Modélisation Algébrique

Thierry EXCOFFIER

May 23, 2000

La première partie définira la modélisation algébrique.

La deuxième présentera quelques concepts de NanoÉdit.

On terminera par une démonstration.

1 Modèle

Un modèle est une représentation formelle souvent simplifiée de la réalité.

- Bleu
- Union(Sphère, Cube)
- ”P6\n320 200\n255\nrvbrvbrvb...
- “Cet <EM>a priori</EM> est un mot latin.”
- F est transitive si F(A,B) & F(B,C) => F(A,C)
- for(i=0;i<10;i++) printf(”%d”, i) ;

1.1 Méta-Modèle

Le Méta-modèle d'un modèle est la formalisation de la représentation formelle du modèle.

- Rouge, Vert, Bleu, Blanc, Gris, ...
- CSG
- PPM
- HTML
- Mathématique
- C

Le méta-modèle de HTML est SGML et SGML est son propre méta-modèle.

1.2 Modélisation

La modélisation est l'acte de création d'un modèle.

Un méta-modèle riche (expressif) autorise une grande finesse de modélisation.

Un méta-modèle pauvre implique une plus grande perte d'information dans l'acte de modélisation.

Si le modèle permet de modifier le méta-modèle (langage de programmation Dylan) il est possible de modéliser sans perte de sémantique.

1.3 Structures de donnée

Une structure de donnée est une représentation informatique d'un méta-modèle.

```
struct liste_int {
  struct liste *suivant ; /* NULL si pas de suivant */
  int valeur ;
} ;
```

Une instance de structure de donnée est une représentation informatique d'un modèle.

```
Adresse 4  {suivant=29, valeur=1996}
Adresse 20 {suivant=0 , valeur=1998}
Adresse 29 {suivant=20, valeur=1997}
```

1.4 Modélisation par langage

Les instances de structures de donnée sont générées par l'exécution d'un programme.

La donnée du programme et de ses entrées contient plus de sémantique que les données engendrées.

```
l = nouvelle_liste() ;
ajoute_fin(l, 1997) ;
ajoute_fin(l, 1997+1) ;
ajoute_debut(l, 1996) ; /* On a retrouvé le backup :-) */
```

C'est toujours le cas car la structure de donnée est forcément créée séquenciellement.

1.5 Modélisation algébrique

Pour avoir plus de sémantique, il ne faut pas instancier les données mais le programme permettant de les instancier.

C'est encore mieux d'instancier le programme permettant de générer le programme permettant de les instancier...

Les langages de programmation courants et le langage des mathématiques peuvent être mis sous une forme algébrique.

```
FOR(:=(i,0), <(i,10), INC(i), PRINTF("%d", i))
```

L'avantage d'un expression algébrique est sa syntaxe uniforme et son affichage graphique trivial.

1.6 Édition de modèle

Elle se fait en éditant le programme générant le modèle.

Si la modification représente une démarche de construction elle devra figurer explicitement dans le programme.

Si la modification fait une modification sémantiquement sans intérêt, elle ne figurera pas dans le programme.

Par exemple, on peut remplacer 1998 par 1999 :

- par substitution dans le modèle
- en ajoutant une fonction de substitution
- en détruisant/recréant l'élément
- ...

1.7 Traitements

Les traitements que l'on fait se classent en deux catégories : ceux qui sont fait une seule et unique fois et les autres.

Comme on ne sait pas à l'avance si un traitement sera refait, on considérera donc que tous les traitements sont répétitifs si on les fait une fois.

Les traitements doivent donc être inclus dans le modèle avec les données à traiter.

En conséquence, le modeleur n'implémente pas les traitements sur le modèle.

1.8 Compression

Une instance de structure de donnée classique contient beaucoup de redondance. On ne sait pas si elles ont un sens ou non.

La modélisation par langage permet d'éliminer ces redondances en ajoutant de la sémantique.

```
adresse="1, Av. des Chants Elisés" ;
téléphone="555 685 456" ;
personne("François Mitterrand", adresse, téléphone) ;
personne("Jacques Chirac", adresse, téléphone) ;
```

On voit donc que l'utilisation d'un langage permet d'ajouter une sémantique quelconque non prévue dans le modèle initial. (*cf.* JavaScript/HTML)

1.9 État courant

On peut ajouter au programme précédent la notion d'état courant permettant encore de le simplifier.

```
adresse_courante("1, Av. des Chants Elisés") ;
téléphone_courant("555 685 456") ;
personne("François Mitterrand") ;
personne("Jacques Chirac") ;
```

En poussant la démarche à l'extrème.

```
nom_courant("François Mitterrand") ;
personne() ;
nom_courant("Jacques Chirac") ;
personne() ;
```

1.10 Instanciation de la structure de donnée

Elle n'est ni obligatoire ni nécessaire.

La plupart des traitements peuvent s'effectuer directement en interprétant le programme dans le contexte spécifique du traitement.

Certaine accélérations sont possibles car les traitements sont fait au niveau "méta" donc formellement quand c'est possible.

1.11 Avantages de l'approche

- Le modèle prend peu de place.
- On perd moins de sémantique.
- L'historique de la modélisation est inutile car il est inclus dans le modèle.
- On peut travailler sur le modèle algébrique sans générer son instance (qui peut être de taille infini).

