Modélisation Conceptuelle des Bases de Données: Techniques de Méta-Modélisation

Houari A. Sahraoui, Nicolas Revault
{sahraoui, revault}@laforia.ibp.fr

Laforia-IBP
Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS Cedex 05 FRANCE

Résumé:

La complexité des nouveaux systèmes à automatiser, combinée au changement de mentalité des utilisateurs, exige des outils de conception spécifiques, simples à comprendre et faciles à utiliser. Dans cet article, nous présentons une démarche et un *méta-outil* appliqués à la conception de schémas de bases de données et des traitements s'y rapportant. Le type de conception proposée est basé sur le vocabulaire du domaine. Notre outil permet, de plus, de transformer automatiquement les modèles conceptuels établis en programmes de description et de manipulation de données.

Mots clés: Méta-modélisation, modélisation conceptuelle, acquisition des connaissances, bases de données, systèmes d'information, systèmes de règles.

Abstract:

The complexity of new systems to automate, combined to the change of user's mentality, requires some specific design tools, simple to understand and easy to use. In this paper, we present a method and a *metatool* for databases and appropriate processing design. The type of design proposed is based on the domain vocabulary. Moreover, our tool enables the automatic transformation of the established conceptual models into data description and manipulation programs .

Keywords: Metamodelling, conceptual modelling, knowledge acquisition, databases, information systems, rule systems.

INTRODUCTION

Les méthodes de conception des systèmes d'information, qu'elles soient supportées ou non par des outils, impliquent la construction de modèles conceptuels de données et de traitements. Ces modèles sont indépendants de toute contrainte d'implémentation. Leur construction se fait en utilisant des formalismes[1] divers (entité/association [Chen76], réseaux de Pétri, systèmes de représentation objet, etc.). Les représentations résultant de cette construction sont généralement transformées en programmes. La transformation se réalise en plusieurs étapes, suivant un certain nombre de règles plus ou moins automatisables.

Les critères qui contribuent à juger de la pertinence de ces méthodes sont principalement liés aux propriétés des formalismes de modélisation qu'elles mettent en oeuvre. Les trois critères suivants nous paraissent déterminants:

1. Le pouvoir d'expression déterminé par la distance entre les formes langagières du formalisme de modélisation et le discours du domaine;
2. Les garanties offertes par le formalisme de modélisation de disposer des règles de transformations des modèles en programmes;
3. La convivialité de l'environnement de modélisation caractérisé par le degré de simplicité des formalismes employés.

Les méthodes de conception répertoriées intègrent ces trois critères de manière plus ou moins discutable. La volonté de mieux prendre en compte chacun de ces critères, ainsi que celle d'augmenter l'automatisation d'une plus grande partie du cycle de vie des logiciels, ont introduit l'utilisation des techniques de l'intelligence artificielle dans de nombreux outils. Le système SECSI [Bouzeghoub88] par exemple, fournit un formalisme de modélisation est de type *réseau sémantique* (MORSE [Bouzeghoub84]). Il utilise, de plus, un *système expert* pour la transformation des modèles conceptuels en schéma de base de données relationnelles.

Malgré ces améliorations, SECSI propose au concepteur un formalisme prédéterminé et figé pour modéliser son application. Ce constat est également valable pour les outils issus des méthodologies de conception telles que MERISE [Tardieu85], AXIAL [Pellaumail86] ou REMORA [Rolland88].

---

[1] *Dans la littérature le terme "modèle" est utilisé indifféremment pour designer le langage de la modélisation ou bien l'objet même de celle-ci. Dans cet article nous utilisons le terme "formalisme" pour désigner le langage, et le terme "modèle" pour l'objet de la modélisation.*

D'autre part, les différents outils utilisés comme supports de ces méthodologies, présentent des formalismes de modélisation relativement analogues. Ces formalismes sont constitués d'éléments dont la sémantique et les règles d'usage qui en découlent, ne varient que légèrement d'un outil à l'autre [Hull87]. Les principales différences résident dans la précision de détails permise, dans l'architecture globale véhiculée et enfin dans les possibilités d'expression de contraintes qui sont offertes au concepteur [Cooper91].

Le constat de la convergence entre les différents formalismes a amené une partie des chercheurs à travailler sur un principe de *méta-modélisation.* Selon ce principe, les formalismes de modélisation d'applications font eux-mêmes l'objet d'une modélisation préalable. Pour matérialiser ce principe, des outils d'une nouvelle génération ont été développés: les *méta-outils* [Morejon91], [Jeulin89], [Atzeni93]. Ces outils offrent la possibilité d'adapter les formalismes à la nature des objets de la modélisation. Cela permet d'optimiser le premier des critères retenus pour juger de la pertinence d'une méthode: l'expressivité du formalisme.

