

# Apports de la modélisation algébrique pour la représentation de connaissances par objets : illustration en AROM

Bogdan Moisuc<sup>1</sup>, Philippe Genoud<sup>2</sup>, Danielle Ziébelin<sup>2</sup>, Jérôme Gensel<sup>1</sup> et Cécile Capponi<sup>3</sup>

<sup>1</sup> Laboratoire LSR - IMAG, BP 72, 38402 Saint Martin d'Hères cedex, Prénom.Nom@imag.fr

<sup>2</sup> UJF projet Hélix INRIA-RA, 655 avenue de l'Europe, Montbonnot, 38334 Saint Ismier cedex, Prénom.Nom@imag.fr

<sup>3</sup> Laboratoire d'Informatique Fondamentale, 39 avenue. Joliot-Curie, 13453 Marseille cedex 13, capponi@cmi.univ-mrs.fr

## Résumé

AROM est un système de représentation de connaissances reposant, à l'image des diagrammes de classes d'UML, sur deux types d'entités de modélisation complémentaires : les classes et les associations. Il intègre un langage de modélisation algébrique (ou LMA) qui sert de support à différents mécanismes d'inférence. Ce langage permet l'écriture d'équations, de contraintes, et de requêtes, impliquant les instances des classes et des associations. La présence d'un module de types en AROM permet d'étendre l'ensemble des types (donc des valeurs et des opérateurs) supportés par le LMA. A travers la description du LMA d'AROM, cet article souligne l'apport d'un langage de modélisation algébrique pour un système de représentation de connaissances tant au niveau de la déclarativité qu'en termes des inférences possibles.

**Mots clés :** représentation de connaissances, modèles à objets, langage de modélisation algébrique, types extensibles.

## 1 Introduction

Issus des langages de *frames* [16] et voisins des logiques de description [4] et des langages à ontologies [22], les Systèmes de Représentation de Connaissances par Objets (ou SRCO) ([8], [20]) décrivent, organisent, et stockent des connaissances en s'appuyant sur des principes généraux du paradigme *objet* (notions de classe, d'instance, de hiérarchie de spécialisation, d'héritage...). Ils disposent de divers mécanismes d'inférence (héritage, méthodes, filtrage, classification...) permettant de compléter et de rendre explicites les connaissances. Parmi les SRCO, AROM [23] se distingue par deux structures de représentation centrales et complémentaires – les classes et les associations, à l'image des entités utilisées dans les diagrammes de classe d'UML.

Issus de la Recherche Opérationnelle, les langages de modélisation algébrique ([3][14][19][21]), quant à eux, proposent de formuler, de manière formelle, dans un langage proche des notations généralement employées en mathématiques et adapté aux structures de données cou-

ramment utilisées en programmation (par exemple, les tableaux), des systèmes d'équations ou d'inéquations, des contraintes, ou bien encore des requêtes. Ils permettent donc d'énoncer des connaissances opératoires qui, parcourues et validées par un analyseur sont envoyées à un solveur pour exécution ou résolution.

Une autre caractéristique originale d'AROM est la présence d'un langage de modélisation algébrique, appelé LMA, qui permet de décrire des contraintes, de formuler des requêtes ou bien de calculer des valeurs d'attributs à l'aide d'équations algébriques. Dans sa première version, le LMA permettait de construire des expressions algébriques intégrant les types de base gérés par AROM, les valeurs et les opérateurs associés. Or, AROM dispose dans sa nouvelle version d'un module de types extensibles.

Nous montrons dans cet article comment le LMA d'AROM peut être étendu avec de nouveaux types et opérateurs. Nous présentons les principes généraux du LMA extensible d'AROM et nous donnons un aperçu des possibilités offertes par ce langage pour la modélisation de domaines d'application particuliers et complexes tels que la géomatique et la bioinformatique.

Le papier est organisé de la façon suivante : la section 2 rappelle les principes généraux de la représentation de connaissances par objets. La section 3 est une introduction aux langages de modélisation algébrique. La section 4 décrit le système AROM. La section 5 présente les principes du LMA d'AROM et de son extensibilité. Des exemples illustrent son intérêt en termes de déclarativité et de support à différents mécanismes d'inférence.

## 2 Représentation de Connaissances par Objets

### 2.1 Principes

Parmi les différents paradigmes de représentation de connaissances, la Représentation de Connaissances par Objets (ou RCO) permet de modéliser la connaissance relative à un domaine d'application particulier en

s'appuyant sur des entités de représentation appelés *objets*. Parmi ces objets, en général sont distinguées les *classes* et les *instances* (à l'exception des langages à *prototypes* où ces entités sont confondues). Une classe décrit un concept, une famille d'individus en donnant l'ensemble des propriétés ou *attributs* qui les caractérisent. Une instance est un individu effectivement représenté. On distingue l'*intention* d'une classe (l'ensemble des descriptions de ses *attributs*), de son *extension* (l'ensemble de ses instances). Un attribut est décrit par un nom et un ensemble de *facettes*. Les systèmes de RCO (ou SRCO) se distinguent par les facettes qu'ils proposent. Les *facettes de typage* sont essentielles, elles précisent le type et l'ensemble (appelé *domaine*) des valeurs possibles de l'attribut, ou, dans le cas d'un attribut multivalué, le nombre de valeurs possibles. Le type d'un attribut peut être défini parmi ceux (*integer, boolean, string...*) présents dans le langage de programmation d'implémentation utilisé. Il peut être élaboré à partir de constructeurs (comme *set-of* ou *list-of*). Il peut également faire référence à une classe présente dans la base de connaissances. Dans ce cas, l'attribut modélise le lien orienté d'une relation binaire quelconque (ou parfois distinguée, comme la composition) établie entre la classe de l'attribut et la classe-domaine désignée par le type. Les *facettes d'inférence* décrivent un moyen d'obtenir la valeur d'un attribut lorsque celle-ci n'est pas présente. Valeur par défaut, *attachement procédural* (qui invoque une méthode), filtrage (qui associe à la valeur le résultat d'une requête), passage de valeur (qui affecte à un attribut la valeur d'un autre attribut) sont les principaux mécanismes d'inférence désignés par des facettes dans les SRCO. Également, des facettes *réflexes* permettent d'activer une action avant ou après l'accès, la modification, l'ajout ou le retrait d'une valeur. Enfin, des facettes secondaires permettent, par exemple, d'associer une description html à un attribut.

