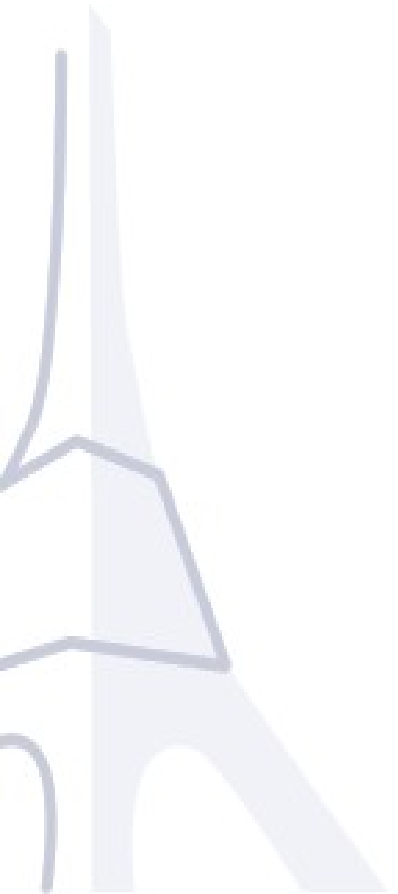
Programmation Orientée Aspects en PHP

William Candillon {wcandillon@elv.enic.fr}
Gilles Vanwormhoudt {vanwormhoudt@enic.fr}

Forum PHP 2005

Plan

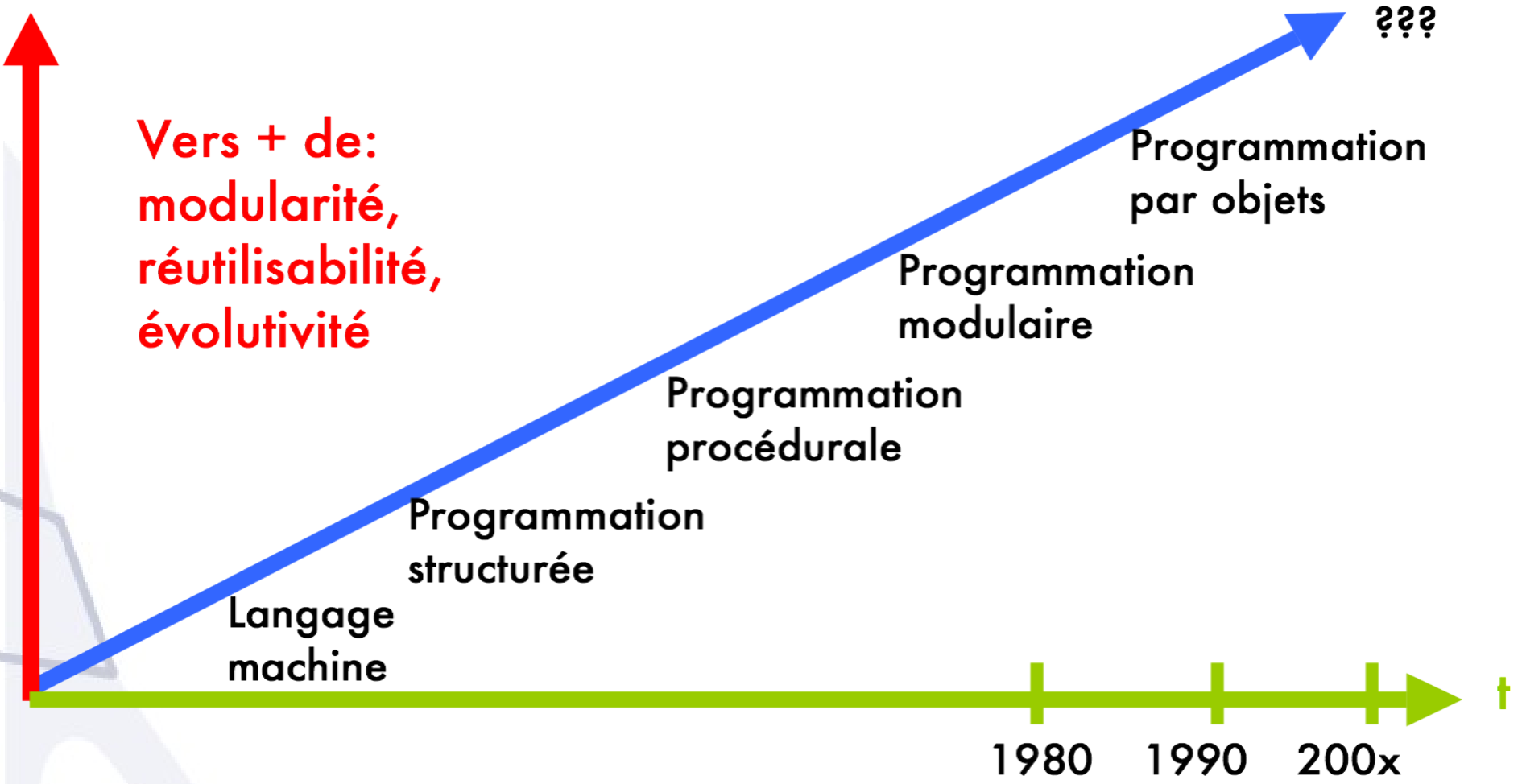
- Introduction à la POA
- PHPAspect
- Applications