Par ailleurs, à travers cette optimisation, le principe de méta-modélisation présente un intérêt par rapport à la complexité croissante des applications envisagées. Cette complexité découle de deux aspects complémentaires: d'une part on cherche à informatiser des activités de moins en moins bien formalisées au sein des organisations, d'autre part les utilisateurs supportent de moins en moins bien les systèmes prescriptifs qui leur imposent des démarches et procédés figés. Les nouvelles applications doivent être adaptables, évolutives, partenaires de l'utilisateur final. Ces impératifs entraînent le transfert progressif de l'activité de conception des systèmes d'information, depuis les spécialistes externes aux organisations (informaticiens, analystes/concepteurs), vers les acteurs internes à celles-ci (futurs utilisateurs, prescripteurs). Pour ce faire, ces acteurs doivent avoir à leur disposition des outils basés sur des formalismes adéquats, simples à comprendre et faciles à utiliser.

C'est dans cette optique qu'ont été développés des environnements logiciels comme GraphOR [Morejon91] qui couvre les activités de méta-modélisation, de modélisation et de génération de code, ou comme GraphTalk [Jeulin89] qui se présente comme un générateur d'ateliers de génie logiciel (AGL) en construisant des méta-modèles des différentes méthodologies de conception proposées sur le marché. L'outil proposé par [Atzeni93] adopte un point de vue différent en associant au niveau "méta" une abstraction des modèles des différents niveaux de la modélisation (conceptuel, logique et physique), ainsi que des règles de passage d'un modèle à un autre.

Malgré l'adaptabilité que leur confère le niveau "méta", ces environnements restent, dans les formalismes qu'ils proposent aux concepteurs, trop dépendants des démarches théoriques qui sous-tendent les principales méthodologies. En effet, la tâche d'instanciation des éléments du formalisme prédomine parfois sur celle de la description de la réalité de l'univers de l'application. Par exemple, GraphTalk ne fait principalement valoir que les possibilités qu'il offre de méta-modéliser les méthodes de conception standards comme OOA [Coad90], OOD [Booch90], Merise, Merise C/S, AXIAL, ...

Pour pallier à cette limitation, nous proposons à travers cet article, une démarche de méta-modélisation " orientée domaine ". Nous présentons le prototype d'un méta-outil permettant de construire économiquement des éditeurs de modèles conceptuels de bases de données et des traitements afférents. Ces éditeurs mettent en oeuvre des éléments de représentation qui s'appuient sur les concepts du domaine d'application. Ils sont donc facilement utilisables par les experts. Notre environnement permet aussi de spécifier efficacement une transformation automatique des modèles conceptuels énoncés en textes interprétables par un sytème de gestion de bases de données (SGBD).

Dans la première partie, nous montrons les problèmes rencontrés dans la construction des systèmes à base de connaissances et la manière dont ils sont contournés par les méthodes mises au point en acquisition des connaissances. Dans la deuxième partie nous nous appuyons sur les résultats de cette étude, pour proposer une démarche similaire pour la construction des applications qui mettent en oeuvre des SGBD. Cette partie contient une description générale de l'architecture de notre environnement. Le niveau méta-modélisation est développé dans la troisième partie et le niveau modélisation dans la quatrième partie. Nous terminons cet article par une conclusion dans laquelle nous évoquons la pertinence de notre démarche ainsi que les perspectives de notre travail.

1. LA MODÉLISATION CONCEPTUELLE DANS LES SBC

Contrairement à ce qui s'est produit dans le domaine des systèmes d'information en général, le processus de modélisation conceptuelle dans la construction des SBC s'est imposé beaucoup plus tard. En effet, la construction des premiers SBC consistait en la transcription de "morceaux" de l'expertise dans des formalismes de représentation dépendants étroitement de l'implémentation (règles, frames). Cette pratique posait beaucoup de problèmes concernant notamment la validation, la maintenance, l'extension et l'explication. Pour résoudre ces problèmes - en phase avec les travaux de Newell [Newell82] - les concepteurs repensèrent les modalités et l'outillage de construction des

SBC. Plus particulièrement, ils s'appliquèrent à l'introduction de la notion de modèle conceptuel dans le but de faciliter entre autres les opérations de validation et de transformation. De plus, il s'est avéré que le fait de disposer d'un modèle conceptuel du système facilitait considérablement les opérations de maintenance et d'extension.

1.1. Première génération d'outils

Un nombre important d'outils d'acquisition des connaissances ont vu le jour en reprenant l'idée de modèle conceptuel. Les principes de modélisation mis en oeuvre dans ces outils s'appuient généralement sur des méthodes de résolution de problèmes. Ces méthodes fondent l'opérationalité des éditeurs d'acquisition tout en guidant le processus de modélisation lié à l'acquisition des connaissances[1]. Nous citerons à titre d'exemple SALT [Marcus89] qui utilise la méthode *propose-and-revise*, MOLE [Eshelman88] avec la méthode *cover-and-differentiate* et ROGET [Bennett85] qui repose sur la méthode *heuristic classification*.

