D'UML à COBOL

Où comment passer d'un modèle UML au code COBOL

Auteur : Erik Gollot (mars 2007)

Non, non, il ne s'agit pas de faire un cours sur l'histoire de l'informatique ou de remonter le temps mais bien de voir comment une application écrite en Cobol peut avoir été modélisée en UML.
Ce n'est pas non plus un article fumeux (heu, là c'est peut être à vous de juger) et commercial pour vous vendre un outil de modélisation UML, à vous les Cobolistes qui êtes persuadés que UML et Cobol c'est comme chien et chat.
Je reconnais que parler d'UML et Cobol dans un même article, autrement que pour les opposer, n'est pas très classique. Mais j'espère que si vous prenez un peu de temps pour lire la suite, vous serez convaincu qu'il n'y a pas vraiment d'autre incompatibilité que celle que vous imaginez.

Pourquoi on modélise ?

Que ce soit avec UML ou Merise, on modélise avant tout pour comprendre un besoin et pour mettre « noir sur blanc » les idées que chacun a en tête. L'objectif étant in fine de mieux se comprendre au sein de l'équipe et de partager tout en utilisant le même langage.
A ce niveau, le langage que l'on va utiliser pour le codage n'a que peu d'importance.

Heu, ok mais si je fais de l'UML et que j'utilise les concepts objets pour décrire les choses, vous aller commencer par me dire que je fume un peu car on est loin des concepts disponibles dans Cobol. Ouhai, vous avez en partie raison, des données avec des traitements à l'intérieur ça n'existe pas en Cobol, de même que la notion de généralisation/spécialisation (l'héritage en langage courant).

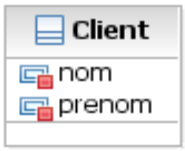
Donc, on est d'accord, il est peut être envisageable d'utiliser UML mais en faisant attention de ne pas utiliser ou plutôt abuser de notions trop éloignées des concepts que l'on a en Cobol. Regardons donc ce que peut être une modélisation UML pragmatique dans un contexte Cobol.

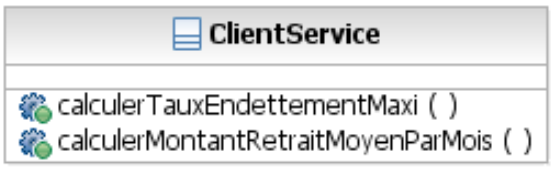
Une modélisation pragmatique

Ce que je propose ici est de voir comment utiliser UML, finalement comme vous avez peut être mis en oeuvre Merise, pour rendre accessible la notation UML à des personnes seulement habituées à Cobol.
Dans Cobol, tout comme dans des langages comme le C, on a uniquement les notions de structures de données et de traitements. L'idée est donc de conserver ces concepts dans nos modèles UML et de se restreindre uniquement à ceux-là (cf. cet autre article). Nous pouvons donc créer des modèles UML où l'on va utiliser le concept de classe mais en ne créant que des **classes de type « données »**, c'est à dire des classes ne contenant que des attributs et pas d'opération et des **classes de type « paquet de traitements »**, c'est à dire des classes ne comportant que des opérations et pas d'attribut.

Le « passage » à Cobol (on détaillera ceci dans les chapitres suivants) devient alors assez simple :

UML	Cobol

Classe « données »
Exemple :
 
	Structure de données Cobol
	Exemple :
	`01 (PREF)-CLIENT.`
	`  02 (PREF)-NOM  PIC X(32).`
	`  02 (PREF)-PRENOM  PIC X(32).`

Classe « traitements » Exemple : 	Plusieurs cas peuvent être envisagés en fonction des habitudes et/ou besoins d'implémentation : **Cas 1** : Chaque opération donne lieu à un programme Cobol (2 programmes ici). - Programmes : - « calculerTauxEndettementMaxi » - « calculerMontantRetraitMoyenParMoisi » **Cas 2** : La classe donne lieu à un programme Cobol et chaque opération est un paragraphe de ce programme. Un paramètre de type « code opération » est passé en entrée du programme pour savoir qu'elle « fonction » du programme on veut exécuter. - Programme « ClientService » - Paragraphes : - « calculerTauxEndettementMaxi » - « calculerMontantRetraitMoyenParMoisi »

