samedi 3 octobre 2009

afpy

Association Francophone Python

Christophe Combelles

Programmation par composants en Python

Alter Way

- polyvalence
  - web
  - calcul scientifique
  - interfaces graphiques
  - daemons / services
  - admin sys
  - pilotage d'applications
- Interopérabilité
  - ctypes
  - Java
  - .NET
- forte demande

Créons une application

```
class MonPetitCanard(object):
    def dis_coincoin(self):
        print 'coin'
```

```
class MonPetitCanard(object):

    def dis_coincoin(self):
        print 'coin'

    def mange(self):
        print 'miam'

    def se_reproduire(self, canard):
        print 'mmh yaaa mmh yaaa'
        return copy.deepcopy(self)

    (...)
```

```
class Animal(object):
    def mange(self):
        print 'miam'

    def se_reproduire(self, animal):
        print 'mmh yaaa mmh yaaa'
        return copy.deepcopy(self)
```

```
class MonPetitCanard(Animal):

    def dis_coincoin(self):
        print 'coin'
```

```
class Animal(object):
    def mange(self):
        print 'miam'

    def se_reproduire(self, animal):
        print 'mmh yaaa mmh yaaa'
        return copy.deepcopy(self)

    def est_lourd(self):
        return "j'en sais rien"
```

```
class MonPetitCanard(Animal):

    def dis_coincoin(self):
        print 'coin'

    def est_lourd(self):
        print 'pas trop lourd'
```

```
class Poids(object):

    def est_lourd(self):
        print 'pas trop lourd'
```

```
class Animal(object):
    def mange(self):
        print 'miam'

    def se_reproduire(self, animal):
        print 'mmh yaaa mmh yaaa'
        return copy.deepcopy(self)
```

```
class MonPetitCanard(Animal, Poids):

    def dis_coincoin(self):
        print 'coin'
```

```
class Poids(object):

    def est_lourd(self):
        print 'pas trop lourd'
```

```
class Masse(object):

    def est_lourd(self):
        print 'pas trop lourd'
```

```
class Animal(object):
    def mange(self):
        print 'miam'

    def se_reproduire(self, animal):
        print 'mmh yaaa mmh yaaa'
        return copy.deepcopy(self)
```

```
class MonPetitCanard(Animal, Poids, Masse):

    def dis_coincoin(self):
        print 'coin'
```

Héritage multiple / mixins

conflit de méthodes

ordre de résolution ?

mauvaise isolation pour les tests

obligation de retoucher le code

difficulté pour connaître les fonctionnalités

difficulté pour ajouter/retirer des fonctionnalités

difficulté pour changer d'implémentation

Réutilisabilité douteuse

Canard qui grossit

découpage en fonctionnalités

=> création de **composants**

- **testés**
- **documentés**
- **réutilisables**
- **pluggables**

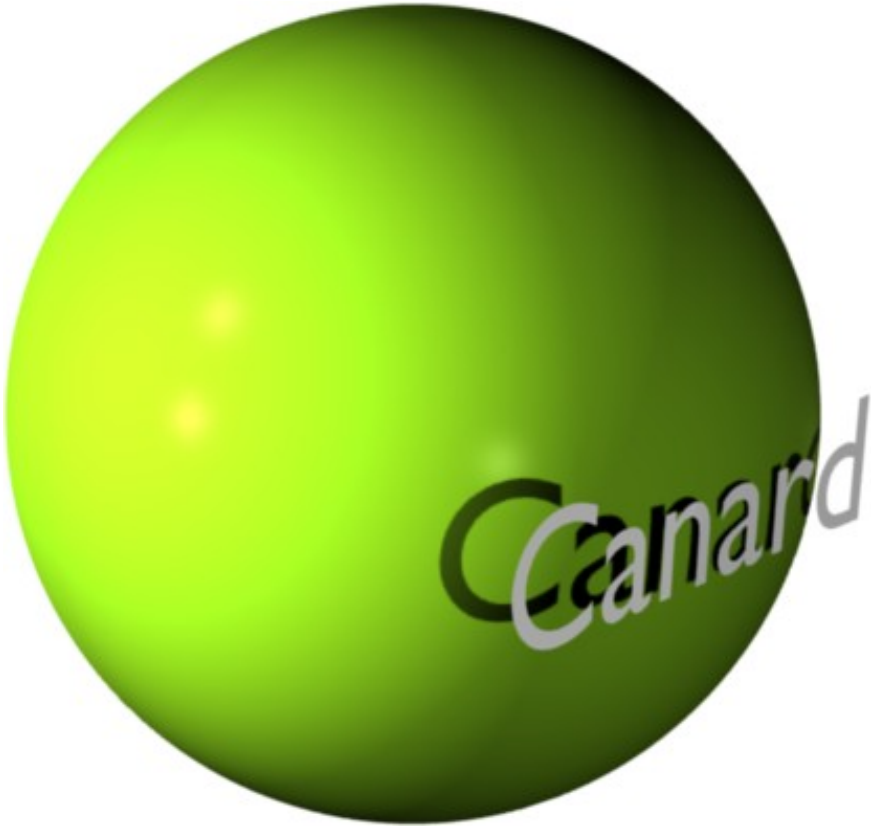
Comment créer des composants ?

