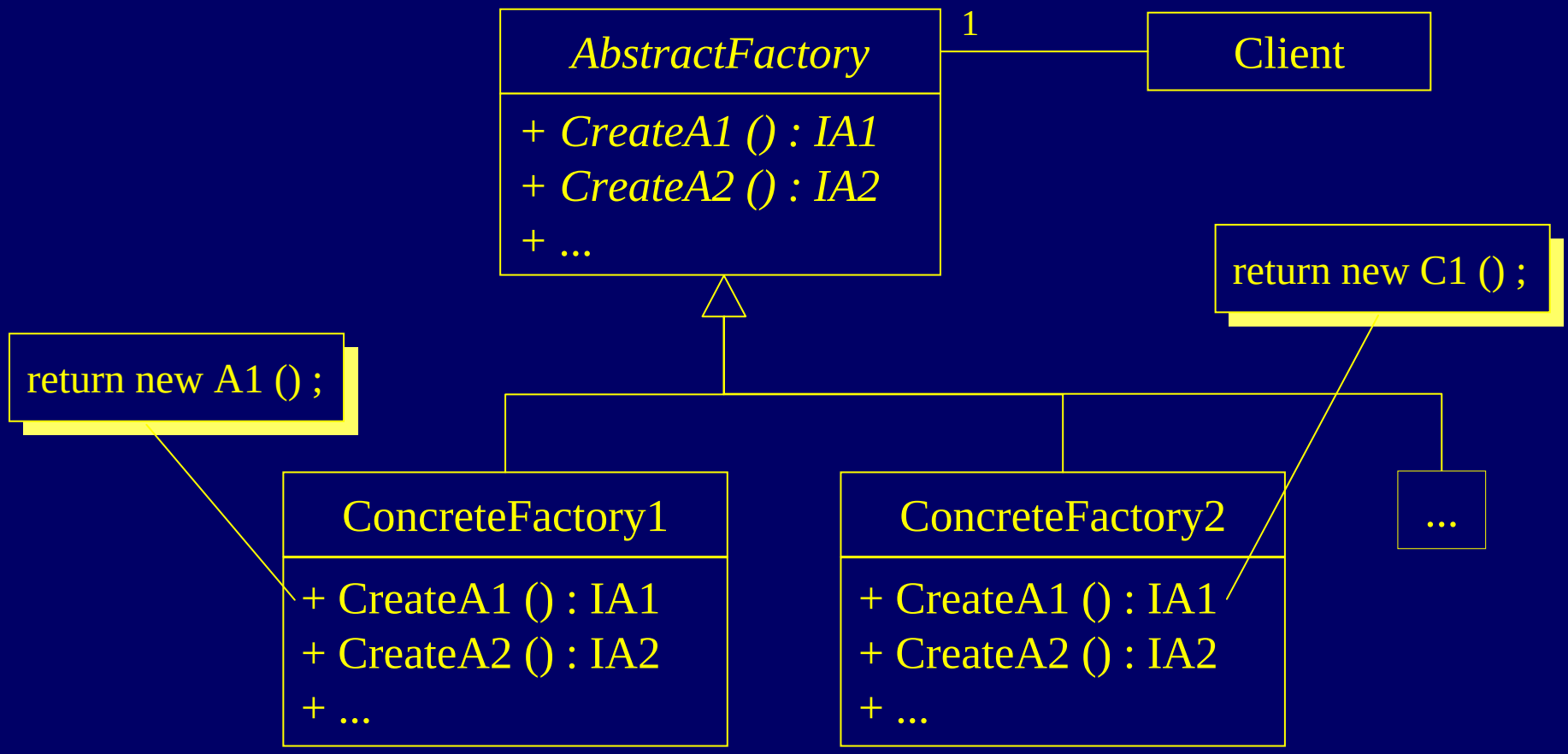
Structure du « Proxy »



L'« Abstract Factory » (GoF 87)

- Pattern « Creational »
- But, intention :
  - fournir une interface pour créer des familles d'objets (liés ou dérivés) sans avoir à spécifier leurs classes concrètes
- Dans notre cas :
  - cela permettra de substituer aux objets initiaux les contrôles (qui implémentent la même interface que les objets initiaux dont ils sont les proxys) des objets interactifs PAC du contrôleur de dialogue

Structure de l'« Abstract Factory »



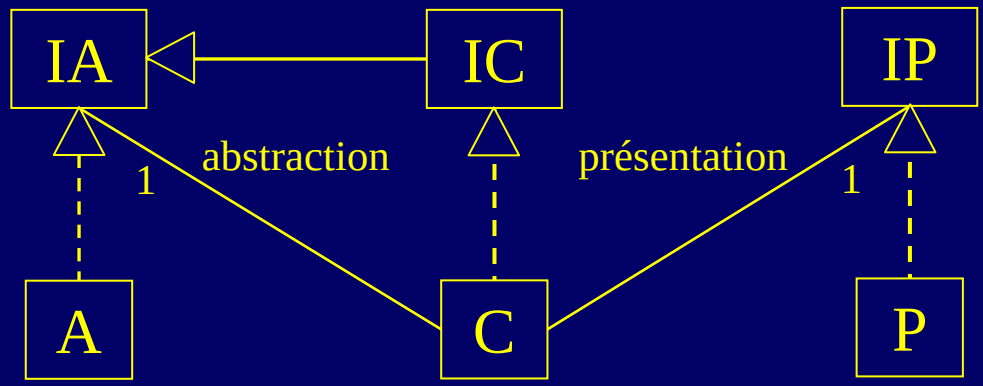
Plan

- Problématique
- Rappel des modèles PAC et PAC-Amodeus
- Mauvaises utilisations de ces modèles
- Mise en œuvre de PAC et PAC-Amodeus
  - les patrons de conception à appliquer
  - trois mises en œuvres différentes :
    - Proxy classique (avec usage du patron Delegation)
    - Proxy + héritage (implémentation non classique du Proxy)
    - Proxy + Observer (dans le cadre de PAC-Amodeus)
  - l'exemple : les tours de Hanoï en java avec Swing

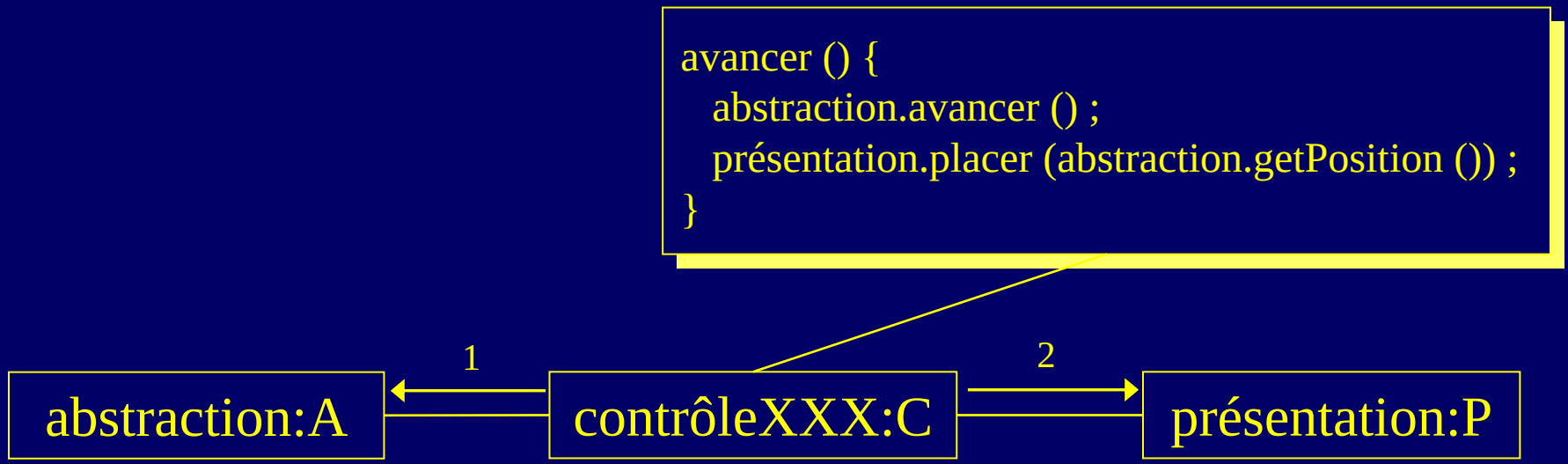
1ère implémentation proposée

- Utilisation du modèle PAC (ou PAC-Amodeus)
  - utilisation d'interfaces :
    - de façon à pouvoir utiliser le polymorphisme et la liaison dynamique
  - le design pattern « Proxy » + « Delegation »
  - le design pattern « Abstract Factory »
- Avec l'utilisation de PAC-Amodeus :
  - faire des optimisations pour réduire :
    - les coûts de développement (le nombre de classes)
    - le manque d'efficacité à l'exécution

PAC + Proxy + Délégation



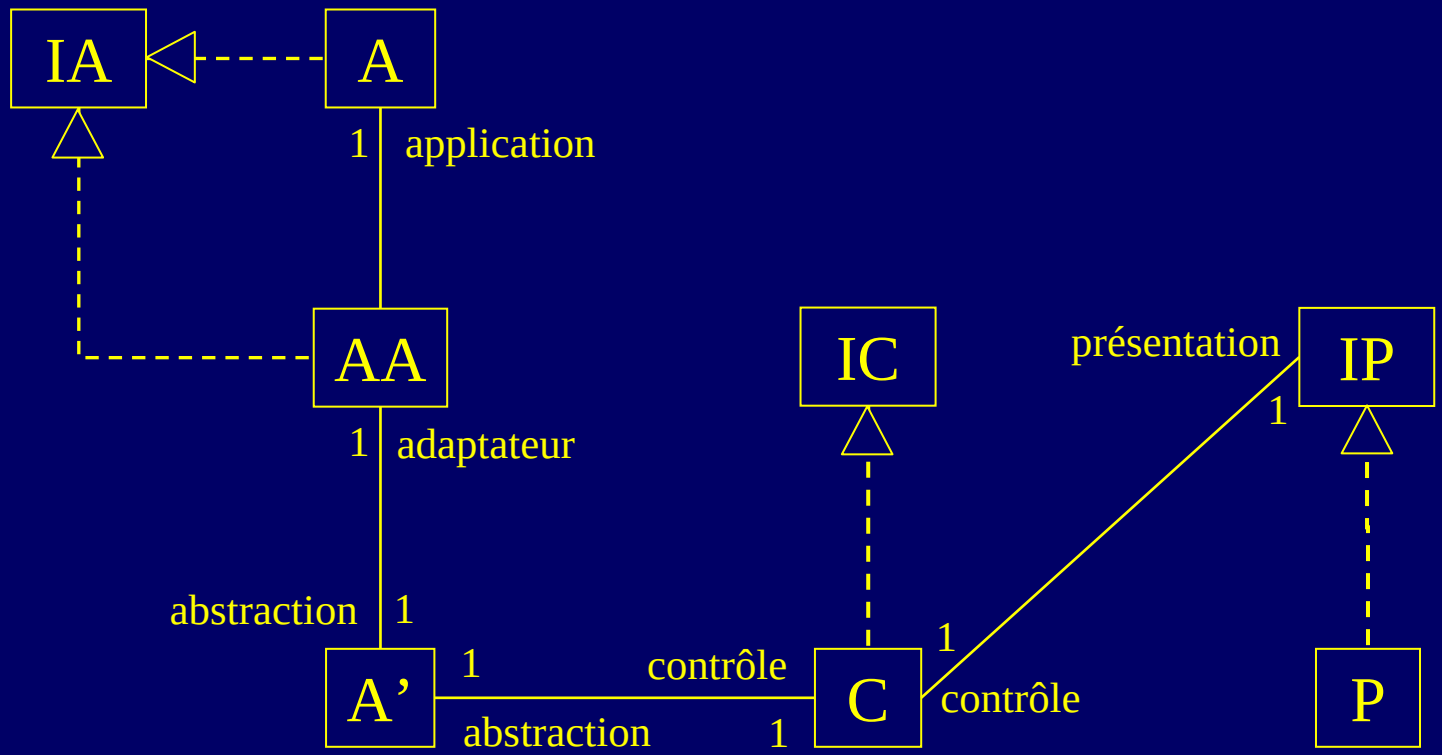
PAC + Proxy + Délégation



PAC + Proxy + Délégation

- Définition d'une interface IA :
  - A et C implémentent IA
  - C est un proxy de A
  - on remplacera les A par des C :
    - à l'aide du patron de conception Abstract Factory
- Définition d'une interface IP :
  - P implémente IP :
    - C sera indépendant le l'implémentation de P (API graphique !)
- Définition d'une interface IC (qui hérite de IA) :
  - C implémente IC :
    - P sera indépendant de l'implémentation de C

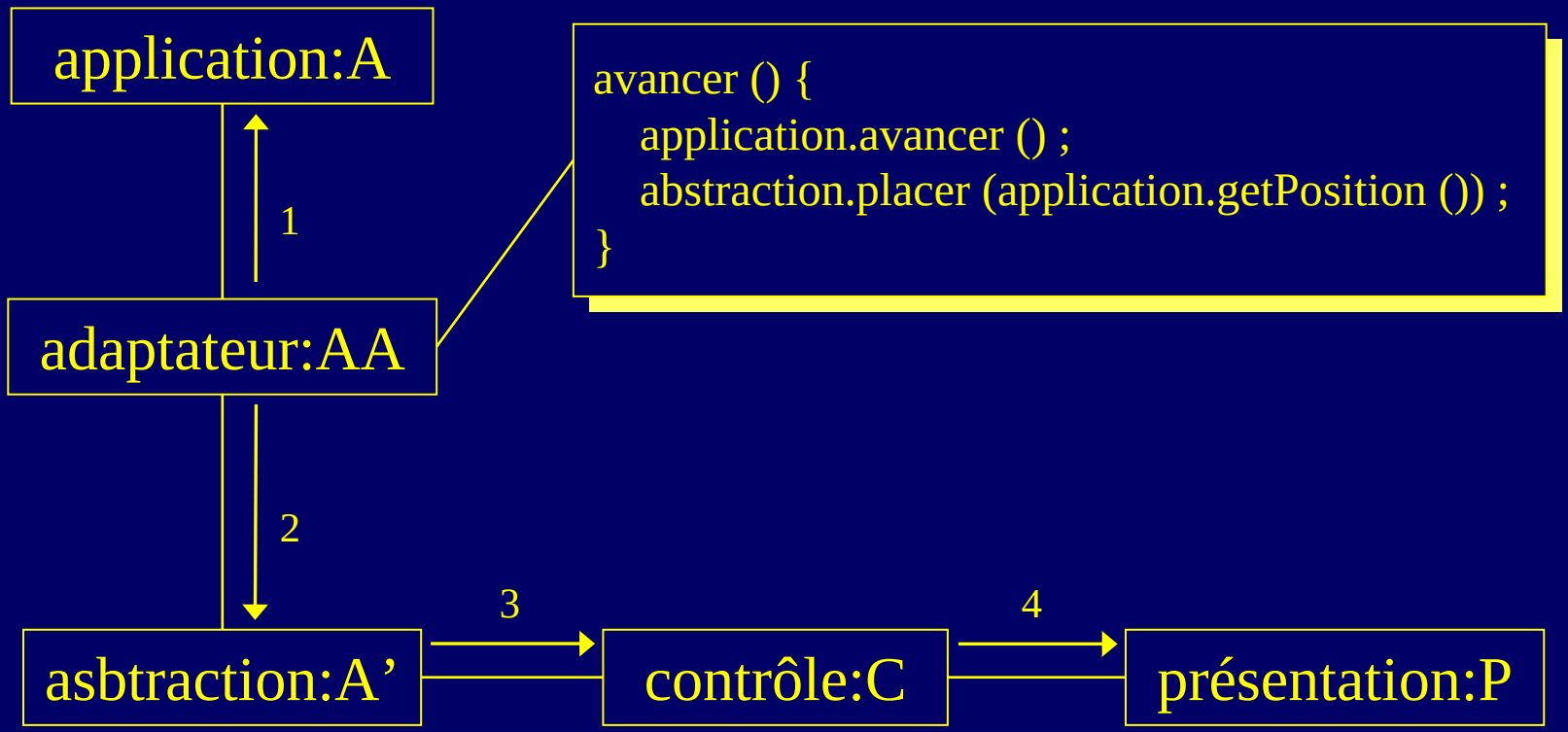
PAC-Amodeus + Proxy + Délégation



PAC-Amodeus + Proxy + Délégation

- Définition d'une interface IA :
  - A et AA implémentent IA
  - AA est un proxy de A
  - on remplacera les A par des AA :
    - à l'aide du patron de conception Abstract Factory
- Définition d'interfaces IP et IC :
  - comme pour l'utilisation du modèle PAC

PAC-Amodeus + Proxy + Délégation



PAC-Amodeus + Proxy + Délégation

- Beaucoup (trop ?) de classes à implémenter…
- Possibilité de supprimer la facette A' (cf PAC)
  - A' ne fait que des relais entre AA et C
- Possibilité de regrouper AA et C (« Slinky » Arche)
  - AA régule l'accès aux méthodes de A
    - propagation vers C pour le maintien de la cohérence
  - C fait les autres contrôles d'accès à AA
    - Contrôles suite aux entrées de l'utilisateur
- Une fois optimisé, on retrouve le modèle PAC !

Dernier problème : la création ...

- Dans la nouvelle application (interactive) :
  - il faut être capable de remplacer la création des objets initiaux par celle des objets interactifs (les proxys)
- Utilisation d'un « Creational Pattern » :
  - le patron de conception « Abstract Factory »
  - l'application initiale doit utiliser ce patron de conception
  - c'est la seule contrainte qui lui est imposée...

Créer les contrôles et les présentations

- Utiliser également des fabriques de composants :
  - pour une indépendance du contrôle vis à vis de l'implémentation effective de la présentation
  - pour une indépendance de la présentation vis à vis de l'implémentation effective du contrôle

Lien avec les composants présentation

- Usage d'une API graphique 2D de type Swing :
  - notions de composants et de containers
  - besoin d'ajouter les présentations les unes dans les autres
- Chaque composant contrôle devra :
  - permettre l'accès à son composant présentation
  - gérer l'ajout des présentations de ses sous-composants à l'intérieur de sa propre présentation

Conclusion sur cette première méthode

- Nombre important de classes et interfaces à définir
- Dépendances minimales entre composants :
  - grâce aux interfaces, aux proxys et aux fabriques de composants
- Contraintes minimales pour l'application initiale :
  - usage d'une fabrique de composants
- Équivalence totale entre PAC et PAC-Amodeus optimisé :
  - utiliser plutôt PAC…
  - ne serait-ce pas plutôt une implémentation de Arche ?