Détails sur les concepts de données et de traitements

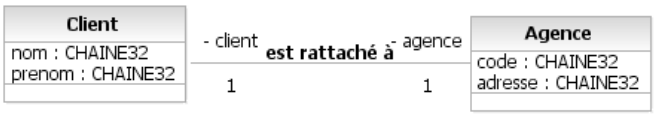
Bien, au premier abord je pense que vous avez vu que le passage d'UML à Cobol n'a rien d'impossible dès lors que l'on fait une modélisation UML « raisonnable » sans abuser des concepts objets purs et durs. Regardons maintenant, de manière plus précise la modélisation des données et la modélisation des traitements.

Note : Nous verrons en fin d'article comment typer les données avec des « PIC » Cobol. Ne vous formalisez donc pas pour le moment avec les types utilisés dans les exemples.

Les données

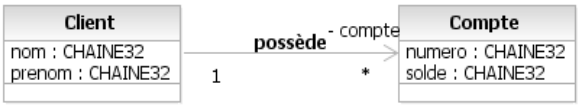
On a vu que le concept de classe UML se transforme simplement en Cobol sous la forme d'une structure de données. Un modèle UML, tout comme un modèle MCD Merise, peut aussi définir des relations entre classes (UML parle d'association).
La transformation du concept d'association UML et finalement tout aussi simple que celui de classe et dépend de la multiplicité.

UML	Cobol

Cas d'une multiplicité = 1
 
	`01 (PREF)-CLIENT.`
	`  03 (PREF)-NOM PIC X(32).`
	`  03 (PREF)-PRENOM PIC X(32).`
	`  03 (PREF)-AGENCE.`
	`  05 (PREF)-CODE PIC X(32).`
	`  05 (PREF)-ADRESSE PIC X(32).`

Cas d'une multiplicité > 1	`01 (PREF)-CLIENT.`
	`  03 (PREF)-NOM PIC X(32).`
	`  03 (PREF)-PRENOM PIC X(32).`

UML	Cobol

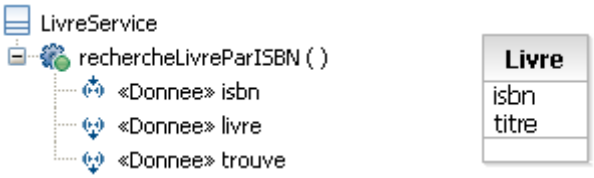
	`03 (PREF)-Q-COMPTE PIC S9(4) BINARY.`
	`03 (PREF)-COMPTE OCCURS 20.`
	`05 (PREF)-NUMERO PIC X(32).`
	`05 (PREF)-SOLDE PIC X(32).`
	
	Dans le cas d'une multiplicité > 1 il y a génération d'un tableau. La taille maximum du tableau est définie dans le modèle UML (cf. plus loin) et on utilise alors l'ordre **OCCURS** de Cobol.

Les traitements

Côté traitement, il faut considérer 2 aspects principaux et ce, que ce soit en UML, Merise, Cobol, ou Java ou...bref, quelque soit le langage : les **paramètres** d'entrée/sortie et **l'algorithme** du traitement.
On a vu précédemment qu'un programme Cobol pouvait être issu soit directement d'une opération d'une classe soit d'une classe elle-même.
Regardons dans un premier temps l'aspect paramètres et dans un second temps l'aspect algorithme.

Transformation Opération – Programme

En UML, on parle de **signature** d'opération, en Cobol on parle de **LINKAGE SECTION** d'un programme. La transformation de la signature d'une opération consiste donc à construire la « linkage section » du programme correspondant. Rappelons nous que les paramètres d'une opération peuvent être en entrée ou en sortie et que le type de ces paramètres peut être un type dit « de base » ou du type « classe ». Côté Cobol on aura donc des paramètres de type « PICTURE » (donnée « simple ») ou « groupe de PICTURE » (structure de données).