Introduction à la POA

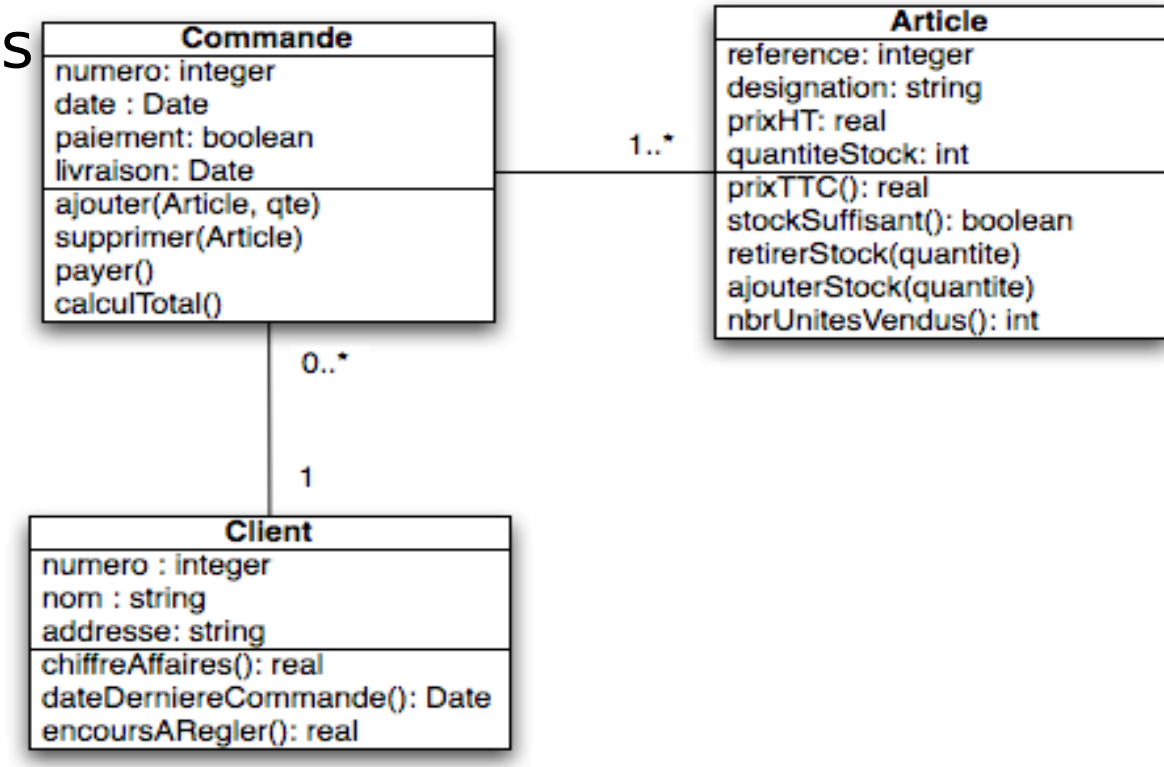
Forum PHP 2005

Evolution de la programmation



Programmation Orientée Objets

- Bien adapté pour
  - Représentation des parties métiers
  - Composants techniques réutilisables
- Mais aussi
  - Design pattern
  - Frameworks

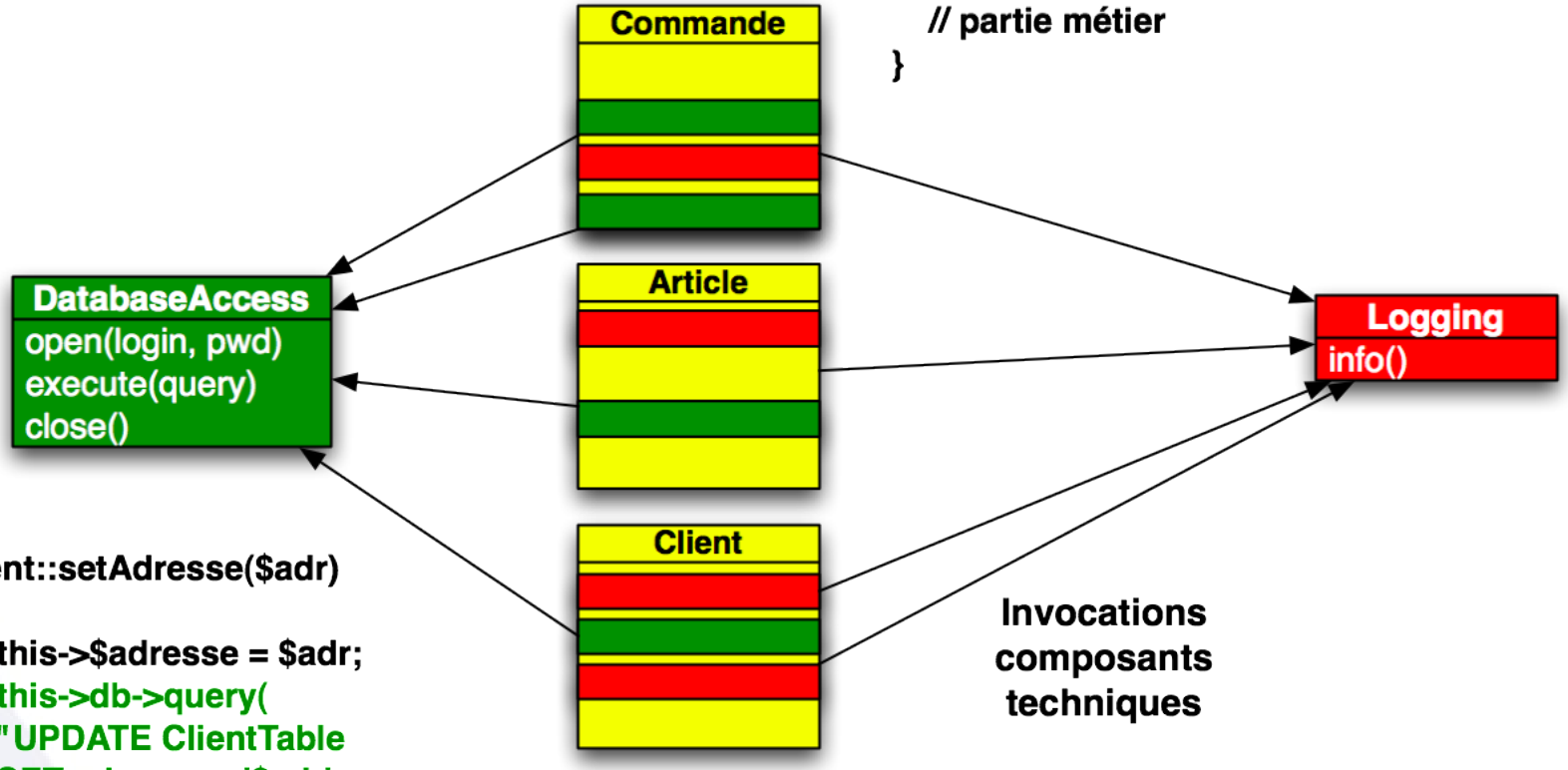


Quid des parties techniques ?