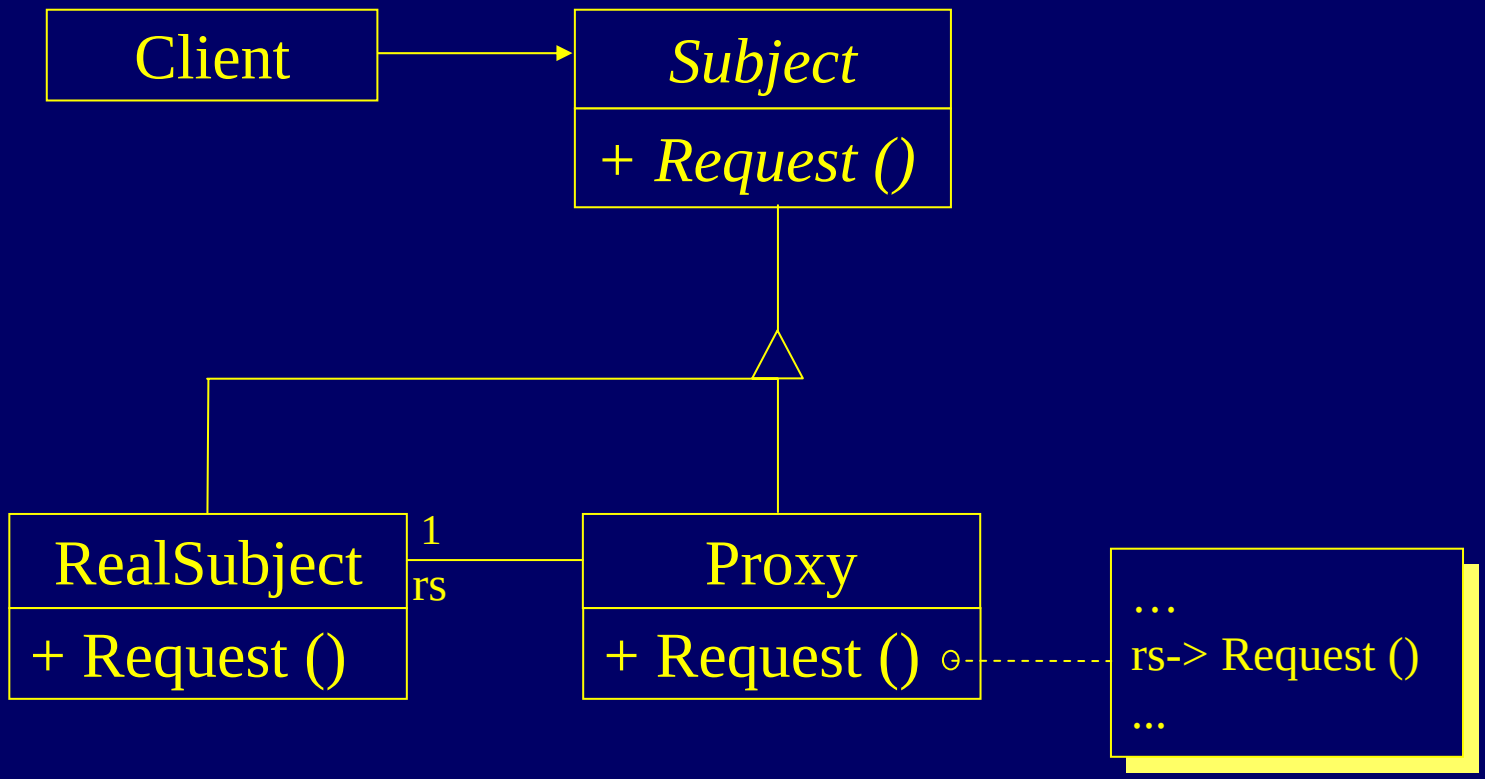
2ème implémentation proposée

- Utilisation du modèle PAC (ou PAC-Amodeus)
  - utilisation d'interfaces :
    - de façon à pouvoir utiliser le polymorphisme et la liaison dynamique
  - le design pattern « Proxy » + « Héritage »
  - le design pattern « Abstract Factory »
- Avec l'utilisation de PAC-Amodeus :
  - faire des optimisations pour réduire :
    - les coûts de développement (le nombre de classes)
    - le manque d'efficacité à l'exécution

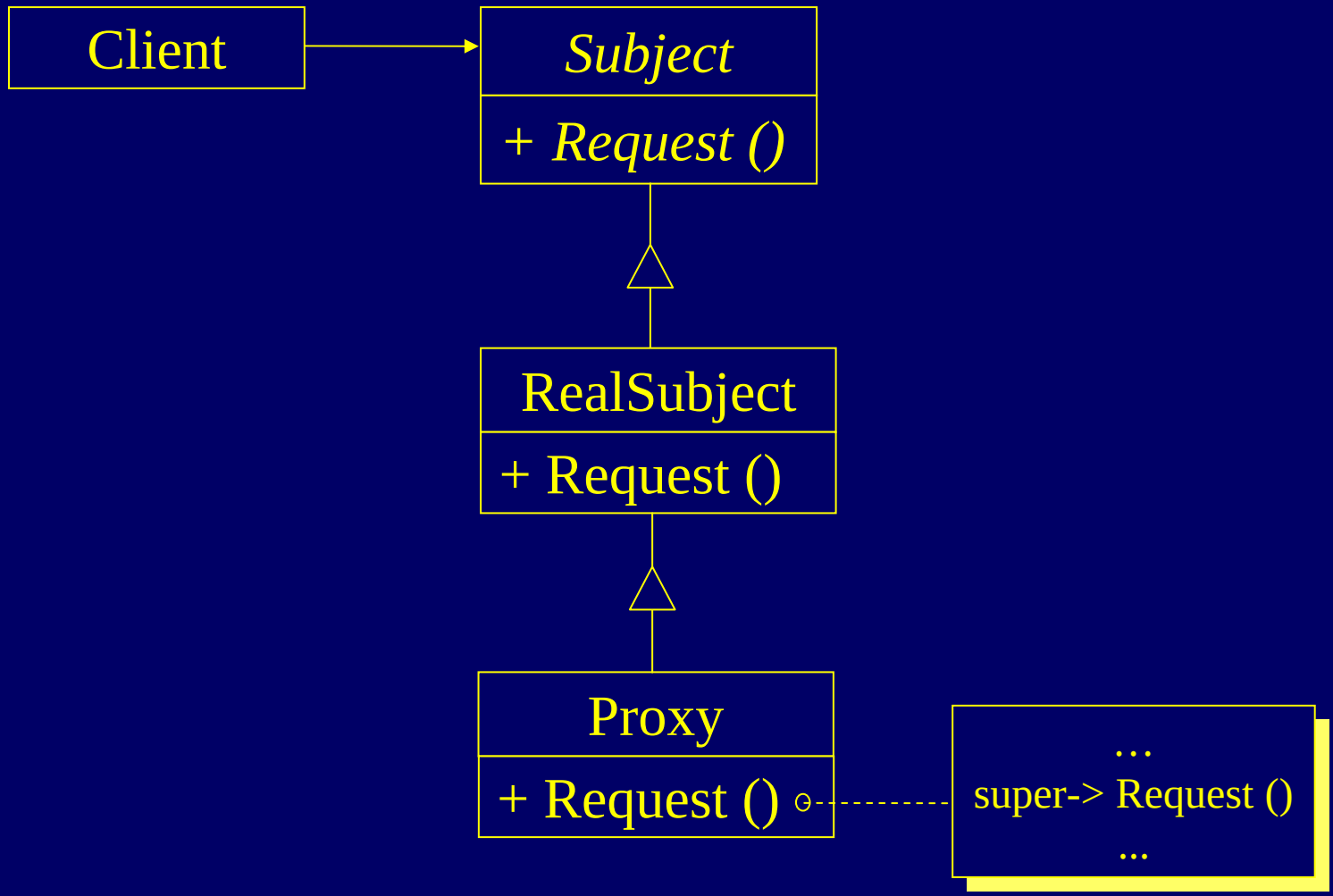
Le patron « Proxy » + « Héritage »

- Pattern « Structural »
- But, intention :
  - fournir un représentant à un autre objet de façon à ce que ce représentant en contrôle l'accès
- Notre approche est particulière :
  - impose une définition plus précise pour le représentant :
    - il héritera de l'objet initial
  - permet une réutilisation optimale :
    - on ne redéfinit ainsi que le strict nécessaire dans le représentant...

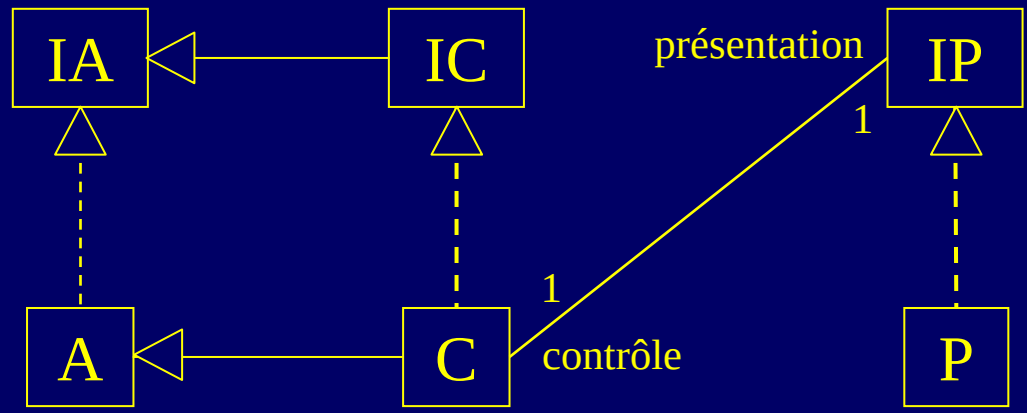
Structure classique du « Proxy »



Structure de notre « Proxy »

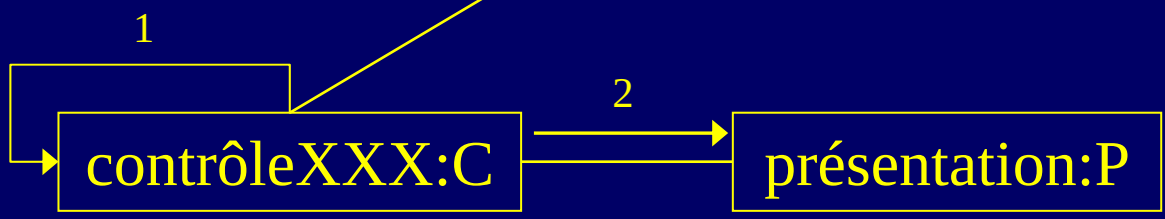


PAC + Proxy + Héritage



PAC + Proxy + Héritage

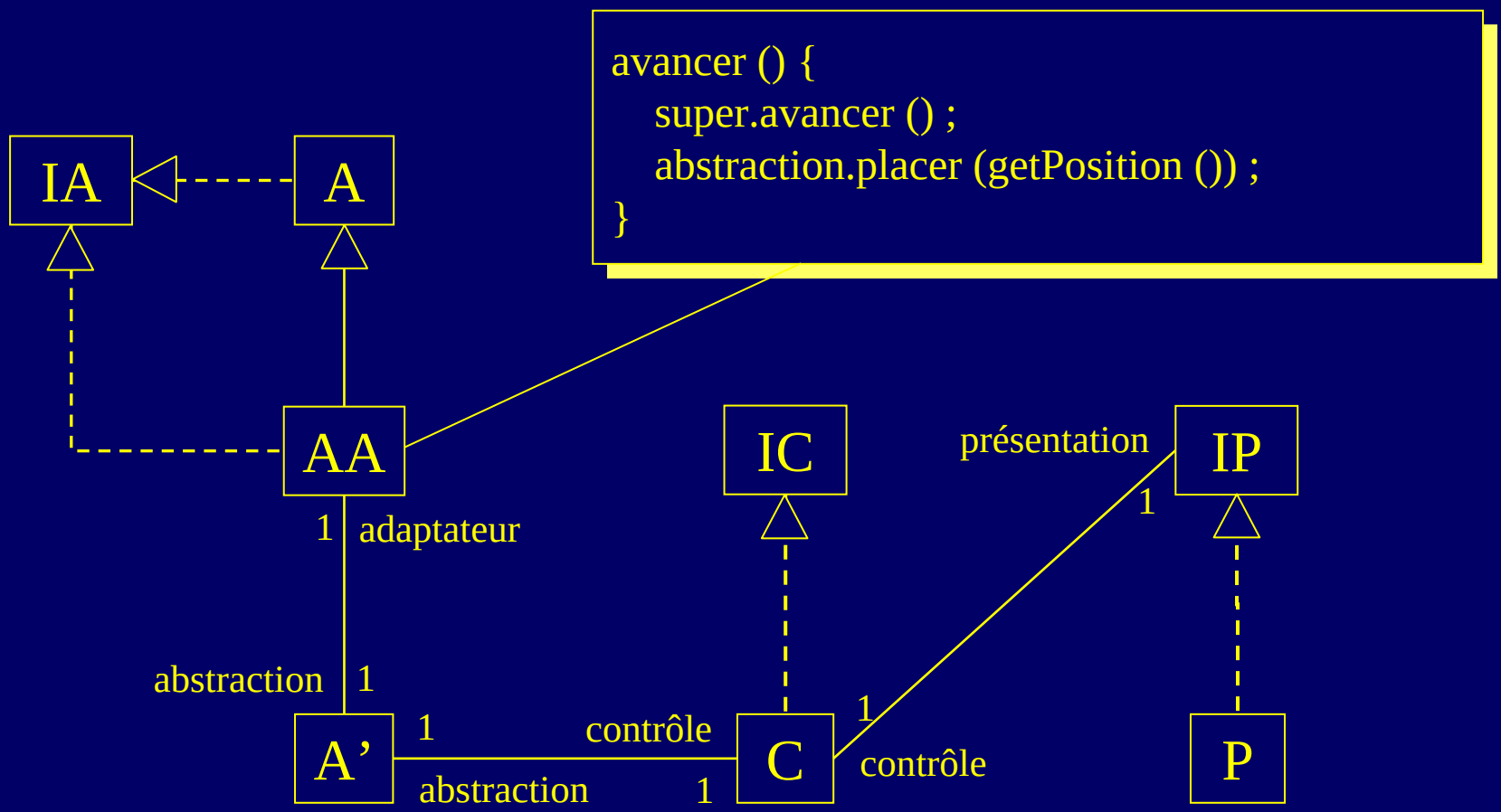
```
avancer () {
    super.avancer () ;
    présentation.placer (getPosition ()) ;
}
```



PAC et cette seconde méthode

- Nombre important de classes et interfaces à définir :
  - mais grâce à l'héritage on ne redéfinit que ce qui est absolument nécessaire dans les contrôles
- Dépendances entre certains composants :
  - à cause de l'héritage on induit des dépendances fortes entre les composants de contrôle et l'implémentation des composants abstraction dont ils héritent
- Facile et rapide à implémenter :
  - même s'il n'y a pas eu d'interfaces de définies
  - grâce à l'héritage

PAC-Amodeus + Proxy + Héritage



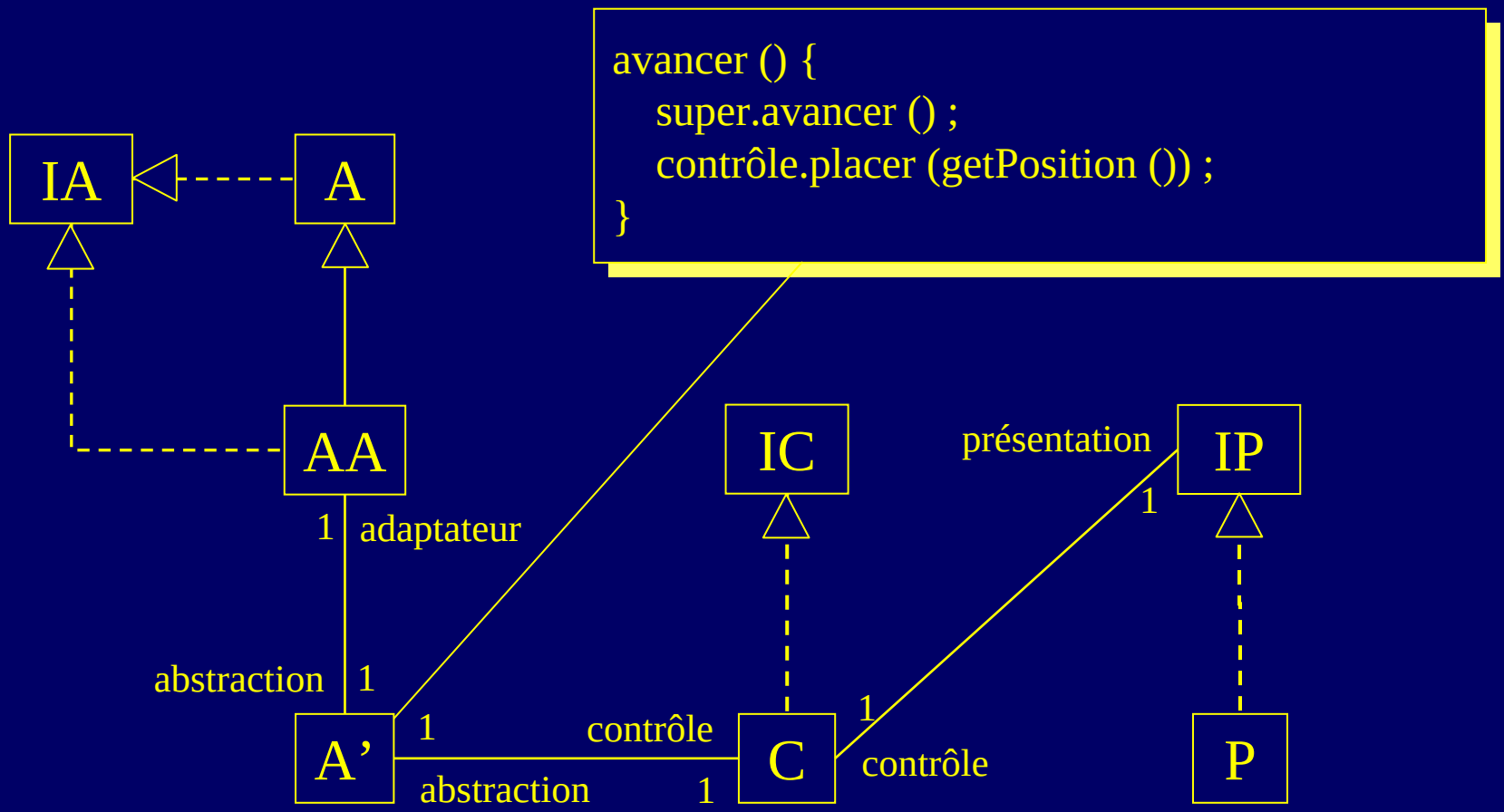
Discussion sur la méthode

- Avantages :
  - les classes initiales ne sont pas modifiées :
    - cela facilite l'évolution du noyau initial, du point de vue de ses créateurs
  - ceux du modèle PAC-Amodeus :
    - principalement une bonne séparation entre le noyau et l'interface utilisateur
- Inconvénients :
  - toujours beaucoup de classes à implémenter...
  - toujours assez inefficace à l'exécution...