Architecture de composants

- interfaces
  - fonctionnalités
  - documentation
- composants
  - implémentent les interfaces
- registres
  - branchements

Objet

```
class Canard(object):
    def fais_coincoin(self):
        print 'coin'
```

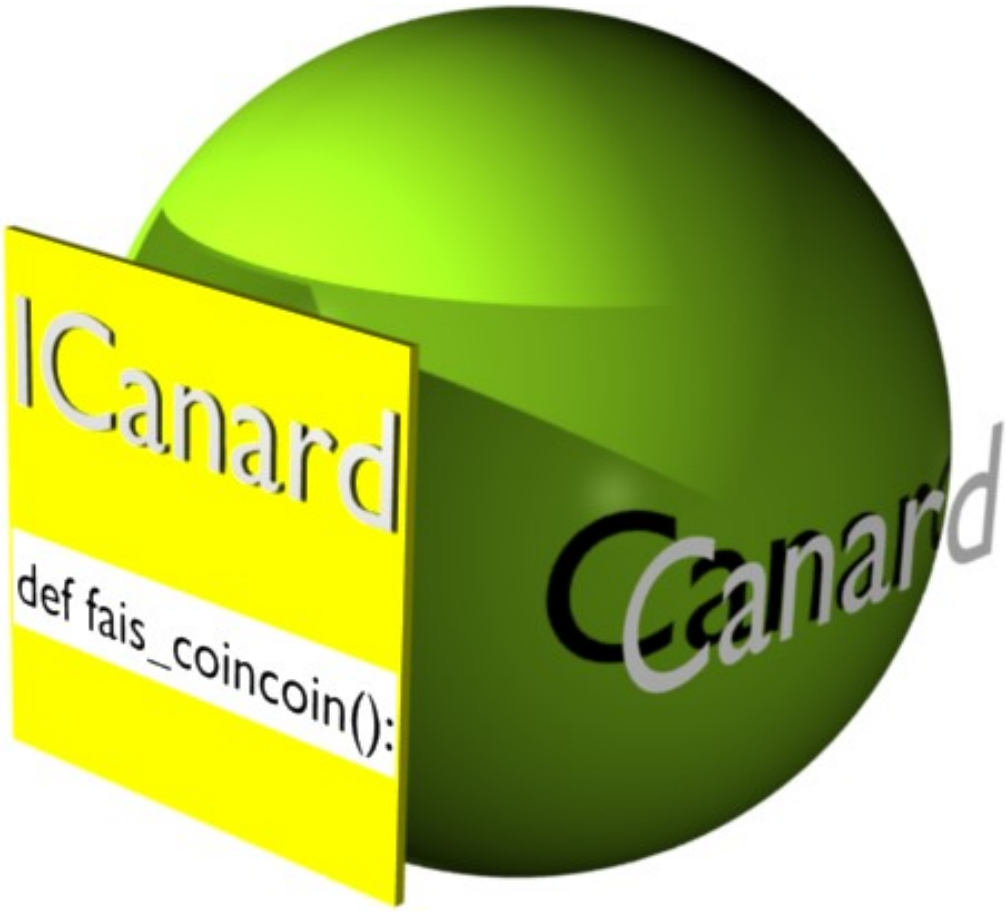


Interface

```
class ICanard(Interface):
    def fais_coincoin():
        "le bruit du canard"
```