Au sein d'une base de connaissances, les classes sont organisées en hiérarchies de spécialisation qui forment, soit des arbres (chaque classe a alors au plus une super-classe directe), soit des graphes (chaque classe peut avoir plusieurs super-classes directes). La spécialisation est une relation d'ordre partiel qui repose sur l'affinement des intentions. Ainsi, chaque description d'un attribut dans une classe soit *i*) est identique à la description de ce même attribut dans la super-classe, *ii*) a pour type un sous-type de la description de ce même attribut dans la super-classe, *iii*) correspond à la définition d'un nouvel attribut. De fait, en termes d'extensions de classe, la spécialisation équivaut à l'inclusion ensembliste : les instances d'une classe sont instances de ses super-classes. Sur les hiérarchies de spécialisation se greffe un mécanisme d'héritage permettant de factoriser les informations. Ces hiérarchies sont également le support d'un mécanisme d'inférence central dans les RCO : la *classifi-*

*cation* qui revêt deux formes. La *classification d'instance* vise à « faire descendre » une instance depuis sa classe d'attachement vers des sous-classes plus spécialisées, en activant au besoin les mécanismes d'inférence nécessaires à la détermination des valeurs d'attributs manquantes. Il s'agit là d'un processus d'identification utile, par exemple, pour l'aide au diagnostic. La *classification de classe*, quant à elle, détermine la ou les positions d'insertion d'une classe dans la hiérarchie de spécialisation par recherche des super-classes les plus spécialisées et des sous-classes les plus générales. Outre les vérifications de type associées à toute affectation de valeur à un attribut, certaines RCO intègrent des mécanismes de maintien de cohérence visant à propager les conséquences de tout changement d'un objet de la base de connaissances aux objets attenants. Des SRCO ont été utilisés dans divers domaines d'application (bioinformatique, géographie, biologie, traitement du signal...). Parmi les principaux représentants des SRCO, citons KRL, FRL, SRL, SHIRKA, YAFOOL, FROME, ou encore TROPES (pour une revue voir [5] et [15]).

## 2.2 Langages et Systèmes

Historiquement, la RCO trouve son origine dans les réseaux sémantiques [23] et les *frames* de Minsky [16]. Elle est apparue à travers des langages de représentation dits *hybrides* [15] qui empruntent aux langages de programmation orientés objets la distinction entre *classe* et *instance*, l'organisation des classes par une relation de généralisation/spécialisation, un mécanisme d'héritage de propriétés ou *attributs* entre classes, ou encore l'encapsulation de méthodes. Parallèlement, du souci de doter ce type de langage de représentation d'une sémantique bien définie, sont nées les *logiques de description* (ou LD) [1]. Dans cette famille prédominante de langages de représentation de connaissances, l'accent est mis sur la formalisation par une sémantique dénotationnelle et sur l'évaluation de la complexité et la décidabilité de la *subsumption*. Cette relation de spécialisation entre termes (concepts ou rôles) est interprétable comme l'inclusion ensembliste. La plupart des mécanismes d'inférence disponibles, notamment la classification, reposent sur elle. Face à cette position hégémonique des LD, que prolonge aujourd'hui OWL [22], standard de description d'ontologies pour le Web sémantique, les systèmes de RCO, dont le système AROM présenté section 4, ont tenté de résister par des efforts de formalisation (notion de *systèmes classificatoires* [5]), et en continuant à privilégier la puissance d'expression (déclarativité), les mécanismes d'inférence et l'utili(sabilité) résultante [20].

### 3 Langage de Modélisation Algébrique

Pour représenter des connaissances de calcul comme la résolution d'une équation, la vérification d'une propriété, la satisfaction d'une contrainte, le calcul d'une inférence impliquée dans une simulation, on distingue essentiellement deux approches : l'une qualifiée de déclarative, l'autre d'algorithmique.

Les principales approches déclaratives ont été développées par des concepteurs de langages algébriques, de programmation logique ou de programmation par contraintes. Ces approches considèrent que le concepteur de la base de connaissances fait constamment un parallèle entre les descriptions fonctionnelles et physiques des connaissances du domaine d'application. Un modèle algébrique décrit les relations établies par le concepteur de la base de connaissances entre le domaine fonctionnel (le *comment*) et le domaine physique (le *quoi*). Chacun de ces domaines est caractérisé par des paramètres spécifiques intervenant dans l'expression mathématique. Les approches déclaratives reposent sur l'utilisation d'axiomes, de formules ou d'équations formées de variables et d'opérateurs, susceptibles de mener à une «bonne modélisation». Le support de cette modélisation est assuré par un Langage de Modélisation Algébrique (ou L-MA).

Par opposition, les approches algorithmiques se caractérisent par un ensemble de tâches distinctes. A chaque tâche est associé un ensemble d'étapes permettant de fournir les résultats indispensables au déclenchement de la tâche suivante. Une démarche algorithmique tente d'identifier ou d'imposer un processus déterministe constitué d'activités ordonnées jalonnées d'étapes de validation.

#### 3.1 Définition

La difficulté d'une approche déclarative réside dans l'élaboration, l'analyse, la compréhension et la formulation des expressions algébriques ainsi que dans le choix de leur rattachement aux classes et aux instances qui composent le modèle de connaissances.

Pour faciliter cette étape de modélisation, des systèmes de modélisation mettant en oeuvre des techniques soit d'optimisation, soit de simulation, soit de satisfaction de contraintes, ont été conçus. Pour renforcer la déclarativité l'accent est mis sur la capacité d'expression du langage

1. Il est primordial, de disposer d'un moyen d'exprimer le modèle mathématique explicitement et une fonction de vérification, qui permet de vérifier que le modèle est valide. Il pourra adjoindre une description textuelle du modèle permettant d'aider l'utilisateur final de la base de connaissances.

2. L'expression algébrique doit être parfaitement intégrée dans le modèle de connaissances. Le langage algébrique peut inclure différentes représentations (schématiques, graphiques ou en langage naturel...) et doit être intégré dans le langage de représentation de connaissances.

3. Outre les capacités d'expression précédemment décrites, le langage algébrique propose un support opérationnel composé d'un éditeur de formules, d'un interpréteur et d'un mécanisme de résolution ou solveur.

4. L'interpréteur de formules doit vérifier que la formule est bien établie et faire appel à une fonction de simplification pour en améliorer la compréhension de l'utilisateur. Des règles de réécriture de simplification des formules algébriques peuvent être proposées afin d'éliminer les redondances ou d'améliorer la lisibilité de la formule.

5. L'interpréteur doit permettre d'éliminer les erreurs de calcul dans une expression algébrique et, parfois, dans le modèle lui-même. La correction de modèles s'effectue à l'aide d'outils d'analyse qui détectent l'incohérence, le bouclage, l'inconsistance.

6. Dans bon nombre d'applications, on recueille, on génère, et on utilise, un large volume de données. La présentation des résultats est une fonction importante qui peut inclure l'interrogation interactive au moyen de l'interpréteur de formules, et la production de fichiers internes pour une analyse ultérieure ou une réutilisation des résultats par d'autres outils.

7. La gestion de versions. En règle générale, on résout plusieurs versions d'un même modèle, chacune représentant une hypothèse. La gestion des versions consiste à organiser une collection de versions connexes, en reprenant les parties identiques. La grande quantité d'informations produite de cette manière doit être traitée et filtrée afin que l'analyse puisse facilement permettre d'en tirer des conclusions discriminantes.

#### 3.2 Représentants

Les études nombreuses (parmi lesquelles [1], [11],[26]) montrent la diversité d'usage des portant sur les langages de modélisation algébrique. Certains sont dédiés à des domaines mathématiques particuliers comme la programmation linéaire, les statistiques, la résolution d'équations... D'autres portent spécifiquement sur les logiciels qui implantent des langages génériques. Ces langages génériques sont plus ou moins complexes et portent sur différents aspects : la modélisation algébrique (AMPL [3], GAMS [10], AIMMS [1]), la programmation logique (MPL [19]), la programmation par contraintes (OPL [21]), ou encore des équations, les contraintes numériques et surtout symboliques (LINGO [14]).

