

Introduction Non-Problème Utilisation Implémentation Conclusion

Approches Objet de la Programmation

~ Étude de Cas : les Méthodes Binaires ~

Didier Verna
EPITA / LRDE

didier@lrde.epita.fr

 lrde/~didier
 @didierverna
 didier.verna
 in/didierverna

Introduction Non-Problème Utilisation Implémentation Conclusion

Plan

Introduction

Un Non-Problème
Types, Classes, Héritage
Corollaire : Combinaisons de Méthodes

Validation du Concept – Niveau Utilisateur
Introspection
Une Méta-Classe de Fonctions Binaires

Validation du Concept – Niveau Programmeur
Implémentation Non-Contractuelle
Implémentation Manquante

Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 2/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Plan

Introduction

Un Non-Problème

Validation du Concept – Niveau Utilisateur

Validation du Concept – Niveau Programmeur

Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 3/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Introduction

- ▶ **Opération Binaire** : 2 arguments de même type
Opérations arithmétiques, relations d'ordre (=, +, > etc.)
- ▶ **Programmation OO** : 2 objets de même classe
Tirer parti du polymorphisme
- ▶ D'où le terme de « méthode binaire »
- ▶ **[Bruce et al., 1995]** :
 - ▶ Concept problématique dans l'approche objet traditionnelle
 - ▶ Relation classe / type dans un contexte d'héritage

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 4/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Plan

Introduction

Un Non-Problème

Types, Classes, Héritage
Corollaire : Combinaisons de Méthodes

Validation du Concept — Niveau Utilisateur

Validation du Concept — Niveau Programmeur

Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 5/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Contexte

La Hiérarchie Point

```

classDiagram
    class Point {
        x : int
        y : int
        +equal(p : Point) : Bool
    }
    class ColorPoint {
        color : String
        +equal(cp : ColorPoint) : Bool
    }
    Point <|-- ColorPoint
  
```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 6/34

Introduction Non-Problème Utilisation Implémentation Conclusion

C++ : Tentative #1

```

class Point
{
    int x, y;

    bool equal (Point& p)
    { return x == p.x && y == p.y; }
};

class ColorPoint : public Point
{
    std::string color;

    bool equal (ColorPoint& cp)
    { return color == cp.color && Point::equal (cp); }
};
  
```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 7/34

Introduction Non-Problème Utilisation Implémentation Conclusion

C++ : Tentative #1

Échec

```

int main (int argc, char *argv[])
{
    Point& p1 = * new ColorPoint (1, 2, "red");
    Point& p2 = * new ColorPoint (1, 2, "green");

    std::cout << p1.equal (p2) << std::endl;
    // => True. ### Wrong !
}
  
```

- ColorPoint::equal masque Point::equal (statiquement)
- Seule Point::equal est vue depuis la classe de base
- On a besoin de la version exacte

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 8/34

Introduction Non-Problème Utilisation Implémentation Conclusion

C++ : Tentative #2

```

class Point
{
    int x, y;

    virtual bool equal (Point& p)
    { return x == p.x && y == p.y; }
};

class ColorPoint : public Point
{
    std::string color;

    virtual bool equal (ColorPoint& cp)
    { return color == cp.color && Point::equal (cp); }
};

```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 9/34

Introduction Non-Problème Utilisation Implémentation Conclusion

C++ : Tentative #2

Un ColorPoint

Un Point

Échec

```

bool foo (Point& p1, Point& p2)
{
    return p1.equal (p2);
}

```

ColorPoint::equal attends un ColorPoint...

...mais n'obtient qu'un Point !

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 10/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Contexte

La Hiérarchie Point

Point
x : int
y : int
+equal (p : Point) : Bool

ColorPoint
color : String
+equal (cp : ColorPoint) : Bool

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 11/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Rappels : Covariance / Contravariance

▶ Règle Théorique :

```

virtual bool equal (Point& p);
virtual bool equal (ColorPoint& cp);

```

- ▶ *contravariance* sur les arguments
- ▶ *covariance* sur les valeurs de retour

▶ En Pratique :

- ▶ *invariance* sur les arguments
- ▶ Ambiguïtés dues à la surcharge

▶ Remarque :

- ▶ Eiffel autorise la covariance sur les arguments
- ▶ Erreurs potentielles à l'exécution

▶ Analyse : [Castagna, 1995]

- ▶ Manque d'expressivité
- ▶ *sous-typage (par sous-classage) ≠ spécialisation*
- ▶ Défaut du modèle object classique
- ▶ Absence des multi-méthodes