Optimisation de la méthode

- Utilisation du mécanisme « Slinky » :
  - suppression des composants de l'ANF :
    - conduit à une perte de portabilité : les agents PAC du contrôleur de dialogue dépendent maintenant directement des objets de simulation
  - les composants Abstraction peuvent hériter directement des objets initiaux
- Les agents PAC peuvent être « incomplets » :
  - ici encore, on peut supprimer les facettes abstraction
  - les composants Contrôle peuvent hériter directement des objets initiaux

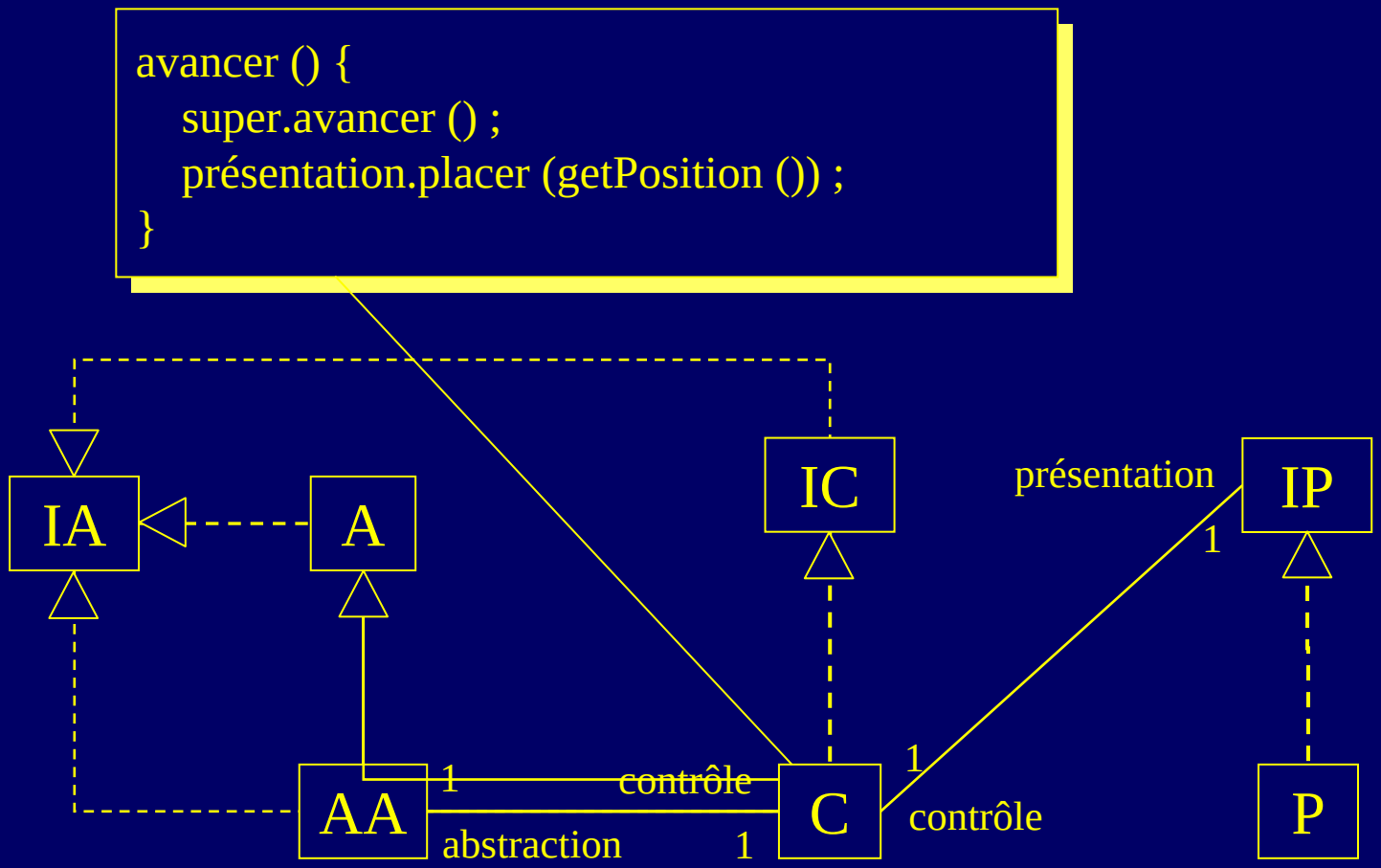
PAC-Amodeus : optimisation Slinky



PAC-Amodeus : optimisation Slinky

- Regroupement de l'ANF et des abstractions
- Indépendance préservée entre contrôle et implémentation de l'application
- Perte de l'indépendance des abstractions vis à vis de l'implémentation des composants applicatifs
- Moins de classes à écrire
- Le contrôle n'est pas forcément le proxy de l'objet applicatif associé

PAC-Amodeus : optimisation PAC



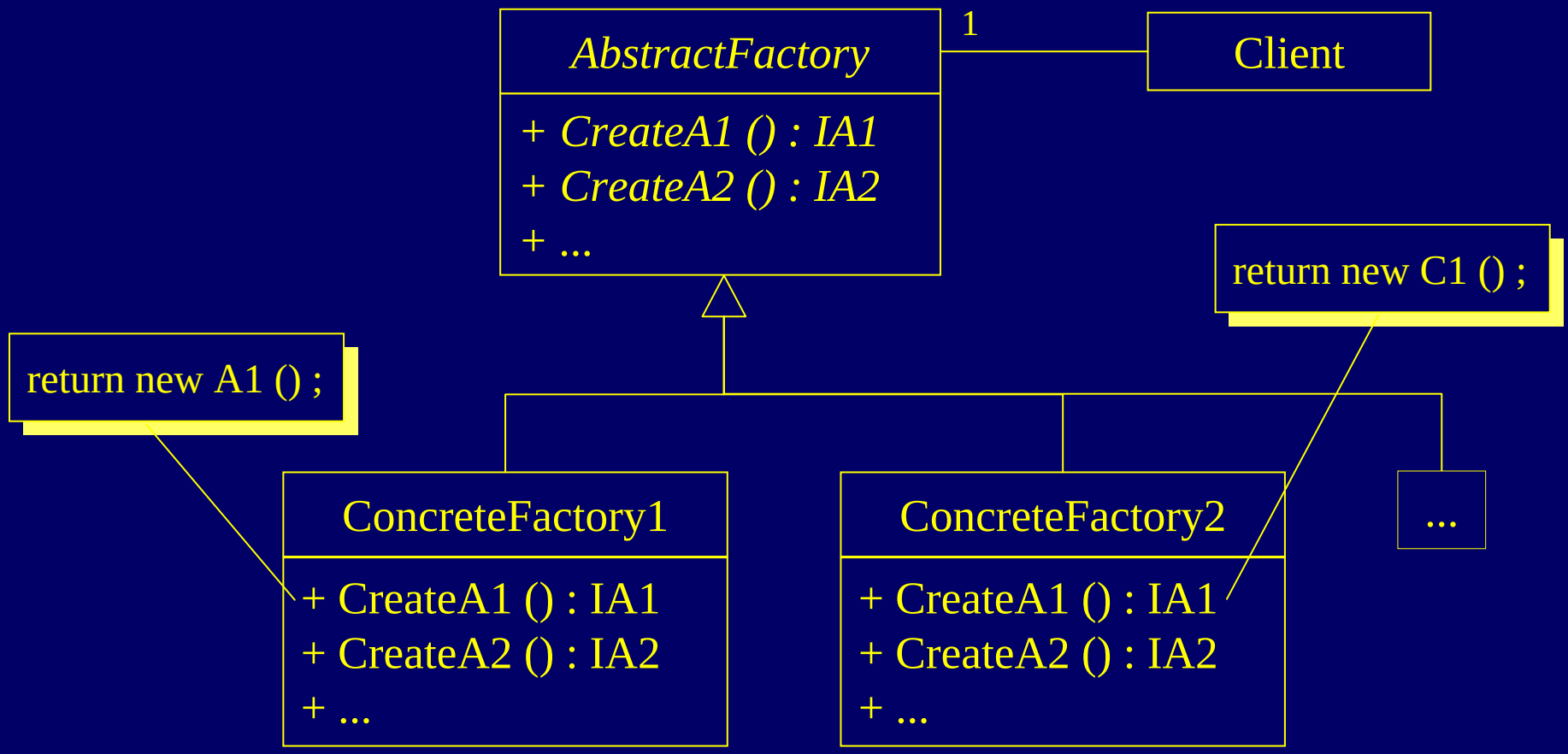
PAC-Amodeus : optimisation PAC

- Indépendance perdue entre contrôle et implémentation de l'application :
  - à cause de l'héritage
- Encore moins de classes et de méthodes à écrire :
  - grâce à l'héritage
- Le contrôle devient le proxy de l'objet applicatif associé :
  - comme c'est le cas avec le modèle PAC
- Équivalence avec l'usage du modèle PAC

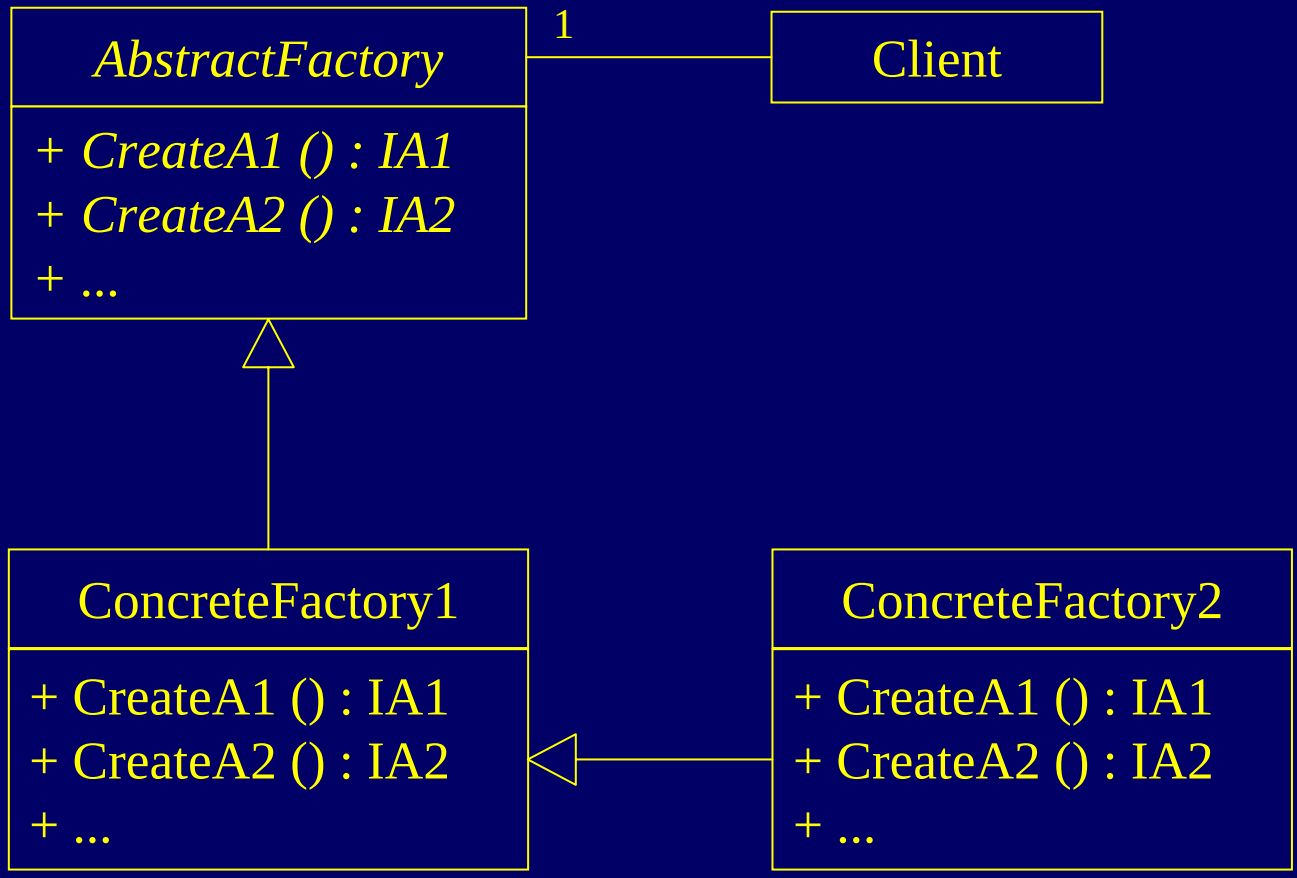
Notre patron « Abstract Factory » ...

- Ici encore, utilisation « particulière » de ce pattern :
  - utilisation de l'héritage entre les différentes « fabriques » d'objets
- Toujours dans un but de réutilisation :
  - pour ne redéfinir que ce qui est vraiment nécessaire...

Patron « Abstract Factory » classique



Notre patron « Abstract Factory »



Conclusion sur la seconde méthode

- Objets initiaux inchangés
- Application clairement distinguée de l'interface
- Coût de développement minimisé :
  - 2 nouvelles classes pour rendre interactif un type
- 2 catégories d'agents PAC :
  - agents initiaux rendus interactifs, liés au NF
  - agents dédiés au dialogue, indépendants du NF
- Structure PAC finale dédiée au dialogue

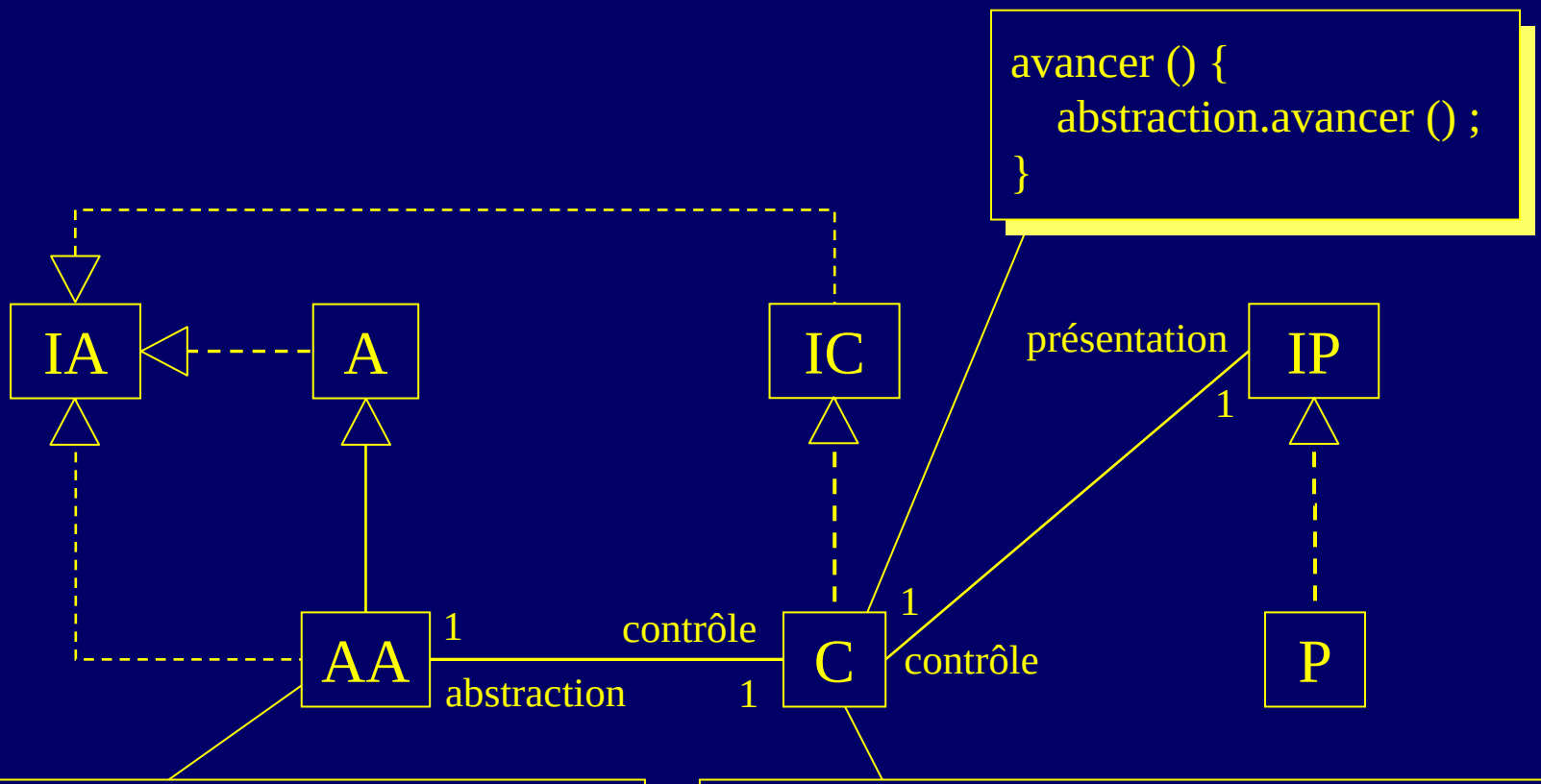
3ème implémentation proposée

- Utiliser le modèle PAC-Amodeus et :
  - le patron Abstract Factory avec des interfaces :
    - de façon à pouvoir utiliser le polymorphisme et la liaison dynamique
  - le patron Proxy + Héritage :
    - pour les composants ANF si c'est nécessaire
    - pour propager les changements d'état à visualiser
  - le patron Observateur + Proxy + Délégation :
    - pour les composants de contrôle
    - pour observer si nécessaire les composants abstraction associés

3ème implémentation proposée

- Faire des optimisations raisonnables :
  - pour réduire les coûts de développement (le nombre de classes) et le manque d'efficacité à l'exécution
  - tout en restant indépendant vis à vis du noyau initial
- Fusion de l'abstraction et de l'ANF
- Ne faire des composants ANF/abstraction que si c'est nécessaire :
  - seulement s'il faut propager des changements d'état à visualiser
  - autrement le contrôle est simple proxy de l'objet initial

PAC-Amodeus + Proxy + Observateur



Conclusion sur la troisième méthode

- Structure assez proche de la seconde méthode utilisant PAC-Amodeus et une seule optimisation
- Implémentation assez différente
- Codage un peu alourdi par l'Observateur
- Méthode plus générique mais moins efficace ?
  - peu adaptée aux simulations temps-réel…
- Mêmes caractéristiques d'indépendance vis à vis du noyau applicatif que la méthode 1 ou que la méthode 2 non (ou peu) optimisée

Plan

- Problématique
- Rappel des modèles PAC et PAC-Amodeus
- Mauvaises utilisations de ces modèles
- Mise en œuvre de PAC et PAC-Amodeus
  - les patrons de conception à appliquer
  - trois mises en œuvres différentes
  - l'exemple : les tours de Hanoï en java avec Swing :
    - trois façons de visualiser la résolution du problème
    - comment aller jusqu'à la manipulation directe ?