### 3.3 LMA et RCO : un couple légitime

Le processus de représentation des connaissances consiste tout d’abord, à identifier les entités du domaine que l’on veut distinguer. Ces entités sont désignées par des classes, auxquelles sont liées leurs réalisations, qui sont les instances. Puis on définit une représentation abstraite des relations entre ces classes, afin de produire un ensemble de classes et d’associations, exprimées dans un langage formel. Le résultat de cette étape est souvent appelé « modèle »

L’étape suivante consiste à définir une représentation opérationnelle du modèle. Pour cela, on produit un ensemble d’algorithmes, de schéma logique et de déduction s’appuyant sur des structures de données. Cette étape transpose le modèle composé d’entité-relation, conçu à l’étape précédente, dans un contexte particulier de résolution ou de déduction.

Représenter les entités-relations et leurs réalisations par des classes et des instances au sein du RCO d’une part, et utiliser un langage algébrique, pour établir l’implication des classes/instances-relations dans le processus de résolution d’autre part, permet d’étendre l’expressivité du RCO.

Lier un RCO et un LMA permet d’utiliser des entités mathématiques telles que des ensembles et des index, des paramètres, des variables, des contraintes, et des fonctions directement au niveau des attributs des classes ; mais également, de supporter le calcul des opérations tels que l’union, la différence, l’intersection, le produit cartésien ... à ce même niveau, plutôt que de reporter la déduction sur un mécanisme d’inférence. Le modèle résultant y gagne en déclarativité. Dans la suite du document nous illustrons ce couplage par l’intermédiaire du SRCO AROM et son LMA.

## 4 AROM

### 4.1 Modèle AROM

AROM [23], acronyme de Allier Relations et Objets pour Modéliser, est un système de représentation de connaissances qui s’inscrit dans la lignée des SRCO (Système de Représentation de Connaissances par Objets). Il reprend les grands principes : distinction entre classes et instances, spécialisation de classes en une hiérarchie d’héritage, présence de facettes de typage et d’inférence (valeur par défaut, attachement procédural, etc.). Cependant, il se distingue de ses prédécesseurs dans sa manière de représenter les relations entre objets : AROM interdit l’utilisation d’attributs liens (attributs typés par une classe) et impose de modéliser de manière explicite les relations entre objets à l’aide d’une seconde entité de représentation complémentaire de la structure de classe :

*l’association.* Une association AROM est une entité nommée qui représente un ensemble de  $n$ -uplets d’objets appartenant aux extensions des  $n \geq 2$  classes qu’elle relie. Elle est décrite par ses rôles (connexion entre l’association et l’une des classes reliées) à chacun desquels est associée une multiplicité (similaire aux cardinalités d’UML) et un ensemble de variables (identiques aux variables des classes) définissant les propriétés associées aux  $n$ -uplets. Un héritage des rôles, variables et facettes est possible entre associations au travers d’une hiérarchie de spécialisation analogue à la hiérarchie de spécialisation entre classes. La figure (FIG. 1) présente, en utilisant la notation graphique d’AROM proche de UML, un modèle AROM représentant sous forme de classes des enseignants et des formations et sous forme d’associations les différentes relations qui les lient.

Des travaux sont actuellement en cours pour étendre le méta-modèle AROM afin d’offrir le support pour différentes sémantiques de relations entre objets, en particulier en ce qui concernent les relations de type Partie-Tout (agrégation, composition, ...) [12].

Une seconde originalité du modèle AROM est l’utilisation d’un langage de modélisation algébrique (LMA) pour définir les variables des classes et associations sous forme d’équations. Ce point sera développé plus en détail dans la partie 5 de cet article.

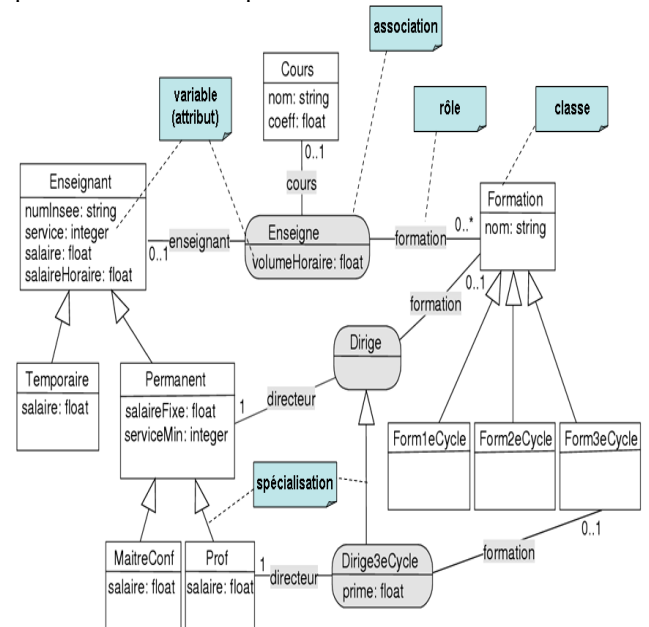


FIG. 1 – Exemple de modèle AROM

### 4.2 Système AROM

La représentation des connaissances en AROM est mise en œuvre au travers d’un environnement réalisé en langage Java. Dans sa version actuelle (AROM2) [5], le système AROM est l’implémentation du modèle de re-

présentation de connaissances : il prend en charge la représentation sous la forme d'objets informatiques des entités du modèle AROM. L'API AROM2, est l'interface de programmation Java de ce système. Au dessus de ce noyau, la plate-forme AROM2 propose un certain nombre de bibliothèques pour l'exploitation des bases de connaissances, chaque bibliothèque étant bâtie sur l'API AROM2 et proposant elle-même sa propre API. Parmi les différents modules ainsi proposés, citons *AROMClassif*, API pour la classification d'objets et de tuples avec propagation [7], *AROMQuery* pour la définition de requêtes sur une base AROM, *XAROM* API pour l'interfaçage de bases AROM avec le format XML, etc (FIG. 2).