- Application = partie métier + partie technique
- Exemples d’aspect technique
  - Sécurité
  - Persistence
  - Authentification
  - Gestion transactionnelle
  - Performance
  - Journalisation
  - IHM
  - Gestion des erreurs, …

Aspects techniques avec la POO

```
Commande::payer()
{
   $this->log->info("Paiement commande:".num);
   // partie métier
}
```

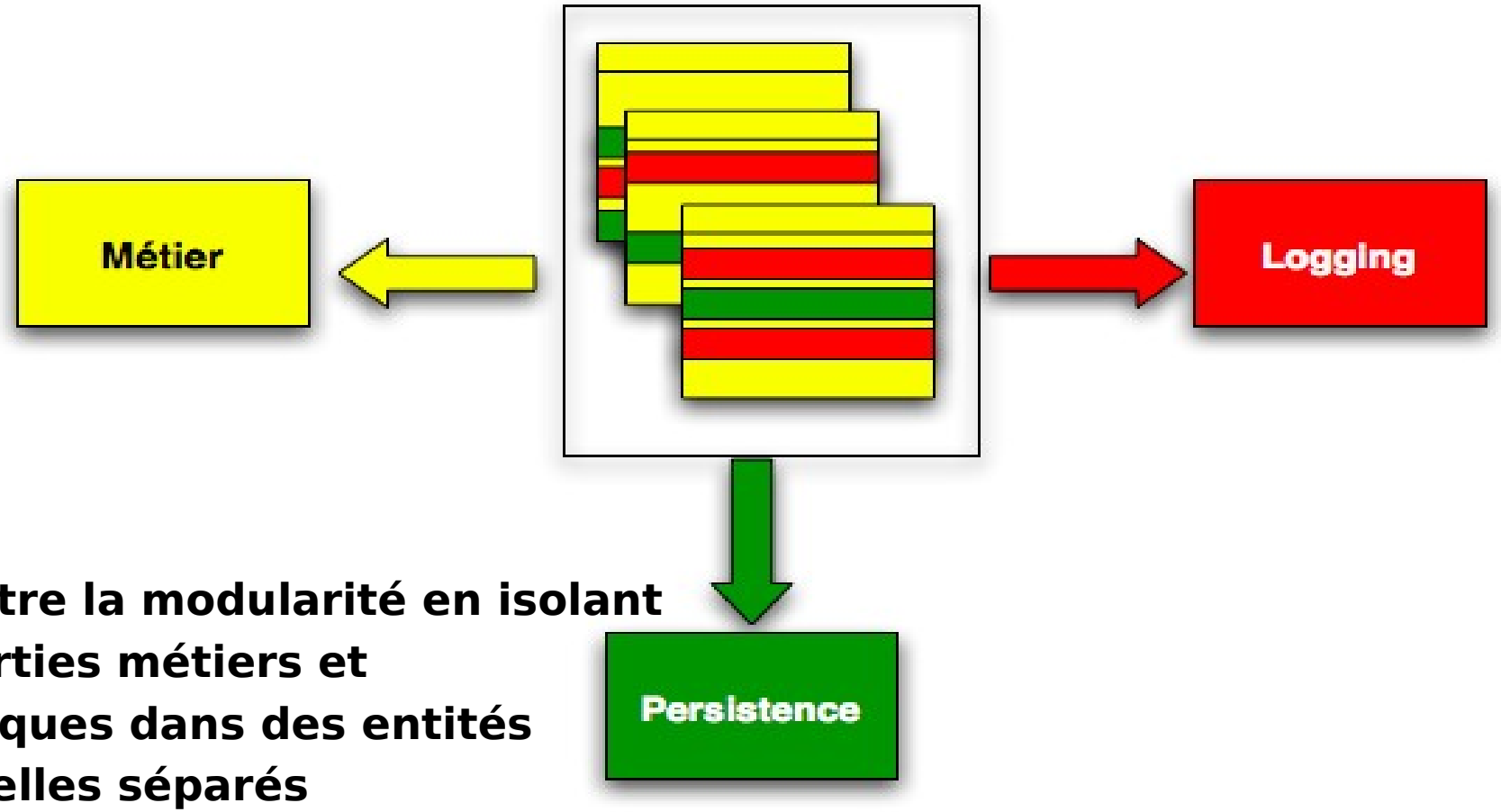



```
Client::setAdresse($adr)
{
   $this->$adresse = $adr;
   $this->db->query(
     "UPDATE ClientTable
     SET adresse = '$adr' .
     WHERE  id ={$this->id}"
   );
}
```

Les symptômes

- Entrelacement des aspects techniques avec le code métier (code tangling)
- Eparpillement des aspects à travers la structure d'objets (code scattering)
- Redondance du code : Héritage ?
- Difficultés
  - Pour la compréhension
  - Pour la réutilisation de la partie métier
  - Pour l'évolution des parties métiers et techniques

Vers plus de modularité



Accroitre la modularité en isolant les parties métiers et techniques dans des entités Logicielles séparés

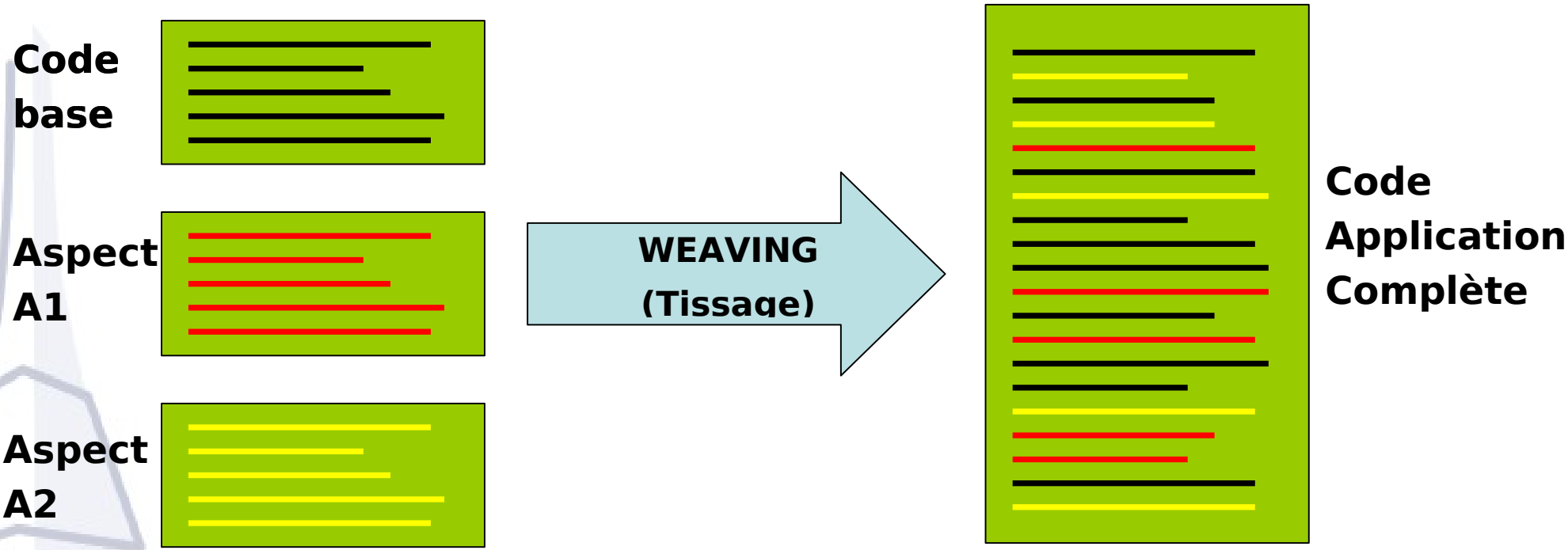
Programmation Orientée Aspects

- Une nouvelle technique de programmation
  - Complémentaire de la POO
  - Permettant de séparer la description des aspects métiers et techniques
- Des constructions et mécanismes pour
  - Capturer les points transversaux
  - Décrire le code des aspects dans un nouveau type d'entité logicielle
  - Fusionner le code des aspects techniques et le code métier selon les points transversaux

