# POOGL

Yves Caniou

24 mars 2005

# Modélisation objet UML

- Intro historico-sémantique
- Le vocabulaire de l'orienté-objet
- Quelques exemples de modélisation ratées et réussies
- Conclusion

# Première partie I

## Introduction

Où les étudiants apprennent avec consternation qu'ils vont devoir faire de la modélisation avant de se jeter sur leurs clavier.

# Un peu d'histoire (1) : la préhistoire

Il fut un temps :

- cobold
- Fortran
- Basic
- plein d'assembleur
- et Lisp pour embêter les étudiants

# Un peu d'histoire (2) : l'antiquité

Quand j'étais petit, la mode c'était Pascal :

- Programme = structures de données + algorithmes (Wirth)
- Typage fort et types structurés
- Code aussi structuré : *`Goto` considered harmful* (Dijkstra)
- Et donc sémantique compositionnelle

Et si on voulait des programmes efficaces malgré tous ces progrès, on avait C qui permettait de programmer proprement, ou bien cochonnement. Selon le besoin.

C'est là aussi que les langages fonctionnels sont devenus civilisés, mais pas au point que les gens s'en servent, (à part pour programmer des éditeurs de texte à 10Mo).

# Un peu d'histoire (2) : l'antiquité

Quand j'étais petit, la mode c'était Pascal :

- Programme = structures de données + algorithmes (Wirth)
- Typage fort et types structurés
- Code aussi structuré : *`Goto` considered harmful* (Dijkstra)
- Et donc sémantique compositionnelle

Et si on voulait des programmes efficaces malgré tous ces progrès, on avait C qui permettait de programmer proprement, ou bien cochonnement. Selon le besoin.

C'est là aussi que les langages fonctionnels sont devenus civilisés, mais pas au point que les gens s'en servent, (à part pour programmer des éditeurs de texte à 10Mo).

# Un peu d'histoire (2) : l'antiquité

Quand j'étais petit, la mode c'était Pascal :

- Programme = structures de données + algorithmes (Wirth)
- Typage fort et types structurés
- Code aussi structuré : *`Goto` considered harmful* (Dijkstra)
- Et donc sémantique compositionnelle

Et si on voulait des programmes efficaces malgré tous ces progrès, on avait C qui permettait de programmer proprement, ou bien cochonnement. Selon le besoin.

C'est là aussi que les langages fonctionnels sont devenus civilisés, mais pas au point que les gens s'en servent, (à part pour programmer des éditeurs de texte à 10Mo).

# Un peu d'histoire (2) : parenthèse 1

Pourtant, en 1965, résolution du mutex sur machine multiproc à mémoire partagée par Dijkstra :

```
boolean array B,C [1:N] ;
integer K,J ;

Tag1 : B[I]:=false ;
Tag2 : if( K != I )
       then
           begin
           C[I]:=true ;
           if( B[K] ) then K:=I
           goto Tag1
       end
       else
Tag3 :     begin
           C[I]:=false ;
           for J:=1 to N do
               if( (J!=I) && !(C[J]) ) then goto Tag1
       end
       { Section Critique }
       C[I]:=true ;
       B[I]:=true ;
```

# Un peu d'histoire (2) : parenthèse 1

Pourtant, en 1965, résolution du mutex sur machine multiproc à mémoire partagée par Dijkstra :

```
boolean array B,C [1:N] ;
integer K,J ;

Tag1 : B[I]:=false ;
Tag2 : if( K != I )
       then
           begin
           C[I]:=true ;
           if( B[K] ) then K:=I
           goto Tag1
       end
       else
Tag3 :     begin
           C[I]:=false ;
           for J:=1 to N do
               if( (J!=I) && !(C[J]) ) then goto Tag1
       end
       { Section Critique }
       C[I]:=true ;
       B[I]:=true ;
```

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ? hmm, la ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ? non, max et min
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ? sémantiquement plus proche de la réalité que "real" (Pascal)
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ? en plus, 24 bits $\longrightarrow$ 53 bits
- long double ?

# Un peu d'histoire (2) : parenthèse 2

Sémantique : Késako ?

- Si char a,b, alors que fait a+b ?
- int est-il un type proche de la réalité ?
- float ?
- double ?
- long double ?

# Un peu d'histoire (2) : parenthèse 2, suite

Sémantique compositionnelle : Késako ?