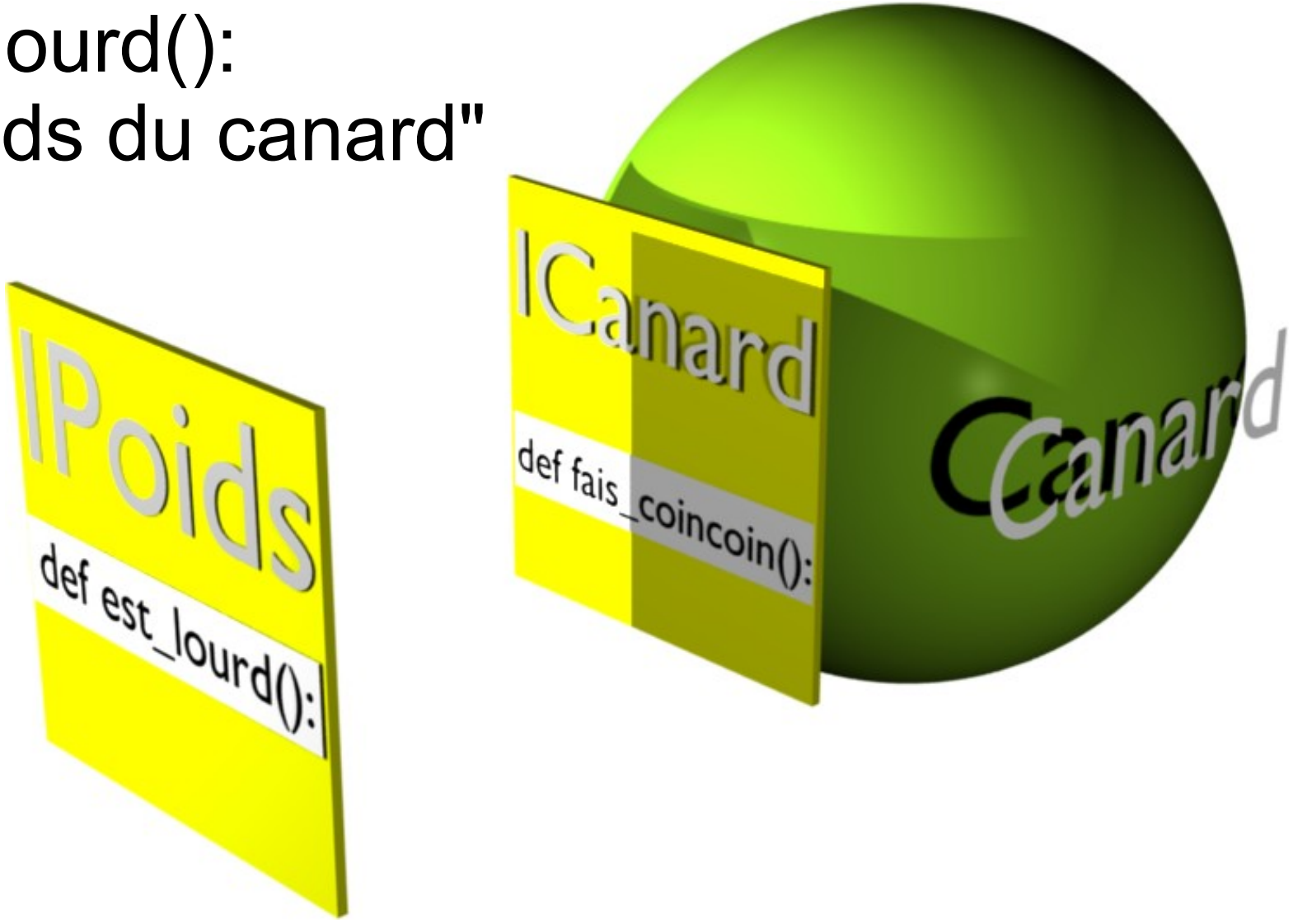
Objet avec interface

```
class Canard(object):
    implements(ICanard)
    def fais_coincoin(self):
        print 'coin'
```



Poids du canard ?

```
class IPoids(Interface):
    def est_lourd():
        "le poids du canard"
```



