UML, le langage de modélisation objet unifié

par Laurent Piechocki

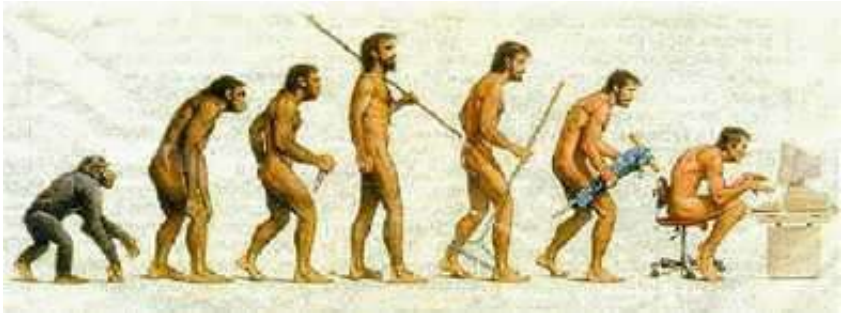
Date de publication : 22/10/07

Dernière mise à jour : 14/09/09

Né de la fusion des méthodes objet dominantes (OMT, Booch et OOSE), puis normalisé par l'OMG en 1997, UML est rapidement devenu un standard incontournable. UML n'est pas à l'origine des concepts objet, mais il en en donne une définition plus formelle et apporte la dimension méthodologique qui faisait défaut à l'approche objet.
Le but de cette présentation n'est pas de faire l'apologie d'UML, ni de restreindre UML à sa notation graphique, car le véritable intérêt d'UML est ailleurs !
En effet, maîtriser la notation graphique d'UML n'est pas une fin en soi. Ce qui est primordial, c'est d'utiliser les concepts objet à bon escient et d'appliquer la démarche d'analyse correspondante.
Cette présentation a donc pour objectif, d'une part, de montrer en quoi l'approche objet et UML constituent un "plus" et d'autre part, d'exposer comment utiliser UML dans la pratique, c'est-à-dire comment intégrer UML dans un processus de développement et comment modéliser avec UML.
Avertissement :
Les textes qui composent la présentation sont (volontairement) très synthétiques, à la manière de transparents qu'on projette au cours d'une formation.
Il faut donc savoir lire entre les lignes, car il ne s'agit là que d'un "tour d'horizon". Cette présentation ne se substitue donc ni aux formations plus "académiques", ni aux ouvrages de référence.

I - PRESENTATION D'UML

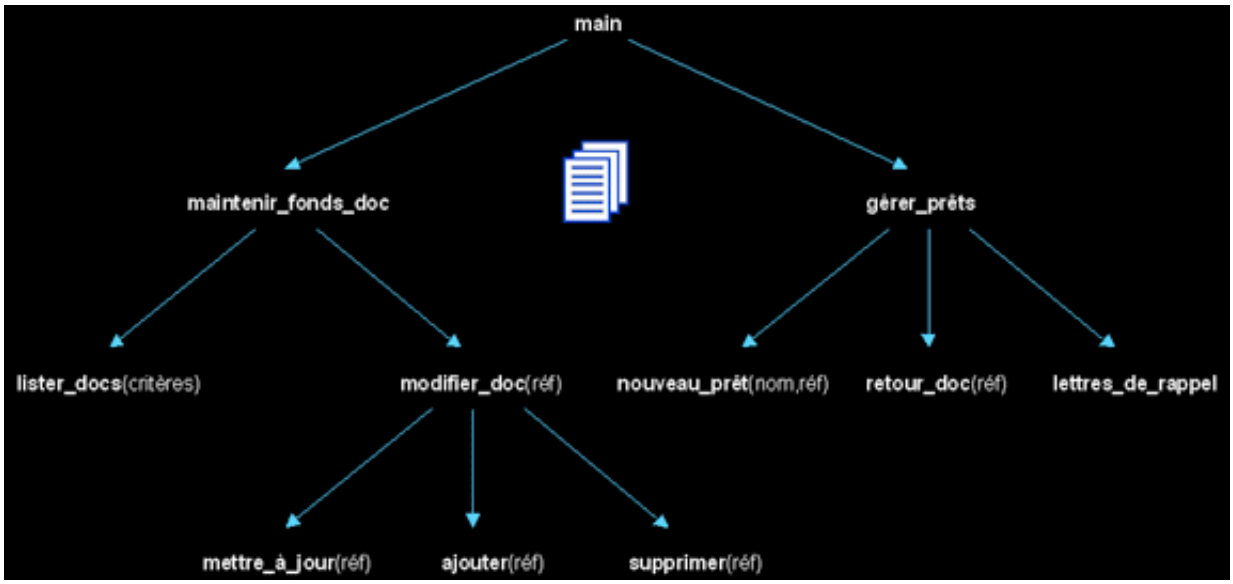
I-A - Un peu d'Histoire...



