Langage à objets et logique de descriptions : un schéma d'intégration

Roland Ducournau* **— Gabriel Pavillet*,****

* *L.I.R.M.M. – 161, rue Ada*
34392 Montpellier cedex 5

** *C3T*
Immeuble Strategie Concept, 1300 av Albert Einstein
34000 Montpellier

ducournau@lirmm.fr
gpavillet@c3t.fr

RÉSUMÉ. *La programmation et la représentation des connaissances constituent des paradigmes opposés qui partagent pourtant, dans le cas particulier de l'approche objet, un modèle commun. Nous proposons, dans cet article, un schéma d'intégration d'un langage à objets et d'une logique de descriptions, donnant ainsi à un sous-ensemble des classes la sémantique et les capacités déductives des logiques de descriptions, tout en conservant les spécificités du langage à objets telles que la mono-instanciation. Ce schéma repose sur une association de concepts aux classes, et sur l'identification des rôles à des attributs et des individus à des objets. Les propriétés spécifiques comme la mono-instanciation sont maintenues par des axiomes de subsomption généraux d'une* TBox *libre.*

ABSTRACT. *Programming and knowledge representation are two opposite paradigms, despite the fact that, in the particular case of object orientation, they share a common model. In this paper, we propose an integration scheme of an object-oriented language with a description logic, giving to a subset of the classes in the language the semantics and the deductive power of the logic, while preserving the specificity of object-oriented languages, such as mono-instanciation. Concepts are associated to classes, roles are identified to attributes and individuals to objects. Specific properties like mono-instanciation are maintained by general subsomption axioms in a free* TBox.

MOTS-CLÉS : *langage à objets, logiques de descriptions, systèmes classificatoires, réflexivité.*

KEYWORDS: *object-oriented languages, description logics, classification schemes, meta-object protocol.*

1. Introduction

La programmation et la représentation des connaissances constituent deux paradigmes que tout oppose :

– la programmation est procédurale, avec une sémantique opérationnelle et un mode opératoire non monotone, par changement d'états ;
– la représentation des connaissances, dans son idéal le plus déclaratif, a une sémantique logique et un mode opératoire monotone, par déduction pour compléter la connaissance portant sur l'état courant.

Cette opposition a fait les beaux jours de la célèbre controverse opposant un mode *déclaratif* et un mode *procédural* de la représentation des connaissances [WIN 75, CAR 95].

Pourtant, tout concourt à faire coexister ces deux paradigmes :

– d'une part, la phase d'analyse d'un développement logiciel relève en grande partie de la représentation des connaissances, au moins dans le sens informel du terme ;
– d'autre part, toute utilisation de techniques de représentation des connaissances dans un logiciel nécessite une bonne dose de programmation.

L'opposition déclaratif/procédural est certes un vieux débat. L'approche objet lui donne un relief particulier du fait que, dans ce cadre spécifique, les deux paradigmes partagent un modèle commun. Qui plus est, ce modèle commun remonte à la nuit des temps ou, pour être plus précis, à la syllogistique aristotélicienne [WAN 89, PER 98, RAY 00]. L'existence de ce modèle commun et sa proximité avec la façon dont l'humanité appréhende le monde — au moins dans la culture occidentale — est certainement le facteur principal du succès de l'approche objet.

Dans un travail antérieur [DUC 96], nous avons proposé le modèle des systèmes classificatoires[1] comme modèle général commun, sous-jacent à l'approche objet dans ses différentes incarnations (programmation, bases de données, représentation des connaissances, etc.) et fournissant un cadre unificateur, de C++ à KL-ONE. Ce modèle, assez essentiellement abstrait, a été par la suite utilisé [DUC 98a] pour analyser la sémantique des systèmes comme SHIRKA [REC 88], TROEPS [PRO 95] ou FROME [DEK 94].

Nous poursuivons ce travail, dans cet article, en proposant un schéma d'intégration entre, d'une part, un langage de programmation par objets raisonnablement quelconque — la seule chose que nous lui demandons est d'être suffisamment extensible, par exemple au moyen d'un *meta-object protocol* (MOP) — et, d'autre part, une logique de descriptions, dont le modèle constitue actuellement la forme la plus aboutie de représentation des connaissances par objets au sens large du terme [WOO 92].

Le plan de l'article est le suivant. Dans la section 2, nous décrivons, successivement et succinctement, les modèles des systèmes classificatoires, des logiques de descriptions et de l'approche objet « standard ». Nous terminons par la discussion des

1. Reprenant ainsi, dans une acception plus spécifique, un terme introduit par [EUZ 93].

principales différences entre les deux modèles à intégrer. La section 3 décrit ensuite ce que nous entendons par intégration et le schéma particulier que nous proposons, en passant en revue, terme à terme, l'appariement des entités des deux modèles. La section 4 présente ensuite nos réalisations en cours, avec le langage CLAIRE [CAS 99] et la logique de descriptions FACT [HOR 97a, HOR 97b], et la section suivante discute de travaux comparables. Nous terminons par quelques perspectives.

2. Le modèle objet

Les modèles objet sont supposés bien connus, au moins pour ce qui concerne le modèle objet standard et les logiques de descriptions : seules les grandes lignes et les traits distinctifs seront rappelés.

2.1. *Les systèmes classificatoires*

Le modèle des systèmes classificatoires est constitué des éléments suivants :

– des *classes*, interprétées *extensionnellement* et munies d'une description *intensionnelle*, au moyen d'*attributs*, eux-mêmes décrits par un *domaine* ;
– une relation de *spécialisation* entre ces classes, impliquant une inclusion des extensions et une inclusion, en sens inverse, des intensions, caractérisée par un « héritage » des attributs et une inclusion de leurs domaines respectifs ;
– deux relations extensionnelles sur les classes : une relation d'*incompatibilité*, impliquant l'exclusion des extensions, et une relation dite de *subsomption*, caractérisée par l'inclusion des extensions ;
– des *instances*, décrites par un certain nombre d'assertions portant sur leur appartenance à l'extension d'une classe et sur les valeurs de leurs attributs.