Dans l'exemple, nous avons une classe « **LivreService** » qui possède plusieurs opérations dont une « **rechercheLivreParISBN** » qui permet, à partir d'un code ISBN en entrée, de retrouver un « Livre » (en sortie donc). Le paramètre « trouve » est un sorte de booléen permettant de dire si on a trouvé ou non le livre recherché.

La LINKAGE SECTION du programme associé à l'opération « rechercheLivreParISBN » peut donc être écrite comme suit :

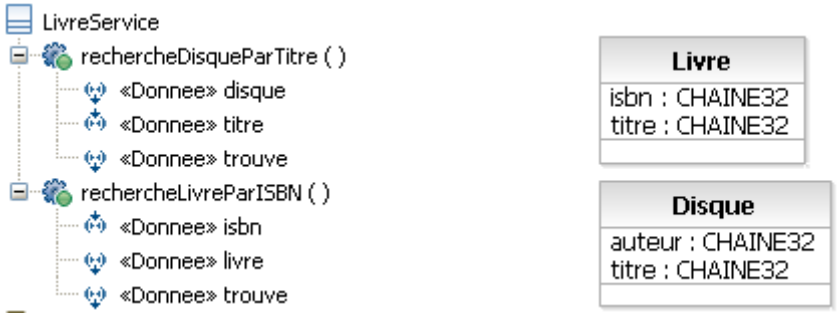
```
01 (PREF)-IN.
 03 (PREF)-ISBN PIC X(32).
01 (PREF)-OUT.
 03 (PREF)-LIVRE.
   05 (PREF)-ISBN PIC X(32).
   05 (PREF)-TITRE PIC X(32).
 03 (PREF)-TROUVE PIC X(1).
```

Transformation Classe – Programme

Dans le cas de la transformation Classe – Programme, nous avons défini que chaque opération de la classe représente une fonctionnalité du programme. Cette fonctionnalité est représentée par un « code opération » que l'on passe en paramètre du programme Cobol. Ce code a pour objectif de dire au programme quelle fonctionnalité (= opération) il doit exécuter.

Autre point, chaque fonctionnalité possède ses propres données en entrée / sortie. Ces données sont représentés en UML par les paramètres de l'opération qui représente la fonctionnalité.

Les données en entrée / sortie du programme sont donc « l'union » des données en entrée / sortie de chaque fonctionnalité.



Notre « **LivreService** » possède une nouvelle opération permettant de retrouver un « **Disque** » en fonction du titre. Nous voulons par contre transformer l'ensemble de la classe en un seul programme Cobol.

Voici ce que peut donner la LINKAGE SECTION de ce programme :