Ces premiers outils ont effectivement permis d'améliorer l'enchaînement des étapes de conception des SBC. Ils n'ont toutefois pas résolu la problématique de la modélisation. En effet, les formalismes proposés dans ces outils, aussi complets soient ils, demeurent rigides et contraignants pour leurs utilisateurs. Ils imposent à ces derniers de raisonner en termes des concepts du formalisme lui même et non en termes des concepts du domaine caractérisant l'application [Musen89]. On retrouve là les mêmes défauts que ceux des premiers outils de conception des systèmes d'information. Ce phénomène est dû en grande partie à l'inadéquation du formalisme par rapport à la nature des concepts à modéliser. Ce problème touche particulièrement les nouveaux systèmes à automatiser. Ces derniers sont en effet de nature très complexe et nécessitent des outils plus spécifiques.

L'un des axes exploités par les chercheurs en IA est celui de la méta-modélisation. L'idée en est simple à première vue: elle consiste à introduire un niveau d'abstraction supplémentaire (le niveau "méta") permettant de générer des outils de modélisation adaptés à la nature de l'application à construire.

---

[1] *Les méthodes de résolution fournissent en effet des rôles qui doivent être remplis par des connaissances du domaine. Ces rôles définissent le langage - ou formalisme - de l'acquisition des connaissances. Les opérateurs associés aux rôles dans la méthode étant implémentés, ils définissent la sémantique opérationnelle du langage, dans le but de générer la base de connaissances visée.*

1.2. Les méta-outils

Il existe actuellement un nombre important de travaux autour du développement de méta-outils dans le domaine de la construction des SBC [Klinker91], [Walter92], [Linster92]. Etudions plus précisément l'un des plus connus: le système PROTÉGÉ [Musen93]. PROTÉGÉ est un méta-outil d'acquisition basé sur la méthode de résolution de problèmes ESPR (*Episodic Skeletal-Plan Refinement*). Contrairement aux outils de première génération, il ne permet pas de générer une base de connaissances, mais il est utilisé pour construire un outil d'acquisition. C'est en cela qu'il est qualifié de méta-outil.

Le principe de PROTÉGÉ est de construire des modèles de tâches. Ces modèles de tâches sont en fait caractérisés par une description des concepts du domaine et de ceux de la méthode de résolution de problèmes, et par des liens de mise en correspondance (*mappings*) entre ces deux types de concepts. Ces modèles sont destinés à piloter la génération des éditeurs d'acquisition à partir de composants génériques d'interface[1]. Les éditeurs d'acquisition générés sont ensuite utilisés pour modéliser l'expertise dans les termes habituellement employés dans la communauté concernée. Les modèles conceptuels ainsi obtenus sont enfin transformés automatiquement en bases de connaissances.

La construction des modèles de tâches introduit les concepts du domaine comme éléments de base du processus de modélisation. C'est en partie en cela que ce type d'outils constitue une nouvelle alternative aux méthodes de construction des SBC du type " première génération ".

La démarche constitutive de l'approche PROTÉGÉ permet de mieux gérer la répartition des responsabilités entre les différents acteurs intervenant dans le processus de construction des SBC. Dans une première phase, les cogniticiens et les spécialistes du domaine interagissent au niveau méta du système pour créer les modèles de tâche. Par la suite, les experts utilisent les éditeurs d'acquisition pour modéliser leurs expertises. Ils instancient en fait les modèles de tâches adaptés au domaine de leur application qui permettront finalement de générer les bases de connaissances sur lesquelles travaillent les utilisateurs finaux.

---

[1] *Ces composants sont par exemple des éditeurs élémentaires de listes, de tableaux, de graphes, etc...*

1.3. Résumé

En résumé, nous retenons de cette étude les éléments suivants comme points clés des démarches de modélisation:

- La maîtrise de la répartition des responsabilités entre les différents acteurs.
- L'introduction des concepts du domaine dans l'outil de modélisation.
- L'articulation explicite entre 1) la description du domaine et 2) la sémantique opérationnelle du langage de modélisation (ici la méthode de résolution).
- La perspective de réutilisation des différents modules intervenants dans le processus de modélisation (celui décrivant le domaine, celui définit par la méthode de résolution, ainsi que de ceux relatifs à l'interface).

Nous considérons que ces différents points relèvent d'une problématique de modélisation plus générale que celle de la construction des SBC. Ils doivent être pris en compte pour la définition de tous systèmes d'aide à la construction d'applications informatiques.

2. PRINCIPES DE LA DÉMARCHE PROPOSÉE

Nous nous proposons d'étudier le processus de construction d'applications mettant en oeuvre un SGBD. Dans ces applications nous appellerons interprète le moteur permettant de traiter des requêtes qu'elles soient de définition ou de manipulation de données.

Notre outil est basé sur trois niveaux de description: *méta-modélisation*, *modélisation* et enfin *application* (figure 2.1).

Le processus de méta-modélisation a une double fonction: 1- il permet de définir le formalisme utilisé lors de la modélisation conceptuelle; 2- il permet d'établir les règles de transformation des modèles conceptuels en textes effectivement interprétables (programmes). Nous disposons d'un éditeur de méta-modèles grâce auquel sont d'abord représentés les concepts liés au *vocabulaire du domaine*. On associe ensuite une sémantique opérationnelle à ces représentations en les mettant en correspondance, par l'écriture de règles de transformation, avec celles élaborées dans un autre méta-modèle prédéfini: celui de l'interprète cible. Comme précisé au paragraphe 3.1, ce dernier méta-modèle est déjà *opérationnel*. Il se présente en effet comme une abstraction de l'interprète cible, et intègre un processus de génération de code.