La cohérence entre ces différents éléments repose sur le fait qu'une *instance* possède les *attributs* décrits par ses *classes*, avec des *valeurs* appartenant au *domaine* de l'attribut dans la classe.

La *sémantique* d'un système classificatoire est calquée sur celle des logiques de descriptions : voir par exemple [NEB 90, DON 96]. Une interprétation est définie par un *domaine* (sans rapport avec celui des attributs) et une fonction qui interprète chaque classe par une partie du domaine, chaque instance par un élément du domaine et chaque attribut par une relation binaire sur le domaine. Un *modèle* est une interprétation vérifiant des contraintes liées aux relations entre les classes, les instances et les attributs. Il est alors possible de définir diverses notions correspondant à des problèmes qu'un mécanisme de déduction *ad hoc* peut résoudre :

– la *subsomption* et l'*incompatibilité* de deux classes sont définies respectivement par l'inclusion ou la disjonction de leurs interprétations, *dans tous les modèles* ;
– l'*appartenance* d'une instance à une classe est définie par le fait que l'interprétation de la première appartient à l'interprétation de la seconde, *dans tous les modèles* ;

$syntaxe$	sémantique	
$C, D \longrightarrow$		*concepts*
A	$A^I \subseteq \Delta$	*concept atomique*
$\top$	Δ	*top*
$\bot$	$\emptyset$	*bottom*
$\neg C$	$\Delta - C^I$	*négation*
$C \sqcap D$	$C^I \cap D^I$	*conjonction*
$C \sqcup D$	$C^I \cup D^I$	*disjonction*
$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^I \Rightarrow d_2 \in C^I\}$	*universelle*
$\exists R.C$	$\{d_1 \mid \exists d_2 \in C^I : (d_1, d_2) \in R^I\}$	*existentielle*
$\geq nR.C$	$\{d_1 \mid \lvert\{d_2 \in C^I \mid (d_1, d_2) \in R^I\}\rvert \geq n\}$	*cardinalité min*
$\leq nR.C$	$\{d_1 \mid \lvert\{d_2 \in C^I \mid (d_1, d_2) \in R^I\}\rvert \leq n\}$	*cardinalité max*
$R, Q \longrightarrow$		*rôles*
P	$P^I \subseteq \Delta \times \Delta$	*rôle atomique*
$Q \sqcap R$	$Q^I \cap R^I$	*conjonction*
$R\vert_C$	$\{(d_1, d_2) \in R^I \mid d_2 \in C^I\}$	*restriction*
R^{-1}	$\{(d_1, d_2) \mid (d_2, d_1) \in R^I\}$	*inverse*
$Q \circ R$	$\{(d_1, d_2) \mid \exists d_3 : (d_1, d_2) \in Q^I \ \ \& \ \ (d_3, d_1) \in R^I\}$	*composition*
$C \times D$	$C^I \times D^I$	*produit de concepts*

Figure 1. *Syntaxe et sémantique du langage de termes de la famille $\mathcal{AL}$*

– la *consistance* (ou *satisfiabilité*) du système classificatoire est définie par l'existence d'au moins un modèle ;
– la *cohérence* (ou *satisfiabilité*) d'une classe est définie par l'existence d'au moins un modèle dans lequel l'interprétation de la classe n'est pas vide.

2.2. *Les logiques de descriptions*

Les logiques de descriptions sont fondées sur les notions de *concepts*, d'*individus* et de *rôles*, qui correspondent respectivement aux classes, instances et attributs des systèmes classificatoires.

Une logique de description est constituée :
– d'un langage de *termes*, constitué d'un certain nombre de *constructeurs* permettant de construire des termes de concepts ou de rôles à partir de termes de concepts et de rôles ; la sémantique est définie constructivement, l'interprétation d'un terme étant fonction de l'interprétation de ses opérandes ; la figure 1 décrit ainsi la syntaxe et la sémantique des langages de la famille $\mathcal{AL}$;
– d'*axiomes* terminologiques permettant d'introduire de nouveaux (noms de) concepts ou de rôles à partir de termes ; ces nouveaux concepts ou rôles peuvent être *définis* ($A \doteq C$) ou *primitifs* ($A \overset{.}{\leq} C$), au sens où il peut s'agir d'une condition nécessaire et suffisante (égalité des interprétations) ou d'une condition nécessaire seulement (inclusion des interprétations) ; les axiomes terminologiques forment ce qu'il est l'usage d'appeler la *TBox* ;

– un ensemble d'*assertions*, constituant la *ABox*, permet d'introduire des individus en précisant leurs relations d'instanciation aux concepts ($a : C$) et à d'autres individus par des rôles (aRb).

Chaque langage particulier est constitué d'un sous-ensemble de constructeurs : pour $\mathcal{AL}$ il sagit de A, $\perp$, $\top$, $C \sqcap D$, $\forall RC$, $\exists R\top$ (existentielle non qualifiée) et $\neg A$ (concept atomique seulement). $\mathcal{ALE}$ rajoute $\exists RC$ (existentielle qualifiée), $\mathcal{ALU}$ $C \sqcup D$, et $\mathcal{ALC}$ $\neg C$ (négation généralisée).

La sémantique est analogue à celle des systèmes classificatoires mais elle s'exprime constructivement sur le langage de termes (figure 1) et la négation remplace l'incompatibilité.

Les logiques de descriptions expriment ainsi un point de vue particulier sur les systèmes classificatoires, avec deux traits spécifiques :

– la relation d'appartenance d'un individu à ses concepts relève de la *multi-instanciation* et elle est dynamique tout en restant monotone ;
– les rôles d'un individu ne sont pas expliqués par l'appartenance de l'individu à un concept : tout individu peut avoir n'importe quel rôle, ou bien, dit autrement, le concept universel $\top$ décrit tous les rôles. En ce sens, le modèle des logiques de descriptions est un modèle autant voire plus relationnel que conceptuel.