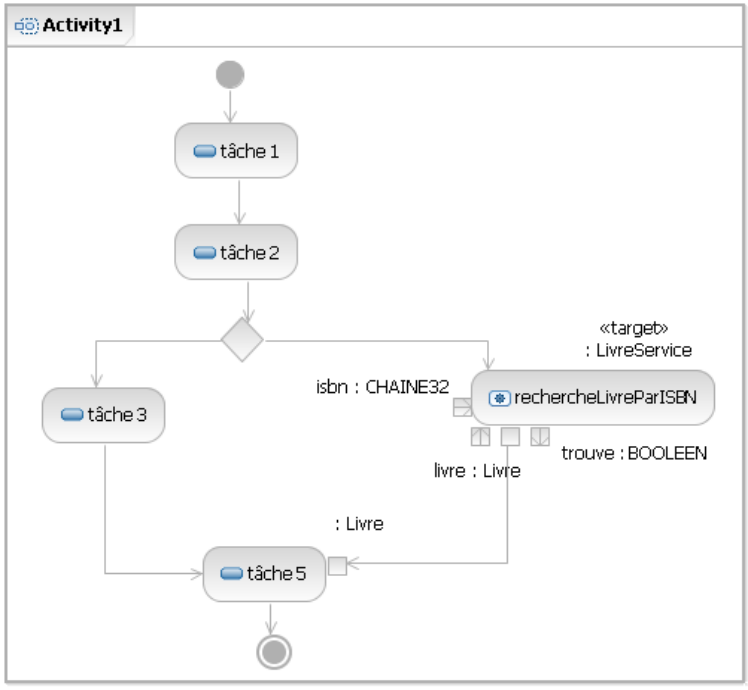
```
01 (PREF)-IN.
 03 (PREF)-CODEFONC PIC X(32).
 03 (PREF)-G-RECHLIVRE.
   05 (PREF)-ISBN PIC X(32).
 03 (PREF)-G-RECHDISQUE.
   05 (PREF)-TITRE PIC X(32).
01 (PREF)-OUT.
 03 (PREF)-G-RECHLIVRE.
   05 (PREF)-LIVRE.
     07 (PREF)-ISBN PIC X(32).
     07 (PREF)-TITRE PIC X(32).
   05 (PREF)-TROUVE PIC X(1).
 03 (PREF)-G-RECHDISQUE.
   05 (PREF)-DISQUE.
     07 (PREF)-AUTEUR PIC X(32).
     07 (PREF)-TITRE PIC X(32).
   05 (PREF)-TROUVE PIC X(1).
```

Dans la partie « IN », nous avons mis une données pour passer le « code opération » que le programme doit exécuter puis des groupes de données correspondant aux entrées de chaque « fonctionnalités » offertes par le programme.

De la même manière, des groupes de données ont été créés dans la partie « OUT » permettant d'accepter soit un « Livre » soit un « Disque ».

L'algorithme du programme / opération

Là, pas de grosse différence par rapport à notre cousin Merise. L'algorithme d'un programme peut être décrit par un diagramme d'activités (= le bon vieux flowchart) ou un diagramme de séquence.
Dans le cas du diagramme d'activité, on va décrire les différentes tâches que l'opération (donc le programme ou la fonctionnalité du programme) doit effectuer. Si l'une de ces tâches correspond à l'appel à une autre opération, et donc à un autre programme en Cobol, on peut dire que la tâche est un appel à une opération d'une classe (à partir de la norme UML 2.0).



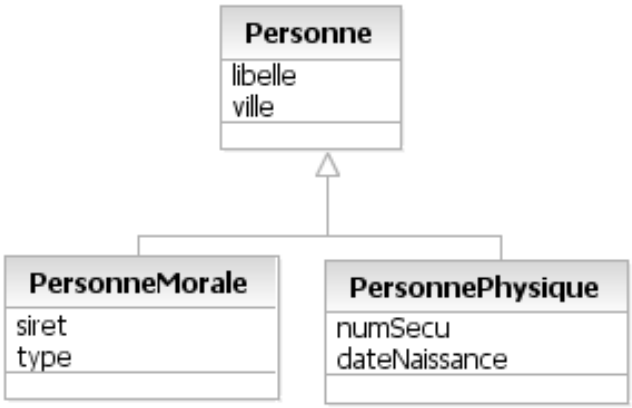
Dans l'exemple ci-dessus, on a un programme qui fait une série de tâches et fait appel, éventuellement, au programme « rechercheLivreParISBN ». Le « Livre » trouvé sert comme paramètre en entrée de la tâche 5.

Et l'héritage dans tout ça ?

Venons en à cette notion UML que vous imaginez peut être être totalement incompatible avec les capacités de Cobol, je vous présente la notion de généralisation / spécialisation ou communément appelée héritage.

Partons donc d'un cas concret, vous avez des « **Personne** », ces personnes se déclinent en « **PersonneMorale** » et « **PersonnePhysique** » et vous voulez réaliser un traitement qui retourne toutes les « Personne » par leur libelle. Regardons comment on modélise cela en UML :

Partie « données »



Partie « traitements »

«libelle» est un paramètre d'entrée et « personne » un paramètre de sortie.