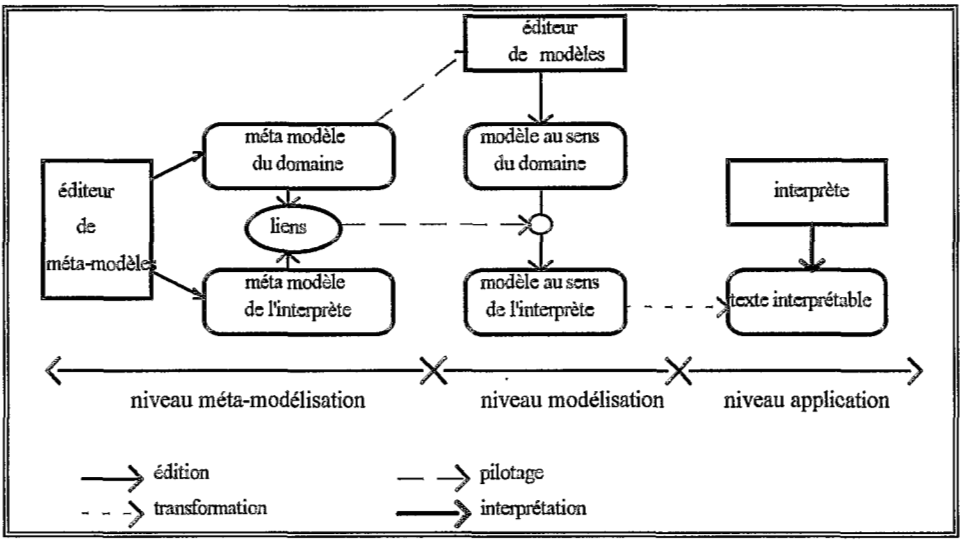

figure 2.1: Architecture générale

Dans l'activité de méta-modélisation, deux acteurs interviennent: le *spécialiste du domaine* et *l'informaticien*. Le premier a pour tâche de construire le méta-modèle du domaine. Les deux travaillent en commun pour établir les règles de mise en correspondance entre les deux méta-modèles (domaine et interprète). La conjonction de ces deux tâches permet d'engendrer un ensemble d'éditeurs de modèles conceptuels, et de compléter le transformateur de ces modèles en programmes.

Au cours de l'étape de modélisation, les concepteurs établissent leurs modèles conceptuels grâce aux éditeurs résultants de la méta-modélisation.

Le formalisme utilisé étant basé sur le vocabulaire du domaine, il favorise une modélisation conceptuelle plus "naturelle". Une fois le modèle conceptuel d'une application édité, il est transformé en programme. Cette transformation est en partie réalisée par les règles de mise en correspondance. Elle est automatique, et les utilisateurs finaux peuvent ainsi valider[1] l'application dès la fin de la conception. Nous détaillons les principes mis en jeu dans notre outil dans les parties 3 et 4.

Pour illustrer notre démarche, nous allons utiliser le même exemple tout au long de l'article. Il traite de la construction d'une application d'établissement

---

[1] *Nous parlons à ce niveau de validation expérimentale par opposition à la notion de validation formelle.*

de polices d'assurance (cet exemple a été établi à partir d'extraits de l'expertise recueillie dans le cadre de l'élaboration du logiciel GESTRISK[1]).

3. LE NIVEAU "META"

L'éditeur de méta-modèles met en oeuvre une variante du formalisme entité/association. Les acteurs qui interviennent dans la phase de méta-modélisation peuvent décrire des méta-individus et des méta-relations[2]. Méta-individus et méta-relations sont caractérisés par des rubriques typées (méta-rubriques). Les méta-relations sont porteuses de cardinalités. De plus, il est possible de définir des contraintes d'intégrité portant sur les futurs modèles conceptuels. On peut résumer la nature de ces contraintes par les catégories suivantes: *intra-individus, intra-relations, inter-individus et inter-relations.*

Le processus de méta-modélisation se déroule en trois étapes. Il faut d'abord choisir le méta-modèle d'un type d'interprète cible (SGBDR, SGBDOO, etc.). Il est ensuite nécessaire de construire le méta-modèle du domaine. Il faut enfin écrire les règles de mise en correspondance entre les éléments de représentation de chacun des méta-modèles.

3.1. L'interprète

Un méta-modèle d'interprète est réalisé indépendamment de toute application. Il représente une abstraction de l'interprète considéré, décrite dans le formalisme implémenté dans l'éditeur de méta-modèles.