2.3. *Le modèle objet « standard »*

Le modèle objet standard, c'est-à-dire le modèle des langages dits « de classe » suivant [MAS 89], est constitué :

– d'une hiérarchie de spécialisation de classes : on parle aussi de hiérarchie d'héritage, cet héritage pouvant être simple ou multiple suivant qu'une classe peut avoir une unique super-classe directe ou plusieurs ;
– chaque classe décrit des propriétés, attributs et méthodes, qui s'ajoutent aux propriétés qu'elle hérite de ses super-classes ;
– des objets, instances de ces classes, sont créés par un procédé d'instanciation d'une classe ;
– enfin le modèle peut être *réflexif*, incluant un méta-modèle : en particulier, les classes elles-mêmes peuvent être des objets, instances de méta-classes (fig. 2).

Il s'agit là encore, mais partiellement seulement, d'un point de vue particulier sur les systèmes classificatoires avec des traits spécifiques :

– notons d'abord que la problématique des procédures (donc des méthodes) n'est pas couverte par les systèmes classificatoires, sans en être exclue pour autant ;
– la sémantique de la réflexivité non plus : il faudrait sans doute faire intervenir une théorie d'ensembles *non fondés* [ACZ 87] pour pouvoir prendre en compte les méta-classes ; pour faire du modèle OBJVLISP un système classificatoire, il faut donc considérer que la racine de la hiérarchie de spécialisation (la classe $\top$) est une sous-classe de `Object`, distincte, mais instance, de `Class`.
– le modèle standard pratique la *mono-instanciation* : chaque objet est instance d'*une* classe, c'est-à-dire d'une unique classe minimale : dans la plupart des

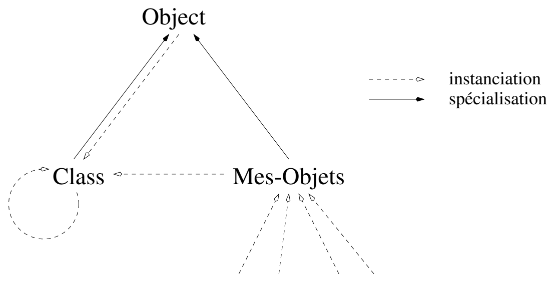

Figure 2. *Le modèle réflexif de* OBJVLISP

langages, ce rattachement est immuable mais des capacités marginales de *migration d'instances* existent dans plusieurs langages (CLOS, SMALLTALK) ;
- chaque attribut d'un objet est expliqué par l'appartenance de l'objet à une classe qui introduit cet attribut : un objet ne peut avoir que les attributs décrits par sa classe (typage).

2.4. *Les principales différences*

Les énoncés de ces trois modèles se paraphrasent mutuellement : en fait, le modèle des systèmes classificatoires est formé du point de vue sémantique et déductif des logiques de descriptions appliqué au point de vue de la programmation par objets sur l'expression des classes. Si l'on met de côté les éléments du modèle standard qui ne sont pas pris en compte par les systèmes classificatoires (méthodes et réflexivité), on peut conclure à deux grandes différences entre les langages à objets et les logiques de descriptions, différences qui portent sur les relations qui lient les objets, d'une part aux classes, d'autre part aux attributs :
- la mono-instanciation plutôt statique des premiers s'oppose à la multi-instanciation complètement dynamique des secondes.
- les langages à objets font l'hypothèse d'une « explication des attributs », qui relève du typage et qui fait défaut aux logiques de descriptions.

3. Le schéma d'intégration

L'objectif de ce travail est de donner à un langage de programmation par objets, de façon limitée mais non triviale, une sémantique et des moyens de déduction analogues à ceux d'une logique de descriptions, tout en conservant les spécificités de la programmation par objets (réflexivité, méthodes, etc.).

Pratiquement, cela signifie que nous voulons *des* objets (pas forcément tous) qui se comportent comme des objets « normaux » — typiquement, on peut leur envoyer des messages auxquels ils répondent par l'activation de la méthode correspondante

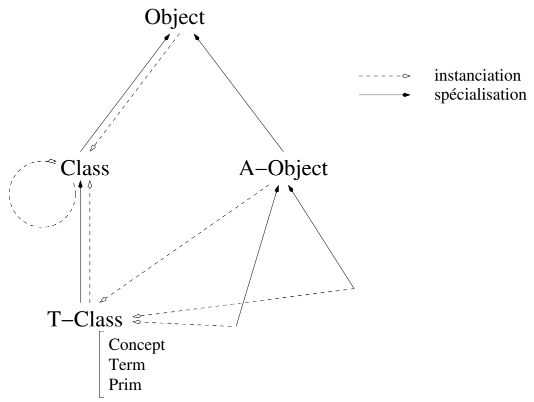

Figure 3. *Le modèle réflexif de l'intégration*

— mais aussi comme des individus d'une logique de descriptions : ils peuvent faire l'objet d'assertions et d'inférences — typiquement, de la classification d'instances, ce qui nécessitera des capacités de migration.

Pour ce faire, nous avons choisi d'intégrer une logique de descriptions dans un langage à objets, en unifiant les deux modèles de façon à conserver les spécificités du langage à objets telles que mono-instanciation — mais en conservant la dynamicité de l'instanciation dans les logiques de descriptions — et explication des attributs. Nous allons donc décrire l'appariement des entités des deux modèles : concepts et classes, individus et instances puis rôles et attributs.

3.1. *Classes et concepts*

L'objectif est de pouvoir considérer certaines classes — pas toutes, réflexivité oblige — comme des concepts, ou, ce qui est équivalent, d'associer à ces classes des concepts, de telle sorte que les sémantiques des spécialisations coïncident.