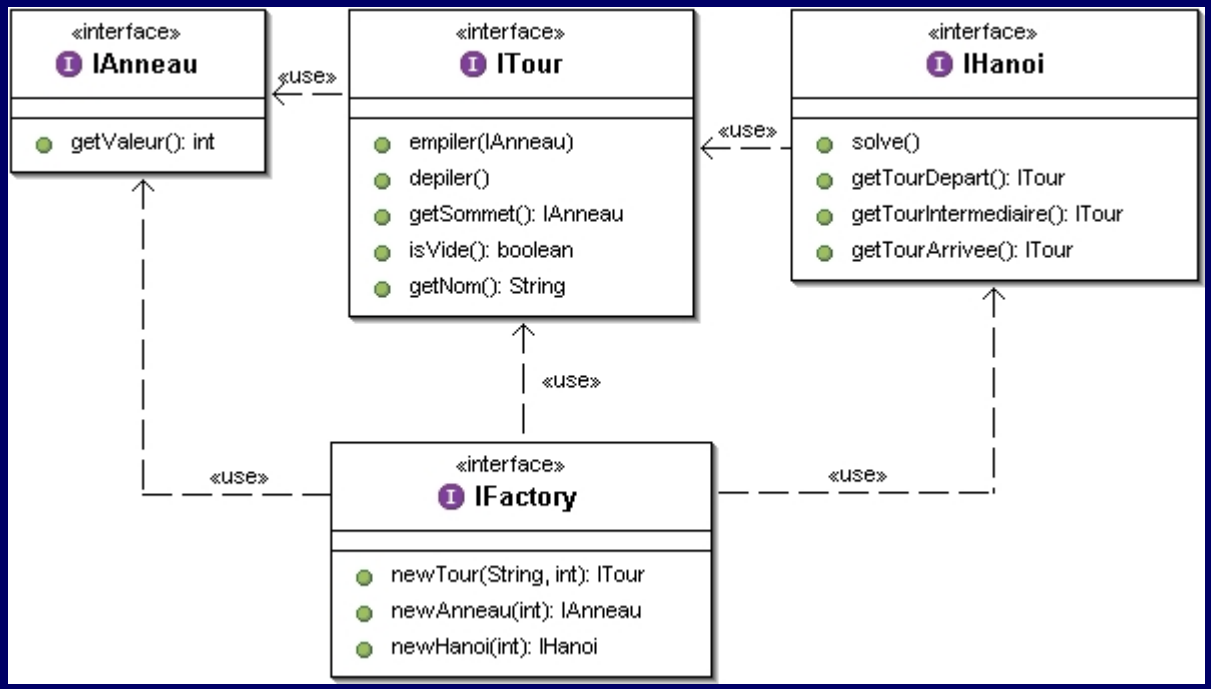
Exemple : les tours de Hanoï

- Une application Java permettant de décrire les actions à réaliser pour résoudre le problème des tours de Hanoï avec N anneaux
- Un package décrivant les concepts à manipuler
- Un package les implémentant
- Utilisation d'une fabrique de composants pour en faciliter l'évolution
- Un programme principal chargé d'instancier la bonne fabrique de composants

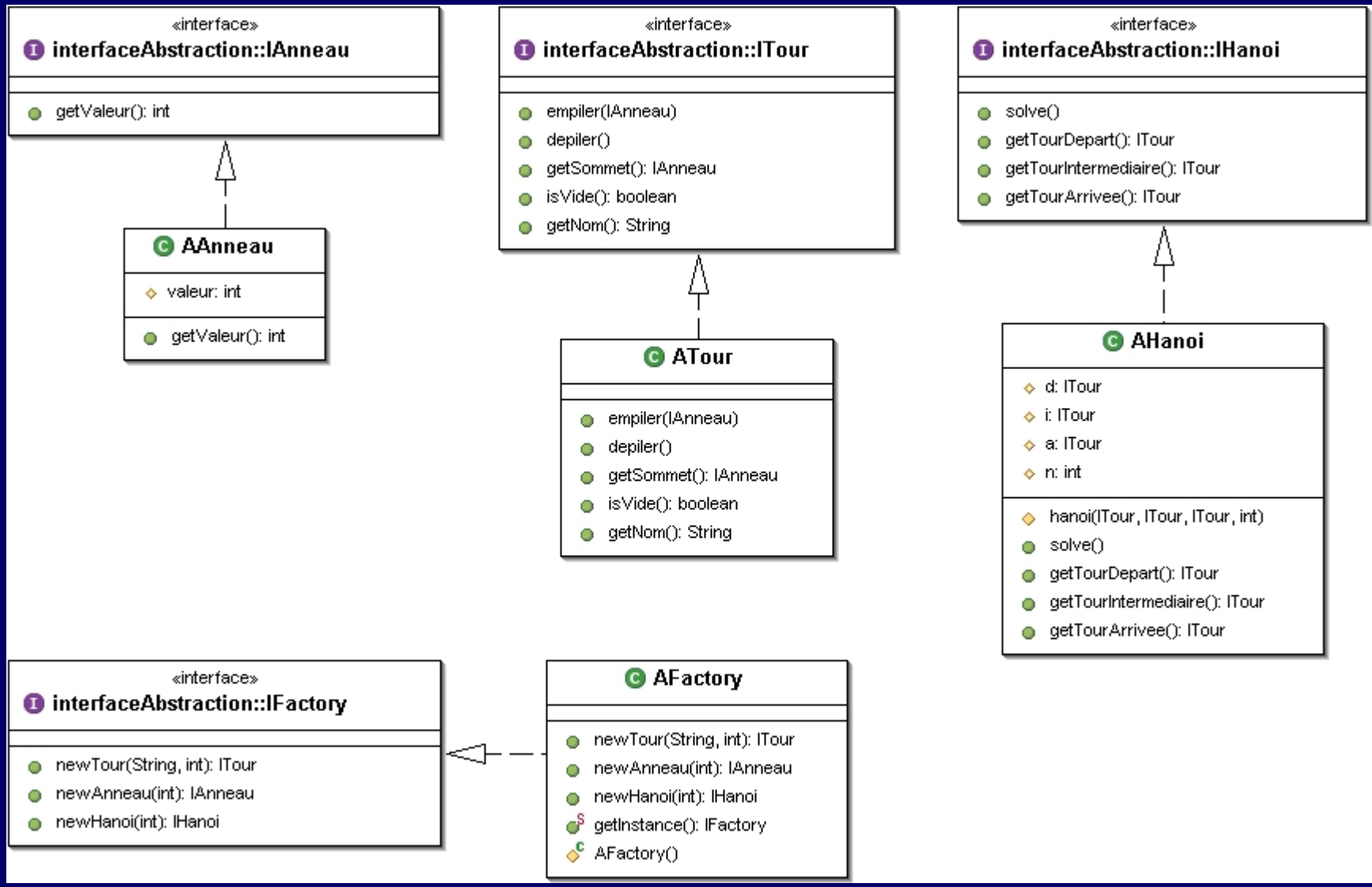
Le noyau applicatif initial

- Déclare les concepts applicatifs nécessaires
  - IAnneau : accès à la valeur d'un anneau
  - ITour : modification du contenu et obtention d'infos
  - IHanoi : accès aux 3 tours et résolution du problème
  - IFactory : création des composants applicatifs
- Implémente ces concepts de façon adéquate :
  - AAnneau, ATour, AHanoi et AFactory
- Un programme principal stocke une fabrique de création des composants dans une classe ConcreteFactory

Le package interfaceAbstraction



Le package abstraction



Utilisation du noyau applicatif

```
public class MainHanoi {
   public static void main (String args []) {
      int n = 2 ;
      if (args.length > 0) {
        n = Integer.parseInt (args [0]) ;
      }
      concreteFactory.ConcreteFactory.setFactory
         (abstraction.AFactory.getInstance ()) ;
      IHanoi h = concreteFactory.ConcreteFactory.
         getFactory ().newHanoi (n) ;
      h.solve () ;
   }
}
```

Aperçu de la classe Hanoi

```
public AHanoi (int n) {
   IFactory f = concreteFactory.ConcreteFactory.getFactory () ;
   this.n = n ;
   d = f.newTour ("d", n) ;
   i = f.newTour ("i", n) ;
   a = f.newTour ("a", n) ;
   for (int ii = n ; ii > 0 ; ii --) {
      d.empiler (f.newAnneau (ii)) ;
   }
}
```

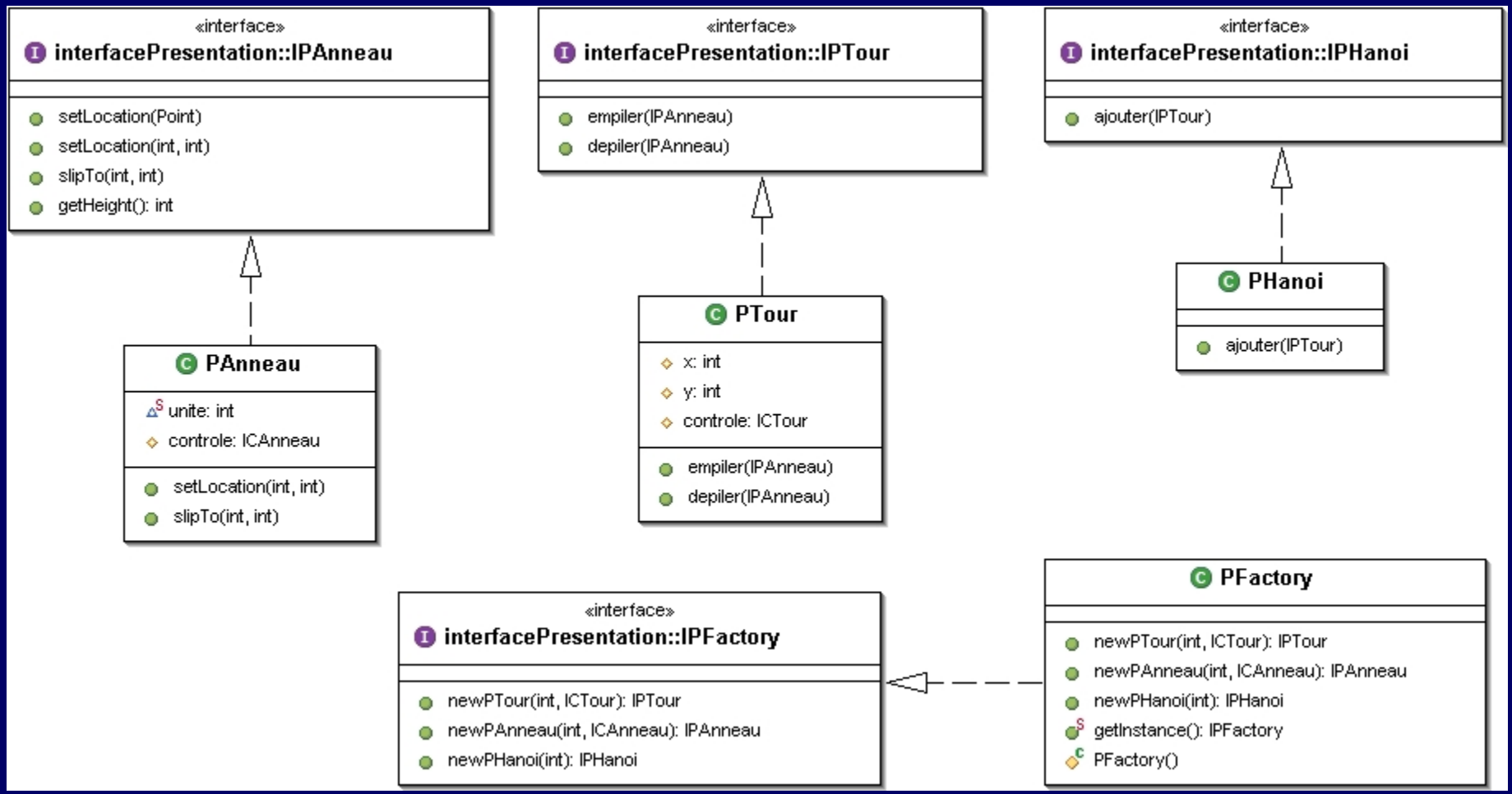
Aperçu de la classe Hanoi

```
protected void hanoi (ITour d, ITour i, ITour a, int n) {
   if (n > 0) {
      hanoi (d, a, i, n - 1) ;
      System.out.println ("déplacer l'anneau "
         + d.getSommet ().getValeur () + " de la tour "
         + d.getNom () + " vers la tour " + a.getNom ()) ;
      IAnneau anneau = d.getSommet () ;
      d.depiler () ;
      a.empiler (anneau) ;
      hanoi (i, d, a, n - 1) ;
   }
}

public void solve () {
   hanoi (d, i, a, n) ;
}
```

Les composants de présentation

- Masquent l'utilisation d'une API graphique :
  - via des interfaces de présentation logique
- Définissent les besoins logiques de présentation :
  - IPAnneau : placement et glissement d'un anneau
  - IPTour : empilement et dépilement d'un IPAnneau
  - IPHanoi : ajout d'un composant IPTour
  - IPfactory : création des présentations
- Sont implémentés (à l'aide de Swing) de façon adéquate :
  - PAnneau, PTour, PHanoi, PFactory

Le package présentation



Aperçu de la classe PAnneau

```
static final int unite = 30 ;

public PAnneau (int v, ICAnneau controle) {
   setSize (unite * v, unite) ;
   setPreferredSize (getSize ()) ;
   setBackground (Color.blue) ;
   this.controle = controle ;
}

public void setLocation (int x, int y) {
   super.setLocation (x - getWidth () / 2, y - getHeight ()) ;
}
```

Aperçu de la classe PTour

```
public PTour (int n, ICTour controle) {
   this.controle = controle ;
   setSize (n * PAnneau.unite + PAnneau.unite / 3,
                (n + 2) * PAnneau.unite) ;
   setPreferredSize (getSize ()) ;
   setBackground (Color.orange) ;
   setLayout (null) ;
   x = getWidth () / 2 ;
   y = getHeight () ;
}
```

Aperçu de la classe PTour

```
public void empiler (IPAnneau pa) {
   Point pos = new Point (getX () + x, pa.getHeight () + getY ()) ;
   pa.slipTo (pos.x, pos.y) ;
   add ((PAnneau)pa, 0) ;
   repaint () ;
   pa.setLocation (x, pa.getHeight ()) ;
   pa.slipTo (x, y) ;
   y = y - pa.getHeight () ;
}
```

Aperçu de la classe PTour

```
public void depiler (IPAnneau pa) {
   pa.slipTo (x, pa.getHeight ()) ;
   remove ((PAnneau)pa) ;
   Point pos = new Point (getX () + x, pa.getHeight () + getY ()) ;
   pa.setLocation (pos.x, pos.y) ;
   getParent ().add ((PAnneau)pa, 0) ;
   getParent ().repaint () ;
   y = y + pa.getHeight () ;
}
```

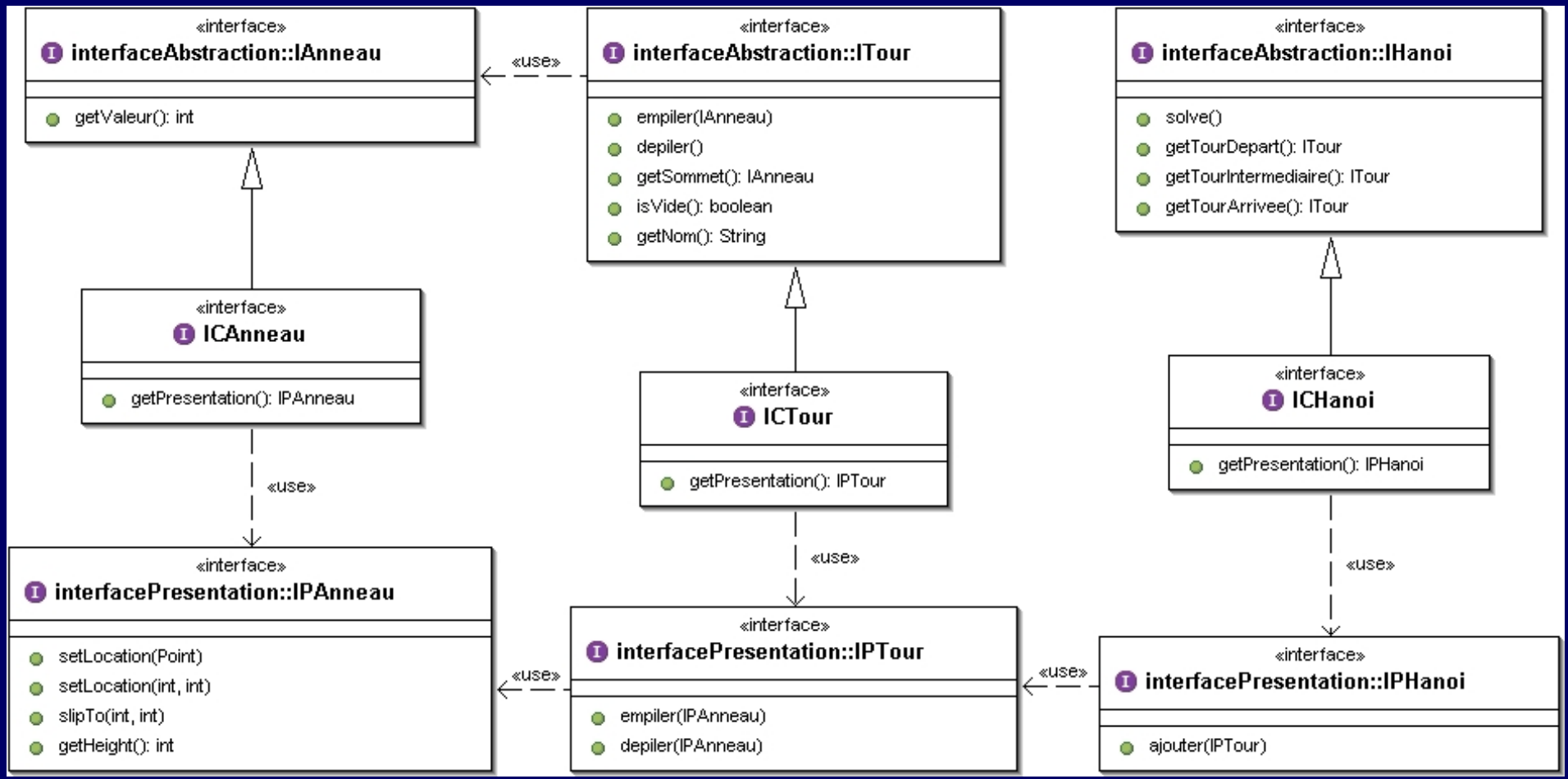
Aperçu de la classe PHanoi

```
public PHanoi (int n) {
   super ("Les tours de Hanoï avec " + n + " anneaux") ;
   getContentPane ().setLayout (new FlowLayout ()) ;
   addWindowListener (new WindowAdapter () {
      public void windowClosing (WindowEvent e) {
         System.exit (0) ; } }) ;
   pack () ;
   setVisible (true) ;
}

public void ajouter (IPTour p) {
   getContentPane ().add ((PTour)p) ;
   pack () ;
   setVisible (true) ;
}
```