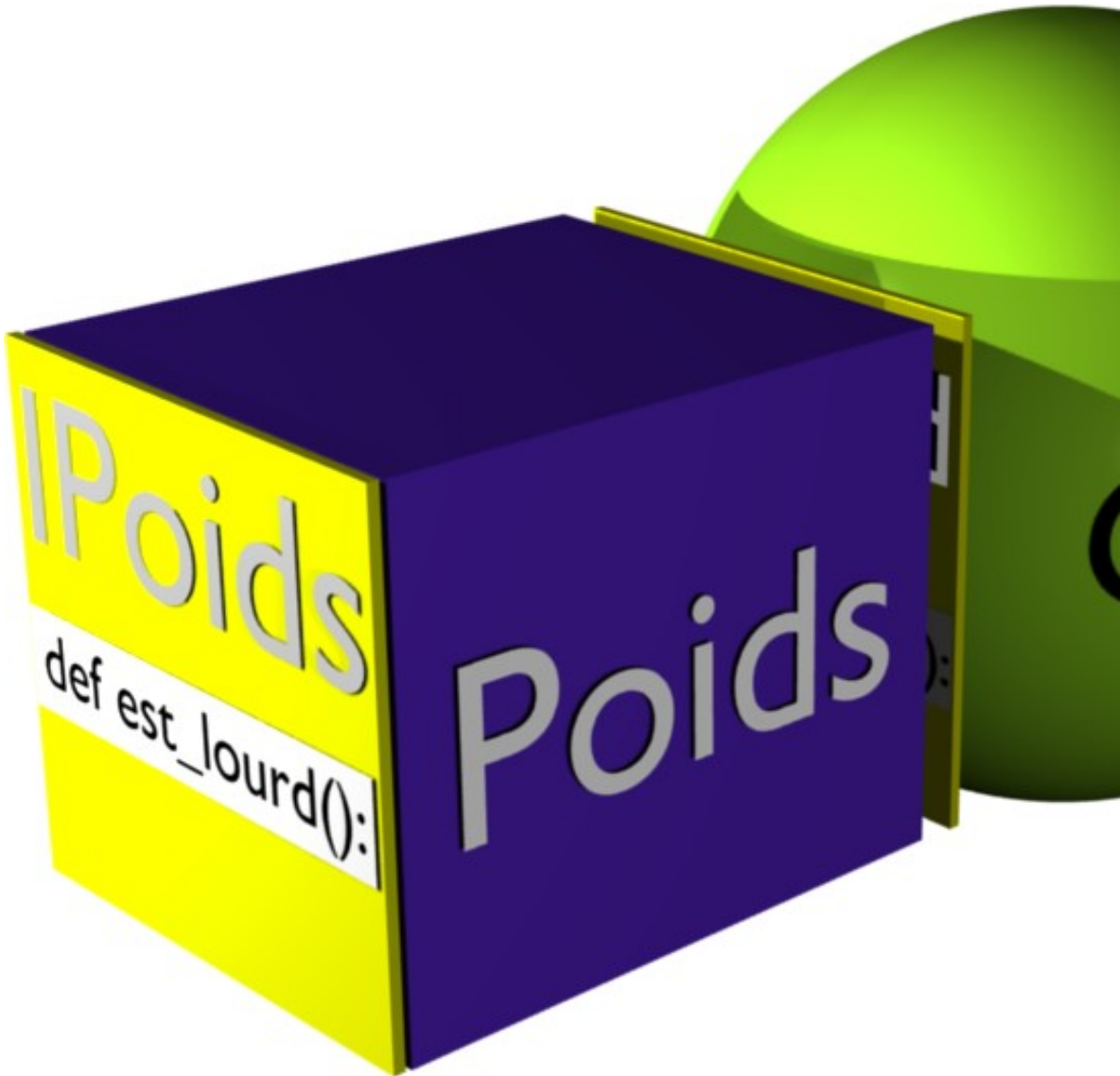
Adaptateur !

```
class Poids(object):

    implements(IPoids)
    adapts(ICanard)

    def __init__(self, context):
        self.context = context

    def est_lourd(self):
        #...utilise self.context...
        print "non pas trop"
```



Cas d'utilisation des adaptateurs

Métadonnées

Commentaires

URL d'un objet

Traversing

Taille

Catégorie

Texte à indexer

Vignette d'aperçu

...