Ces classes doivent être particulières, donc avoir leur propre méta-classe, et il paraît raisonnable de leur donner une super-classe commune. On définit donc (fig. 3) :

- une méta-classe `T-Class`, classe de toutes les classes-concepts, instance et sous-classe de `Class` ;
- une classe `A-Object`, classe de tous les objets-individus et super-classe de toutes les classes-concepts, sous-classe de `Object` et instance de `T-Class`.

On associe à une classe-concept C un concept $Concept(C)$ comme suit. Lors de la définition d'une nouvelle classe-concept C, sous-classe (indirecte) de `A-Object` et instance de T-Class, on associe à C un terme $Term(C)$. Le concept associé à C est alors défini récursivement par :

$$Concept(\texttt{A-Object}) \quad = \quad \top \tag{1}$$

$$Concept(C) \quad = \quad Term(C) \sqcap \big(\bigsqcap_{C' \in Super(C)} Concept(C') \big) \tag{2}$$

où $Super(C)$ désigne les super-classes directes de C.

On introduit enfin le nom de concept C par un axiome terminologique :

$$C \doteq Concept(C) \tag{3}$$

Par la suite, on pourra donc utiliser, dans les termes, les expressions C et $Concept(C)$, qui sont parfaitement équivalentes. Si l'on veut que la classe-concept soit primitive et non pas définie, on pourrait remplacer $\doteq$ par $\dot{\leq}$ dans (3). Une solution alternative sera préférée, en introduisant, comme le fait [NEB 90], un composant primitif $Prim(C)$ pour chaque classe-concept primitive C, en posant alors :

$$Concept(C) \quad = Term(C) \sqcap Prim(C) \sqcap \big(\bigsqcap_{C' \in Super(C)} Concept(C') \big) \tag{4}$$

A ce stade, on voit que la méta-classe T-Class doit décrire trois attributs spécifiques, pour $Term(C)$, $Concept(C)$ et le caractère primitif ou défini de la classe.

3.1.1. *Incompatibilité et mono-instanciation*

Cette première définition est insuffisante parce qu'elle n'assure pas une sémantique en mono-instanciation. Il faut donc compliquer les choses.

3.1.1.1. En héritage simple

Deux classes incomparables par la spécialisation sont alors incompatibles : on se contentera donc de rajouter la négation des concepts des classes sœurs déjà définies, en remplaçant (2) par :

$$\begin{aligned} Concept(C) = Term(C) \sqcap Concept(Super(C)) \\ \sqcap \big(\bigsqcap_{C' \in Sub(Super(C))} \neg Concept(C') \big) \end{aligned} \tag{5}$$

où $Sub(D)$ désigne les sous-classes directes de D (déjà définies). On transformera (4) de façon analogue. En pratique, on pourra simplifier utilement cette expression en remplaçant $Concept(C')$ par $Term(C')$ ou par $Prim(C')$, suivant que C' est définie ou primitive.

3.1.1.2. En héritage multiple

Le problème se complique alors considérablement, ce qui est un peu étonnant puisque les logiques de descriptions sont plus proches de l'héritage multiple que de l'héritage simple.

Le principe est le suivant : deux classes incomparables sont incompatibles si elles n'ont pas de sous-classe commune. Il apparaît ici un phénomène inconnu des logiques de descriptions : il faut d'abord assurer que la hiérarchie de classes est *fermée*, c'est-à-dire que l'on ne définira plus de nouvelles sous-classes. En effet, en mono-instanciation, la définition d'une nouvelle classe n'est pas une *extension conservative* : elle a une influence sur la sémantique des classes existantes. Ainsi, si les classes C_1 et C_2 n'ont pas de sous-classe commune, les assertions $a : C_1$ et $a : C_2$ rendent le système inconsistant, mais cette inconsistance est levée dès que l'on définit une sous-classe D commune à C_1 et C_2. De plus, on peut en déduire $a : D$. Il n'est donc pas possible de faire des inférences avant d'avoir fermer la hiérarchie de classes.

Le problème dépasse ainsi celui de l'incompatibilité : toute instance de deux classes incomparables doit être instance de l'une de leurs sous-classes communes.

Étant donné deux classes C et C', soit $Inf(C, C')$ l'ensemble des minorants maximaux de C et C' (au sens de la relation de spécialisation). Si cet ensemble est vide, les deux classes sont incompatibles. S'il n'est pas vide, toute instance de ces deux classes est instance de l'un des éléments de l'ensemble. Il n'est pas possible d'exprimer *simplement* cette information dans le concept des classes. En revanche, si la logique de descriptions admet une *TBox* dite *libre* [DON 96] au sens où les axiomes terminologiques $C \dot{\leq} D$ ne servent pas seulement à introduire de nouveaux noms de concepts mais peuvent être utilisés comme *axiomes de subsomption* entre deux termes quelconques, il est alors possible d'exprimer la contrainte de mono-instanciation sous la forme de l'axiome de subsomption :

$$C \sqcap C' \dot{\leq} \bigsqcup_{D \in Inf(C,C')} D \tag{6}$$

le terme de droite étant $\perp$ lorsque l'ensemble $Inf(C, C')$ est vide. La nécessité de la fermeture apparaît bien ici : les classes du terme de gauche sont des super-classes des classes du terme de droite.

Pour résumer, en héritage simple, on génère les concepts au fur et à mesure de la définition des classes, par (5), en assurant directement la mono-instanciation. En héritage multiple, en revanche, on génère les concepts au fur et à mesure sans tenir compte de l'incompatibilité, par (2) ou (4). Une fois la hiérarchie fermée, on calcule l'ensemble des axiomes de subsomption qui assurent à la fois l'incompatibilité et la mono-instanciation.

Il est donc possible d'encoder une hiérarchie de classes dans une logique de descriptions, sous réserve qu'elle soit suffisamment expressive : en héritage simple, il faut la négation généralisée d'$\mathcal{ALC}$, mais en héritage multiple, il faut en plus les axiomes de subsomption généraux d'une *TBox* libre.

3.2. *Objets et individus*

Il s'agit maintenant d'établir une correspondance biunivoque entre les instances indirectes de `A-Object` et les individus de la *ABox*.