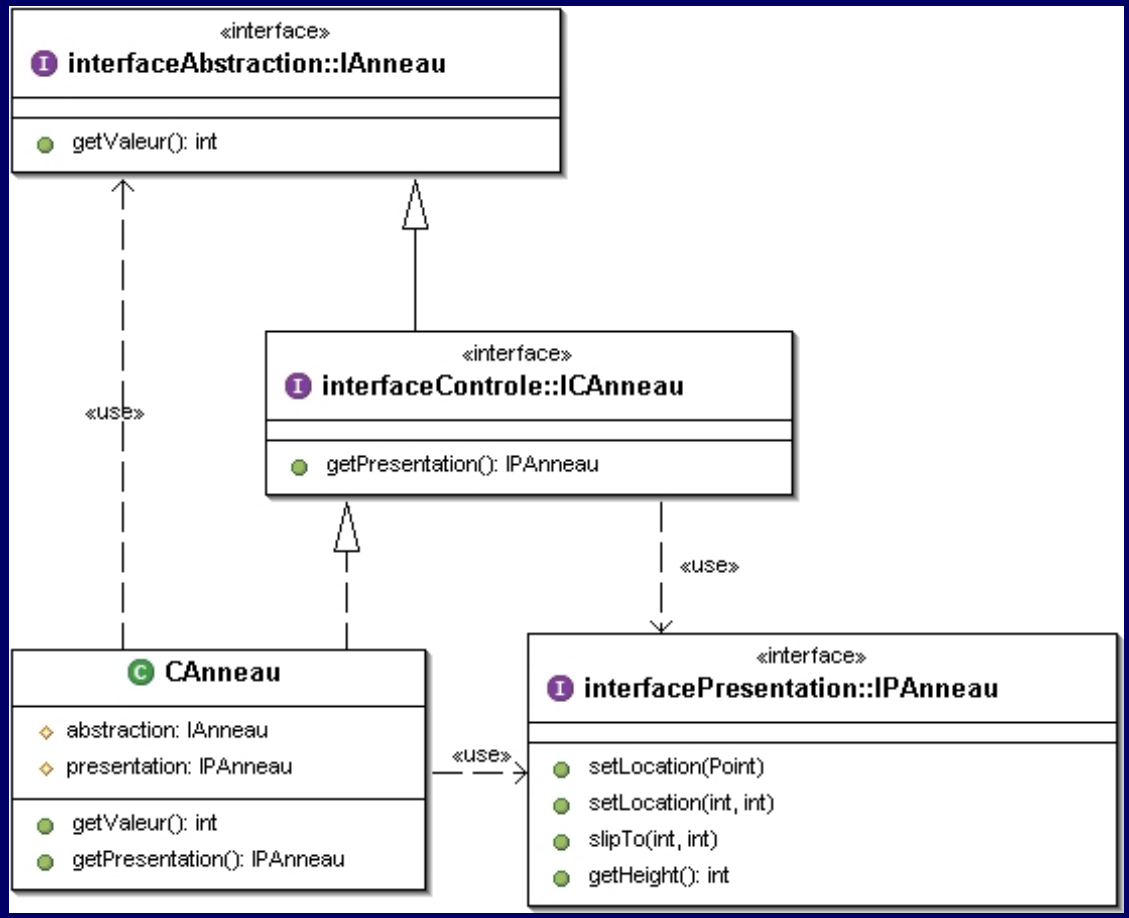
Les composants de contrôle (m1)

- Vont être des proxys des composants applicatifs associés et se substitueront à eux grâce à l'usage d'une fabrique de composants
- Définissent les besoins au niveau du contrôle :
  - ICAnneau, ICTour, ICHAnoi
- Sont implémentés de façon adéquate :
  - CAnneau, CTour, CHanoi

Le package interfaceControle (m1)



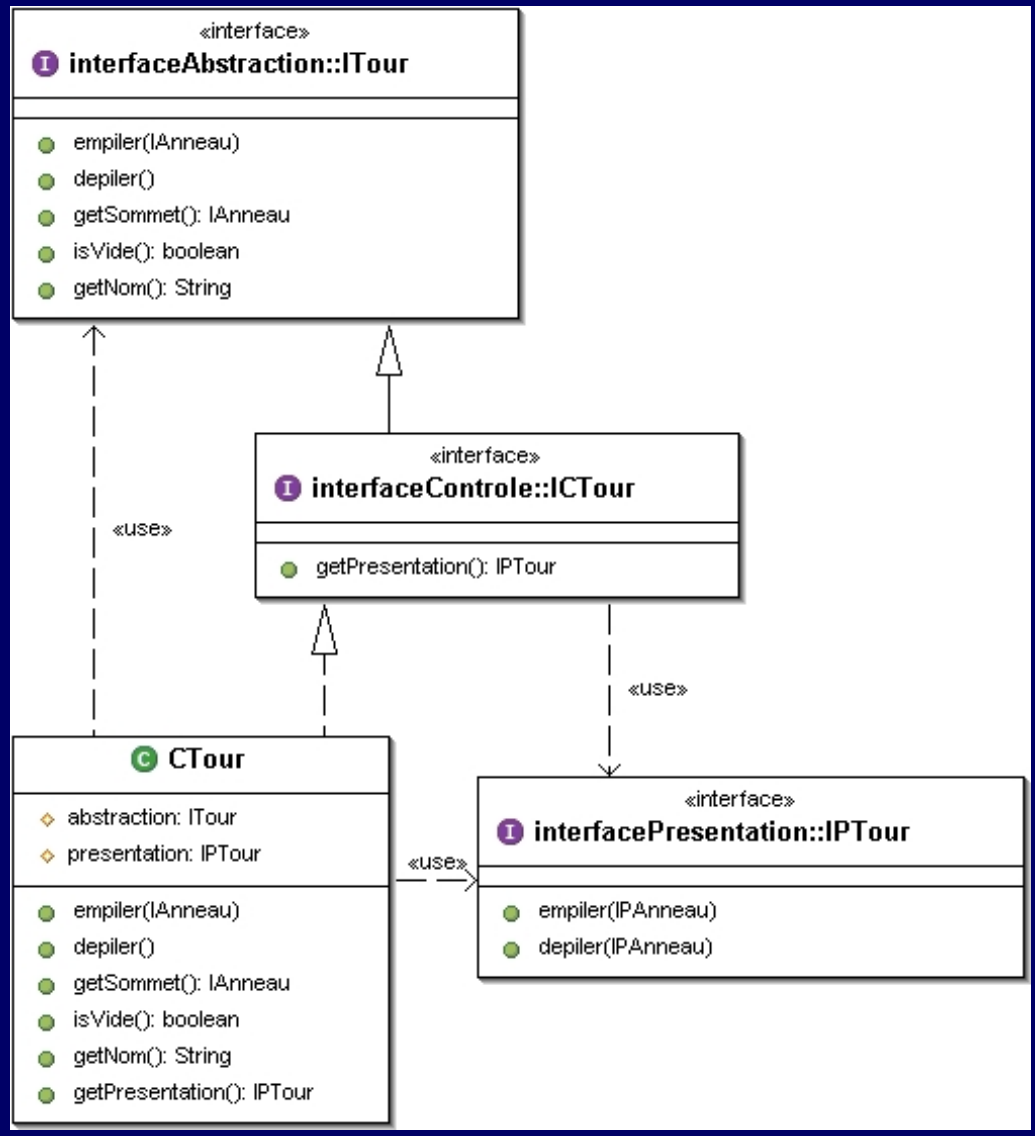
Le package contrôle (m1)



Aperçu de la classe CAnneau (m1)

```
public CAnneau (int v) {
   abstraction = ConcreteFactory.getAFactory ().
      newAnneau (v) ;
   presentation = ConcreteFactory.getPFactory ().
      newPAnneau (v, this) ;
}

public int getValeur () {
   return abstraction.getValeur () ;
}

public IPAnneau getPresentation () {
   return presentation ;
}
```

Le package contrôle (m1)



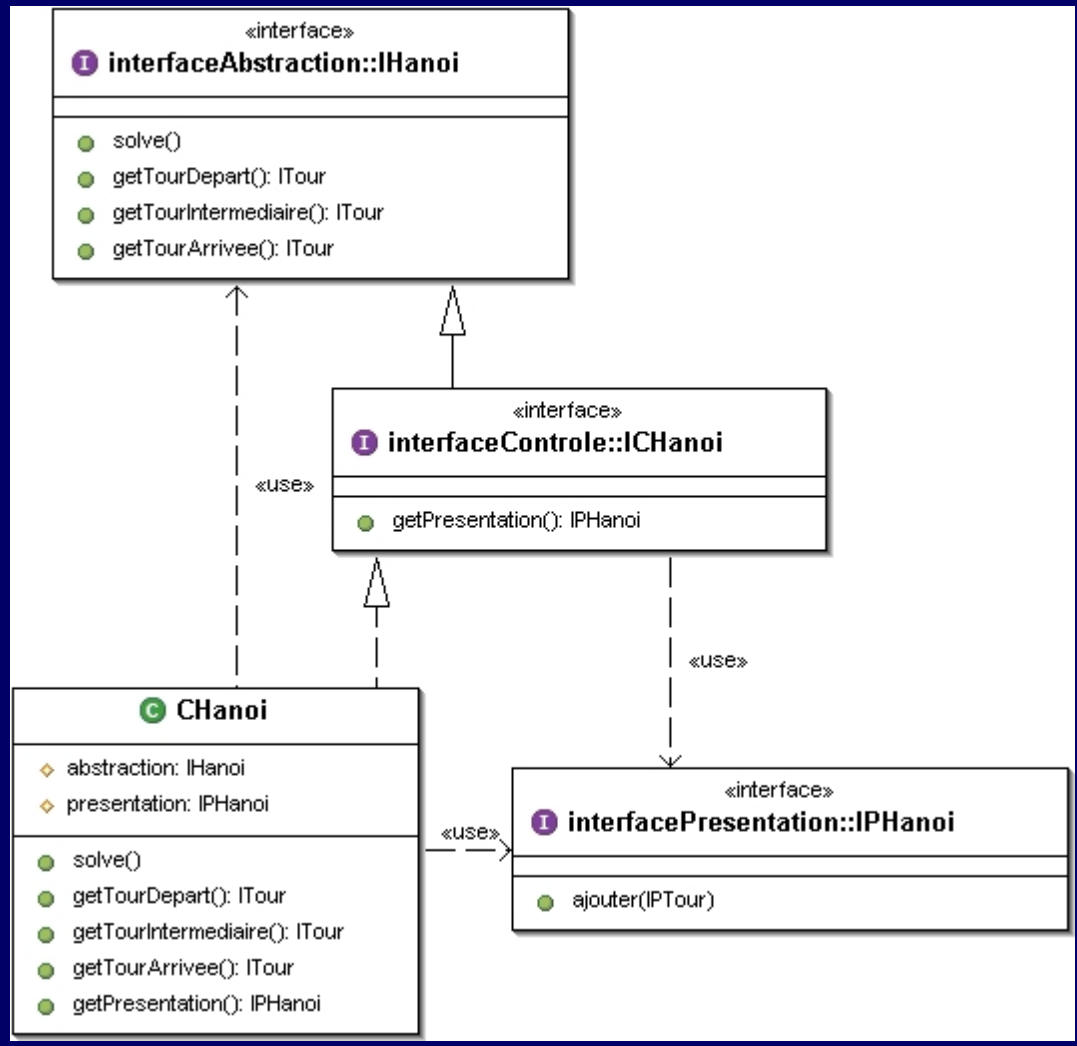
Aperçu de la classe CTour (m1)

```
public CTour (String nom, int nbAnneauxMax) {
   abstraction = ConcreteFactory.getAFactory ().
      newTour (nom, nbAnneauxMax) ;
   presentation = ConcreteFactory.getPFactory ().
      newPTour (nbAnneauxMax, this) ;
}

public void empiler (IAnneau aa) {
   abstraction.empiler (aa) ;
   IPAnneau pa = ((ICAnneau)aa).getPresentation () ;
   presentation.empiler (pa) ;
}

public void depiler () {
   ICAnneau ca = (ICAnneau)getSommet () ;
   abstraction.depiler () ;
   presentation.depiler (ca.getPresentation ()) ;
}
```

Aperçu de la classe CTour (m1)

```
public IAnneau getSommet () {
   return abstraction.getSommet () ;
}

public boolean isVide () {
   return abstraction.isVide () ;
}

public String getNom () {
   return abstraction.getNom () ;
}

public IPTour getPresentation () {
   return presentation ;
}
```

Le package contrôle (m1)



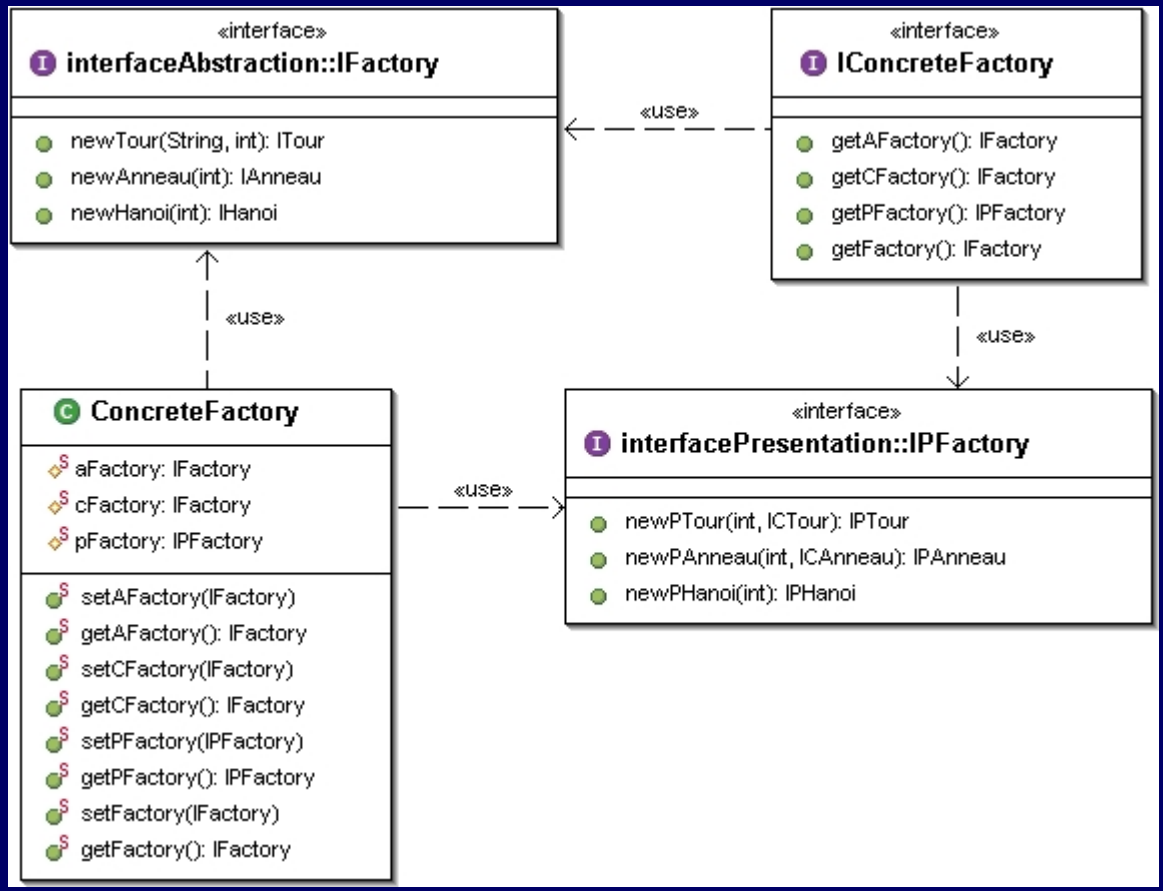
Aperçu de la classe CHanoi (m1)

```
public CHanoi (int n) {
   abstraction = ConcreteFactory.getAFactory ().
      newHanoi (n) ;
   presentation = ConcreteFactory.getPFactory ().
      newPHanoi (n) ;
   presentation.ajouter (((ICTour)abstraction.
       getTourDepart ()).getPresentation ()) ;
   presentation.ajouter ((ICTour)abstraction.
      getTourIntermediaire ()).getPresentation ()) ;
   presentation.ajouter (((ICTour)abstraction.
      getTourArrivee ()).getPresentation ()) ;
}

public void solve () {
   abstraction.solve () ;
}
```

Aperçu de la classe CHanoi (m1)

```
public ITour getTourDepart () {
   return abstraction.getTourDepart () ;
}

public ITour getTourIntermediaire () {
   return abstraction.getTourIntermediaire () ;
}

public ITour getTourArrivee () {
   return abstraction.getTourArrivee () ;
}

public IPHanoi getPresentation () {
   return presentation ;
}
```

