

Le modèle des traits : présentation et discussion

Simon Denier & Pierre Cointe

Journée OCM
Lille - 16 mars 2004

Les traits de Schärli et al.

- Motivation
 - pallier aux défauts de l'héritage pour la réutilisation (duplication de code, conflit de nom)
- Objectifs
 - modèle pour une sémantique formelle
 - modèle le plus simple possible
 - outils pour le programmeur

Définition d'un trait

- Une entité :
 - sans état
 - contenant des définitions de méthodes fournies
 - déclarant des méthodes requises
- Les méthodes fournies :
 - n'ont pas accès à des variables globales
 - peuvent utiliser les méthodes requises

Exemple Java-like

```
trait TColor {  
    int getRed() { return getRGB().red(); }  
    int getHue() { return getRGB().hue(); }  
  
    // méthode requise  
    Color getRGB();  
}
```

Utilisation d'un trait

```
class Circle uses TColor {  
    // variable d'instance  
    Color color;  
  
    // satisfaction requête TColor  
    Color getRGB() { return this.color; }  
}
```

- N'importe quelle classe peut utiliser un trait si elle répond à ses requêtes

Propriétés de l'utilisation

- `getRed()` et `getHue()` se comportent comme si `Circle` les définissaient :
 - résolution des appels de méthode, `this`, `super`
- Utilisation = composition classe/trait :
 - Classe **uses** Trait → Classe
 - `uses` : somme asymétrique (non commutative) avec masquage de la classe sur le trait

Composition de traits

- Composition de plusieurs traits :
 - $\text{Trait} + \text{Trait} \rightarrow \text{Trait}$
 - somme symétrique (commutative)
 - mais génère des conflits entre méthodes
- Opérateurs de composition “gros grain”
 - Classe uses { Trait + Trait + ... }
 - limites liées à ce grain

Opérateurs à grain fin

- Deux pour Smalltalk : exclusion et aliasing
 - Trait – a → Trait (méthode a inaccessible)
 - Trait [b:=a] → Trait (b “pointe” sur la méthode a)
- But : modeler/adapter l'interface du trait à l'usage voulu, résoudre les conflits
 - par exemple, alias et exclusion permettent de fusionner deux méthodes en conflit

Combinaison des opérateurs

- Composition Trait/Trait
- Composition Classe/Trait
- Résolution conflit (fusion) par exclusion et alias

```
class Circle uses
{ (Tcolor - equals) [colorEquals:=equals] } +
{ (TLocation - equals) [locationEquals:=equals] } {
  boolean equals(Object anObject) {
    return colorEquals(anObject)
      && locationEquals(anObject); }
}
```

Conclusion

- Un modèle qui offre un grain intermédiaire entre la classe et la méthode :
 - unités composables, formellement définies
- Des opérateurs pour la composition et le remodelage :
 - unités malléables (interface) et paramétrables (méthodes requises)

Limites et évolutions

- Typage des traits :
 - quels problèmes ?
 - usage dans un RTTI ?
- Limite : conflit de noms entre traits
 - nouveau modèle/nouveaux opérateurs pour la résolution
 - évolution vers un système de module plus complexe : Jigsaw [Bracha 92] ?

Discussion : traits et aspects

- Les traits visent la réunification de structures dispersées dans l'arbre d'héritage
- Mais il ne s'agit pas d'exprimer une coupe (utilisation invasive et non quantifiée)
 - Une forme minimale d'aspects structurels ?
- Impact d'un passage à l'échelle ?

Discussion : traits et composants

- Transition vers un modèle à composant ?
 - dichotomie services requis / services fournis
 - passer d'une boîte blanche sans état à une boîte noire (possiblement avec état)
- les traits peuvent-ils servir de support pour la colle des interfaces (malléabilité et paramétrabilité) ?

BONUS

L'introduction avec AspectJ

- Mécanisme d'extension de classes (par des états, méthodes) – réflexion structurelle
- Complément de la programmation par aspects d'AspectJ
 - Un lien avec les traits ?

Faire un module trait

- Trait \simeq interface + aspect

```
interface Tcolor { // interface + méthodes requises
    Color getRGB();
}
```

```
aspect TColorRepository { // implémentations
    int TColor.getRed() { return getRGB().red(); }
    int TColor.getHue() { return getRGB().hue(); }
}
```

Utiliser le module

- implements = uses

```
class Circle implements TColor {  
    Color color;  
  
    Color getRGB(){ return color; }  
}
```

Composer les modules

```
class Circle implements Tcolor, TLocation {  
  
    // Conflit equals : comment fusionner ?  
  
}
```

- Solution biaisée : utiliser la délégation pour résoudre les conflits