On suppose que, dans le langage à objets considéré et comme en CLOS, l'interface des classes comporte deux fonctions (ou méthodes) `make-instance` et `change-class`. Alors, lorsque C est une `T-Class` :
- `make-instance(C,a)`, lorsque a n'existe pas, entraîne l'assertion $a : C$;
- `change-class(C,a)`, lorsque a existe, entraîne l'assertion $a : C$;
- inversement, l'assertion $a : C$ — qu'elle provienne de l'utilisateur, du programme ou du moteur d'inférences — provoque l'appel de `change-class` ou de `make-instance` suivant que a existe ou pas.

Le cas de `change-class` est un peu délicat : son usage n'est admissible que si C et la classe courante de a, C', ont une borne inférieure, c'est-à-dire un unique minorant maximal — $Inf(C, C')$ est un singleton. Si ce n'est pas le cas, le résultat est momentanément indéterminé du point de vue du langage à objets, puisque c'est une disjonction de classes. Lorsque C est une sous-classe de C', la méthode `change-class` des `T-Class` provoquera donc une migration en faisant appel à la méthode usuelle (par `call-next-method` pour rester en CLOS). Sinon, elle n'entraînera qu'une assertion $a : C$, charge aux inférences ultérieures de provoquer une véritable migration.

Dans le cas où C n'est pas une `T-Class`, mais un terme de concept quelconque, l'assertion $a : C$ provoque un `make-instance(A-Object,a)` lorsque a n'existe pas, et rien dans le cas contraire.

Notons pour finir, que l'inconsistance de la logique de descriptions se traduit par la levée d'une exception et le retrait de la dernière assertion.

3.3. *Attributs et rôles*

A chaque rôle de la logique de descriptions est associé un attribut du langage à objets. L'inverse n'est pas vrai puisque l'on garde la possibilité d'avoir des propriétés, attributs ou méthodes, en dehors de la sémantique de la logique de descriptions. L'intégration des attributs et des rôles pose deux problèmes :
- un problème de sémantique déjà cité, celui de l'explication des attributs ;
- un problème d'implémentation : faut-il implémenter physiquement les rôles dans les objets, comme des attributs normaux, ou peut-on, voire doit-on, se contenter d'y accéder par une indirection dans la *ABox* ?

Dans tous les cas, l'accès en écriture par le langage à objets est limité : la sémantique étant monotone, il est possible de rajouter des valeurs, pas d'affecter directement un ensemble de valeurs (ce qui poserait un problème de révision qu'il serait prématuré d'aborder ici : voir par exemple [NEB 90] ou [CRA 96]). *Il s'agit là de la seule limitation de programmation de cette intégration.*

3.3.1. *Explication des attributs*

Rappelons qu'il s'agit du fait que, dans les logiques de descriptions, il est possible d'affirmer aRb sans que a soit tenu d'appartenir à une classe (concept) possédant l'attribut (rôle) R alors que, dans un langage à objets, accéder à R impose que R soit une propriété de la classe de a (on a sinon une erreur de type). Le compromis que nous visons est le suivant : l'assertion aRb doit permettre de déduire que a est instance de l'une des classes qui introduisent R.

Etant donné un rôle R, on définit $Def(R)$ comme l'ensemble de toutes les classes-concepts C telles que l'un des termes $\forall R.D$, $\exists R.D$ ou une restriction de cardinalité apparaisse dans la conjonction qui constitue $Term(C)$, sans apparaître dans le concept des super-classes de C.

Une fois la hiérarchie fermée, on peut rajouter l'axiome de subsomption :

$$\exists R.\top \ \dot{\leq} \bigsqcup_{D \in Def(R)} D \tag{7}$$

3.3.2. *Implémentation des attributs*

Deux solutions sont envisageables : l'implémentation physique dans les objets, ou des accès logiques à la *ABox*. Dans les deux cas et comme en CLOS, on peut accéder aux attributs par une primitive comme `slot-value` ou par un accesseur. L'usage de variables d'instances relèverait du « sucre syntaxique ».

Les rôles sont en général multi-valués, mais ils peuvent aussi être « fonctionnels », c'est-à-dire mono-valués. On peut donc les associer à des attributs uniformément multi-valués, ou faire la distinction. L'implémentation peut être panachée : implémentation physique pour les rôles fonctionnels, accès logique pour les autres.

L'implémentation physique nécessite deux mécanismes :
- lors de la définition d'une classe C, on rajoute l'attribut R dans la définition de la classe si C appartient à $Def(R)$[2] ;
- toute assertion aRb se traduit par la rajout physique de b dans la valeur de $R(a)$: le langage doit donc posséder une primitive comme `add-slot-value`.

Quant aux accès logiques, si l'on suppose que la logique possède une primitive `role-value` telle que `role-value(a,R)` retourne $\{b \mid aRb\}$, ils sont immédiats. Si l'on désire une meilleure intégration, par l'uniformisation du style d'accès, il suffit de définir des accesseurs pour les rôles lors de la définition des classes.

2. Comme le montre [PAV 99], l'ajout d'un nouvel attribut en cours de *meta-object protocol* est possible mais en général *ad hoc*.

4. Etat des réalisations

Le schéma d'intégration, tel qu'il a été spécifié dans la section précédente, permet de préciser les fonctionnalités nécessaires, tant du point de vue de la logique de descriptions que du langage à objets.