La classe ConcreteFactory



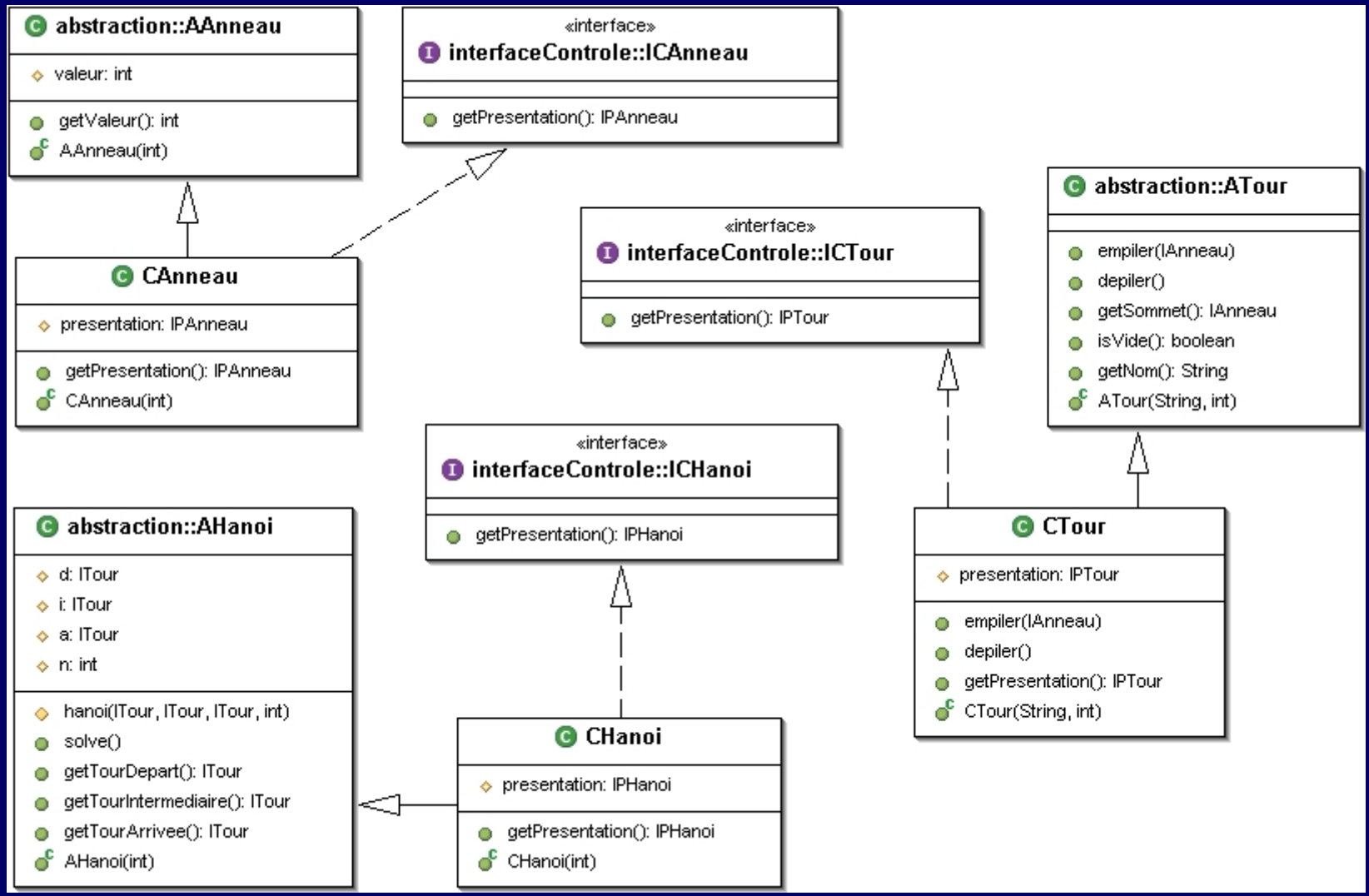
Aperçu du programme principal (m1)

```
public class MainHanoiPAC {
   public static void main (String args []) {
      int n = 5 ;
      if (args.length > 0) {
         n = Integer.parseInt (args [0]) ;
      }
      concreteFactory.ConcreteFactory.
         setAFactory (abstraction.AFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setCFactory (controle.CFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setPFactory (presentation.PFactory.getInstance ()) ;
      IHanoi h = concreteFactory.ConcreteFactory.
         getFactory ().newHanoi (n) ;
      h.solve () ;
   }
}
```

Synthèse de la première méthode

- PAC + proxy + délégation + fabrique de composants
- Abstraction inchangée
- Contrôle indépendant des implémentations :
  - de l'application
  - de la présentation
- Les contrôles offrent l'accès à leur présentation :
  - pour permettre les inclusions de présentations

Le package contrôleHéritage (m2)



Aperçu de la classe CAnneau (m2)

```
public class CAnneau extends AAnneau
                             implements ICAnneau {

   public CAnneau (int v) {
      super (v) ;
      presentation = ConcreteFactory.getPFactory ().
         newPAnneau (v, this) ;
   }

   public IPAnneau getPresentation () {
      return presentation ;
   }

   protected IPAnneau presentation ;

}
```

Aperçu de la classe CTour (m2)

```
public class CTour extends ATour implements ICTour {

  public CTour (String nom, int nbAnneauxMax) {
    super (nom, nbAnneauxMax) ;
    presentation = ConcreteFactory.getPFactory ().
      newPTour (nbAnneauxMax, this) ;
  }

  public void empiler (IAnneau aa) {
    super.empiler (aa) ;
    IPAnneau pa = (((ICAnneau)aa).getPresentation ()) ;
    presentation.empiler (pa) ;
  }

  ...
```

Aperçu de la classe CTour (m2)

```
public void depiler () {
   ICAnneau ca = (ICAnneau)getSommet () ;
   super.depiler () ;
   IPAnneau pa = ca.getPresentation () ;
   presentation.depiler (pa) ;
}

public IPTour getPresentation () {
   return presentation ;
}

…
```

Aperçu de la classe CHanoi (m2)

```
public class CHanoi extends AHanoi implements ICHanoi {
   public CHanoi (int n) {
      super (n) ;
      presentation = ConcreteFactory.getPFactory ().
         newPHanoi (n) ;
      presentation.ajouter (((ICTour)getTourDepart ()).
         getPresentation ()) ;
      presentation.ajouter (((ICTour)getTourIntermediaire ()).
         getPresentation ()) ;
      presentation.ajouter (((ICTour)getTourArrivee ()).
         getPresentation ()) ;
   }

   public IPHanoi getPresentation () {
      return  presentation ;
   }

   …
```

Aperçu de la classe CFactory (m2)

```
public class CFactory implements IFactory {

   public ITour newTour (String nom, int nbAnneauxMax) {
      return (new CTour (nom, nbAnneauxMax)) ;
   }

   public IAnneau newAnneau (int v) {
      return (new CAnneau (v)) ;
   }

   public IHanoi newHanoi (int n) {
      return new CHanoi (n) ;
   }
```

Aperçu de la classe CFactory (m2)

```
    protected CFactory () {
    }

    private static IFactory instance = new CFactory () ;

    public static IFactory getInstance () {
       return instance ;
    }

}
```

Aperçu du programme principal (m2)

```
public class MainHanoiHeritage {
   public static void main (String args []) {
      int n = 5 ;
      if (args.length > 0) {
         n = Integer.parseInt (args [0]) ;
      }
      concreteFactory.ConcreteFactory.
         setAFactory (abstraction.AFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setCFactory (controleHeritage.CFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setPFactory (presentation.PFactory.getInstance ()) ;
      IHanoi h = concreteFactory.ConcreteFactory.
         getFactory ().newHanoi (n) ;
      h.solve () ;
   }
}
```

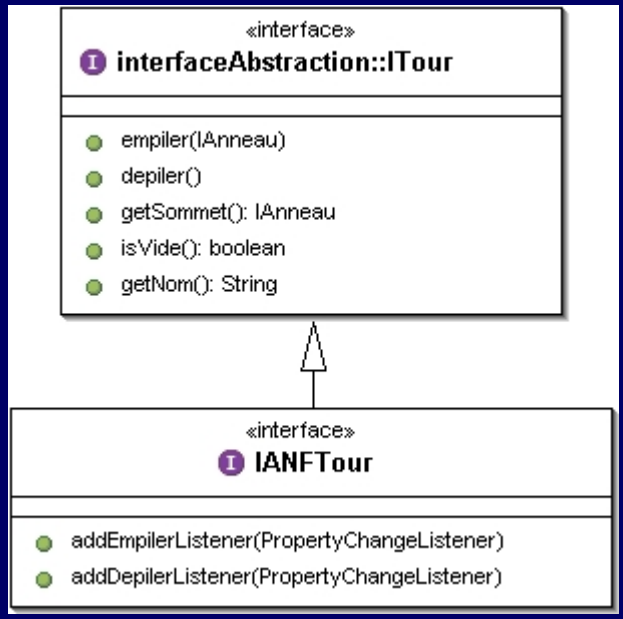
Synthèse de la seconde méthode

- PAC + proxy + héritage + fabrique de composants
- Abstraction inchangée
- Contrôle devenu dépendant de l'implémentation :
  - de l'application
- Plus léger à mettre en œuvre
  - pas absolument besoin des interfaces applicatifs

Adaptation du noyau fonctionnel (m3)

- Seule la notion de tour est concernée :
  - c'est le seul composant qui évolue dynamiquement au cours de l'application (empilement et dépilement d'anneaux)
- Définition de méthodes permettant de s'abonner aux changements d'états suite à :
  - un empilement
  - un dépilement
- Définition de classes pouvant s'abonner à ces changements d'états :
  - on se base ici sur le concept de PropertyChangeListener

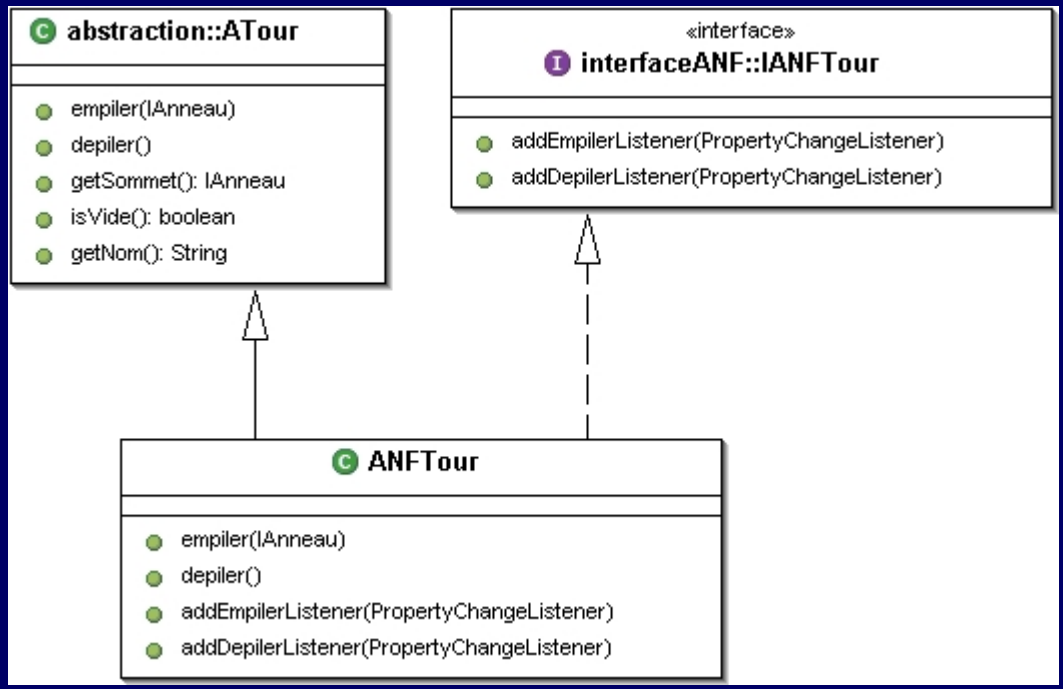
Le package interfaceANF (m3)



Adaptation du noyau fonctionnel (m3)

- Adaptation effective par la classe ANFTour
- Implémentation via les méthodes :
  - addEmpilerListener
    - abonnement aux changements suite aux empilements
  - addDepilerListener
    - Abonnement aux changements suite aux dépilements
  - empiler
    - notification des modifications aux abonnés après empilement
  - depiler
    - notification des modifications aux abonnés après dépilement

Le package ANF (m3)



Aperçu de la classe ANFTour (m3)

```
public class ANFTour extends ATour implements IANFTour {

   public ANFTour (String nom, int n) {
      super (nom, n) ;
   }

   private PropertyChangeSupport support =
      new PropertyChangeSupport (this) ;

   public void addEmpilerListener (PropertyChangeListener s) {
      support.addPropertyChangeListener ("empiler", s) ;
   }

   public void addDepilerListener (PropertyChangeListener s) {
      support.addPropertyChangeListener ("depiler", s) ;
   }
```

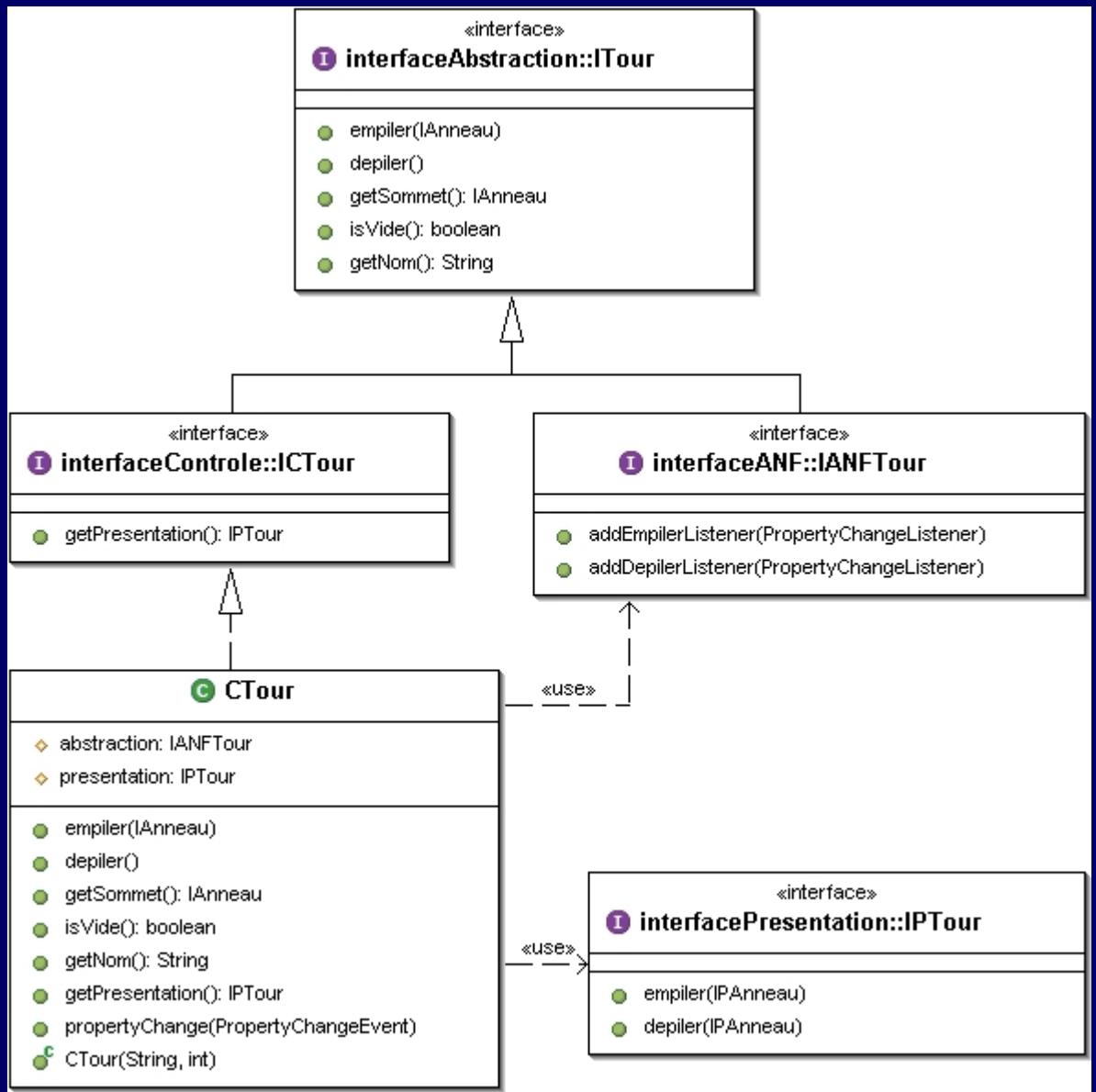
Aperçu de la classe ANFTour (m3)

```
    public void empiler (IAnneau a) {
       super.empiler (a) ;
       support.firePropertyChange ("empiler", null, a) ;
    }


    public void depiler () {
       IAnneau a = getSommet () ;
       super.depiler () ;
       support.firePropertyChange ("depiler", null, a) ;
    }

}
```

Le package controleANF (m3)

- Le contrôle de tour peut s'abonner aux évolutions de l'ANFTour associé :
  - en implémentant PropertyChangeListener
    - méthode propertyChange

Le package contrôleANF (m3)



Aperçu de la classe CTour (m3)

```
public class CTour implements ICTour,
                                        propertyChangeListener {

  public CTour (String nom, int nbAnneauxMax) {
    abstraction = (IANFTour)ConcreteFactory.
      getAFactory ().newTour (nom, nbAnneauxMax) ;
    presentation = ConcreteFactory.getPFactory ().
     newPTour (nbAnneauxMax, this) ;
    abstraction.addDepilerListener(this) ;
    abstraction.addEmpilerListener(this) ;
  }

  public IAnneau getSommet () {
    return abstraction.getSommet () ;
  }
```

Aperçu de la classe CTour (m3)

```
public void propertyChange (PropertyChangeEvent evt) {
   if (evt.getPropertyName ().equals ("empiler")) {
      ICAnneau a = (ICAnneau)evt.getNewValue () ;
      presentation.empiler (a.getPresentation ()) ;
   } else if (evt.getPropertyName ().equals ("depiler")) {
      ICAnneau a = (ICAnneau)evt.getNewValue () ;
      presentation.depiler (a.getPresentation ()) ;
   }
}

public void empiler (IAnneau aa) {
   abstraction.empiler (aa) ;
}

public void depiler () {
   abstraction.depiler () ;
}
...
```

Aperçu du programme principal (m3)

```
public class MainHanoiANF {
   public static void main (String args []) {
      int n = 3 ;
      if (args.length > 0) {
         n = Integer.parseInt (args [0]) ;
      }
      concreteFactory.ConcreteFactory.
         setAFactory (ANF.ANFFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setCFactory (controleANF.CFactory.getInstance ()) ;
      concreteFactory.ConcreteFactory.
         setPFactory (presentation.PFactory.getInstance ()) ;
      IHanoi h = concreteFactory.ConcreteFactory.
         getFactory ().newHanoi (n) ;
      h.solve () ;
   }
}
```