Si $[P] : \sigma \mapsto \sigma$ retourne un état correct
Alors $[P, Q] = f([P, Q])$, i.e. composition des états du système, retourne un état du système correct

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (3) : les temps modernes

Un jour, tout-à-coup, la mode est passée aux langages orientés-objets :

- encapsulation : on ne voit d'un objet ce qu'on a besoin de voir
- héritage : permet l'abstraction et la réutilisation
- polymorphisme : permet malgré tout du code simple/lisible
- tout cela compatible avec les progrès précédents.

Pascal imposait la séparation données/algorithmes, les langages 00 imposent en plus la séparation fonctionnalité/implémentation.

*Un programme ne s'écrit pas mais se réécrit*

Et si on veut des programmes efficaces malgré tous ces progrès, on a C++ qui permet de programmer proprement, ou bien cochonnement. Selon le besoin.

Même Caml est devenu orienté objet ! ? ! ? ? ! ! !

# Un peu d'histoire (4) : encore plus loin

Fini les *langages*, bonjour les *méthodologies* de programmation
On dit souvent conception orientée objet au lieu de programmation :

- Depuis toujours, avant de se jeter sur leur clavier, les gens sérieux réfléchissent et font des petits dessins (modélisation)
  - Assembleur/Basic/Fortran : organigramme pour s'y retrouver dans les gotos.
  - Pascal : en plus, définition des structures de données, représentations graphiques des pointeurs...
  - Modèles de programmation exotiques : graphes de flots de données, automates, ...
- En principe, cette étape de modélisation est indépendante du langage.
- La suite de l'histoire fut de concrétiser ce "en principe".

# Un peu d'histoire (4) : encore plus loin

Fini les *langages*, bonjour les *méthodologies* de programmation
On dit souvent conception orientée objet au lieu de programmation :

- Depuis toujours, avant de se jeter sur leur clavier, les gens sérieux réfléchissent et font des petits dessins (modélisation)
  - Assembleur/Basic/Fortran : organigramme pour s'y retrouver dans les `goto`s.
  - Pascal : en plus, définition des structures de données, représentations graphiques des pointeurs...
  - Modèles de programmation exotiques : graphes de flots de données, automates, ...
- En principe, cette étape de modélisation est indépendante du langage.
- La suite de l'histoire fut de concrétiser ce "en principe".

# Un peu d'histoire (4) : encore plus loin

Fini les *langages*, bonjour les *méthodologies* de programmation
On dit souvent conception orientée objet au lieu de programmation :

- Depuis toujours, avant de se jeter sur leur clavier, les gens sérieux réfléchissent et font des petits dessins (modélisation)
  - Assembleur/Basic/Fortran : organigramme pour s'y retrouver dans les `goto`s.
  - Pascal : en plus, définition des structures de données, représentations graphiques des pointeurs...
  - Modèles de programmation exotiques : graphes de flots de données, automates, ...
- En principe, cette étape de modélisation est indépendante du langage.
- La suite de l'histoire fut de concrétiser ce "en principe".

# Un peu d'histoire (4) : encore plus loin

Fini les *langages*, bonjour les *méthodologies* de programmation
On dit souvent conception orientée objet au lieu de programmation :

- Depuis toujours, avant de se jeter sur leur clavier, les gens sérieux réfléchissent et font des petits dessins (modélisation)
  - Assembleur/Basic/Fortran : organigramme pour s'y retrouver dans les `goto`s.
  - Pascal : en plus, définition des structures de données, représentations graphiques des pointeurs...
  - Modèles de programmation exotiques : graphes de flots de données, automates, ...
- En principe, cette étape de modélisation est indépendante du langage.
- La suite de l'histoire fut de concrétiser ce "en principe".

# Un peu d'histoire (7) : l'époque contemporaine

Après il y a eu d'autres langages objets, par exemple Java.

Mais ils n'inventent plus rien.
Ils font juste tout un peu mieux..

Je ne vais donc pas vous apprendre Java (ni C++, ni SmallTalk, ni rien). Je vais donc vous expliquer les Grands Principes de l'Orienté Objet. Ensuite vous vous débrouillerez pour apprendre les langages que vous voudrez vous-mêmes, sur le tas.

# Un peu d'histoire (7) : l'époque contemporaine

Après il y a eu d'autres langages objets, par exemple Java.

Mais ils n'inventent plus rien.
Ils font juste tout un peu mieux.. - Troll ? ☺

Je ne vais donc pas vous apprendre Java (ni C++, ni SmallTalk, ni rien). Je vais donc vous expliquer les Grands Principes de l'Orienté Objet. Ensuite vous vous débrouillerez pour apprendre les langages que vous voudrez vous-mêmes, sur le tas.