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 12/34

Introduction Non-Problème Utilisation Implémentation Conclusion

CLOS : Tentative #1

```

(defgeneric point= (a b))

(defclass point ()
  ((x :initarg :x :reader point-x)
   (y :initarg :y :reader point-y)))

(defmethod point= ((a point) (b point))
  (and (= (point-x a) (point-x b))
        (= (point-y a) (point-y b))))

(defclass color-point (point)
  ((color :initarg :color :reader point-color)))

(defmethod point= ((a color-point) (b color-point))
  (and (string= (point-color a) (point-color b))
        (call-next-method)))

```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 13/34

Introduction Non-Problème Utilisation Implémentation Conclusion

CLOS : Tentative #1

Succès

```

(let ((p1 (make-point :x 1 :y 2))
      (p2 (make-point :x 1 :y 2))
      (cp1 (make-color-point :x 1 :y 2 :color "red"))
      (cp2 (make-color-point :x 1 :y 2 :color "green")))
  (values (point= p1 p2)
          (point= cp1 cp2)))
;; => (T NIL)

```

- ▶ Sélection de la méthode appropriée sur les 2 arguments
Multiple dispatch
- ▶ Syntaxe d'appel de fonction plus satisfaisant
p1.equal(p2) or p2.equal(p1) ?
- ▶ « Fonction binaire »

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 14/34

Introduction Non-Problème Utilisation Implémentation Conclusion

Rappel : Combinaisons de Méthodes

- ▶ Éviter la duplication de code :
 - ▶ C++ : Point::equal()
 - ▶ CLOS : (call-next-method)
- ▶ Méthodes Applicables :
 - ▶ Toutes les méthodes compatibles avec les classes des arguments
 - ▶ Classées par spécificité décroissante
 - ▶ call-next-method exécute la méthode suivante dans cet ordre
- ▶ Combinaisons de méthodes :
 - ▶ Appeler plusieurs méthodes applicables
Pas seulement la plus spécifique
 - ▶ Combinaisons prédéfinies : and, or, progn etc.
 - ▶ Programmable

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 15/34

Introduction Non-Problème Utilisation Implémentation Conclusion

La Combinaison and

```

(defgeneric point= (a b)
  (:method-combination and))

(defmethod point= and ((a point) (b point))
  (and (= (point-x a) (point-x b))
        (= (point-y a) (point-y b))))

(defmethod point= and ((a color-point) (b color-point))
  (and (call-next-method)
        (string= (point-color a) (point-color b))))

```

- ▶ **Note** : le dispatch de CLOS est programmable

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 18/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Plan

Introduction

Un Non-Problème

Validation du Concept – Niveau Utilisateur

- Introspection
- Une Méta-Classe de Fonctions Binaires

Validation du Concept – Niveau Programmeur

Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 19/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Utilisation Non Contractuelle

Succès non mérité

```
(let ((p (make-point :x 1 :y 2))
      (cp (make-color-point :x 1 :y 2 :color "red")))
  (point= p cp))
;; => T #### Wrong !
```

- ▶ (point= <point> <point>) est applicable
Un color-point est un point
- ▶ Se prémunir contre une telle situation

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 20/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Introspection dans CLOS

```
(assert (eq (class-of a) (class-of b)))
```

- ▶ **Quand vérifier ?**
 - ▶ Chaque méthode : duplication
 - ▶ Dernière méthode : pas efficace / pas toujours possible
 - ▶ :before : pas disponible hors combinaison standard
 - ▶ Nouvelle combinaison : pas le lieu
 - ▶ :around : oui mais...
- ▶ **Où vérifier ?** (meilleure question)
 - ▶ Nulle part dans le code de point=
 - ▶ Lié au concept de fonction binaire, pas à point=
- ▶ Exprimer le concept de fonction binaire lui-même
 - ▶ *Les fonctions binaires sont des fonctions génériques particulières*

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 21/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Une Méta-Classe de Fonctions Binaires

- ▶ Fonctions génériques : standard-generic-function
- ▶ Objets-fonctions : funcallable-standard-class

```
(defclass binary-function (standard-generic-function)
  ()
  (:metaclass funcallable-standard-class))

(defmacro defbinary (function-name lambda-list &rest options)
  `(defgeneric ,function-name ,lambda-list
    (:generic-function-class binary-function)
    ,@options))
```

- ▶ **Remarque** : le concept impose une lambda-list particulière...