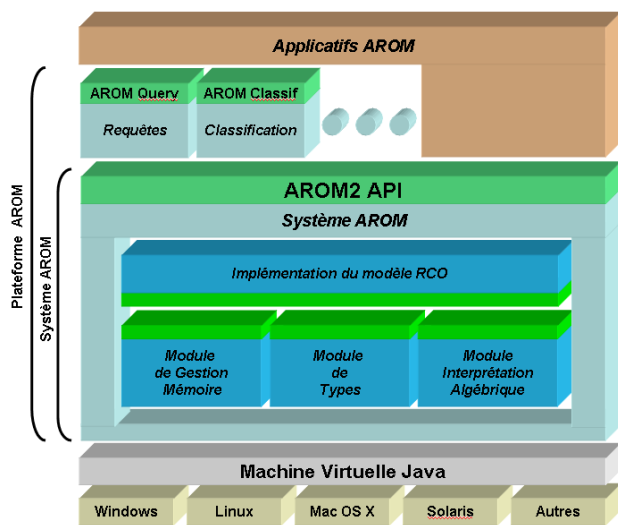


FIG. 2 – Architecture de la plateforme AROM

Le système AROM est lui-même conçu de façon modulaire, chaque module étant en charge d'une fonction précise du système. Le module de gestion mémoire assure l'accès en mémoire aux instances d'objet AROM, le module de types définit l'ensemble des types reconnus dans une base AROM et les opérations possibles sur ces types [6] le module d'interprétation algébrique assure l'interprétation d'équations de Langage de Modélisation Algébrique (LMA) d'AROM. Chaque module ne communique avec les autres modules qu'au travers d'une API interne clairement isolée. De cette manière il est possible de modifier la plateforme AROM en changeant l'implémentation de l'un des modules par une autre implémentation. Cette possibilité a par exemple été exploitée pour le système GénoStar ([9], [11]) dédié à l'analyse exploratoire de séquences génomique. GénoStar utilisant le système AROM pour des bases de connaissances qui impliquent un très grand nombre d'instances, un module de gestion mémoire spécifique permet d'optimiser l'occupation mémoire de la machine virtuelle Java. Un mécanisme de persistance assure le chargement et le déchargement des instances depuis le disque vers la mé-

moire. De même, un module de types particulier, intégrant des types spécifiques à la génomique, est utilisé pour cette application.

La configuration du système AROM2 s'effectue de manière transparente (via des fichiers de ressources) pour les applicatifs AROM. En effet, l'API AROM2 étant constituée uniquement d'interfaces Java et de classes « fabriques », le code des applicatifs AROM ne fait jamais référence aux classes d'implémentation du système. Il est ainsi possible d'exécuter, sans modifications, un même code avec des implémentations différentes du système AROM. Cette nouvelle architecture modulaire et configurable et ces nouvelles bibliothèques, distinguent AROM2 de sa version préalable, la sémantique en elle-même n'ayant pas changé.

## 5 Le LMA d'AROM

### 5.1 Présentation

Le LMA d'AROM vise à définir et manipuler des expressions algébriques contenant des constantes, variables, ou opérateurs de n'importe quel type géré par AROM. Intégrant des identifiants de classes, d'associations, d'instances et de tuples, ces expressions correctement formées doivent permettre d'interroger et d'exploiter les entités stockées dans une base de connaissances.

Pour les types de base AROM (*integer*, *boolean*, *double*, *string*...) et les constructeurs liste et ensemble, le LMA d'AROM fournit : *i*) les *opérateurs arithmétiques de base* (addition, soustraction, puissance, partie entière, etc.), *ii*) les *opérateurs de comparaison* (supérieur, inférieur, différent, etc.), *iii*) les *opérateurs trigonométriques* (sinus, cosinus, tangente, etc.), *iv*) les opérateurs pour des ensembles et des listes (constructeurs d'ensembles et de listes, n-ième élément, union, intersection etc.), *v*) les *opérateurs logiques* (conjonction, disjonction, négation, etc.), *vi*) les *opérateurs de manipulation de chaînes* (partie gauche d'une chaîne, concaténation de chaînes, etc.), *vii*) les *opérateurs itérés* (somme itérée, produit itéré, moyenne, etc.), et des *opérateurs de navigation* (afin de naviguer d'un objet à un autre en suivant les liens des associations).

En AROM, le LMA a trois utilisations principales : il est utilisé pour inférer de nouvelles connaissances, pour maintenir et vérifier la cohérence des connaissances et pour interroger les informations stockées dans une base de connaissances.

Le LMA est employé comme mécanisme d'inférence à travers l'écriture d'équations qui permettent de déduire des valeurs d'attributs à partir d'autres valeurs d'attributs, qui sont soit stockées dans la base de connaissances, soit eux-mêmes calculés en cascade.

Nous illustrons notre propos par une application en géomatique qui permet de gérer le partitionnement des communes en parcelles. Le modèle simplifié de cette application, réalisé à l'aide de l'environnement graphique d'AROM, est présenté (FIG. 3). Les attributs des parcelles relatifs à l'occupation du sol, représentent les proportions respectivement de surface boisée, de surface cultivée et de surface bâtie dans la surface totale de la commune. Si l'on suppose que seuls existent ces trois types d'occupation des sols, il est possible d'inférer, par exemple, la proportion de surface boisée d'une parcelle à partir de ses proportions de surface cultivée et de surface bâtie. L'équation LMA pour le calcul de la proportion de surface boisée est la suivante :

$$\text{SurfaceBoisee} = 1 - \text{SurfaceBatie} - \text{SurfaceCultivee}$$

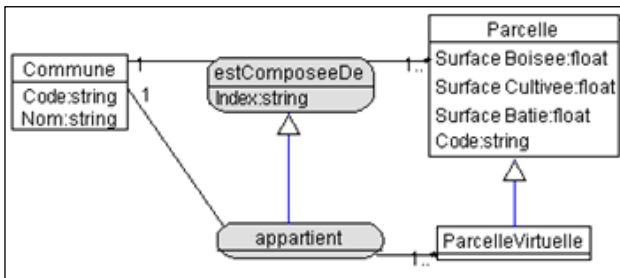


FIG. 3 – Modèle AROM d'une application cadastrale

En second lieu, le LMA permet aussi de vérifier et maintenir la cohérence des données par le biais de contraintes d'intégrité. Pour éviter les erreurs dans l'acquisition et dans la mise à jour des connaissances, il est possible d'introduire une contrainte d'intégrité pour les parcelles virtuelles, du type :  $\text{SurfaceBoisee} < 1$ .

Enfin, il est possible d'utiliser le LMA pour formuler des requêtes traitées par le module AROMQuery. AROMQuery considère une base de connaissances comme un graphe, dont les nœuds correspondent à des objets AROM et dont les arcs correspondent à des tuples AROM. Traiter une requête dans une base de connaissances revient alors à une recherche de chemins dans ce graphe, dont les nœuds finaux sont les résultats attendus de la requête. Le LMA est utilisé ici pour exprimer les conditions de la requête. Par exemple, la requête suivante renvoie les parcelles couvertes à plus de 50% de forêts et qui font partie de communes dont le nom commence par la lettre "A":

```
MaRequete = set (p in Parcelle:
  left(p!EstComposeeDe.Composite.Nom,1)="A"
  and p.SurfaceBoisee>0.5)
```

## 5.2 LMA et types de base

Dans sa version initiale, le système AROM ne supportait qu'un ensemble limité de types pour les attributs des classes et associations (types primitifs entier, chaîne,

booléen, réel, types liste-de ou ensemble-de). Bien que les possibilités de modélisation d'AROM permettent d'implémenter des Types Abstrait de Données (TAD) au travers de classes et d'associations regroupant des attributs de types de bases, cette approche présente rapidement des limites.