La figure 3.1 montre une partie du méta-modèle de l'interprète SQL (en tant que norme). En plus des entités comme RELATION et ATTRIBUT, et des associations telles que POSSEDE, le méta-modèle contient aussi des expressions de contraintes d'intégrité qui ne sont pas représentées graphiquement. Un exemple de contrainte sur l'interprète SQL serait par exemple que l'existence de liens de type AFFECTE entre des instances des entités INSERTION et VALEUR exclue l'existence de liens de type AFFECTE entre des instances de INSERTION et de SELECTION. Cela traduit le fait qu'une insertion de tuple se fait, soit

---

[1] *GESTRISK: logiciel d'aide à la gestion des riques industriels - développé par Y. Gorlin dans le cadre de l'association pour le management des risques et des assurances dans l'entreprise (AMRAE) - diffusé par la société EFFISOFT.*

[2] *Dans la suite de l'article, nous utiliserons indifféremment les termes "entité" et "méta-individu", ainsi que "association" et "méta-relation". Les dénominations préfixées par méta font référence à l'interface d'édition de méta-modèles.*

directement par affectation de valeurs, soit par affectation du résultat d'une sélection.

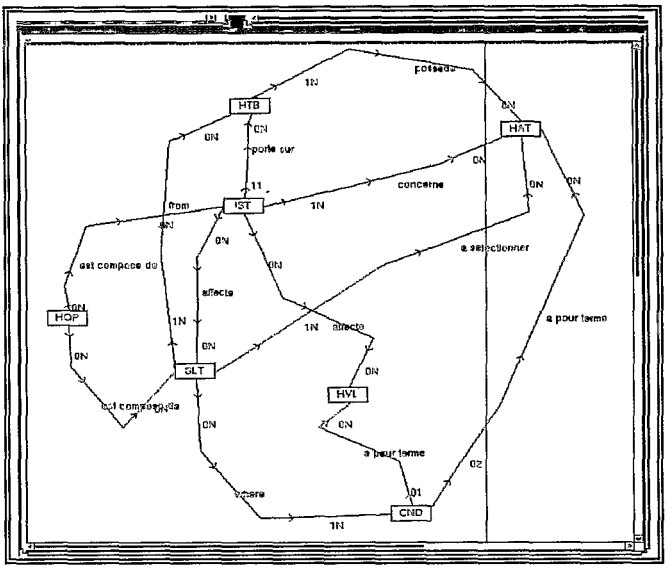

figure 3.1: Un extrait du méta-modèle de l'interprète SQL où:

HTB signifie relation, HAT attribut, HVL valeur, HOP opération ou transaction, IST insertion, SLT sélection et CND condition.

Comme nous l'avons déjà dit, le méta-modèle de l'interprète est indépendant de toute application et de tout domaine. Pour une application donnée, il est possible de choisir un méta-modèle d'interprète dans une bibliothèque. Ce choix est dicté par les moyens informatiques disponibles et par d'autres considérations telles que les performances attendues ou la politique de l'organisation. La bibliothèque contient également, pour le méta-modèle d'un interprète choisi, les transformateurs vers les différentes implémentations de l'interprète. Ces implémentations correspondent aux divers logiciels existants sur le marché. Par exemple, pour SQL, la bibliothèque contient des ensembles de règles permettant d'adapter le standard SQL aux différentes versions commercialisées (ORACLE, INGRES etc.).

Notre outil étant implémenté en Smalltalk 80 [Goldberg89], le méta-modèle de l'interprète est compilé en un ensemble de classes d'objets dont nous présenterons l'usage dans la description du niveau *modélisation* (cf. § 4.1).

3.2. Le domaine

D'après [Loucopoulos92], l'intérêt principal d'un modèle conceptuel est de permettre la compréhension d'un domaine d'application particulier. Cela implique naturellement la communication entre les concepteurs et les utilisateurs de ce domaine. Du fait que la communication a pour support le modèle conceptuel, les concepts que contient ce dernier doivent être en adéquation avec le *milieu* dans lequel le système d'information est utilisé, plutôt qu'en rapport avec sa conception ou son implémentation.

Il est clair que cette communication est d'autant plus efficace que le formalisme de modélisation intègre les notions du domaine.

L'élaboration du méta-modèle d'un domaine n'est pas une tâche aisée. Chaque spécialiste peut avoir sa propre conception des choses. Néanmoins, si l'on considère que le spécialiste détient des connaissances théoriques admises par sa communauté, les risques de subjectivité se trouvent atténués.

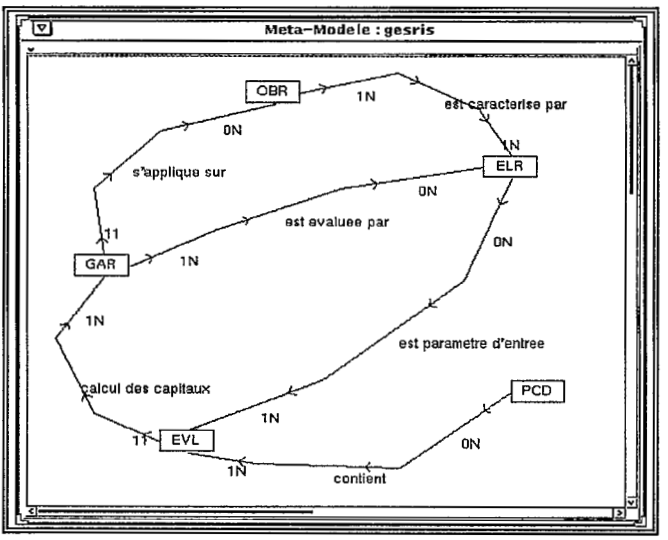

figure 3.2: un extrait du méta-modèle du domaine des assurances où:

OBR désigne OBJET DU RISQUE, ELR ELEMENT DU RISQUE, GAR GARANTIE, EVL EVALUATION DE GARANTIE et PCD PROCEDURE

Pour notre exemple (figure 3.2), le méta-modèle va contenir les concepts généraux manipulés dans le monde des assurances tels que OBJET DU RISQUE sur lequel porte une police d'assurance (une voiture, un local), ELEMENT DU RISQUE qui peut être matériel (une partie de la voiture) ou définissant un usage (déplacement), GARANTIE et EVALUATION D'UNE GARANTIE.

La compilation du méta-modèle du domaine engendre également un ensemble de classes Smalltalk pour supporter la définition de modèles conceptuels, et un ensemble d'éditeurs pour construire ces modèles. Les contraintes d'intégrité exprimées sont regroupées dans une base de règles servant à la validation des modèles.

3.3. Spécification de la transformation

La mise en correspondance entre les concepts du domaine et ceux de l'interprète est une étape importante. En effet, elle est nécessaire à la génération des textes interprétables (programmes) à partir des modèles conceptuels. Cette transformation constitue la première étape vers la génération du code: elle permet de passer d'une représentation en termes du domaine à une représentation en termes de l'interprète (cf. § 4.2).

Dans l'état actuel de nos travaux, nous envisageons de spécifier la transformation par l'application des étapes suivantes:

1- recenser chaque élément de représentation du méta-modèle du domaine directement représentable par un élément du méta-modèle de l'interprète (par exemple un objet du risque correspond à une relation).

2- pour tout autre élément de représentation du méta-modèle du domaine, identifier une configuration d'éléments du méta-modèle de l'interprète pouvant supporter la même information.

L'application de ces deux premières étapes doit permettre d'écrire une base de règles de transformation. Plusieurs règles peuvent éventuellement s'appliquer à un même élément de représentation. Cela introduit la nécessité d'organiser la base de règles en définissant un processus de contrôle de la manière suivante:

3- regrouper en paquets les règles relatives à un même élément.

4- ordonnancer l'application de la base de règle en séquençant précisément les paquets.

Pour spécifier la transformation, l'informaticien et le spécialiste du domaine disposent d'un éditeur de règles NéOpus[1] [Pachet92]. Cet éditeur permet à la fois d'écrire les règles dans une base (étapes 1 et 2) et de spécifier le contrôle de cette base (étapes 3 et 4).

---

[1] *NéOpus est un système de règles implémenté sous Smalltalk. Par conséquent, il manipule naturellement les objets de cet environnement.*

Sur notre exemple, la correspondance établie entre les éléments OBJET DU RISQUE et RELATION se résume à l'écriture de la règle suivante:

SI une instance de OBJET-DU-RISQUE est rencontrée dans le "modèle du domaine"

ALORS créer une instance de RELATION portant le même nom dans le "modèle de l'interprète"

4. LE NIVEAU MODELE

Dans la phase de modélisation, les concepteurs établissent leurs modèles conceptuels grâce aux éditeurs générés à partir de l'étape de méta-modélisation. Une fois le modèle conceptuel d'une application édité, il est transformé en code exécutable en deux temps. Cette transformation globale est d'abord réalisée grâce à l'application des règles résultant de la mise en correspondance entre les éléments des deux méta-modèles. Elle permet ainsi de générer une représentation *intermédiaire*: "au sens de l'interprète". Elle permet ensuite de transformer cette représentation intermédiaire en *code exécutable*.

4.1. Les éditeurs

L'étape d'édition, ou de construction, de modèles conceptuels consiste à utiliser un éditeur spécialisé. Cet éditeur est en fait généré à partir des descriptions faites au niveau "méta", en particulier celles relatives au méta-modèle du domaine, et celles relatives aux représentations graphiques choisies pour les éléments de représentation mis en oeuvre dans ce méta-modèle.

La figure 4.1 montre une vue d'un éditeur ouvert sur un extrait de modèle conceptuel d'établissement de polices d'assurance automobile. Les représentations graphiques (pictogrammes) ont été associées aux éléments du méta-modèle à l'aide d'un éditeur particulier du niveau méta: le *méta-éditeur graphique*.

L'utilisation des éléments du méta-modèle est donc faite à travers une interface spécifique à un type d'utilisateur. En effet, suivant le métier, ou le profil de l'utilisateur à qui l'éditeur est destiné (le concepteur), les représentations peuvent être adaptées. Au niveau de l'implémentation, la construction d'un modèle conceptuel se traduit par l'instanciation des classes Smalltalk qui sont engendrées par compilation du méta-modèle du domaine.

En résumé, un même méta-modèle permet de construire plusieurs éditeurs de modèles. En effet, notre environnement offre la possibilité d'adapter les représentations pour différents types d'utilisateurs. Par ailleurs, il autorise la

modélisation de plusieurs applications dans un même domaine (pour un même méta-modèle).