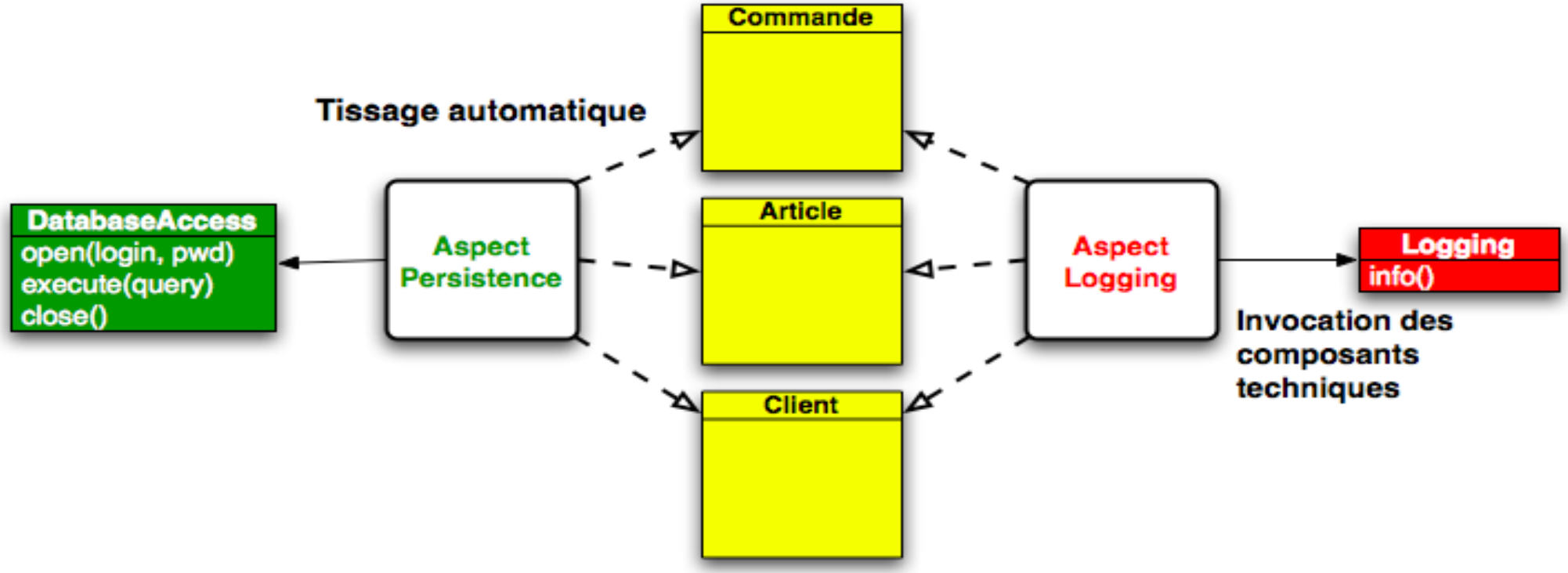
Quelques généralités

- Inventé par G.Kiczales et son équipe au Xerox Parc en 97
- Pointée comme technologie du futur par le MIT
- Champ de recherche actif
- Intégration progressive dans les communautés J2EE/.NET
- Supporter par de grands noms de l'informatique IBM (Eclipse, HyperJ), Microsoft
- De nombreux extensions de langages et de plateformes : AspectJ, JBossAOP, ...