D'un point de vue modélisation, cela induit une « pollution » des bases de connaissance par l'introduction d'entités de modélisation de « bas niveau » qui complexifient le modèle. La nécessité de modéliser et manipuler explicitement les TAD sur lesquels se construit la représentation du domaine éloigne l'utilisateur de ses préoccupations de modélisation premières. Elles devraient se situer au niveau du domaine modélisé, et ne pas intégrer des éléments de programmation. Ainsi si nous prolongeons l'exemple précédent en complétant le modèle afin de représenter la géométrie des communes comme un objet à part, lié à l'objet « commune » par une association. De même, l'objet géométrique qui définit la forme de la commune doit être représenté à l'aide d'associations vers des d'objets « élémentaires » qui sont des « points » (FIG. 4).

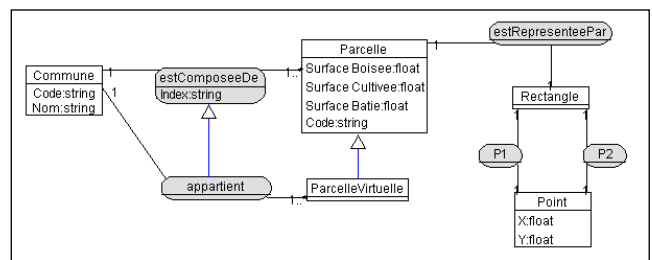


FIG. 4 – Modélisation de parcelles rectangulaires

Le modèle mélange donc à la fois une modélisation liée au domaine des SIG (Systèmes d'Information Géographiques) et une modélisation liée au domaine de la géométrie, ceci est néfaste à sa compréhension.

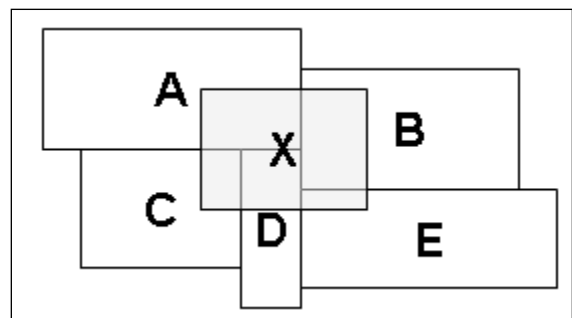


FIG. 5 – Création d'une parcelle virtuelle X

Par ailleurs l'utilisation d'expressions LMA pour représenter des connaissances structurales sur des TAD complexes, si elle demeure possible, peut se révéler d'une lourdeur extrême. Par exemple, pour une parcelle

virtuelle  $X$  (FIG. 5), que les responsables de l'aménagement du territoire envisagent de créer. Supposons qu'ils souhaitent en calculer la surface boisée. L'expression LMA permettant d'inférer la valeur de la surface boisée serait :

```
SurfaceBoisee = sum(parc in this !ap-
partient.composant
    !estComposeeDe.composite:
parc.SurfaceBoisee * (
    if (((this !estRepresentee-
Par.Rectangle !P1.X > parc
!estRepresenteePar.Rectangle
!P1.X and parc !estRepresentee-
Par.Rectangle !P1.X > this !es-
tRepresenteePar.Rectangle !P2.X)
or (this !estRepresentee-
Par.Rectangle !P1.X > parc !es-
tRepresenteePar.Rectangle !P2.X
and parc !estRepresentee-
Par.Rectangle !P2.X > this !es-
tRepresenteePar.Rectangle !P2.X))
and ((this !estRepresentee-
Par.Rectangle !P1.Y > parc !es-
tRepresenteePar.Rectangle !P1.Y
and parc !estRepresentee-
Par.Rectangle !P1.Y > this !es-
tRepresenteePar.Rectangle !P2.Y)
or (this !estRepresentee-
Par.Rectangle !P1.Y > parc !es-
tRepresenteePar.Rectangle !P2.Y
and parc !estRepresentee-
Par.Rectangle !P2.Y > this !es-
tRepresenteePar.Rectangle
!P2.Y)))then (min(this !estRepre-
senteePar.Rectangle !P1.X, parc
!estRepresenteePar.Rectangle
!P1.X)-max(parc !estRepresentee-
Par.Rectangle !P2.X, this !estRe-
presenteePar.Rectangle
!P2.X))*(min(this !estRepresen-
teePar.Rectangle !P1.Y, parc !es-
tRepresenteePar.Rectangle !P1.Y)-
max(parc !estRepresentee-
Par.Rectangle !P2.Y, this !estRe-
presenteePar.Rectangle
!P2.Y))else 0))/ (this !estRepre-
senteePar.Rectangle !P1.X- this
!estRepresenteePar.Rectangle
!P2.X)* (this !estRepresentee-
Par.Rectangle !P1.Y- this !estRe-
presenteePar.Rectangle !P2.Y)
```

Cette expression permet de faire la somme des surfaces boisées dans chaque aire d'intersection entre la parcelle virtuelle (désignée ici par le mot clé *this*) et les parcelles

réelles de la commune. Il est nécessaire de vérifier d'abord si une intersection a lieu (bloc *if*), si c'est le cas l'aire de la surface d'intersection est calculée (le bloc *then*), sinon on lui affecte la valeur 0 (le bloc *else*). Puis pour obtenir la proportion de surface boisée, la somme obtenue est divisée par l'aire totale de la parcelle virtuelle.

Cette complexification du modèle, outre une perte d'expressivité et de déclarativité, peut également avoir un impact négatif sur les performances du système. Le fait de représenter des TAD complexes à l'aide de classes et d'associations multiplie de manière significative le nombre d'objets et de tuples stockées dans les bases de connaissances. De même, l'évaluation des expressions LMA complexes peut se révéler trop coûteuse pour être utilisable par des applications réelles dont l'ensemble des données dépasse quelques centaines d'objets.

C'est pour éliminer ces difficultés que, dans sa version 2, AROM permet la définition de nouveaux types. En donnant la possibilité d'intégrer dans le LMA ces nouveaux types et leurs opérateurs associés, nous le rendons à son tour extensible et augmentons son expressivité.

### 5.3 Extensibilité

L'idée centrale du système de types d'AROM2 est de permettre son extension avec des types adaptés aux besoins d'applications spécifiques. Afin de simplifier la tâche d'extension, l'ajout de nouveaux types et de nouveaux opérateurs exploite un mécanisme d'héritage. L'utilisateur doit spécialiser des classes et implémenter des interfaces de l'API d'AROM pour insérer de manière cohérente de nouveaux types et les opérateurs associés dans la hiérarchie de types et, respectivement, dans la hiérarchie d'opérateurs d'AROM.

#### 5.3.1 Extension des types

Le module de types d'AROM définit l'ensemble des types de données qui peuvent être utilisés dans une base de connaissances. Des opérations sur ces types de données sont également proposées par le module de types. De la même manière que dans le système de types Météo [6], le module de types d'AROM considère deux niveaux de représentation des types :

- Le *C-type* (classe de types) qui représente l'ensemble, fini ou non, des valeurs partageant une même structure, tel que l'ensemble des réels ou l'ensemble des chaînes de caractères. Chaque C-type définit des opérations qui lui sont applicables.
- Les  $\delta$ -types qui représentent des sous-ensembles de C-types. Chaque  $\delta$ -type est associé à un C-type qui représente son type principal. Les informations relatives aux données se trouvent donc dans le C-type. Les opérations sont aussi

définies au niveau des C-types. En revanche, les  $\delta$ -types contiennent des informations relatives à la restriction du domaine défini par le C-type. Pour un même C-type il peut donc exister plusieurs, voir une infinité, de  $\delta$ -types.