– Pour la logique de descriptions, $\mathcal{ALC}$ et une *TBox* libre sont nécessaires, aussi bien pour l'héritage multiple que pour l'explication des attributs : à défaut, il faudrait intervenir dans le moteur d'inférences pour y rajouter des règles ; notre choix s'est finalement porté sur le système FACT [HOR 97a, HOR 97b] dont les sources en COMMON LISP étaient disponibles et l'efficacité prouvée.
– Pour le langage de programmation, un *meta-object protocol* offre la flexibilité nécessaire : CLOS bien sûr mais, à la limite, OPENC++ aussi. Cela étant, la nécessité de la migration d'instances et l'antagonisme entre spécialisation et sous-typage [CAS 96] font plutôt préférer un langage à typage dynamique, au moins dans un premier temps. Notre choix initial s'était porté sur ILOGTALK/POWER-CLASSES [ILO95, ILO96], qui constituait une alternative intéressante à CLOS. La décision de la société ILOG d'en arrêter le développement et la commercialisation nous a conduit à choisir le langage CLAIRE [CAS 99].

FACT a été réimplémenté en CLAIRE et un module de raisonnement sur la *ABox*, inexistant dans la version d'origine, a été réalisé. L'intégration reste suspendue à une modification du noyau de CLAIRE pour permettre l'extension de son MOP. Le schéma d'intégration des concepts ne nécessite aucune modification de la logique de descriptions. Pour les individus et les rôles, les interventions ne sont nécessaires que dans le module de raisonnement sur la *ABox*. Pour plus de détails, voir [PAV 00].

5. Comparaisons avec d'autres travaux

La cohabitation de la représentation des connaissances et de la programmation est, dans l'approche objet, quasiment aussi vieille que l'approche objet elle-même : tout le courant des langages dits de *frames* en témoigne. La trajectoire de D. Bobrow, de KRL [BOB 77] à CLOS [BOB 88, KIC 91], en passant par LOOPS [BOB 83] et COMMON LOOPS [STE 86, BOB 86] est, à cet égard, exemplaire.

Pourtant, un constat d'échec relatif se dégage : les systèmes hybrides comme KEE [FIK 85] ou Y3 [DUC 91] faisaient cohabiter ces deux paradigmes au détriment d'une sémantique fondée de la représentation des connaissances. L'irruption de KL-ONE et les travaux théoriques qui l'ont suivie pour donner naissance aux logiques de descriptions [WOO 92] ont condamné cette cohabitation en privilégiant la rigueur de la sémantique. Parallèlement, des systèmes comme SHIRKA ou TROEPS refusaient de s'abandonner aux délices de la programmation sans pour autant être dotés d'une sémantique formelle satisfaisante [EUZ 98, DUC 98a]. Le travail présenté dans cet article participe ainsi aux tentatives de refonder cette approche hybride sur des bases sémantiques saines.

Si une telle intégration a déjà été tentée, il semble que notre approche soit originale. [RAT 93] constitue surtout une démonstration de la flexibilité du MOP de CLOS pour définir les concepts d'une logique de descriptions. La proposition s'arrête à la *TBox*, avec un langage et un test de subsomption rudimentaires, sans perspectives visibles.

[MÖL 96] est un travail plus abouti que le précédent, qui adopte une démarche inverse de la nôtre. Au lieu d'associer des concepts aux classes d'un langage à objets, l'auteur étend le mécanisme des fonctions génériques de CLOS aux individus de CLASSIC, une logique de descriptions implémentée en COMMON LISP [RES 95]. Des classes sont associées aux concepts nommés de la logique et le problème de la multi-instanciation est réglé par la génération dynamique de classes. L'argumentation de l'auteur est intéressante. Elle vise en particulier à répondre à divers problèmes de réutilisation constatés dans l'approche objet, par exemple le problème « *Make isn't generic* » [KIC 93], dont le *pattern* « *Factory Method* » [GAM 94] est une solution. D'après [MÖL 96], la logique de descriptions LOOM [BRI 93] propose aussi l'utilisation de fonctions génériques sur des individus de la *ABox*, mais dans une forme moins élaborée.

Par ailleurs, cette approche de l'intégration — qu'il s'agisse du présent article ou de [MÖL 96]— diffère fondamentalement de l'implémentation d'un système de représentation des connaissances par objets dans un langage à objets, comme par exemple TROEPS [PRO 95] qui est implémenté en ILOGTALK [ILO95], ou certaines logiques de descriptions qui sont implémentées en C++.

Sur un plan plus théorique, ce travail (et en particulier le modèle des systèmes classificatoires [DUC 96]) doit aussi être comparé avec le langage $\mathcal{ALUNI}$ proposé par [CAL 99] pour unifier divers modèles objet des bases de données, de la programmation et de la représentation des connaissances, et plus précisément pour interpréter dans le modèle des logiques de descriptions les modèles entité-relation et les langages de *frames*. En ce qui concerne ce dernier modèle, la proposition d'$\mathcal{ALUNI}$ est insuffisante. Elle se base en effet sur la sémantique en logique du premier ordre proposée par [HAY 79], tout en visant des systèmes comme KEE et des études comme [KAR 93] ou [FIK 85], tous bien ultérieurs. L'instanciation fait les frais de cet anachronisme, puisque l'article ne propose qu'une modélisation des classes (*TBox*), alors que les langages de *frames* ont souvent distingué classes et instances, en particulier KEE. Le problème de la mono-instanciation, qui invaliderait la proposition d'$\mathcal{ALUNI}$, n'est donc pas abordé, pas plus que la question de l'explication des attributs.

Encore plus récemment, le langage OIL a été conçu comme une synthèse entre logique de descriptions et langages de *frames* : la question de ce que nous appelons « explication des attributs » est explicitement soulevée, dans une démarche qui semble, sur ce point, assez proche de la nôtre [FEN 00].

6. Conclusions et perspectives

Ce travail est né de discussions au sein du groupe *classification et objets* du PRC-IA et constitue, de fait, une première réponse constructive aux débats internes à ce groupe sur la possibilité d'intégrer des fonctionnalités de programmation et de représentation dans un même modèle objet [CAR 95, NAP 97].

La faisabilité est claire, même si l'implémentation reste à terminer. On peut voir aussi cette intégration comme la spécification de la sémantique du modèle objet standard en tant que système classificatoire le plus riche possible.