Idée générale



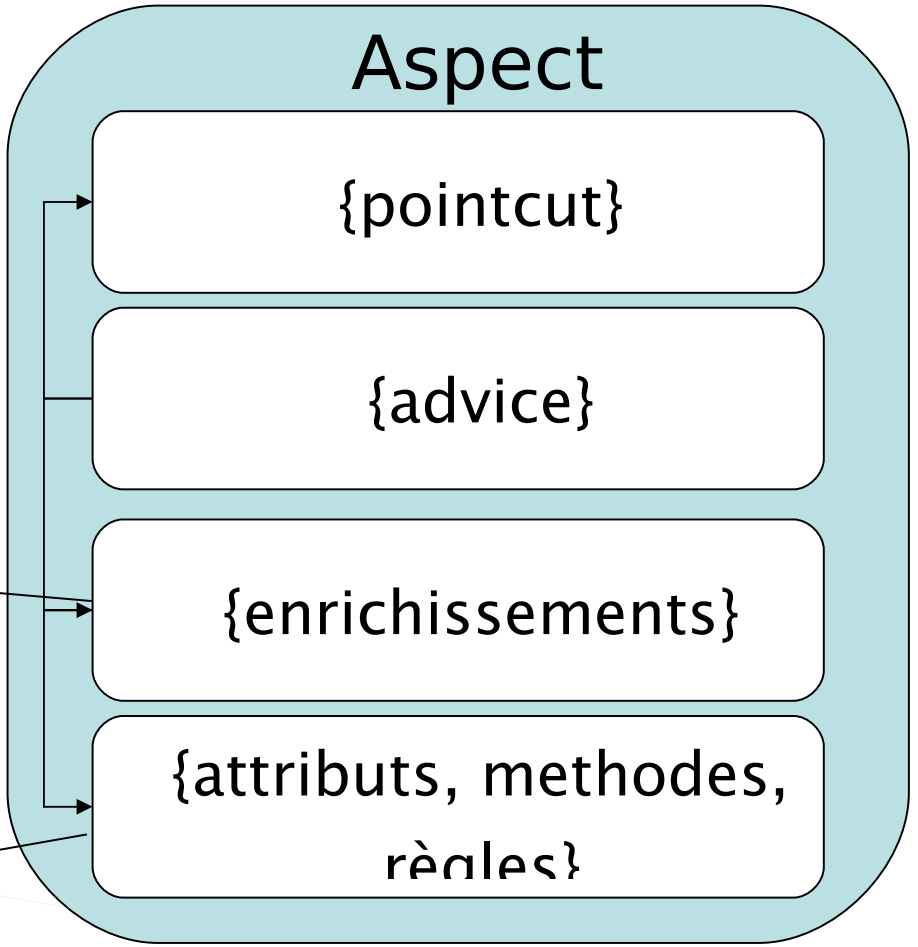
Application à l'exemple



Inversion des dépendances entre parties métiers et techniques

Concepts de l'AOP : Aspect

- Unité logicielle enrichissant le code de base avec des éléments techniques

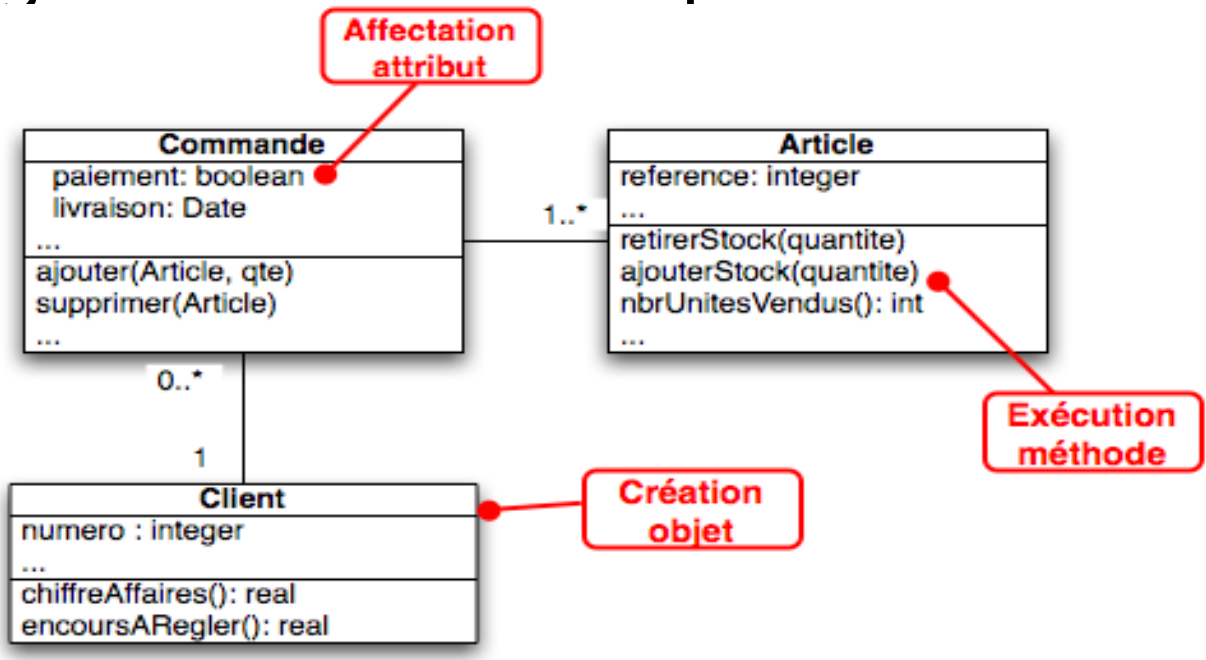


Ajouts structurels au programme base

Caractéristiques de l'aspect

Concepts de l'AOP : Point de jonction (joinpoint)

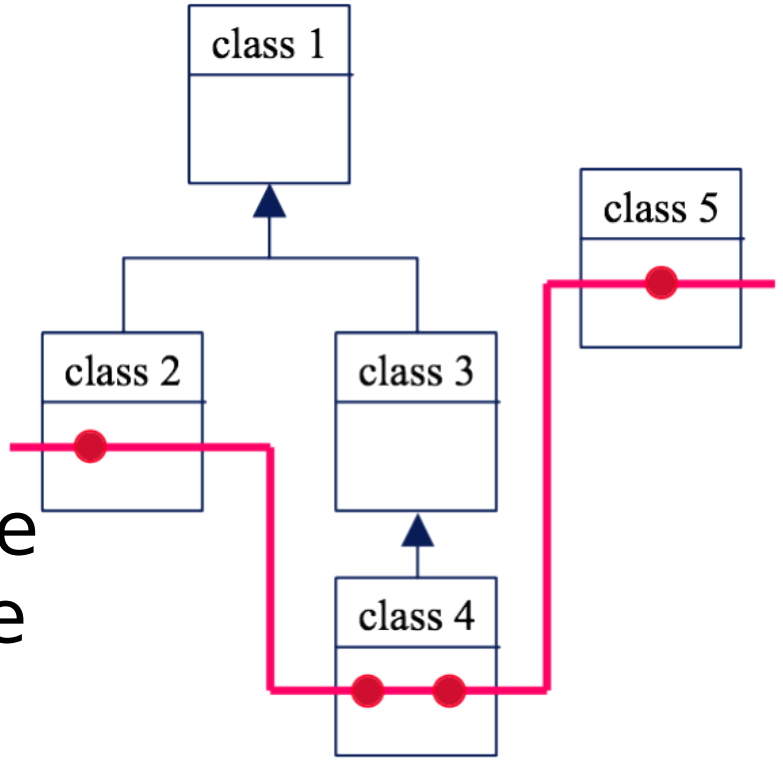
- Point dans la structure ou le flot d'exécution du programme de base qui va être enrichi
- Exemple
- Envoi message
- Execution méthode
- Affectation variable
- Création objet
- ...



- Modèle + ou - riche selon les langages

Concepts de l'AOP : Point de coupe (pointcut)

- Ensemble de points de jonction ayant un lien logique entre eux
- Permet d'identifier les endroits du programme ou se fera l'insertion de code
- Jeu d'opérateurs + ou - riche selon le langage
- filtrage, quantification, combinateur logique, typage ...



Concepts de l'AOP : Greffon (advice)

- Morceaux de code injectés au niveau des points de jonction utilisés dans un point de coupe
- Types de greffon
  - Avant (before)
  - Apres (after)
  - Autour (around)

```
$myOrder = new Order;
<- greffon avant ->
$myOrder->addItem('Largo Winch', 2);
<- greffon apres ->
$myOrder->catalog = new Catalog;
```

{ greffon around}

- Capacité d'exploitation du contexte des points de jonction
  - Arguments, objet receveur,...

Utilisation des aspects

- Développement
  - Trace (debug), Couverture, Profilage
  - Programmation par contrats (assertions), Test
- Production
  - Mise en œuvre d'aspects techniques
    - Sécurité, Persistence, Journalisation, Concurrence, ...
  - Application de patrons de conception
    - Observer, Singleton, Visitor, ...