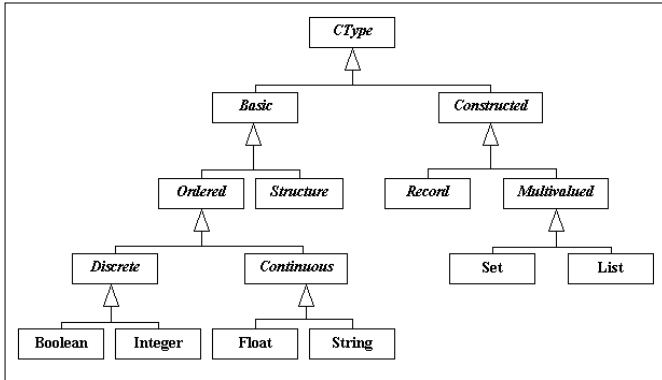


FIG. 6 – La hiérarchie des types prédéfinis d'AROM

Des classes abstraites de C-types sont prédéfinies et organisées hiérarchiquement afin de regrouper les C-types selon leurs caractéristiques (FIG. 6). A un premier niveau les C-types sont séparés en C-types de base (*basic*) et construits (*constructed*). On trouve ensuite :

- Les *C-types de structures* qui permettent de représenter des structures (classes).
- Les *C-types ordonnés* qui représentent les C-types acceptant des valeurs ordonnées. On peut distinguer les C-types ordonnés discrets et les C-types ordonnés continus.
- Les *C-types multivalués*, eux-mêmes divisés en *C-types List* et *C-types Set*, qui représentent des collections de valeurs de même type.
- Les *C-types record* qui représentent des ensembles de couples étiquettes/types.

Par défaut, seuls les C-types simples (*String*, *Boolean*, *Float*, *Integer*) sont définis dans le module de types et peuvent être utilisés afin de typer les variables AROM. Pour ajouter un nouveau type au système AROM, il faut étendre le module de types en spécifiant une des interfaces de la hiérarchie de types du système AROM (FIG. 6). Pour spécifier un nouveau type l'utilisateur doit définir son format à travers deux types de fonctions: une fonction qui permet de créer un objet quelconque de ce type et des fonctions pour la lecture/écriture sur le disque d'un objet de ce type.

Par exemple, le type *Date* peut être introduit comme un type *basic* en s'appuyant sur le type *Date* de Java, qui offre beaucoup de précision (jusqu'au millisecondes) et qui offre aussi une implémentation en Java pour certaines opérations. Si l'utilisateur souhaite gérer des dates imprécises (le fait qu'on on peut connaître seulement l'année ou

seulement le mois quand un événement s'est produit), le type *Date* peut être introduit comme type structuré de la forme (jour: *Integer*, mois: *Integer*, année: *Integer*, siècle: *Integer*).

### 5.3.2 Extension des opérateurs

L'évaluation d'une expression LMA dans le système AROM (qu'il s'agisse d'une équation, d'une contrainte ou d'une requête) comprend trois composantes principales :

- L'analyse lexicale et syntaxique de l'expression, pour valider le fait que les expressions LMA sont correctement formées. Cela permet d'éviter les erreurs du type `MaVariable = 2 5 +`.
- L'interprétation des expressions pour valider le typage cohérent des expressions. Cela permet d'éviter les erreurs du type `MaVariable = 5 + "abc"`
- L'évaluation des expressions pour calculer les résultats des équations ou des requêtes ou pour vérifier les contraintes. Cela permet de résoudre des équations telles que `MaVariable = 2 + 5` et de donner à la variable `MaVariable` la valeur 7.

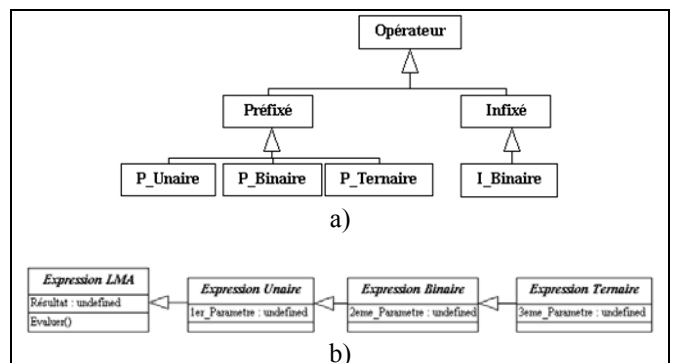


FIG. 7 – Types d'opérateurs et d'expressions LMA en AROM

Pour étendre le LMA d'AROM, il est nécessaire d'étendre les trois composantes. L'utilisateur doit créer un nouvel opérateur, en spécifiant son nom, son type (préfixé ou infixé), son arité (unaire, binaire, ternaire) et sa signature (les types des paramètres et le type du résultat). Le nouvel opérateur est ajouté dans la hiérarchie d'opérateurs du système (voir FIG. 7 a). Le système effectue automatiquement les extensions nécessaires à l'analyseur syntaxique et à l'analyseur de types. Ensuite, il faut spécialiser une classe abstraite de l'interpréteur du LMA d'AROM (le composant chargé de l'évaluation des expressions) qui contient une seule méthode, la fonction d'évaluation de l'opérateur respectif (voir FIG. 7 b).

Pour réussir l'évaluation d'une expression LMA le système AROM doit accomplir plusieurs étapes. L'analyse lexicale et syntaxique permet de créer un arbre syntaxique abstrait de l'expression à évaluer (si l'expression est correctement formée), sinon une exception est levée.

L'arbre abstrait syntaxique sert ensuite de support pour la vérification de types. Si l'expression LMA est cohérente du point de vue des types, la vérification de types réussit, sinon une exception est levée. Il est important d'observer que l'analyse syntaxique comme la vérification de types ne sont effectuées qu'une seule fois pour chaque expression LMA.

La troisième étape est plus complexe, elle s'exécute une fois pour chaque attribut d'AROM dont on doit inférer la valeur. L'évaluation de l'arbre abstrait peut donner un résultat (lorsque les données existantes sont suffisantes), peut lever une exception lorsque certaines opérations sont incorrectes (comme la division par 0) ou peut ne pas produire de résultat. L'absence de résultat peut avoir deux origines : l'une liée à l'absence de données (lorsque, par exemple, le calcul d'une variable nécessite la valeur d'une autre variable qui n'est pas connue), l'autre liée à l'incapacité du système à résoudre le problème posé. Un tel cas peut se produire, par exemple, lorsque sont présentes des références cycliques entre variables. Considérons l'exemple suivant,

```
PremiereVariable = DeuxiemeVariable + 5
DeuxiemeVariable = 2 * PremiereVariable - 5
```

En essayant d'inférer la valeur de la première variable, le système cherche à déterminer la valeur de la deuxième variable. Comme la valeur de celle-ci dépend à son tour de celle de la première variable, l'interpréteur d'AROM, dans la version actuelle, détecte ce cycle et s'arrête. Cependant, le système d'équations linéaires décrivant les valeurs des deux variables peut être résolu. Dans une version future de l'interpréteur des solveurs seront inclus, avec des stratégies capables de résoudre des problèmes plus complexes.