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 22/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Appels Génériques

- ▶ `compute-discriminating-function` : calcul, classement et exécution des méthodes applicables
- ▶ `compute-applicable-methods-using-classes` : calcul et classement des méthodes applicables
- ▶ `compute-effective-method` : Application des méthodes (combinées) aux arguments
- ▶ **Note** : fonctions génériques *spécialisables*

Protocole

```

graph TD
    A[generic call] --> B[comp-dscr-func]
    B --> C[comp-app-meth-usg-cls]
    C --> D[comp-eff-meth]
  
```

Couche Utilisateur
 Couche 3

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 23/34

Introduction Non-Problème **Utilisation** Implémentation Conclusion

Application aux Fonctions Binaires

```

(defmethod compute-applicable-methods-using-classes
  :before ((binary-function binary-function) classes)
  (assert (= (length classes) 2))
  (assert (apply #'eq classes)))
  
```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 24/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Plan

Introduction

On Non-Problème

Validation du Concept – Niveau Utilisateur

Validation du Concept – Niveau Programmeur

- Implémentation Non-Contractuelle
- Implémentation Manquante

Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 25/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Implémentation Non-Contractuelle

```

;; #### WRONG SPEC!!
(defmethod point= ((p1 point-3d) (p2 point-2d))
  #/ ... /#)
  
```

- ▶ Précédemment : protection contre les appels illégaux
- ▶ Question : protection contre les implémentations illégales ?

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 26/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Définition de Méthodes

- ▶ `ensure-method` : création et ajout de la nouvelle méthode
- ▶ `make-method` : création du méta-objet
- ▶ `add-method` : mise à jour de la fonction générique
- ▶ **Note** : fonctions génériques *spécialisables*

Protocole

```

graph TD
    A[defmethod] --> B[ensure-method]
    B --> C[make-method]
    C --> D[add-method]
  
```

Couche 1
 Couche 2
 Couche 3

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 27/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Application aux Fonctions Binaires

```

(defmethod add-method :before ((bf binary-function) method)
  (let ((method-specializers (method-specializers method)))
    (assert (= (length method-specializers) 2))
    (assert (apply #'eq method-specializers))))
  
```

- ▶ `method` : méta-objet (Cf. `standard-method`)
- ▶ `method-specializers` : accesseur

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 28/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Implémentation Manquante

- ▶ Chaque sous-classe de `point` doit spécialiser `point=`
- ▶ Vérification tardive : lors d'un appel générique *préserver la haute dynamicité*
- ▶ « Binaire-complétude » : il existe une méthode primaire applicable pour chaque classe de la hiérarchie

```

(defmethod compute-applicable-methods-using-classes
  :around ((bf binary-function) classes)
  (multiple-value-bind (methods ok) (call-next-method)
    ;; 1. Check for binary completeness on METHODS
    ;; 2. If everything goes well, simply return
    ;; what we got from CALL-NEXT-METHOD
    (values methods ok)))
  
```

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 29/34

Introduction Non-Problème Utilisation **Implémentation** Conclusion

Mise en oeuvre

```

(loop :for class :in (class-precedence-list (car classes))
  :until (eq class (find-class 'standard-object))
  :do (assert
    (find-if
      (lambda (method)
        (let ((qualifiers (method-qualifiers method))
              (and (equal (method-specializers method)
                          (list class class))
                    (not (member :around qualifiers))
                    (not (member :before qualifiers))
                    (not (member :after qualifiers))))))
      methods)))
  
```

- ▶ `class-precedence-list` : accesseur
- ▶ `method-qualifiers` : accesseur

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 30/34

Introduction Non-Problème Utilisation Implémentation **Conclusion**

Plan

- Introduction
- Un Non-Problème
- Validation du Concept — Niveau Utilisateur
- Validation du Concept — Niveau Programmeur
- Conclusion

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 31/34

Introduction Non-Problème Utilisation Implémentation **Conclusion**

Conclusion

- ▶ **Méthodes binaires** : un concept très simple
 - ▶ Difficile à implémenter dans un système OO classique
 - ▶ Trivial avec des multi-méthodes
- ▶ **Grâce à CLOS** :
 - ▶ Validation de l'utilisation
 - ▶ Validation de l'implémentation
- ▶ **Grâce au MOP** :
 - ▶ Concept explicite
 - ▶ Extension du système OO de base

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 32/34

Plan

Bibliographie

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 33/34

Bibliographie

-  Bruce, K. B., Cardelli, L., Castagna, G., Eifrig, J., Smith, S. F., Trifonov, V., Leavens, G. T., and Pierce, B. C. (1995)
 On Binary Methods
Theory and Practice of Object Systems, 1(3):221–242
-  Castagna, G. (1995)
 Covariance and Contravariance : Conflict Without a Cause
ACM Transactions on Programming Languages and Systems, 17(3):431–447
-  Verna, D. (2008)
 Binary Methods Programming : the CLOS Perspective
Journal of Universal Computer Science, Vol. 14.20

AOP / Étude de Cas : les Méthodes Binaires – Didier Verna 34/34