Conclusion sur la POA

- Technique de programmation prometteuse
- Prise de conscience du besoin de séparation des préoccupations
- Focus actuel
  - Aspects réutilisables
  - Composition d'aspects
  - Elargissement à toutes les étapes du cycle de vie du logiciel
  - Langages spécialisés (Domain Specific Languages)

PHPAspect

Forum PHP 2005

Aspect Oriented PHP

- Approche minimaliste de la POA.
- Fonctionne avec le module de réécriture d’url apache.
- Analyse du code PHP via des expressions régulières (très peu fiable).
- Absence de la notion d’enrichissement.
- Interception d’appels de fonctions uniquement.

aspectPHP

- Extension du langage PHP. Prototype de recherche du docteur Yijun Yu .
- Compilateur PHP 4 modifié : tissage à la génération de l’opcode.
- Analyse syntaxique Lex et Yacc (très fiable).
- Absence de la notion d’enrichissement.
- Interception d’appels de fonctions uniquement.

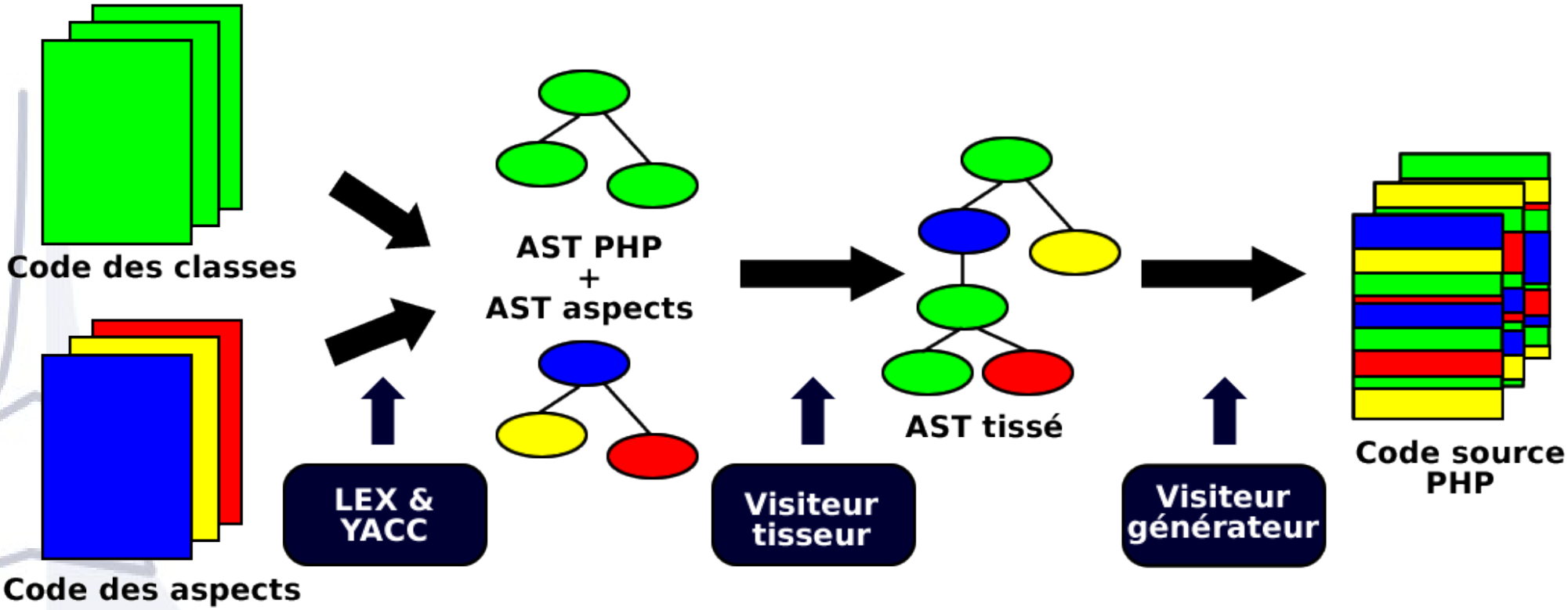
PHPaspect - motivations

- Solutions existantes trop limités et peu fiables.
- Proposer une nouvelle solution pour faire de la POA en PHP :
  - POA riche.
  - Intégration forte avec le langage.
  - Tissage basé sur :
    - Une analyse syntaxique (Lex & Yacc).
    - Une analyse statique (AST) : performance, indépendance de la plate forme.

PHPaspect - caractèristiques

- Extension du langage PHP 5, dédiée à l'application d'aspects sur les objets.
- Technique de compilation PHP.
- Notion d'enrichissement de classes: attributs et méthodes.
- 5 types de point de jonctions : appel de méthode, exécution de méthode, construction d'objet, écriture et lecture d'attribut.
- Autres notions : jokers et opérateurs, réification des points de jonction, ordonnancement des aspects.

PHPaspect – chaîne de tissage



Un panier virtuel

```php
<?php
class Order{
   public $catalog;
   private $items = array();
   private $amount = 0;

   public function addItem($reference, $quantity){
      $this->items[] = array($reference, $quantity);
      $this->amount +=
        $quantity*$this->catalog->getPrice($reference);
   }

   public function getAmount(){
      return $this->amount;
   }
}

class Catalog{
   private static $priceList = array('Largo Winch'=>9.31,
                          'Astérix'=>8.46, 'XIII'=>8.70);
   public static function getPrice($reference){
      return self::$priceList[$reference];
   }
}

$myOrder = new Order;
$myOrder->catalog = new Catalog;
$myOrder->addItem('Largo Winch', 2);
$myOrder->addItem('Astérix', 1);
?>
```

- Un client ajoute les articles d'un catalogue dans le panier.
- Partie métier de l'application sans aspects techniques.