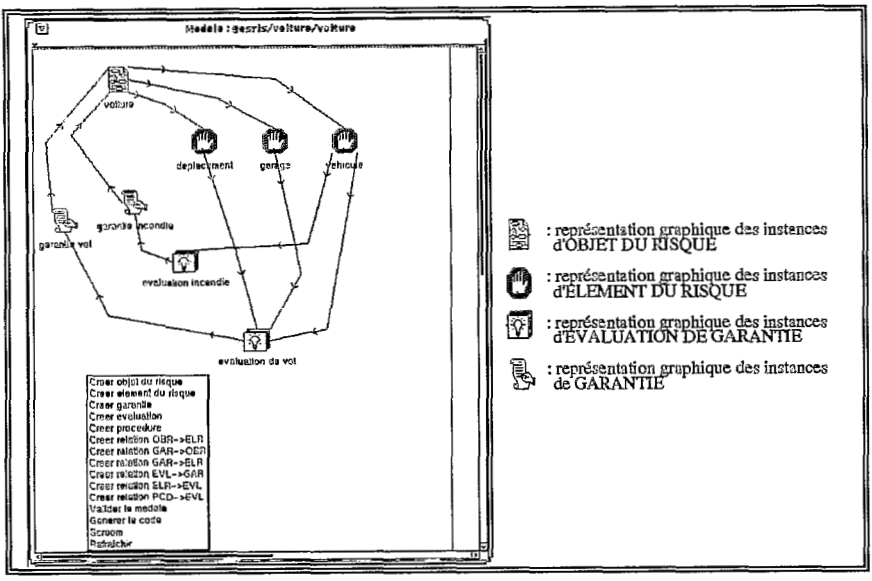

figure 4.1: Extrait du modèle d'établissement des polices d'assurance automobile

4.2. La génération du code

La transformation du modèle conceptuel se fait en deux étapes. Le point de départ est le modèle conceptuel, représenté par un ensemble d'instances Smalltalk. Le résultat est le code du programme, généré pour l'interprète choisi.

La première étape consiste à créer un ensemble d'instances des classes générées par la compilation du méta-modèle de l'interprète. Cette transformation est faite grâce aux règles de mises en correspondance définies au niveau *méta*. Nous appelons le modèle ainsi obtenu *modèle intermédiaire*. Ce dernier pourrait correspondre au modèle logique utilisé dans certaines méthodologies.

La deuxième étape concerne la transformation du modèle intermédiaire en programme. Le module chargé de cette transformation est construit en même temps que le méta-modèle de l'interprète (cf. § 3.1).

Cette transformation en deux temps, est intéressante dans le sens où elle permet de dissocier la *description logique* d'un interprète de sa *description physique*. On peut, par exemple, implémenter une application sous différents SGBD, sans modifier son modèle conceptuel, ni revoir les mises en correspondances définies au niveau "méta", ni même écrire une seule ligne de

code. Pour cela il suffit, dans notre environnement, d'associer le bon générateur de code au méta-modèle de l'interprète choisi. Ceci confère aux applications générées grâce à notre environnement l'importante propriété de portabilité[1].

CONCLUSION

L'orientation initiale de notre travail était de construire un outil de prototypage rapide d'applications de gestion [Krief92]. Après un certain nombre d'expériences, nous nous sommes rendu compte que la conception est plus aisée quand le méta-modèle est « orienté domaine » plutôt que quand il représente un formalisme universel.

Notre objectif actuel est de diviser la modélisation conceptuelle en deux étapes: une modélisation des concepts généraux du domaine (méta-modèle) suivie d'une particularisation de ces concepts pour une application donnée. Cette séparation est rendue possible par l'existence de deux acteurs dont nous différencions clairement les profils. Ceux-ci possèdent des connaissances complémentaires et interviennent généralement dans la construction des systèmes d'information. Ce sont:

- le spécialiste du domaine qui possède des connaissances théoriques, publiques, admises par la communauté du domaine,
- l'expert qui détient des connaissances privées, fruits d'un nombre important d'années d'expérience.

L'élicitation du niveau méta permet de réutiliser une partie - ou la totalité - des concepts décrits par le spécialiste du domaine, pour plusieurs applications différentes.

Pour étendre le champ d'application de notre environnement à la conception globale des systèmes d'information, nous travaillons principalement sur les deux points suivants: 1. Nous travaillons à la construction d'une bibliothèque de modules d'interprètes (SGBD, systèmes de simulations, langages de programmation, etc.). Ainsi pour une application donnée, on pourra combiner ces modules de manière à prendre en compte tous les types de traitements possibles. 2. A partir d'expériences diverses, nous essayons de mettre au point une démarche et un outil d'acquisition des connaissances relatives à la première étape de transformation de modèles. De cette manière,

---

[1] *La portabilité en question est possible entre des SGBD différents. Elle est même rendue possible entre des plates-formes différentes (stations SUN, PC, Mac) grâce à l'utilisation de l'environnement Smalltalk 80.*

nous pensons pouvoir générer efficacement des bases de règles pour réaliser les transformations.