I-A-1 - Approche fonctionnelle vs. approche objet

I-A-1-a - La découpe fonctionnelle d'un problème informatique : une approche intuitive

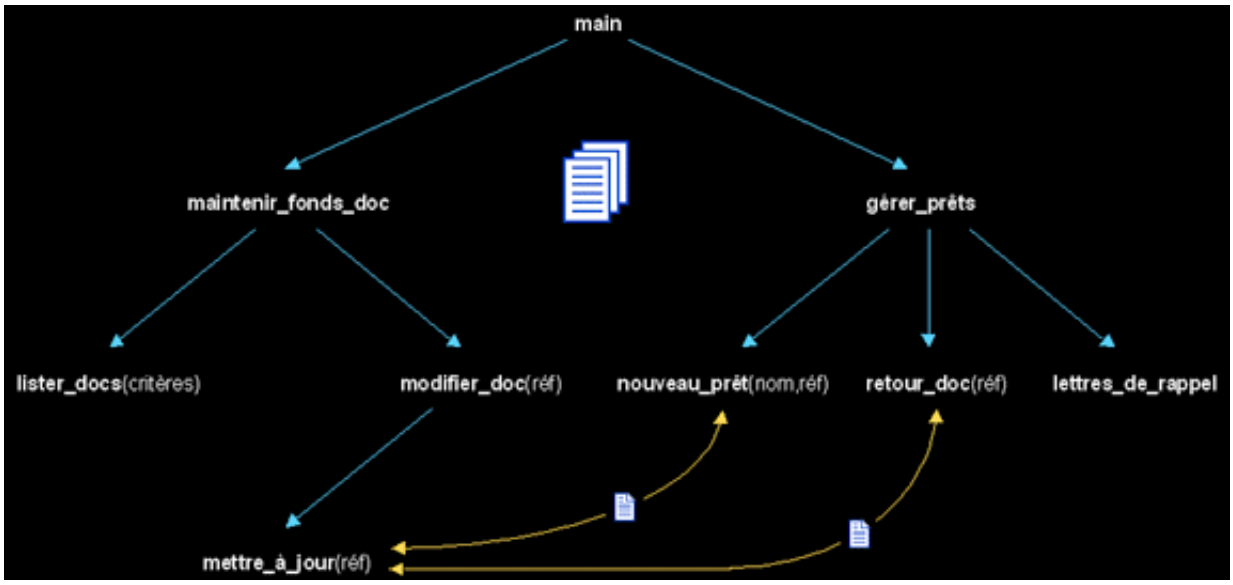
Exemple de découpe fonctionnelle d'un logiciel dédié à la gestion d'une bibliothèque :



Le logiciel est composé d'une hiérarchie de fonctions, qui ensemble, fournissent les services désirés, ainsi que de données qui représentent les éléments manipulés (livres, etc□). Logique, cohérent et intuitif.

I-A-1-b - Le "plus" de l'approche fonctionnelle : la factorisation des comportements