Nous illustrons les possibilités du LMA étendu d'AROM en reprenant l'exemple de l'application cadastrale et en utilisant des opérateurs spatiaux pour réaliser une inférence.

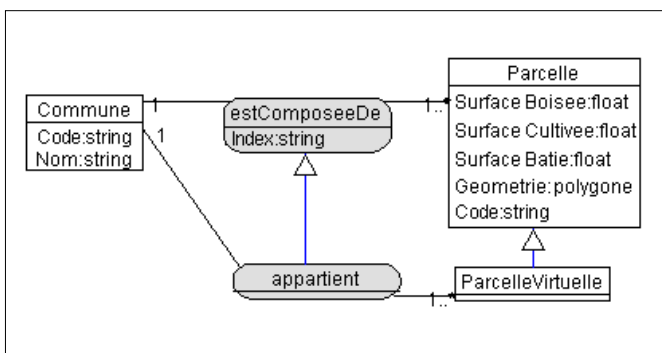


FIG. 8 – Modèle AROM d'une application cadastrale, employant des types extensibles

La modélisation s'effectue en introduisant un nouveau type polygone (FIG. 8).

Supposons que les responsables de l'aménagement de la commune aient l'intention de créer une nouvelle parcelle (FIG. 9 parcelle X, en gris), l'application doit permettre d'estimer les valeurs des attributs dans la nouvelle parcelle.

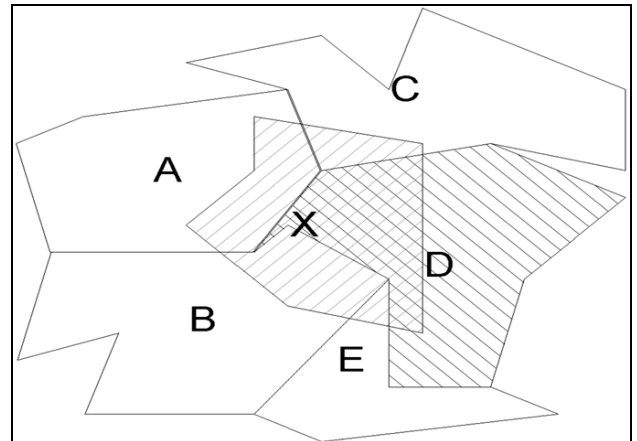


FIG. 9 – Création d'une nouvelle parcelle dans une application cadastrale

Cette estimation est réalisée en calculant la surface affectée à un certain type d'utilisation dans chaque aire d'intersection entre la parcelle virtuelle et les vraies parcelles (FIG. 9 partie avec des hachures croisées).

Pour connaître la proportion finale d'un certain type d'occupation du sol, il suffit de faire le rapport entre la somme des surfaces affectées dans chaque aire et la surface totale de la parcelle virtuelle. Nous donnons ci-dessous l'expression LMA qui permet d'inférer les valeurs d'attributs pour les parcelles virtuelles, pour le modèle présenté en FIG. 8.

```
SurfaceBoisee = sum(parc in this!appartient
.composant!estComposeeDe.composite:
.parc
.SurfaceBoisee * area (intersect (parc
.polygone, this. polygone)))/area(this. polygone)
```

L'opérateur *intersect* permet de calculer l'intersection géométrique entre la parcelle virtuelle et les parcelles réelles appartenant à la même commune (les composants appartenant au même composite dans notre modèle). L'opérateur *area* permet d'obtenir l'aire des polygones d'intersection. L'opérateur itéré *sum* (l'équivalent du symbole  $\sum$  en notation mathématique) permet de calculer la moyenne des proportions de surface boisée dans les parcelles réelles, pondérée par la surface du polygone d'intersection entre la parcelle virtuelle et chaque parcelle réelle.

## 5.4 Applications possibles

En pratique, l'extensibilité du LMA AROM a déjà été utilisée pour deux types d'applications. Une première application a étendu AROM pour la bioinformatique génomique [25] La deuxième application est une exten-

sion spatio-temporelle d'AROM pour des applications d'information géographique, appelée AROM-ST [18]. Cette extension d'AROM contient des types géométriques (point, ligne, polygone, etc.) et temporels (instant, intervalle temporel, etc.), ainsi que des opérateurs associés à ces types. AROM-ST a été employé pour la réalisation d'une application de visualisation d'informations spatio-temporelles liées aux risques naturels [17]).

Les langages de représentation de connaissances sont souvent utilisées en couplage avec des SGBD relationnels afin d'augmenter leurs capacités de représentation. L'expressivité supérieure des langages de représentation permet d'ajouter plus de sémantique aux données relationnelles et de renforcer le maintien de la cohérence.

Nous employons AROM pour la mise en oeuvre d'une architecture permettant de collecter, stocker et manipuler des données géographiques à long terme ([6]). Les problèmes auxquels doit répondre cette architecture sont nombreux et relèvent, plus généralement, de la problématique de la qualité de données :

- les différences entre les méthodes de recensement employées par les différents organismes (d'un pays à l'autre ou d'une période historique à l'autre, par exemple) ;
- les changements territoriaux historiques qui causent des changements (voire des incompatibilités) au niveau des unités statistiques de recensement, comme on a pu le voir lors des recensements de la population allemande au moment de la réunification, ou encore, lors de la scission de la population tchécoslovaque en deux entités distinctes ;
- les erreurs systématiques de recensements (liées, par exemple, aux campagnes de propagande durant guerre froide) ;
- les erreurs non systématiques (accidentelles) de type quantitatif (par exemple, la surestimation d'une population) ou qualitatif (par exemple, l'affectation erronée d'une commune dans un département) ;
- les erreurs induites par les différences linguistiques et culturelles.

Afin de résoudre ce types de problèmes, la base de données doit être intégrée dans une architecture permettant la vérification et la validation de toute opération. Ainsi, toutes les interactions entre la base de données et les utilisateurs pour la mise à jour, la modification, la manipulation et la production de données, passent d'abord par un ensemble de bases de connaissances AROM pour permettre une panoplie de vérifications. Ces bases de connaissances permettent de

- désambiguïser les requêtes utilisateur (établir avec exactitude quelles sont les entités géographiques et quelles sont leurs propriétés auxquelles se réfèrent les requêtes) ;
- trouver les incohérences ou les lacunes existantes dans les données, dans la requête ou dans les deux ;

- chercher les informations disponibles qui pourraient permettre de résoudre les incohérences/ combler les lacunes, et choisir la/les meilleures stratégies à appliquer ;
- appliquer les stratégies d'estimation/ production/ correction de données adéquates.