La LINKAGE SECTION du programme Cobol représenté par notre opération doit donc pouvoir retourner à la fois des « PersonneMorale » et des « PersonnePhysique » puisque c'est ce que nous dit la relation d'héritage.
Eh bien, il n'y a aucun soucis avec Cobol, contrairement à ce que certains peuvent croire; et je suis certain que vous avez déjà mis en oeuvre la solution que je vais vous présenter. Simplement, vous n'aviez peut être pas conscience que cette « forme Cobol » pouvait provenir d'une relation d'héritage UML.

Code de la linkage section :

```
01 (PREF)-IN.
 03 (PREF)-VILLE PIC X(32).
01 (PREF)-OUT.
 03 (PREF)-PERSONNE.
   05 (PREF)-LIBELLE PIC X(32).
   05 (PREF)-VILLE PIC X(32).
   05 (PREF)-C-NHERIT-PERSONNE PIC 9(4) BINARY.
   05 (PREF)-L-NHERIT-PERSONNE PIC X(64).
   05 (PREF)-PERSMORALE REDEFINES (PREF)-L-NHERIT-PERSONNE.
     07 (PREF)-SIRET PIC X(32).
     07 (PREF)-TYPE PIC X(32).
   05 (PREF)-PERSPHYSIQUE REDEFINES (PREF)-L-NHERIT-PERSONNE.
     07 (PREF)-NUMSECU PIC X(32).
     07 (PREF)-DATENAISSANCE PIC X(32).
```

A partir de la structure des classes, on créé une structure de donnée Cobol comportant :

- Les données définies dans la classe de base « Personne » : libelle et ville
- Un « FILLER » : L-NHERIT-PERSONNE permettant d'accueillir les données des sous-classes et un code C-NHERIT-PERSONNE permettant de savoir, lors du retour quel type effectif on a dans la structure « Personne ».
- Les structures de données correspondant aux sous-classes, chaque structure « **REDEFINES** » le « FILLER » (qui a la taille de la plus grande des sous-classes)

Bref, rien de compliqué ici, et finalement, contrairement à ce qui est dit en introduction, il reste tout à fait possible d'utiliser la généralisation / spécialisation en UML avec un codage en Cobol.

Un générateur Cobol pour UML

Bon, j'espère que vous commencez à être un peu convaincu qu'il n'y a pas d'incompatibilité entre UML et Cobol dès lors que l'on se fixe des règles de transformation.
Il reste quand même un obstacle de taille, car une fois que l'on a fait nos beaux modèles avec, bien entendu, sur de vrais projets, pas mal de données et de traitements, comment passer rapidement au code Cobol ? Comment récupérer tout cet investissement dans la modélisation qui du reste est utile à bien des égards (cf. ici) ?