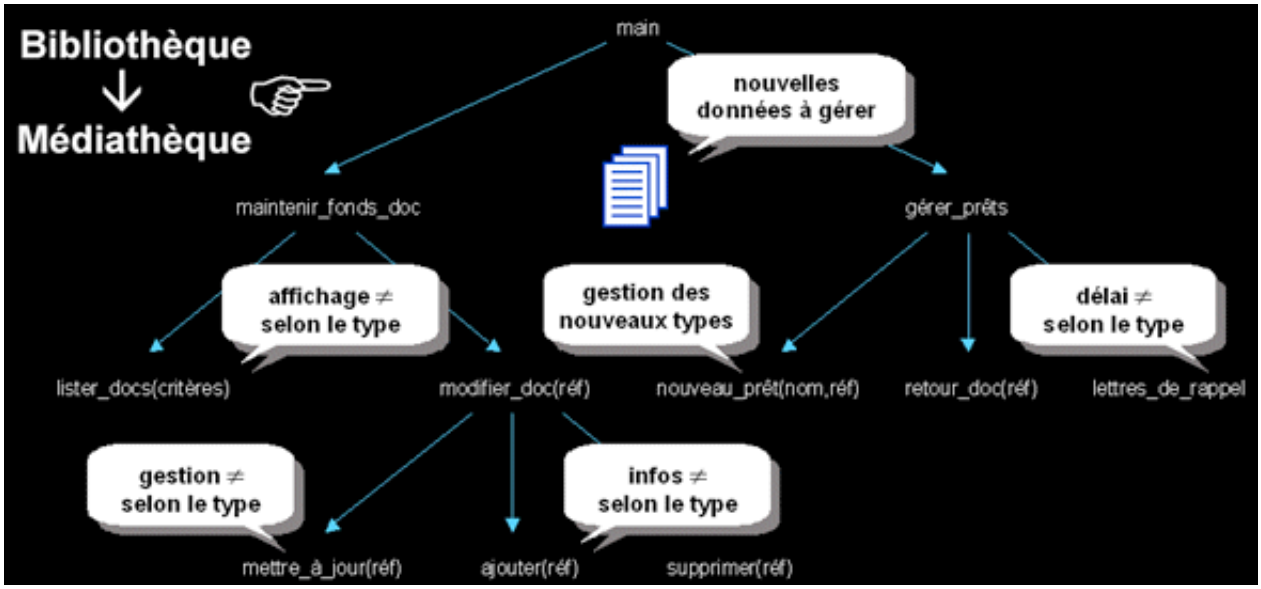
Une découpe fonctionnelle "intelligente" consiste à factoriser certains comportements communs du logiciel. En d'autres termes : pour réaliser une fonction du logiciel, on peut utiliser un ensemble d'autres fonctions, déjà disponibles, pour peu qu'on rende ces dernières un tant soit peu génériques. Génial !



I-A-1-c - Le revers de la médaille : maintenance complexe en cas d'évolution

Factoriser les comportements n'a malheureusement pas que des avantages. Les fonctions sont devenues interdépendantes : une simple mise à jour du logiciel à un point donné, peut impacter en cascade une multitude d'autres fonctions. On peut minorer cet impact, pour peu qu'on utilise des fonctions plus génériques et des structures de données ouvertes. Mais respecter ces contraintes rend l'écriture du logiciel et sa maintenance plus complexe.

En cas d'évolution majeure du logiciel (passage de la gestion d'une bibliothèque à celle d'une médiathèque par exemple), le scénario est encore pire. Même si la structure générale du logiciel reste valide, la multiplication des points de maintenance, engendrée par le chaînage des fonctions, rend l'adaptation très laborieuse. Le logiciel doit être retouché dans sa globalité :



I-A-1-d - La séparation des données et des traitements : le piège !

Examinons le problème de l'évolution de code fonctionnel plus en détail...

Faire évoluer une application de gestion de bibliothèque pour gérer une médiathèque, afin de prendre en compte de nouveaux types d'ouvrages (cassettes vidéo, CD-ROM, etc...), nécessite :

- de faire évoluer les structures de données qui sont manipulées par les fonctions,
- d'adapter les traitements, qui ne manipulaient à l'origine qu'un seul type de document (des livres).

Il faudra donc modifier toutes les portions de code qui utilisent la base documentaire, pour gérer les données et les actions propres aux différents types de documents.

Il faudra par exemple modifier la fonction qui réalise l'édition des "lettres de rappel" (une lettre de rappel est une mise en demeure, qu'on envoie automatiquement aux personnes qui tardent à rendre un ouvrage emprunté). Si l'on désire que le délai avant rappel varie selon le type de document emprunté, il faut prévoir une règle de calcul pour chaque type de document.

En fait, c'est la quasi-totalité de l'application qui devra être adaptée, pour gérer les nouvelles données et réaliser les traitements correspondants. Et cela, à chaque fois qu'on décidera de gérer un nouveau type de document !

```
struct Document
{
  char nom_doc[50];
  Type_doc type;
  Bool est_emprunte;
  char emprunteur[50];
  struct tm date_emprunt;
} DOC[MAX_DOCS];

  void lettres_de_rappel()
{
  /* ... */
  for (i = 0; i < NB_DOCS; i++)
  {
    if (DOC[i].est_emprunte)
    {
      switch(DOC[i].type)
      {
      case LIVRE:
        delai_avant_rappel = 20;
        break;
      case CASSETTE_VIDEO:
        delai_avant_rappel = 7;
        break;
      case CD_ROM:
        delai_avant_rappel = 5;
        break;
      }
    }
  }
/* ... */
}
  void mettre_a_jour(int ref)
   {
     /* ... */
     switch(DOC[ref].type)
     {
       case LIVRE:
         maj_livre(DOC[ref]);
         break;
       case CASSETTE_VIDEO:
         maj_k7(DOC[ref]);
         break;
       case CD_ROM:
         maj_cd(DOC[ref]);
         break;
     }
     /* ... */
   }
```

Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.