Un aspect de trace

```php
<?php
aspect TraceOrder{

   pointcut logAddItem:exec(public Order::addItem(2));
   pointcut logTotalAmount:call(Order->addItem(2));

   after logAddItem{
      printf("%d %s Ajouté à la commande\n",
               $quantity, $reference);
   }

   after logTotalAmount{
      printf("Montant total de la commande : %.2f €\n",
         $thisJoinPoint->getObject()->getAmount());
   }
}
?>
```

- Trace les ajouts d'articles et résume la commande.
- Tissage des appels

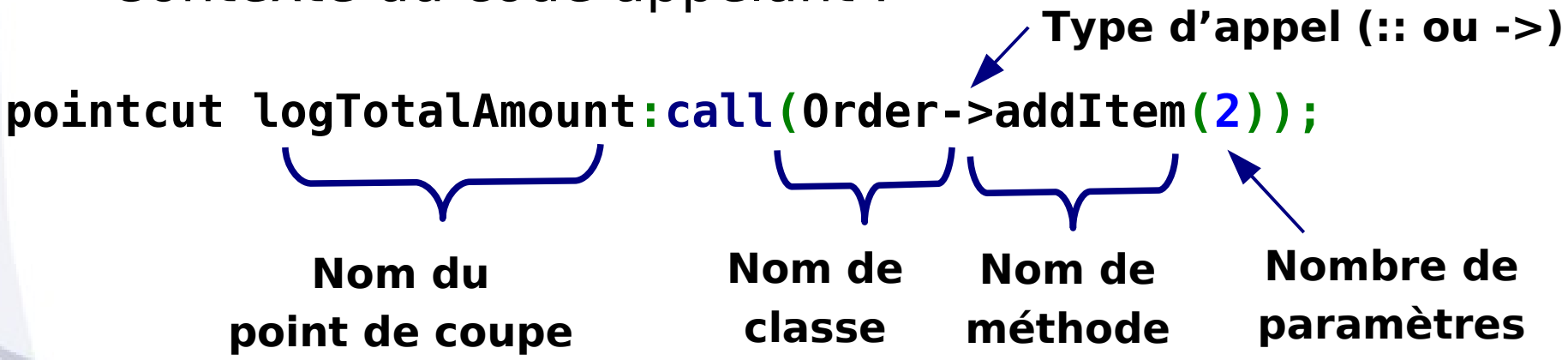
```php
$myOrder->catalog = new Catalog;
$myOrder->addItem('Largo Winch', 2);
printf("Montant total de la commande : %.2f €\n",
         $myOrder->getAmount());
$myOrder->addItem('Astérix', 1);
printf("Montant total de la commande : %.2f €\n",
         $myOrder->getAmount());
```

- Résultat :

```
2 Largo Winch Ajouté à la commande
Montant total de la commande : 18.62 €
1 Astérix Ajouté à la commande
Montant total de la commande : 27.08 €
```

Points de jonction pour les méthodes (call/exec)

- Sélection des appels de méthodes.
  Contexte du code appelant :



- Sélection des exécutions de méthodes.
  Contexte du code appelé :

```
pointcut logAddItem:exec(public Order::addItem(2));
```

Construction (new)

- Sélection des constructions d'objets.
  - Sélection des constructions de la classe *Order* prenant 1 paramètre.

```
pointcut TraceConstruct:new(Order(1));
```

  - Selection d'une construction quelconque :

```
pointcut TraceAnyConstruct:new(*(*));
```

Ecriture/lecture d'attributs (get/set)

- Ecriture d'un attribut.
  - Ecriture statique de l'attribut *$priceList* dans la classe *Catalog* :

```
pointcut TraceSet:set(Catalog::$priceList);
```

  - Ecriture dynamique de l'attribut *catalog* dans la classe *Order* :

```
pointcut TraceSet:set(Order->$catalog);
```

- Lecture d'un attribut.

```
pointcut TraceSet:get(Catalog::$priceList);
pointcut TraceSet:get(Order->$catalog);
```

Les greffons (code-advice)

- **before** et **after** : portions de code greffées avant et après l'exécution du point de jonction.

```
before Foo{
    echo 'Avant Foo';
}
```

```
after Foo{
    echo 'Après Foo';
}
```

- **around** : portion de code greffée autour du point de jonction. L'appel à la fonction ***proceed()*** correspond à l'exécution du point de jonction.

```
Around Foo{
    if(count($thisJoinPoint->getArgs()) >= 2){
        proceed();
    }else{
        echo "Nombre de paramètres insuffisant
              pour l'exécution du joinpoint";
    }
}
```

Enrichissement de classes

- Introduction d'attributs ou de constantes.

```
private Bo*::$pearLog,
             Bo*::$debug=false;
```

Introduction de l'attribut *$pearLog* et *$debug* de visibilités privée dans toutes les classes préfixées par Bo.

```
Log::PEAR_LOG_DEBUG=7,
Log::PEAR_LOG_ERR=3;
```

Introduction de la constante *PEAR_LOG_DEBUG* et *PEAR_LOG_ERR* dans la classe Log.

- Introduction de méthodes.

```
public function Bo*::setLog(Log $log){
   if($this->debug){
      $log->setMask(LOG::PEAR_LOG_DEBUG);
   }else{
      $log->setMark(LOG::PEAR_LOG_ERROR);
   }
   $this->pearLog = $log;
}
```

Introduction de la méthode *setLog()* dans toutes les classes préfixées par Bo.

Joker et opérateurs

- Le joker * peut s'appliquer sur la visibilité d'une méthode, le nom d'une classe ou d'une méthode, le nombre de paramètres d'une méthode.

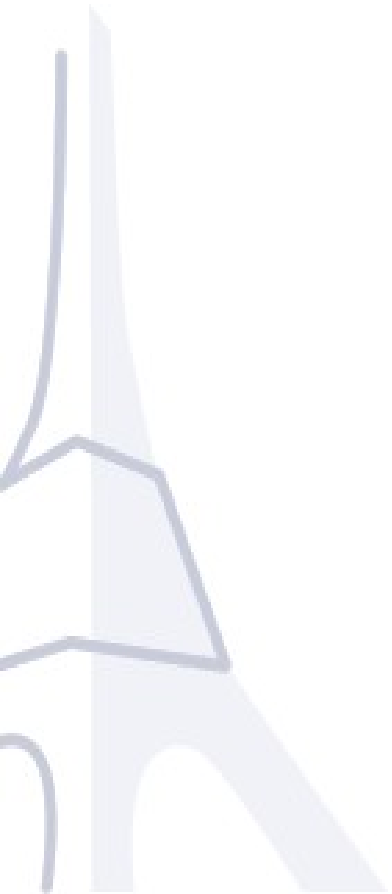
```
exec(* Bo*::set*(*))
call(*::*(*))
```

- L'opérateur + permet de sélectionner tous les sous-types d'une classe (sous-classes et interfaces).

```
call(DB_DataObject+->update(0))
```

- L'opérateur || permet d'appliquer un greffon sur plusieurs points de jonction.

```
before DynamicSet || StaticSet{
    echo "Avant l'appel à un muttateur";
}
```