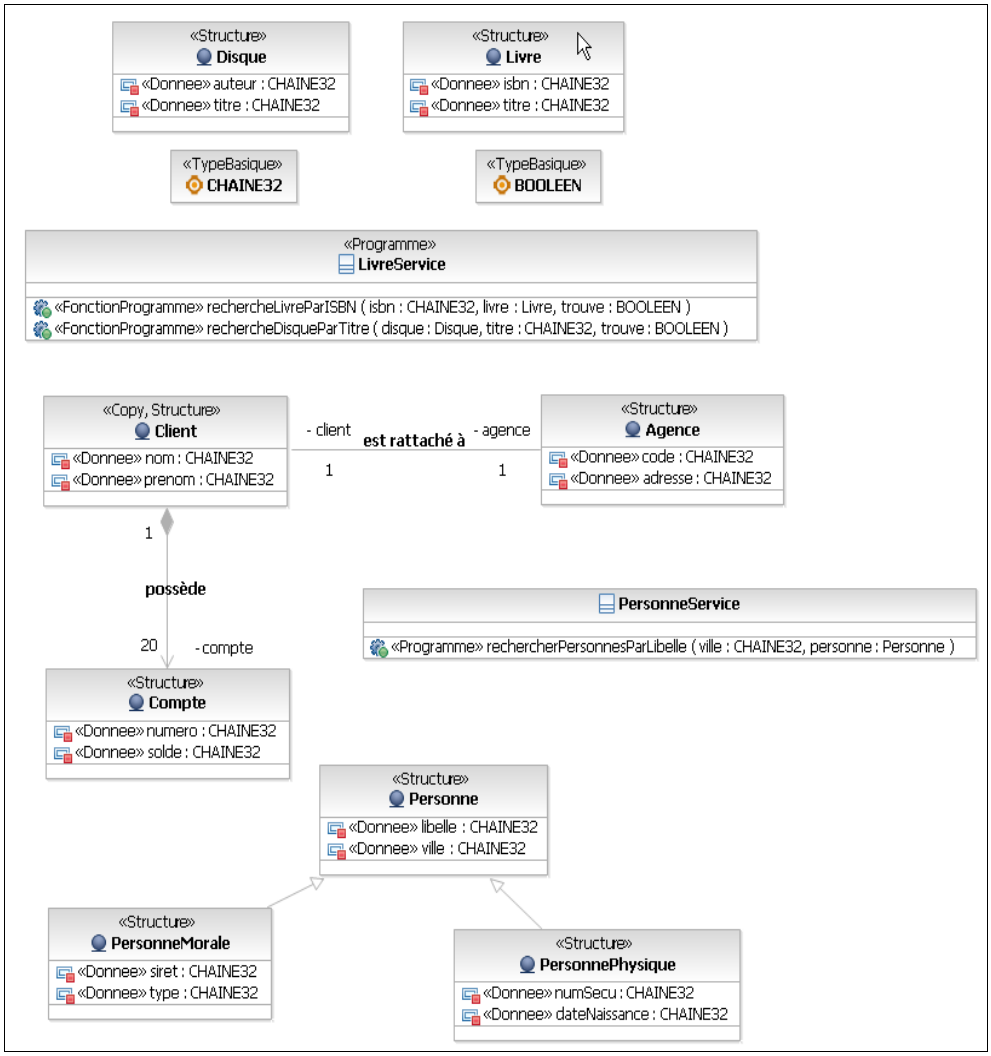
La solution, tout comme pour les autres langages de programmation, réside dans la **génération automatique de code**.

Maintenant que nous avons nos règles de transformation, il ne nous reste plus qu'à écrire un générateur Cobol. Pour cela, il est nécessaire d'ajouter quelques informations spécifiques Cobol à nos éléments UML pour que le générateur sache précisément quoi faire.
Il faut que le générateur sache (non exhaustif) :

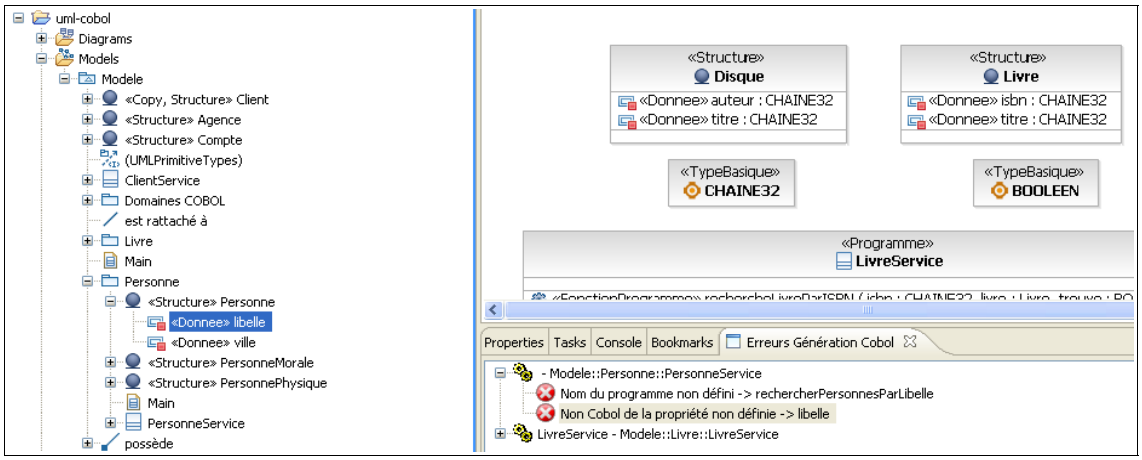
- Que l'on veut transformer une opération en programme
- Que l'on veut transformer une classe en programme
- Quel type Cobol utiliser pour les types de base comme Date, Entier, Booléen, ...?
- Quel est le nombre maximum d'éléments que peut contenir un tableau issu d'une association UML ?
- Quel est le nom Cobol que l'on veut donner à une donnée (s'il y a une différence par rapport au modèle UML) ?
- Quel est le nom du fichier dans lequel générer le code Cobol d'une « LINKAGE SECTION » ?