Outre sa fonction de modélisation orientée domaine, notre environnement peut servir de générateur d'AGL. Ainsi nous pouvons générer des outils tels que INTERSERM [Prud-homme92] avec un formalisme initial de type Entité-Association étendu, et un formalisme cible de type objet.

BIBLIOGRAPHIE

[Atzeni93] P. Atzeni R. Torlone, A Meta Model Approch for the Management of Multiple Models and the Translation of Schemes; Information Systems, vol. 18 N° 6, 1993.

[Bennett85] J. S. Bennett, ROGET : A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system, Journal of Automated Reasoning, 1, 1985.

[Booch90] G Booch, Object Oriented Design with Applcations, Benjamin/Cummings, Menlo Park, CA, 1990.

[Bouzeghoub84] M. Bouzeghoub, The Functional Query Language MORSE; Trends and Application Conf. on Databases, Gaithersburg (MD, USA) IEEE-NBQ; 1984.

[Bouzeghoub88] M. Bouzeghoub F. Pasquer D. Richard, Bases de Données Relationnelles: de la modélisation des schémas à l'exploitation de la base; JIIA; 1988.

[Chen76] P.P. Chen, The entity relationship model - Towards a unified view of data; ACM TODS, vol. 1, N°1, 1976.

[Coad90] P. Coad E. Yourdon, Object Oriented Analyses, Yourdon Press, Prentice HAll, Englewood Cliffs, NJ, 1990.

[Cooper91] R. Cooper, Configurable Data Modeling Systems; Entity-Relationship Approch: The Core of Conceptual Modeling, North-Holland, 1991.

[Eshelman88] L. Eshelman, MOLE : A knowledge-acquisition system for cover-and-differentiate systems, In Automating Knowledge Acquisition for Expert Systems, S. Marcus (ed), Boston, 1988.

[Goldberg89] A. Goldberg D. Robson, Smalltalk-80: The language, Addison Wesley Publishing Company, Reading, MA, 1989.

[Hull87] R. Hull R. King, Semantic Data Modeling: Survey, Applications and Research Issues; ACM Computing Surveys, 19-3, 1987.

[Jeulin89] P. Jeulin M. Khlat L. Wilhem, GRAPHTALK, GQL et GKNOWLEDGE: Des techniques d'Intelligence Artificielle au service d'un environnement de Génie Logiciel; rapport RANK XEROX FRANCE; 1989.

[Klinker91] G. Klinker et al., Usable and reusable programming constructs; Knowledge Acquisition, 3; 1991.

[Krief92] P. Krief, Utilisation des langages objets pour le prototypage; Masson; 1992.

[Linster92] M. Linster, Linking modeling to make sense and modeling to implement systems in an operationnel en vironnement. In Wetter, T., Althoff, K.D., Boose, J., & Gaines, B., editors, Current developments in knowledge acquisition: EKAW92, volume 509 of Lecture Notes in AI. Springer-Verlag.

[Loucopoulos92] P. Loucopoulos R. Zicari, Conceptual Modeling, Databases, and Case: An integrated view of information systems developement, John Wiley & Sons, Inc., 1992.

[Marcus89] S. Marcus J. McDermott, SALT : A knowledge acquisition tool for propose-and-revise systems, Artificial Intelligence, 39, 1989.

[Morejon91] J. Morejon & al, GraphOR: A Meta Design Tool; Entity-Relationship Approch: The Core of Conceptual Modeling, North-Holland, 1991.

[Musen89] M. A. Musen, Automated Support for Building and Extending Expert Models, Machine Learning, 4, 1989.

[Musen93] M. A. Musen S. W. Tu, Problem-Solving Models for Generation of Task-Specific Knowledge-Acquisition Tools, In Knowledge-Oriented Software Design, J. Cuena (ed), Amsterdam, 1993.

[Newell82] A. Newell, The Knowledge Level; Artificial Intelligence, North-Holland, 1982.

[Pachet92] F. Pachet, Représentation de connaissances par objets et règles: le système NéOpus, Thèse de doctorat de l'université de Paris VI, 1992.

[Pellaumail86] P. Pellaumail, La méthode AXIAL; Editions d'Organisation, Paris, 1986.

[Prud-homme92]. B. Prud-homme, R. Missaoui, R. Godin. INTERSEM : une interface sémantique orientée-objet. ICO, Vol. 4, N. 1-2, 1992.

[Rolland88] C. Rolland O. Foucaut G. Benci, Conception des systèmes d'information. La méthode REMORA, Editions Eyrolles, Paris, 1988.

[Tardieu85] H. Tardieu A. Rochfeld R. Colleti, Méthode MERISE: Démarche et pratiques; Edition d'Organisation; 1985.

[Walther92] E. Walther H. Eriksson M. A. Musen, Plug-and-Play: Construction of Task-Specific Expert-System ShellsUsing Sharable Context Ontologies; Proceedings of the AAAI Workshop on Knowledge Representation Aspects of KnowledgeAcquisition, San Jose, CA, 1992.