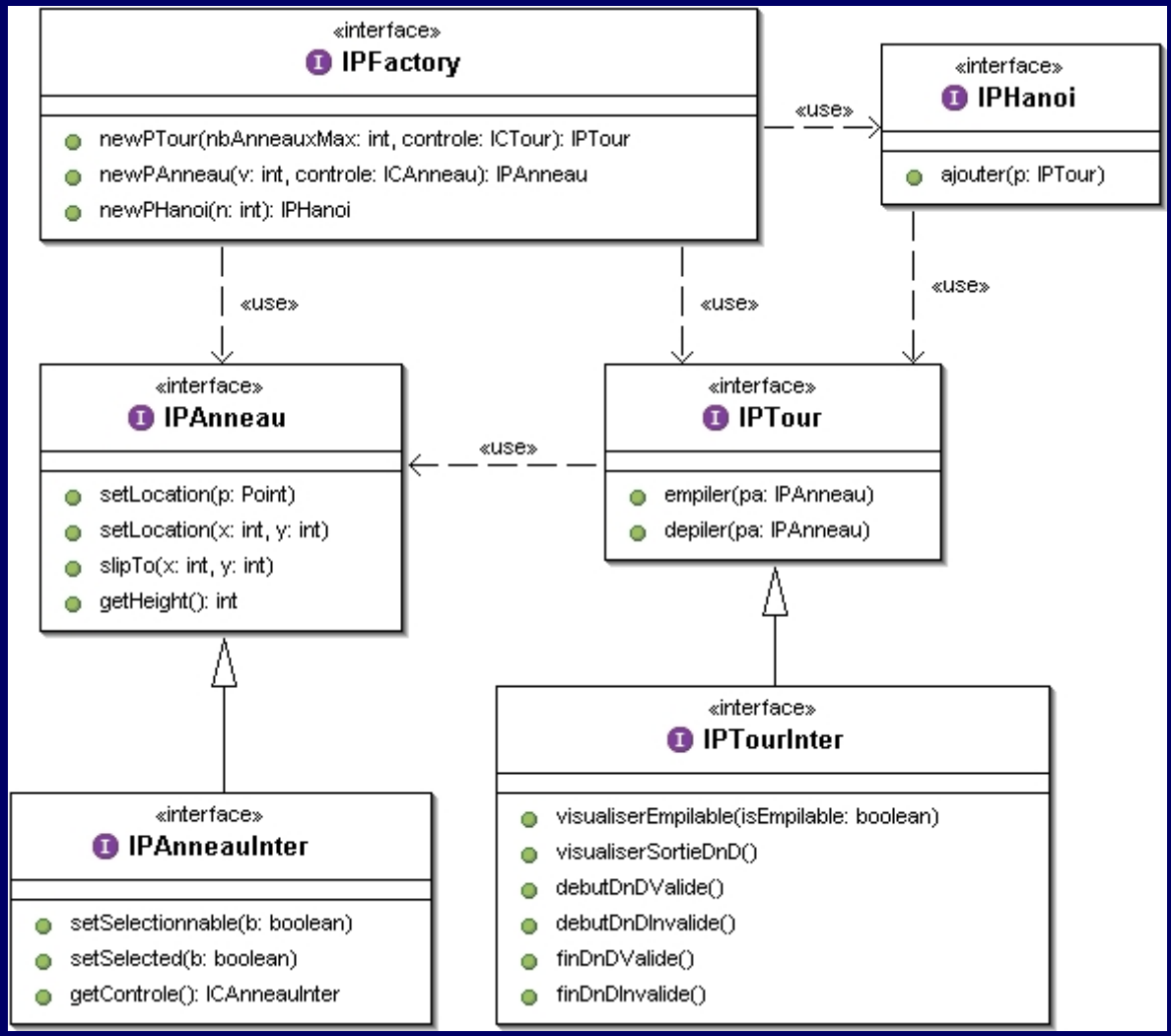
Synthèse de la troisième méthode

- PAC-Amodeus + proxy + héritage + obervateur + fabrique de composants
- Abstraction inchangée
- Contrôle indépendant de l'implémentation :
  - de l'application
  - de la présentation
- Plus lourd à mettre en œuvre :
  - définition de protocoles entre ANF et contrôle

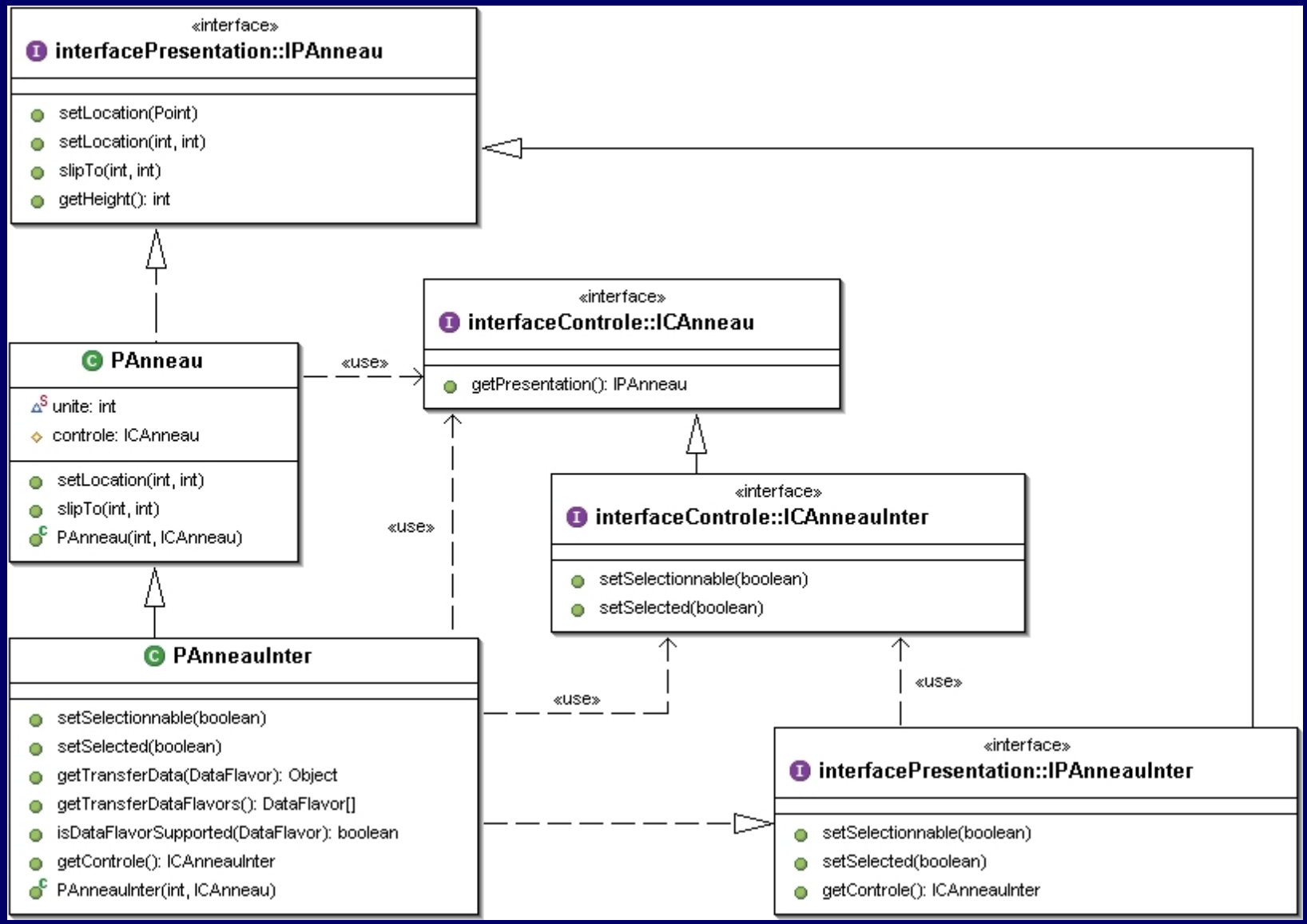
Passage à la manipulation directe (i)

- Nouveaux besoins à prendre en compte :
  - actions de l'utilisateur sur des anneaux
  - reconnaissance de l'entrée ou de la sortie d'un anneau sur une tour
  - nouvelles méthodes de vérification à ajouter
  - nouvelles méthodes de mise en évidence à ajouter

Le package interfacePrésentation (i)



Le package présentation (i)



Aperçu de la classe PAnneauInter

```
public class PAnneauInter
   extends Panneau
   implements IPAnneauInter, Transferable {

   public PAnneauInter (int v, ICAnneau controle) {
      super (v, controle) ;
   }

   public ICAnneauInter getControle () {
      return (ICAnneauInter)controle ;
   }
```

Aperçu de la classe PAnneauInter

```
public void setSelectionnable (boolean b) {
  if (b) {
    setCursor (new Cursor (Cursor.HAND_CURSOR)) ;
    setBackground (Color.cyan) ;
  } else {
     setCursor (new Cursor (Cursor.WAIT_CURSOR)) ;
     setBackground (Color.blue) ;
  }
}

public void setSelected (boolean b) {
   if (b) {
      setBackground (Color.magenta) ;
   } else {
      setBackground (Color.blue) ;
   }
}
```

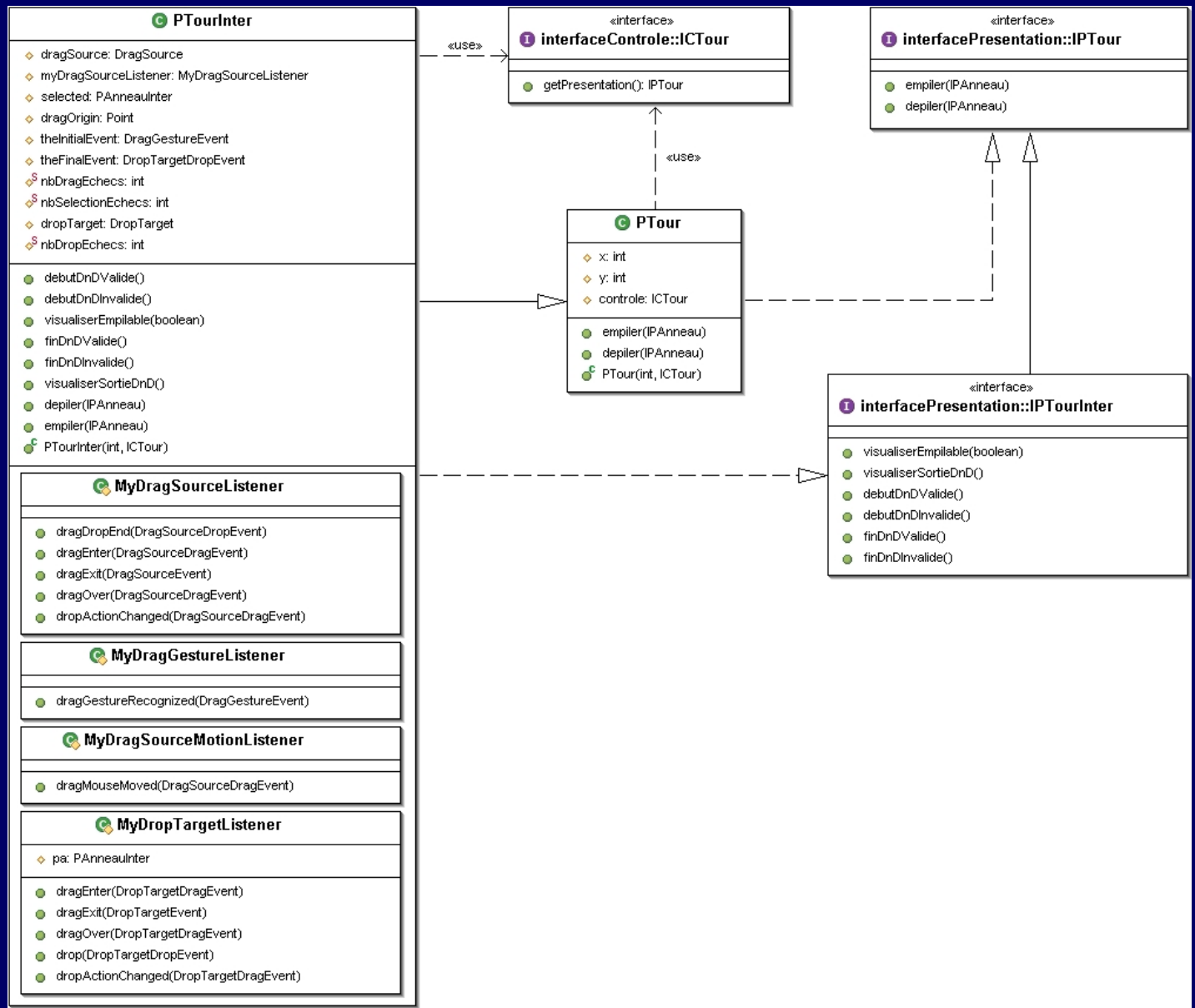
Aperçu de la classe PAnneauInter

```
public Object getTransferData (DataFlavor flavor) {
   Object result = null ;
   if (flavor.isMimeTypeEqual
         (DataFlavor.javaJVMLocalObjectMimeType)) {
     result = this ;
   }
   return (result) ;
}

public DataFlavor [] getTransferDataFlavors () {
   DataFlavor data [] = new DataFlavor [1] ;
   try {
     data [0] = new DataFlavor
        (DataFlavor.javaJVMLocalObjectMimeType) ;
   } catch (java.lang.ClassNotFoundException e) {
   }
   return (data) ;
}
```

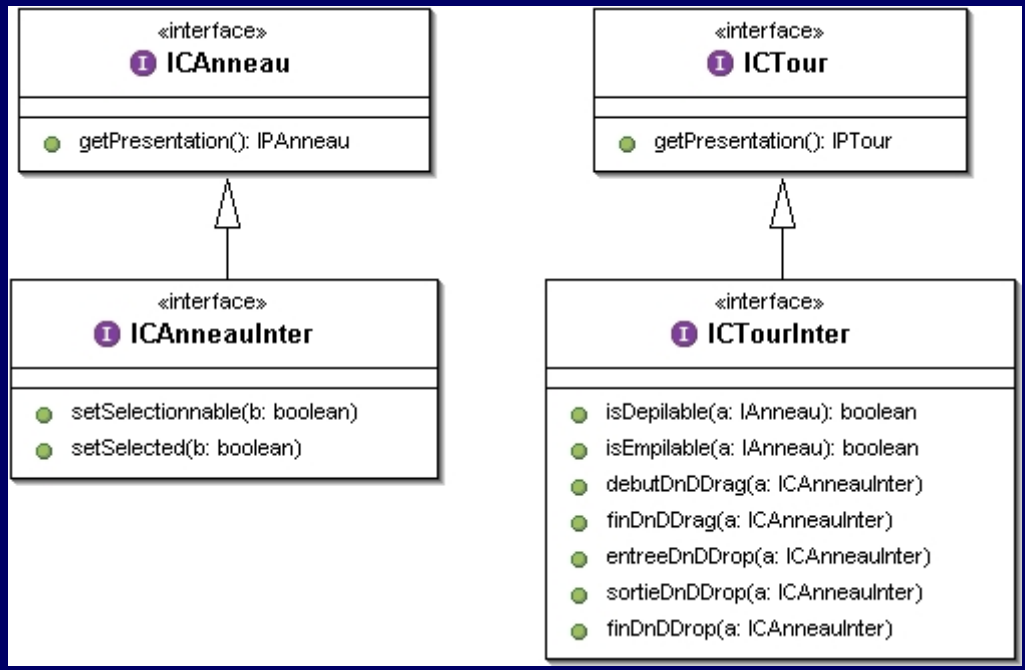
Aperçu de la classe PAnneauInter

```
    public boolean isDataFlavorSupported (DataFlavor flavor) {
       boolean result = false ;
       if (flavor.isMimeTypeEqual
          (DataFlavor.javaJVMLocalObjectMimeType)) {
         result = true ;
       }
       return (result) ;
    }

}
```

Le package présentation (i)



Le package interfaceContrôle (i)



Aperçu de la classe PTourInter

```
public class PTourInter extends PTour
                                implements IPTourInter {

  //-------------------------------------------------------------
  // pour le "drag"
  //-------------------------------------------------------------
  protected DragSource dragSource = null ;
  protected MyDragSourceListener
     myDragSourceListener = null ;
  protected PAnneauInter selected = null ;
  protected Point dragOrigin = new Point (0, 0) ;
  protected DragGestureEvent theInitialEvent ;
  protected DropTargetDropEvent theFinalEvent ;
  protected static int nbDragEchecs = 0 ;
  protected static int nbSelectionEchecs = 0 ;
```

Aperçu de la classe PTourInter

```
protected class MyDragSourceListener implements
   DragSourceListener {

   public void dragDropEnd (DragSourceDropEvent event) {
      if (event.getDropSuccess ()) {
         repaint () ;
      } else {
         ((ICTourInter)controle).finDnDDrag
            (selected.getControle ()) ;
      }
   }
```

Aperçu de la classe PTourInter

```
    public void dragEnter (DragSourceDragEvent event) {
      event.getDragSourceContext ().setCursor
        (new Cursor (Cursor.MOVE_CURSOR)) ;
    }

    public void dragExit (DragSourceEvent event) { }

    public void dragOver (DragSourceDragEvent event) { }

    public void dropActionChanged
      (DragSourceDragEvent event) { }

}
```

Aperçu de la classe PTourInter

```
protected class MyDragGestureListener
   implements DragGestureListener {

   public void dragGestureRecognized
         (DragGestureEvent event) {
      selected = null ;
      try {
         selected = (PAnneauInter)getComponentAt
            (event.getDragOrigin ()) ;
      } catch (Exception e) { }
      if (selected != null) {
         theInitialEvent = event ;
         nbSelectionEchecs  = 0 ;
         dragOrigin = event.getDragOrigin () ;
         ((ICTourInter)controle).debutDnDDrag
            ((ICAnneauInter)selected.getControle()) ;
```

Aperçu de la classe PTourInter

```
        } else {
            nbSelectionEchecs ++ ;
            if (nbSelectionEchecs == 3) {
                JOptionPane.showMessageDialog (null,
    "Il faut cliquer sur un anneau pour amorcer le Drag'n Drop",
                "Avertissement de reconnaissance de Drag'n Drop",
                    JOptionPane.INFORMATION_MESSAGE) ;
                nbSelectionEchecs = 0 ;
            }
        }
    }
}
```

Aperçu de la classe PTourInter

```
protected class MyDragSourceMotionListener
  implements DragSourceMotionListener {

  public void dragMouseMoved
       (DragSourceDragEvent event) {
    selected.setLocation (
        event.getLocation ().x –
        getRootPane ().getParent ().getX (),
        event.getLocation ().y –
        getRootPane ().getParent ().getY ()) ;
  }

}
```