Utilisation de l'*adaptateur*

Instanciation directe :

Poids(canard).est_lourd()

Component Architecture

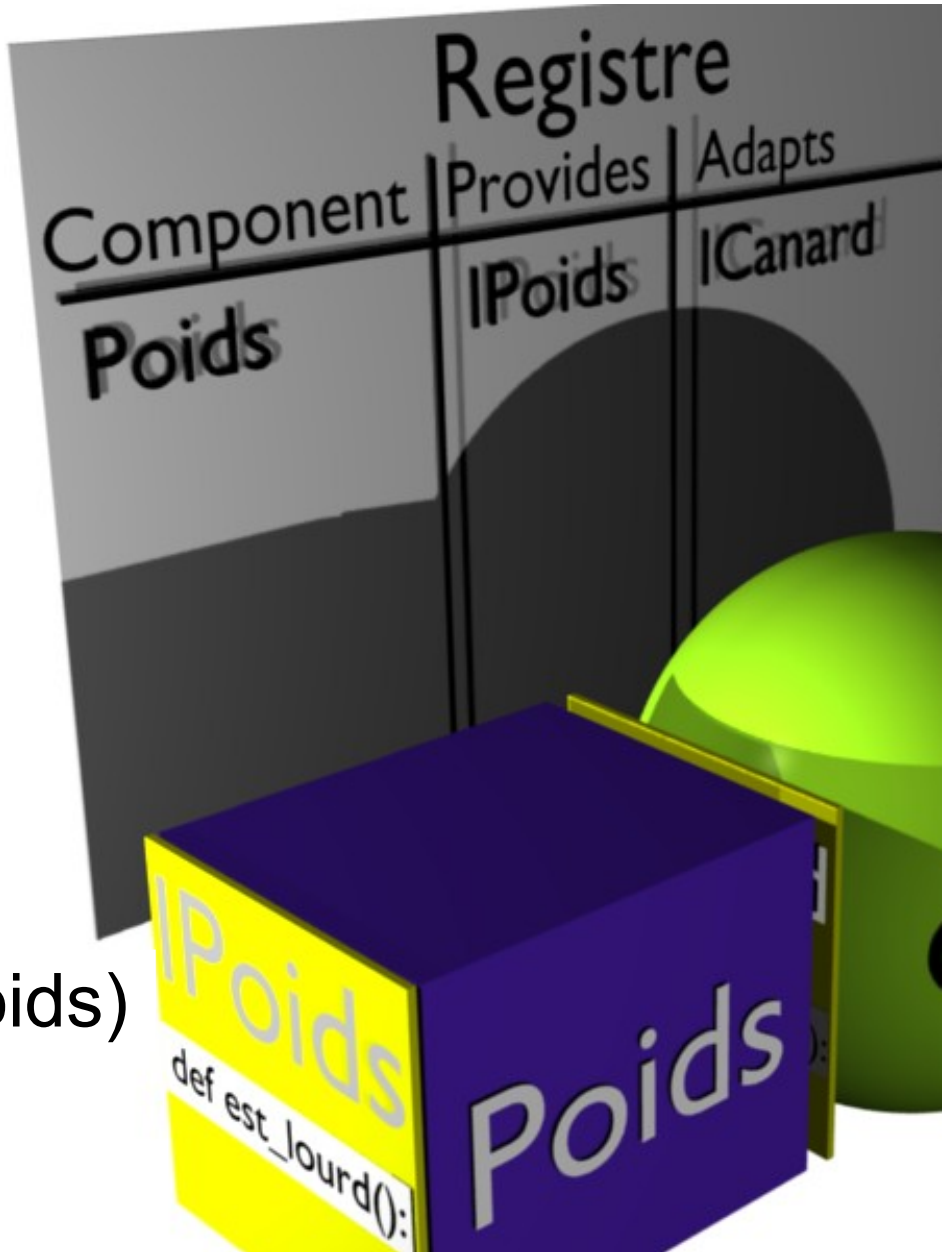
getAdapter(canard, IPoids).est_lourd()

Écriture simplifiée

IPoids(canard).est_lourd()

Comment marche getAdapter ?

Registre de composant



provideAdapter(Poids, ICanard, IPoids)

L'adaptateur a tous les avantages :

pas de conflit avec la classe adaptée

parfaite isolation des tests

classe adaptée n'est jamais touchée

fonctionnalités documentées

l'adaptateur se branche/débranche à volonté

l'implémentation peut changer à volonté

Réutilisabilité parfaite

Component Architecture

Deux paquets Python / C :

zope.interface

zope.component

```
>>> from zope.interface import implements
>>> from zope.interface import Interface

>>> from zope.component import adapts
>>> from zope.component import getAdapter
```

Merci
Thanks
Danke schön
Teşekkür ederim