2 Modeleur algébrique

Un modeleur peut être découpé en :

Un noyau comportant un ensemble de fonctions de base permettant la gestion des classes et leur instanciation.

Un ensemble de classes qui sont instanciées pour créer les noeuds et feuille du modèle algébrique. Les plus particulières sont :

- Interface utilisateur (WWW, MOTIF, ...)
- Modeleur : l'éditeur du modèle
- Vue : un afficheur Open GL
- Opérateur/Variable : définition de fonctions.

Toutes ces classes peuvent être redéfinies.

2.1 Le but du modeleur

On doit pouvoir ajouter facilement de nouveaux type de noeuds sans avoir à modifier des programmes existants.

On ne doit pas à avoir à utiliser des fonctions (programmation) du modeleur.

Il doit être facile à utiliser et offrir toutes les fonctionnalités que l'on attend d'un modeleur.

Il doit permettre de faire **communiquer des fonctions développées dans des environnements différents**.

Le 3D et la modélisation géométrique ne doivent pas faire partie du modeleur afin de ne pas le limiter.

Le noyau du modeleur doit être petit et facilement maintenable.

2.2 La philosophie

Plusieurs idées importantes :

- Le programmeur décrit sa classe, le noyau utilise la classe. Le programmeur n'appelle pas de fonctions du noyau.
- Chaque donnée contenu dans les noeuds à trois types au lieu d'un : Stockage, Sémantique, Édition
- Il y a une hiérarchie des classes (de noeud) et des types sémantiques.
- On utilise l'*abus de langage* pour simplifier la construction du modèle. Le noeud de l'arbre n'attend pas une classe particulière pour son fils mais une donnée particulière dans celui ci.
  Le segment de droite défini par 2 points veut trouver des points dans ses 2 fils quelque soit ses fils

2.3 L'implémentation

Afin de minimiser les risques d'erreur, l'implémentation a été faite le plus proprement possible au détriment de la vitesse.

Pour l'interface, il faut que l'utilisateur puisse éditer textuellement ses données. C'est ce format textuel neutre qui assure la transmission d'informations entre objets.

Le noyau garantit que les structures de données sont initialisées. Il les libère après usage.

Il demande à chaque instance de s'évaluer en la prévenant des changements qui ont eu lieu.

2.4 Le noyau

Le noyau assure :

- L'initialisation des classes.
- La création des instances de classes.
- L'évaluation du modèle.
- La validation du modèle.
- Le transport des informations.
- La répartition des tâches.

2.5 Les classes de noeuds

Il y a quelques classes particulières permettant au modeleur de fonctionner.

Et toute les classes “normales”

2.5.1 Interface utilisateur

L'interface utilisateur ne fait pas parti du noyau.

L'instance de la classe “interface” assure :

- La gestion des ouvertures/fermeture de fenêtre.
- La génération du contenu des fenêtres.
- La mise à jour du contenu.
- Les bulles d'aides.
- Le service WWW pour la navigation dans toutes les structures de données du système.

C'est la classe choisie qui détermine le système de Widget utilisé.

2.5.2 La classe modeleur

Le modeleur ne fait pas parti du noyau.

L'instance de la classe modeleur assure :

- Les fonctions d'édition du graphe du modèle
- Lecture/Sauvegarde du modèle
- Undo/Redo

2.5.3 La classe afficheur

L'affichage et l'interaction 3D sont gérés par des instances d'une classe affichage.

Avantages :

- La vue graphique fait partie du modèle.
- On peut créer d'autres vues

Les autres classes doivent fournir des méthodes afin de pouvoir être affichées (Open GL et POV).

2.5.4 Les classes Opérateur/Variable

Une instance de “Opérateur” est en fait un appel de fonction. Les fils de l’opérateur étant les paramètres de la fonction.

Une instance de “Variable” indique le nom de la variable et sa valeur par défaut.

L’évaluation est faite par substitution.

2.6 Les types des valeurs

- Stockage
- Sémantique
- Éditeur

2.6.1 Le type de stockage des valeurs

On utilise des CoDec analogues à ceux de XDR dans les RPC Sun.

On dispose de nombreux types de bases qui supportent les méthodes :

- Conversion de/vers chaîne de caractères
- Informations sur le stockage (taille, méthodes, ...)
- Libération mémoire
- ...

L'utilisateur définit les nouveaux types en écrivant **une seule** fonction C qui définit toutes les méthodes.

```
codec_int(a->date_de_naissance) ;
codec_linefeed() ;
codec_L(&a->adresses, &a->nb_adresses, codec_adresse) ;
```

2.6.2 Le type sémantique

Il y a une hiérarchie de type sémantiques. A cette hiérarchie s'ajoute deux méthodes permettant d'ajouter des extensions.

Une méthode "prédicat" permettant de réduire la classe (aux nombres positifs par exemple)

Une méthode "sous-classage" permettant de l'augmenter en testant si l'on peut la rattacher ailleurs (variable, opérateur). Ceci permet de faire de l'héritage multiple ou contextuel.

2.7 Le type d'éditeur

Ce type indique quel Widget est utilisé pour modifier la valeur.

Par défaut c'est un éditeur de texte.

Cela peut-être un ascenseur, une palette de couleur, un éditeur d'image, ...

2.8 Démonstration