Applications

Un aspect de persistance

```
aspect Persistance{

   private Bo*::$db;
   private Bo*::$id=false;

   private function Bo*::setStates(){
      if($this->id !== false){
         $sql = sprintf('SELECT * FROM %s WHERE id = %d',
                  get_class($this), $this->id);
         $states = $this->db->getRow($sql);
         foreach($states as $stateName = $state){
            $this->$stateName = $state;
         }
      }
   }

   pointcut SetState:exec(public Bo*::__construct(*));
   pointcut StateChange1:exec(public Bo*::set*());
   pointcut StateChange2:exec(public Bo*::add*());

   before:SetState{
      $this->db = $db;
      $this->id = $id;
      $this->setStates();
   }
   after:StateChange1 || StateChange2{  $this->update(); }

    private function Bo*::update() {
    //Construction de la variable update
      if($this->id === false){
         //INSERT INTO...
         $this->db->query($sql);
      }else{
         $sql = sprintf('UPDATE %s SET  WHERE id = %d',
                  get_class($this), $update, $this->id);
         $this->db->query($sql);
      }
   }
}
```

- La gestion de la persistance des objets est ici hautement modulaire : tout est factorisé dans un aspect.
- L'externalisation de la persistance des objets dans un aspect accroît fortement la réutilisabilité des classes.

Un aspect de gestion des erreurs

```
aspect ErrorHandler{

   pointcut ConnectError:exec(DB_*::connect(*));
   pointcut HandleConnect:call(DB_*::connect(*));

   after HandleDBConnect{
      if(!$this->connection){
         throw(new ConnectException(
                  $this->dbsyntax,
                  $this->getError(),
                  $this->getCodeError()));
      }
   }

   around HandleConnect{
      try{
         proceed();
      }catch(ConnectException $e){
         printf("Impossible de se connecter à %s : fichier %s, ligne %d :\n\t%s",
                     $e->driver(),
                  $e->getFile(),
                  $e->getLine(),
                  $e->getMessage());
         exit;
      }
   }
}
```

- Facilite le respect des standards (PEAR).
- En cas d'évolution complexe, la localisation des exceptions est simplifié.

Design patterns et aspects

- « *Les Patterns offrent la possibilité de capitaliser un savoir précieux né du savoir-faire d'experts* » Buschmann, 1996.
- Solutions orientées objets génériques pour résoudre des problèmes communs.
- Dans certains cas, l'implantation par aspect d'un patron apporte les bénéfices suivants :
  - Le patron est factorisé au sein d'un aspect : gain de modularité.
  - Le couplage entre une classe et les design patterns qu'elle implante devient nul : gain de réutilisabilité.

L’observeur

```
class Client implements Subject{
    private $observers = array();
    private $state = true;

    public function getState(){
        return $this->state;
    }

    public function setState($state){
        $this->state = $state;
        $this->notifyObservers();
    }

    public function addObserver(Observer $o){
        $this->observers[] = $o;
    }

    public function notifyObservers(){
        foreach($this->observers as $o){
            $o->sendNotify($this);
        }
    }
}
```

- « *Définit une interdépendance de type un à plusieurs, de façon telle que, quand un objet change d’état, tous ceux qui en dépendent en soient notifiés et automatiquement mis à jour.* »
- Exemple : méthode *connect()* en PHP-GTK.

L’aspect observeur

```
aspect ObserverClient {
   private Client::$observers = array();

   public function Client::addObserver(Observer $o){
      $this->observers[] = $o;
   }

   public function Client::notifyObservers(){
      foreach($this->observers as $o){
         $o->sendNotify($this);
      }
   }

   pointcut ChangeStage:exec(Client::set*(*));

   after ChangeStage{
      $this->notifyObservers();
   }
}
```

- Le sujet est indépendant de ses observeurs.
- Le mécanisme générique de gestion des observeurs et la détection des changements d’état du sujet est factorisé dans un aspect.

Conclusion

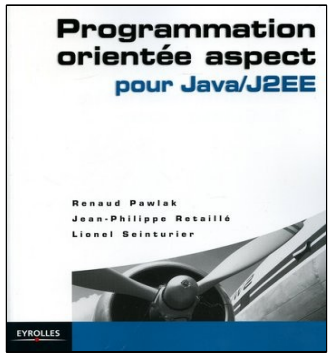
- PHPaspect implante tout les concepts majeurs de la POA.
- Simple à utiliser, PHPaspect apporte un intérêt immédiat dans les développements PHP.
- Le wiki : http://phpaspect.org
- Perspectives :
  - Extension de PHPAspect
    - Inférence de types, Composition d'aspects, plus d'opérateurs, Reification des aspects.
  - Conception par Aspects
    - Intégration frameworks, DSL -> PHPAspect.
- Actuellement à la recherche d'un stage, mon cv : http://wcandillon.netcv.org

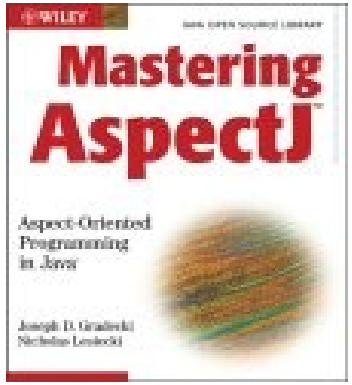
Pour aller plus loin

- Bibliographie

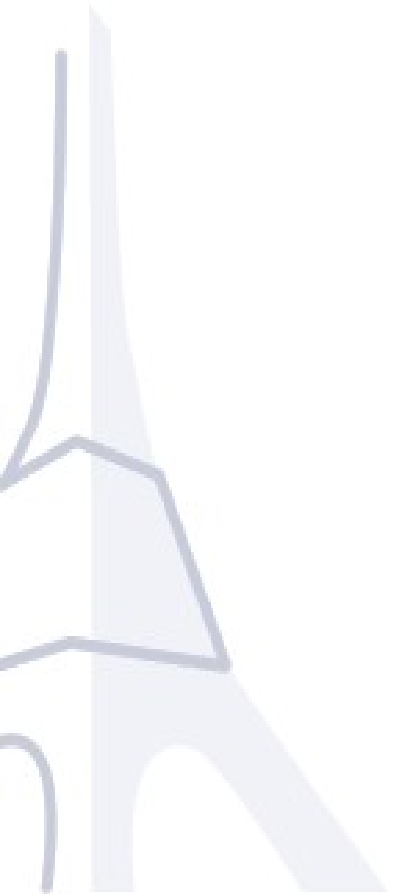
  Programmation Orientée Aspect pour Java/J2EE.
  Eyrolles
  Renaud Pawlak, Jean-Philippe Retaillé, Lionel Seinturier.

  Mastering AspectJ, John & Wiley
  Joseph D. Gradecki, Nicholas Lesiecki





- Sur le Web
  - portail http://aosd.net



Des questions ?

Exemple en AspectJ