Dans le processus de conception et de développement de l'architecture, le couplage SRCO-LMA existant en AROM nous a beaucoup facilité le travail. L'avantage principal du LMA est de permettre de décrire des traitements de données très complexes sous une forme déclarative et simplifiée. Cela aide les développeurs d'applications à avoir une démarche rigoureuse et à rester au niveau conceptuel. Du point de vue pragmatique, l'utilisation du LMA a permis aussi de limiter le recours à la programmation, accélérant ainsi les développements.

## 6 Conclusion et perspectives

Dans cet article, nous avons présenté le Langage de Modélisation Algébrique (LMA) du système de représentation de connaissances par objets AROM. Le LMA est utilisé dans AROM à trois fins : *i*) pour décrire des équations algébriques permettant, au besoin, de calculer la valeur d'un attribut dans un objet, *ii*) pour étendre la définition des entités de représentation (classes, associations, instances et tuples) par des contraintes et *iii*) pour formuler des requêtes sur le contenu d'une base de connaissances.

Nous avons montré comment la présence du LMA augmente le pouvoir expressif (ou déclarativité) d'AROM en donnant la possibilité à un concepteur de formaliser, dès la construction, une base des connaissances opératoires, et, à un utilisateur, d'interroger la base de connaissances à l'aide d'un langage de requêtes formel basé sur le calcul ensembliste.

AROM dispose d'un module de types extensibles et permet la définition de nouveaux types (valeurs et opérateurs associés). Nous avons décrit ici comment le LMA peut, à son tour, bénéficier de cette extensibilité et gérer des expressions algébriques impliquant, outre des objets de la base de connaissances, de nouveaux opérateurs et types définis pour les besoins particuliers d'une application.

Nos recherches s'orientent à présent vers l'utilisation d'AROM pour le Web sémantique pour lequel des langages plus déclaratifs semblent pertinents. Dans le contexte plus général des ontologies, AROM apparaît comme un candidat naturel pour la description des concepts et relations qui définissent une ontologie. De même, son LMA permet d'exprimer des connaissances opératoires sur les concepts d'une ontologie, non seulement des règles, mais également des contraintes ou des équations. Dans ce cadre du Web sémantique, il nous faut étudier la corres-

pondance entre le standard OWL et AROM. L'idée est de traduire en AROM des expressions XML de OWL extraites de pages Web, de construire et de gérer des bases de connaissances équivalentes à ces ontologies OWL, d'exploiter les mécanismes d'inférence et de vérification de cohérence proposés par AROM afin de compléter ces bases de connaissances, puis, en retour d'enrichir ou d'étendre les descriptions OWL par les connaissances inférées ou rendues explicites grâce, notamment, au LMA. Notre objectif, à terme, est de rendre utilisables pour le Web Sémantique les mécanismes d'inférence offerts par AROM.

## Références

- [1] F. AIMMS <http://www.aimms.com/aimms/index.cgi>
- [2] Algebraic Modeling languages <http://www.mat.univie.ac.at/~oleg/AML.html>
- [3] AMPL : <http://www.ampl.com/>
- [4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi et P. Patel-Schneider. The Description Logic Handbook : Theory, Implementation and Applications. Cambridge Univ. Press, 2003.
- [5] C. Bruley, P. Genoud, V. Dupierris V., Guide Utilisateur AROM V2, Rapport Interne INRIA, 2004, <ftp://ftp.inrialpes.fr/pub/romans/logiciels/aron2/ug.pdf>
- [6] C. Capponi. Un système de types pour la représentation des connaissances par objets. Revue d'Intelligence Artificielle 12(3):309-344, septembre 1998.
- [7] J. Chabalière, G. Fichant, C. Capponi. La classification récursive en AROM. *RSTI, série L'Objet*, 9(1-2):, Hermès, Paris, 2003.
- [8] R. Ducournau, J. Euzenat, G. Masini et A. Napoli. Langages et modèles à objets : état des recherches et perspectives. Collection didactique D-019, INRIA, Le Chesnay, 1998.
- [9] P. Durand, C. Médigue, A. Morgat, Y. Vandenbrouck, A. Viari, F. Rechenmann, Integration of data and methods for genome analysis *Current Opinion in Drug Discovery and Development*, 2003, Vol. 6, No. 3, pp. 346-352.
- [10] GAMS <http://www.gams.com/Default.htm>
- [11] Genostar <http://www.genostar.org>
- [12] J. Gensel, C. Capponi, P. Genoud et D. Ziébelin, Vers une intégration des relations Partie-Tout en AROM, Langages et Modèles à Objets (LMO'06), Nîmes, 22 - 24 Mars 2006.
- [13] H J Greenberg The home page of the Intelligent Mathematical Programming System (IMPS). <http://carbon.cudenver.edu/~hgrenbe/imps/imps.html>
- [14] LINGO <http://www.lindo.com/>
- [15] G. Masini, A. Napoli, D. Colnet, D. Léonard et K. Tombre. Les langages à objets : langages de classes, langages de frames, langages d'acteurs. InterEditions, 1989.
- [16] M. Minsky. A Framework for Representing Knowledge. *The Psychology of Computer Vision*, pages 211-281, P. Winston ed, Mc Graw Hill, New York, 1975.
- [17] B. Moïsuc, P.-A. Davoine, J. Gensel et H. Martin. Design of Spatio-Temporal Information Systems for Natural Risk Management with an Object-Based Knowledge Representation Approach, *Geomatica*, Vol. 59, No. 4, 2005.
- [18] B. Moïsuc, J. Gensel et H. Martin. Représentation de connaissances par objets pour les SIG à représentations multiples. Conférence Cassini-SIGMA, Grenoble, 2004.
- [19] MPL <http://www.maximal-usa.com/>
- [20] A. Napoli, B. Carré, R. Ducournau, J. Euzenat, F. Rechenmann. Objets et représentation, un couple en devenir. *RSTI L'Objet*, 10(4), pages 61-81, Hermès (Paris), 2004.
- [21] OPL <http://www.ilog.com/products/oplstudio/>
- [22] OWL, Web Ontology Language, <http://www.w3.org/TR/owl-features>, 2004.
- [23] M. Page, J. Gensel, C. Capponi, C. Bruley, P. Genoud et D. Ziébelin, Représentation de connaissances au moyen de classes et d'associations : le système AROM. *Actes de LMO'00*, pages 91-106, Mont Saint-Hilaire, Canada, 2000.
- [24] R. Quillian. Semantic Memory. *Semantic information processing*, pages 227-270, M. Minsky ed., MIT Press, Cambridge (MA), USA 1968.
- [25] H. Rivière-Rolland, L. Taloc, D. Ziébelin, F. Rechenmann et A. Viari. Modelling metabolism knowledge using objects and associations. *ERCIM News*, n°43, october 2000.
- [26] Tools : Automatic Differentiation, Modeling Systems, and Analysis Tools. <http://plato.asu.edu/topics/tools.html>.
- [27] ESPON 3.2 Project Third Interim Report [http://www.espon.lu/online/documentation/projects/cross\\_tematic/3899/3rd-ir\\_3.2\\_vol4\\_elements-of-support.pdf](http://www.espon.lu/online/documentation/projects/cross_tematic/3899/3rd-ir_3.2_vol4_elements-of-support.pdf)