Les perspectives sont de deux ordres : validation de l'utilité et choix d'implémentation. La faisabilité est une chose : il reste à voir si l'utilité est à la hauteur des espoirs. La convergence de notre démarche avec celle de LOOM, de [MÖL 96] et de OIL est un élément positif mais une application significative reste nécessaire. Dans le cas particulier de notre approche, il faut valider l'intérêt d'une logique de descriptions en mono-instanciation.

En ce qui concerne l'implémentation, nous avons choisi une intégration qui ne modifiait aucun des deux éléments à intégrer. Or, si l'approche est validée, il sera sans doute nécessaire de se pencher sur l'implémentation pour réduire les redondances entre les deux systèmes. Par ailleurs, comme la technique des tableaux utilisée pour décrire et implémenter le système de déduction des logiques de descriptions est très modulaire, ce qui rend ces raisonneurs relativement extensibles, il serait intéressant de voir si certaines spécificités de la sémantique ne pourraient pas être traitées par des règles de déduction à la place des axiomes très généraux comme (6) ou (7).

Un point n'a pas été considéré dans notre travail : le statut des attributs des `A-Objects` qui ne sont pas des rôles. Ces attributs existent puisque rien n'empêche d'en définir dans une `T-Class`. S'ils n'ont pas un statut de condition nécessaire et suffisante, ce sont néanmoins des conditions nécessaires. La valeur dans l'objet doit appartenir au domaine spécifié dans la classe. Si le langage objet autorise une redéfinition covariante, la classification peut entraîner une erreur de type sur un tel attribut. On pourrait donc étendre les inférences de la logique de descriptions pour en tenir compte. Dans le même ordre d'idées, la question se pose de savoir s'il faut considérer comme un rôle tout attribut dont le type est `A-Object`. La réponse à ces questions viendra typiquement d'une application.

Enfin, un dernier point méritera une étude approfondie : quelle forme limitée de non monotonie, donc de révision, est-il possible d'introduire pour réduire la distance entre programmation et représentation ? Il faut, *a minima*, implémenter efficacement le retrait d'assertions.

7. Bibliographie

[ACZ 87] ACZEL P., *Non-well-founded sets*, N° 14 CSLI Lecture Notes, CSLI, 1987.

[BOB 77] BOBROW D., WINOGRAD T., « An Overview of KRL, a Knowledge Representation Language », *Cognitive Science*, vol. 1, n° 1, 1977, p. 3–46.

[BOB 83] BOBROW D., STEFIK M., « The LOOPS Manual : A Data and Object Oriented Programming System for Interlisp », Knowledge-Based VLSI Design Group Memo, 1983, Xerox PARC, Palo Alto (CA), USA.

[BOB 86] BOBROW D., KAHN K., KICZALES G., MASINTER L., STEFIK M., ZDYBEL F., « Commonloops : Merging Lisp and Object-Oriented Programming », *Proc. OOPSLA'86*, ACM Press, 1986, p. 17–29.

[BOB 88] BOBROW D., DEMICHIEL L., GABRIEL R., KEENE S., KICZALES G., MOON D., « Special issue, Common Lisp Object System Specification, X3J13 Document 88-002R », *ACM SIGPLAN Notices*, vol. 23, 1988.

[BRI 93] BRILL D., « LOOM Reference Manual (Version 2.0) », University of Southern California, Marina del Rey (CA), USA, 1993.

[CAL 99] CALVANESE D., LENZERINI M., NARDI D., « Unifying Class-Based Representation Formalisms », *JAIR*, vol. 11, 1999, p. 199–240.

[CAR 95] CARRÉ B., DUCOURNAU R., EUZENAT J., NAPOLI A., RECHENMANN F., « Classification et objets : programmation ou représentation ? », *Actes des 5es Journées Nationales PRC-GDR Intelligence Artificielle, Nancy*, Teknea, Toulouse, 1995, p. 213–237.

[CAS 96] CASTAGNA G., « Le modèle fondé sur la surcharge : une visite guidée », *Technique et Science Informatiques*, vol. 15, n° 6, 1996, p. 673–708.

[CAS 99] CASEAU Y., LABURTHE F., « Introduction to the CLAIRE programming language », rapport, 1999, LIENS.

[CRA 96] CRAMPÉ I., EUZENAT J., « Fondements de la révision dans un langage d'objets simple », DENNEBOUY Y., Ed., *Actes du Colloque Langages et Modèles à Objets (LMO'96), Leysin, Suisse*, École Polytechnique Fédérale de Lausanne, 1996, p. 134–149.

[DEK 94] DEKKER L., « FROME : représentation multiple et classification d'objets avec points de vue », Thèse d'Informatique, Université des Sciences et Technologies de Lille, 1994.

[DON 96] DONINI F.-M., LENZERINI M., NARDI D., SCHAERF A., « Reasoning in Description Logics », BREWKA G., Ed., *Principles of Knowledge Representation*, p. 191–236, CSLI Publications, Stanford (CA), USA, 1996.

[DUC 91] DUCOURNAU R., « Y3 : YAFOOL, le langage à objets, et YAFEN, l'interface graphique », Sema Group, Montrouge, 1991.

[DUC 96] DUCOURNAU R., « Des langages à objets aux logiques terminologiques : les systèmes classificatoires », Rapport de Recherche n° 96-030, 1996, L.I.R.M.M., Montpellier.

[DUC 98a] DUCOURNAU R., « La logique des objets : application à la classification incertaine », Ducournau et al. [DUC 98b], Chapitre 12, p. 351–380.

[DUC 98b] DUCOURNAU R., EUZENAT J., MASINI G., NAPOLI A., Eds., *Langages et modèles à objets : État des recherches et perspectives*, Collection Didactique, INRIA, 1998.

[EUZ 93] EUZENAT J., « Définition abstraite de la classification et son application aux taxonomies d'objets », HABIB M., OUSSALAH M., Eds., *Actes de la Conférence Représentations Par Objets (RPO'93), La Grande Motte, France*, EC2, Nanterre, 1993, p. 235–246.