Ces informations peuvent être ajoutées au modèle par l'intermédiaire d'un mécanisme d'extension d'UML que l'on appelle « **Profil** ».
Ensuite, pour l'écriture du générateur proprement dit, tout dépend de l'outil UML que vous utilisez et de l'indépendance que vous voulez avoir par rapport à cet outil.
Dans mon cas, l'utilisation de Rational Software Modeler, outil qui s'appuie sur la plate-forme Eclipse et donc son système de plugin, m'a permis de réaliser facilement ce générateur.
Sans trop rentrer dans les détails de ce plugin, voici quelques copies d'écran qui vous donneront une idée de ce que cela peut donner :

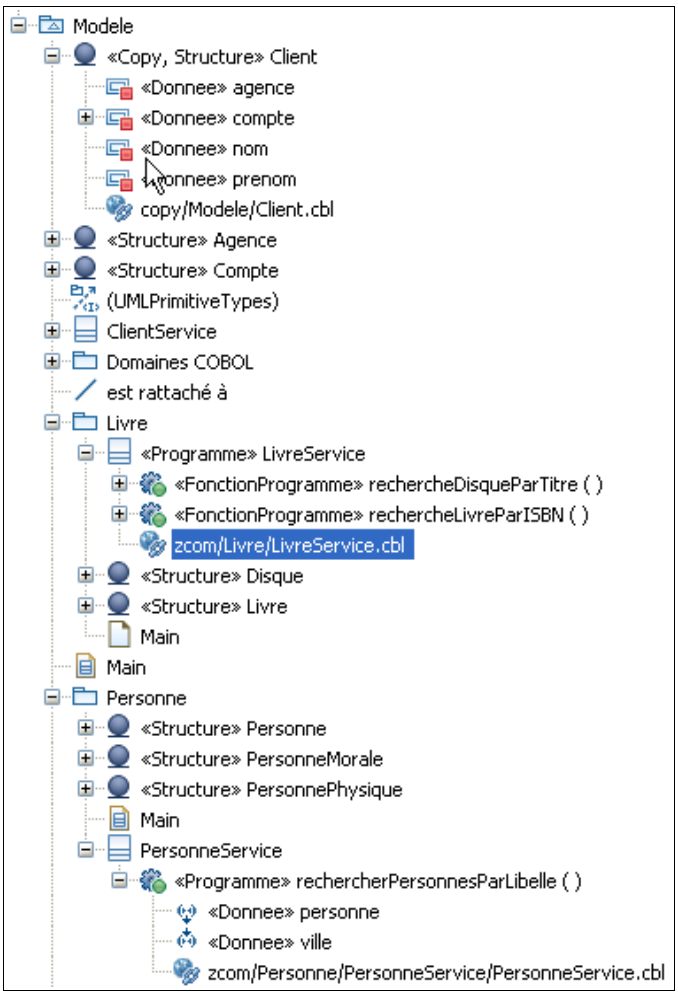
Profil Cobol



Gestion des erreurs de génération



Rattachement des fichiers générés aux éléments UML



Conclusion

On attend tous les adeptes du forum « Cobol » sur le forum « Conception », partie UML.