Aperçu de la classe PTourInter

```
//-------------------------------------------------------------
// pour le "drop"
//-------------------------------------------------------------
protected DropTarget dropTarget = null ;
protected static int nbDropEchecs = 0 ;

protected class MyDropTargetListener
   implements DropTargetListener {
      protected PAnneauInter pa = null ;
      public void dragExit (DropTargetEvent event) {
         ((ICTourInter)controle).sortieDnDDrop
            ((ICAnneauInter)pa.getControle()) ;
      }
```

Aperçu de la classe PTourInter

```
    public void dragOver (DropTargetDragEvent event) {
    }

    public void drop (DropTargetDropEvent event) {
       theFinalEvent = event ;
       ((ICTourInter)controle).finDnDDrop
          ((ICAnneauInter)pa.getControle ()) ;
    }

    public void dropActionChanged
       (DropTargetDragEvent event) {
    }

}
```

Aperçu de la classe PTourInter

```
public void dragEnter (DropTargetDragEvent evt) {
   DropTargetDropEvent event = new
      DropTargetDropEvent (
         evt.getDropTargetContext (),
         evt.getLocation (),
         evt.getDropAction (),
         evt.getSourceActions ()) ;
   try {
      Transferable transferable = event.getTransferable () ;
```

Aperçu de la classe PTourInter

```
            if (transferable.isDataFlavorSupported (new
               DataFlavor
                 (DataFlavor.javaJVMLocalObjectMimeType))) {
               evt.acceptDrag (DnDConstants.ACTION_MOVE) ;
               pa = (PAnneauInter)transferable.
                  getTransferData (new DataFlavor
                     (DataFlavor.javaJVMLocalObjectMimeType)) ;
            }
         } catch (java.io.IOException exception) {
         } catch (UnsupportedFlavorException ufException) {
         } catch (java.lang.ClassNotFoundException e) {
         }
        ((ICTourInter)controle).entreeDnDDrop
          ((ICAnneauInter)pa.getControle ()) ;
    }
}
```

Aperçu de la classe PTourInter

```
public void debutDnDValide () {
   dragSource.startDrag (theInitialEvent,
                         DragSource.DefaultMoveNoDrop,
                         selected, myDragSourceListener) ;
   repaint () ;
   nbDragEchecs = 0 ;
}

public void visualiserEmpilable (boolean isEmpilable) {
   if (isEmpilable) {
      setBackground (Color.green) ;
   } else {
      setBackground (Color.red) ;
   }
   getParent ().repaint () ;
}
```

Aperçu de la classe PTourInter

```
public void debutDnDInvalide () {
   nbDragEchecs ++ ;
   if (nbDragEchecs == 3) {
      JOptionPane.showMessageDialog (null,
         "Il est interdit de déplacer un anneau "  +
         "qui n'est pas au sommet d'une tour",
         "Avertissement de départ de Drag'n Drop",
         JOptionPane.INFORMATION_MESSAGE) ;
      nbDragEchecs = 0 ;
   }
}
```

Aperçu de la classe PTourInter

```
public void finDnDValide () {
   theFinalEvent.acceptDrop
      (DnDConstants.ACTION_MOVE) ;
   theFinalEvent.getDropTargetContext ().
      dropComplete (true) ;
   nbDropEchecs = 0 ;
}

public void visualiserSortieDnD () {
   setBackground (Color.yellow) ;
}
```

Aperçu de la classe PTourInter

```
public void finDnDInvalide () {
   nbDropEchecs ++ ;
   theFinalEvent.rejectDrop () ;
   if (nbDropEchecs == 3) {
      JOptionPane.showMessageDialog (null,
         "Il est interdit de poser un anneau " +
         "sur un plus anneau plus petit que lui",
         "Avertissement d'arrivée de Drag'n Drop",
         JOptionPane.INFORMATION_MESSAGE) ;
      nbDropEchecs = 0 ;
   }
}
```

Aperçu de la classe PTourInter

```
public void depiler (IPAnneau pa) {
   remove ((PAnneau)pa) ;
   pa.setLocation (getX () + dragOrigin.x –
      getRootPane ().getParent ().getX (),
      getY () + dragOrigin.y –
      getRootPane ().getParent ().getY ()) ;
   getParent ().add ((PAnneau)pa, 0) ;
   y = y + pa.getHeight () ;
}

public void empiler (IPAnneau pa) {
   add ((PAnneau)pa, 0) ;
   if (getParent () != null) {
      getParent ().repaint () ;
   }
   pa.setLocation (x, y) ;
   y = y - pa.getHeight () ; repaint () ;
}
```

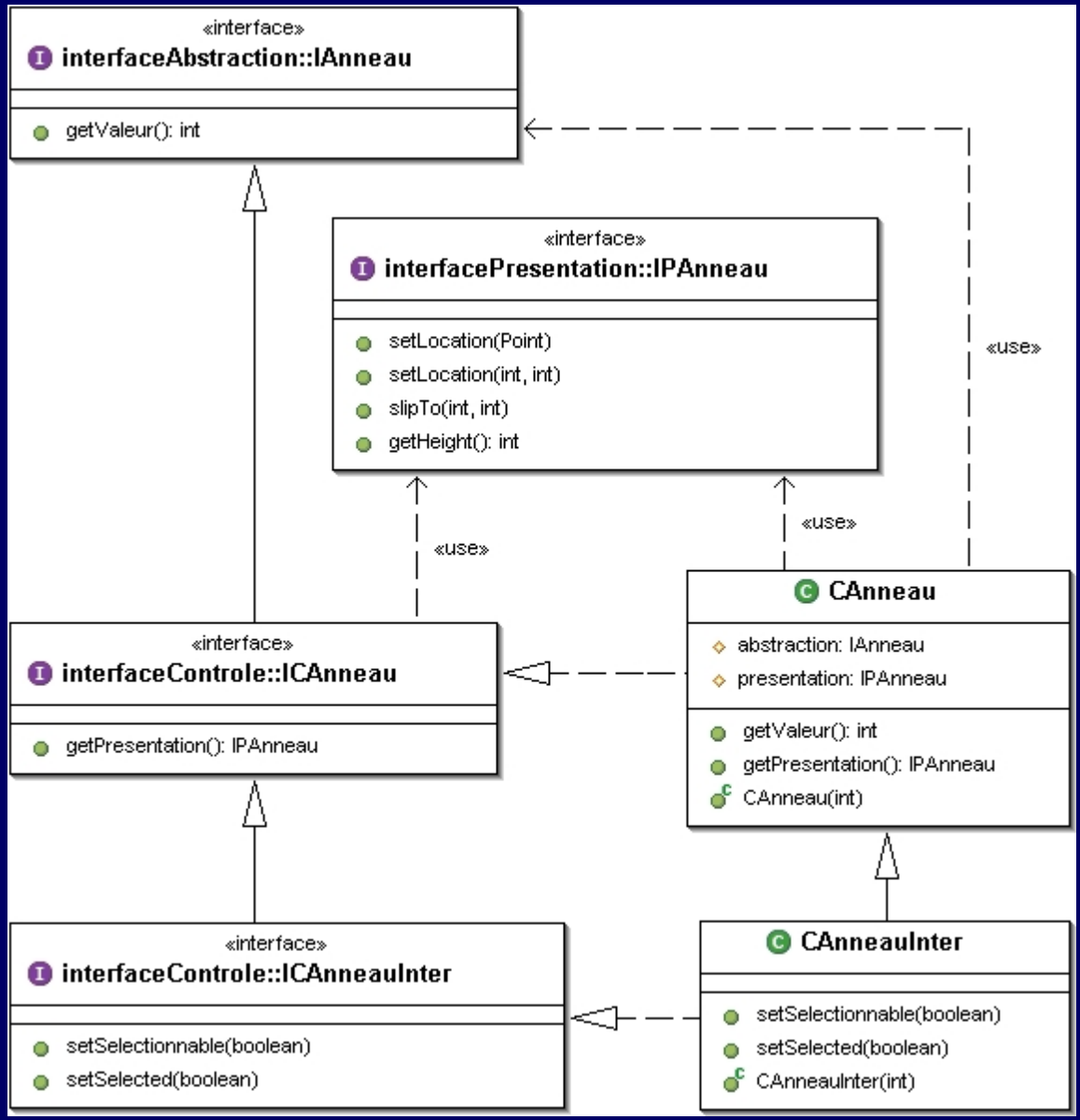
Aperçu de la classe PTourInter

```
public PTourInter (int n, ICTour controle) {
   super (n, controle) ;
   this.controle = controle ;
   setSize (n * PAnneauInter.unite + PAnneauInter.unite / 3,
                (n + 2) * PAnneauInter.unite) ;
   setPreferredSize (getSize ()) ;
   setBackground (Color.yellow) ;
   setLayout (null) ;
   x = getWidth () / 2 ;
   y = getHeight () ;
```

Aperçu de la classe PTourInter

```
        // pour le "drag"
        myDragSourceListener = new MyDragSourceListener () ;
        dragSource = new DragSource () ;
        dragSource.createDefaultDragGestureRecognizer (this,
           DnDConstants.ACTION_MOVE,
          new MyDragGestureListener ()) ;
        dragSource.addDragSourceMotionListener (
           new MyDragSourceMotionListener ()) ;
        // pour le "drop"
        dropTarget = new DropTarget (this,
           new MyDropTargetListener ()) ;
    }

}
```

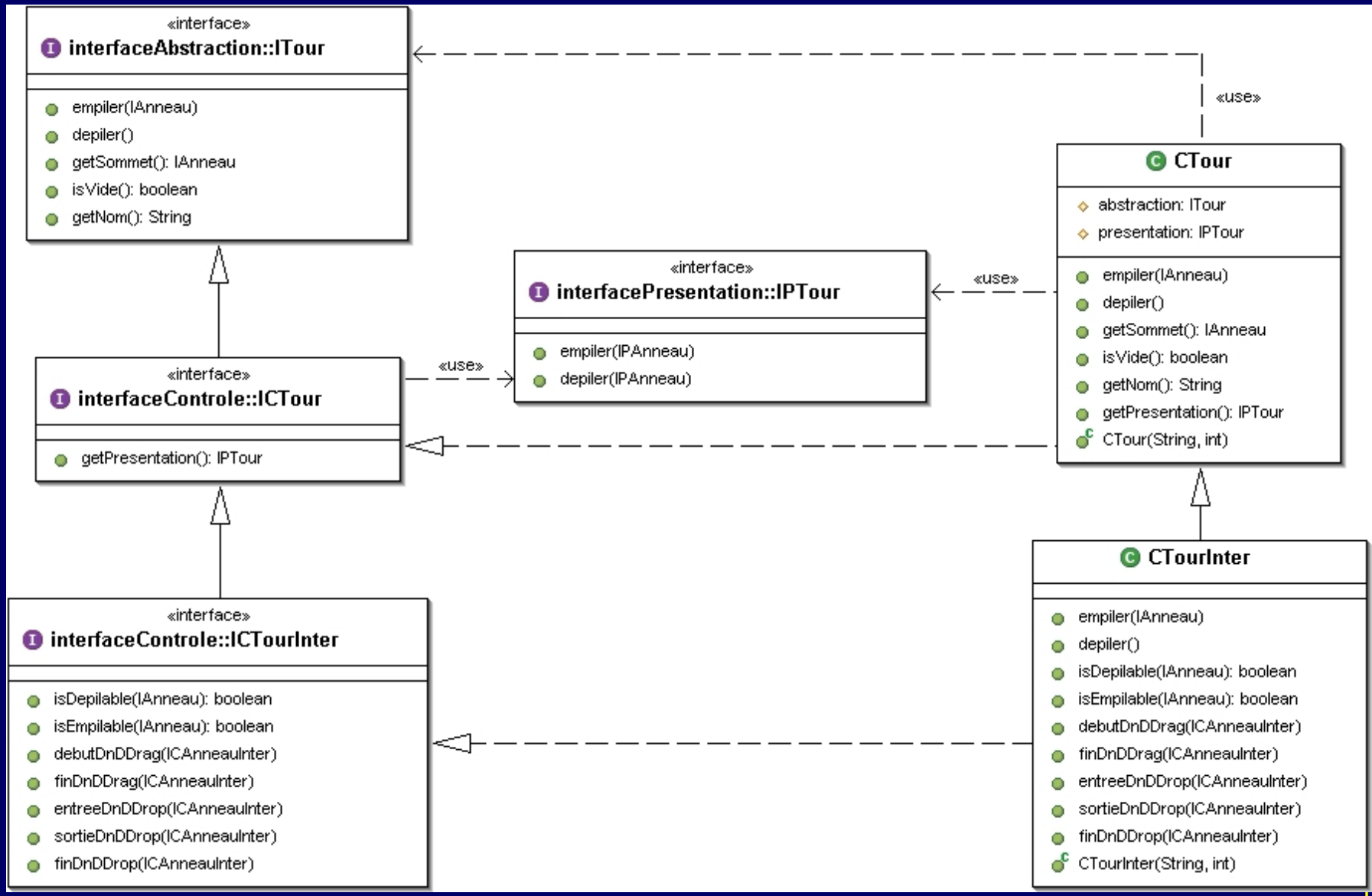
Le package contrôle (i)



Aperçu de la classe CAnneauInter

```
public class CAnneauInter extends CAnneau
                                  implements ICAnneauInter {

   public CAnneauInter (int v) {
      super (v) ;
   }

   public void setSelectionnable (boolean b) {
      ((IPAnneauInter)presentation).setSelectionnable (b) ;
   }

   public void setSelected (boolean b) {
      ((IPAnneauInter)presentation).setSelected (b) ;
   }

}
```

Le package contrôle (i)



Aperçu de la classe CTourInter

```
public class CTourInter extends CTour
                                implements ICTourInter {

  public CTourInter (String nom, int nbAnneauxMax) {
    super (nom, nbAnneauxMax) ;
  }

  public void empiler (IAnneau aa) {
    if (! isVide ()) {
      ((ICAnneauInter)getSommet ()).setSelectionnable (false) ;
    }
    super.empiler (aa) ;
    ((ICAnneauInter)aa).setSelected (false) ;
    ((ICAnneauInter)aa).setSelectionnable (true) ;
  }
```

Aperçu de la classe CTourInter

```
public void depiler () {
   ICAnneauInter ca = (ICAnneauInter)getSommet () ;
   ca.setSelectionnable (false) ;
   ca.setSelected (true) ;
   super.depiler () ;
   if (! isVide ()) {
      ((ICAnneauInter)getSommet ()).setSelectionnable (true) ;
   }
}

public boolean isEmpilable (IAnneau a) {
   boolean result = isVide () ;
   if (! result) {
     result = (a.getValeur () < getSommet ().getValeur ()) ;
   }
   return (result) ;
}
```

Aperçu de la classe CTourInter

```
public boolean isDepilable (IAnneau a) {
    return (a == getSommet ()) ;
}

public void debutDnDDrag (ICAnneauInter a) {
    if (isDepilable (a)) {
        depiler () ;
        ((IPTourInter)presentation).debutDnDValide () ;
    } else {
        ((IPTourInter)presentation).debutDnDInvalide () ;
    }
}

public void finDnDDrag (ICAnneauInter a) {
    empiler (a) ;
}
```

Aperçu de la classe CTourInter

```
public void entreeDnDDrop (ICAnneauInter a) {
   ((IPTourInter)presentation).visualiserEmpilable
      (isEmpilable (a)) ;
}
public void sortieDnDDrop (ICAnneauInter a) {
   ((IPTourInter)presentation).visualiserSortieDnD () ;
}

public void finDnDDrop (ICAnneauInter a) {
   if (isEmpilable (a)) {
      empiler (a) ;
      ((IPTourInter)presentation).finDnDValide () ;
   } else {
      ((IPTourInter)presentation).finDnDInvalide () ;
   }
   ((IPTourInter)presentation).visualiserSortieDnD () ;
}
}
```

Synthèse du passage à l'interaction

- Ajout d'une couche logique pour s'abstraire des mécanismes de drag'n drop offerts par Java :
  - seuls les composants de présentation sont liés à cette technologie
- Tout à fait compatible avec les méthodologies proposées

D’un point de vue génie logiciel...

- On sait enfin mettre en œuvre un modèle d’architecture d’IHM (et de 3 façons…)
- Les démarches sont efficaces (!…?)
- Le noyau applicatif initial est :
  - inchangé
  - clairement séparé de l’interface utilisateur
- Nous avons minimisé :
  - coûts de conception et de développement
  - perte d’efficacité à l’exécution

D’un point de vue IHM...

- L’essentiel est préservé...
- 2 catégories d’agents PAC :
  - les agents liés aux composants de l’application
    - ceux dont on a présenté la construction…
  - les agents dédiés au dialogue utilisateur
    - les autres...
- La structure du contrôleur de dialogue :
  - est indépendante de celle du noyau initial
  - peut être centrée sur l’utilisateur

Conclusion

- La méthode de transformation proposée :
  - est générale, pour tout noyau objet utilisant des patterns de création tels que le pattern Abstract Factory
  - est basée sur les modèles PAC et PAC-Amodeus
  - repose essentiellement sur les patrons Proxy et Abstract Factory :
    - et sur la délégation (méthode 1)
    - ou sur l'héritage (méthode 2)
    - ou sur l'observateur (méthode 3)
  - utilise le polymorphisme et la liaison dynamique
- Cette méthode ne modifie pas le noyau initial !

Au sujet des fabriques

- Si beaucoup de classes dans l'application :
  - la fabrique de composants est énorme…
- Réaliser l'application en plusieurs triplets de packages :
  - (abstraction, contrôle, présentation)
- Faire des « Fabriques de Fabriques » :
  - une fabrique principale pour l'application interactive
  - une fabrique de composants par package

Validation… et perspectives

- Méthode validée dans les langages :
  - Java (AWT : application + applet, Swing)
  - C++ (X11, API graphique 3D OpenInventor)
- Méthode utilisée dans le contexte d'OpenMASK :
  - pour rendre interactifs des objets de simulation
  - C++ (+ API graphique 3D Performer)
- Méthode utilisée dans un contexte industriel :
  - e-manation (Zoé et Léa…)
- Le présenter comme un Design Pattern… ?

Bibliographie

- Thierry Duval, Laurence Nigay :
  - « Implémentation d'une application de simulation selon le modèle PAC-Amodeus », IHM'99
- Thierry Duval, François Pennaneac'h :
  - « Using the PAC-Amodeus Model and Design Patterns to Make Interactive an Existing Object-Oriented Kernel », TOOL'S Europe 2000
- Frantz Degrigny, Thierry Duval :
  - « Utilisation du modèle PAC-Amodeus pour une réutilisation optimale de code dans le développement de plusieurs versions d'un logiciel commercial », IHM'2004