[EUZ 98] EUZENAT J., « Représentation de connaissance par objets », Ducournau et al. [DUC 98b], Chapitre 10.

[FEN 00] FENSEL D., HORROCKS I., HARMELEN F. V., DECKER S., ERDMANN M., KLEIN M., « OIL in a Nutshell », DIENG R., Ed., *Proc. of European Knowledge Acquisition Conference (EKAW'2000)*, LNAI, Springer Verlag, 2000.

[FIK 85] FIKES R., KEHLER T., « The Role of Frame-Based Representation in Reasoning », *Communications of the ACM*, vol. 28, n° 9, 1985, p. 904–920.

[GAM 94] GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading (MA), USA, 1994.

[HAY 79] HAYES P., « The Logic of Frames », METZING D., Ed., *Frame Conception and Text Understanding*, p. 46–61, de Gruyter, Berlin, 1979.

[HOR 97a] HORROCKS I., « Optimisation Techniques for Expressive Description Logics », Technical report, 1997, Dept of Computer Science, University of Manchester.

[HOR 97b] HORROCKS I., « FACT Reference Manual, Version 1.0 », Technical report, 1997, Dept of Computer Science, University of Manchester.

[ILO95] ILOG, Gentilly, « ILOG TALK reference manual, Version 3.2 », 1995.

[ILO96] ILOG, Gentilly, « POWER CLASSES reference manual, Version 1.4 », 1996.

[KAR 93] KARP P., « The Design Space of Frame Knowledge Representation Systems », Technical Note n° 520, 1993, SRI International, Menlo Park (CA), USA.

[KIC 91] KICZALES G., DES RIVIERES J., BOBROW D., *The Art of the Meta-Object Protocol*, MIT Press, Cambridge (MA), USA, 1991.

[KIC 93] KICZALES G., « Traces (A Cut at the "Make Isn't Generic" Problem) », *Proc. of ISOTA'93*, 1993, `ftp ://parcftp.xerox.com/pub/openimplementations /traces.ps.Z`.

[MAS 89] MASINI G., NAPOLI A., COLNET D., LEONARD D., TOMBRE K., *Les langages à objets*, InterEditions, 1989.

[MÖL 96] MÖLLER R., « A Functional Layer for Description Logics : Knowledge Representation Meets Object-Oriented Programming », *Proc. OOPSLA'96*, ACM Press, 1996, p. 198–213.

[NAP 97] NAPOLI A., COLLECTIF D'AUTEURS, « Aspects actuels des représentations de connaissances par objets et de la Classification », PESTY S., SIEGEL P., Eds., *Actes des 6e journées nationales du PRC-GDR Intelligence Artificielle*, Hermès, 1997, p. 289–314.

[NEB 90] NEBEL B., *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence 422, Springer-Verlag, Berlin, 1990.

[PAV 99] PAVILLET G., DUCOURNAU R., « Implémentation des attributs booléens par un Meta Object Protocol », MALENFANT J., ROUSSEAU R., Eds., *Actes LMO'99*, Hermès, 1999, p. 55–68.

[PAV 00] PAVILLET G., « Des langages de programmation par objets à la représentation des connaissances à travers le MOP : vers une intégration », Thèse d'Informatique, Université Montpellier II, 2000.

[PER 98] PERROT J.-F., « Objets, classes, héritage : définitions », Ducournau et al. [DUC 98b], Chapitre 1, p. 3–34.

[PRO 95] PROJET SHERPA, « TROPES 1.0 : Reference Manual », Rapport interne, 1995, INRIA Rhône-Alpes, Grenoble, Version révisée : Tropes 1.1 Reference Manual, 1997. `ftp ://ftp.inrialpes.fr/pub/sherpa/rapports/tropes-manual.ps.gz`.

[RAT 93] RATHKE C., « Implementing Subsumption in an Object-Oriented Language », NAPOLI A., Ed., *Proceedings of the IJCAI Workshop on Object-Based Representation Systems, Chambéry, France*, Rapport CRIN 93-R-156, 1993, p. 101–109.

[RAY 00] RAYSIDE D., CAMPBELL G., « An Aristotelian Introduction to Classification », HUCHARD M., GODIN R., NAPOLI A., Eds., *ECOOP'2000 Workshop on Objects and Classification, a natural convergence*, RR LIRMM 2000- ? ?, 2000, `http ://www.lirmm.fr/~huchard/WorkshopClassif.html`.

[REC 88] RECHENMANN F., « SHIRKA : système de gestion de bases de connaissances centrées-objet. Manuel de référence », INRIA/ARTEMIS, Grenoble, 1988, `ftp ://ftp.inrialpes.fr/pub/sherpa/rapports/manuel-shirka.ps.gz`.

[RES 95] RESNICK L., BORGIDA A., BRACHMAN R., MCGUINNESS D., PATEL-SCHNEIDER P., ZALONDEK K., « CLASSIC. Description and Reference Manual for the Common Lisp Implementation (Version 2.3) », AT&T Bell Laboratories, Murray Hill (NJ), USA, 1995.

[STE 86] STEFIK M., BOBROW D., « Object-Oriented Programming : Themes and Variations », *The AI Magazine*, vol. 6, n° 4, 1986, p. 40–62.

[WAN 89] WAND Y., « A Proposal for a Formal Model of Objects », KIM W., LOCHOVSKY F., Eds., *Object-Oriented Concepts, Databases and Applications*, p. 537–559, Addison--Wesley, Reading (MA), USA, 1989.

[WIN 75] WINOGRAD T., « Frame Representation and the Declarative/Procedural Controversy », BOBROW D., COLLINS A., Eds., *Representation and Understanding : Studies in Cognitive Science*, p. 185–210, Academic Press, New York (NY), USA, 1975.

[WOO 92] WOODS W., SCHMOLZE J., « The KL-ONE Family », *Computers & Mathematics with Applications*, vol. 23, n° 2–5, 1992, p. 133–177.