Exploitation des types dans un modèle de représentation des connaissances par objets.

Cécile CAPPONI

INRIA Rhône-Alpes
LIFIA - IMAG

46 av. Félix Viallet — 38031 Grenoble — France
Tél : (33) 76 57 47 99 — Fax : (33) 76 57 46 95
Email : Cecile.Capponi@imag.fr

RFIA94 — Paris — janvier 1994

1 Introduction

Les systèmes de représentation et manipulation des connaissances offrent des mécanismes puissants s'appuyant sur des relations entre objets, dont la sémantique est rarement définie formellement. Cette absence de formalisme est une réelle lacune car les algorithmes mis en œuvre ne sont pas fondés sur des bases solides et la preuve de l'adéquation de leurs résultats effectifs aux résultats attendus n'est alors pas formellement possible. Pourtant, les domaines d'application modélisés exigent la plupart du temps une rigueur absolue quant aux définitions des entités représentées qu'au niveau des processus mis en œuvre.

Cette rigueur n'est possible que si le modèle de représentation admet un formalisme adapté. Dans le domaine des langages à objets, le formalisme apporté par les types est souvent choisi, puisque d'une part, généralement la notion d'objet contient implicitement la notion de domaines de valeurs, c'est-à-dire de types, et la relation de spécialisation est proche de la relation d'inclusion de domaines, c'est-à-dire de sous-typage. D'autre part, les algorithmes de vérifications de types se révèlent des outils puissants pour la vérification de la cohérence entre les définitions d'entités de représentation.

Ces constatations nous ont naturellement amenés à expliciter la notion de types dans les modèles de représentation des connaissances par objets, bien que l'utilisation qui en est faite n'est pas exactement la même que dans le domaine des langages de programmation. Nous partons cependant de la même hypothèse couramment faite dans le domaine des langages de programmation qui stipule qu'une classe est un moule pour la génération d'objets alors qu'un type est un prédicat que doit satisfaire tout objet de ce type [Wegner& 88]. C'est en ce sens que nous entendons les types : une structure absente de toute signification propre, qui ne supporte pas les divers mécanismes appliqués aux connaisances structurées en classes ou instances, mais qui est toutefois intéressante pour la mise en œuvre de processus de vérification.

Cet article propose ainsi un système de types (section 2) associé à un modèle de représentation des connaissances simple utilisant les notions de classes et d'instances. Les entités et composantes essentielles de ce modèle sont brièvement expliquées et illustrées par des exemples. Le modèle considéré est principalement régi par la relation de spécialisation qui ordonne les classes en une taxonomie compatible avec la relation d'inclusion ensembliste. La relation de spécialisation structure la base de connaissances ; sa définition formelle montre la nécessité de la mise en œuvre d'un processus capable de comparer deux ensembles de valeurs quelconques : une relation de sous-typage est alors introduite (section 3) de telle façon qu'elle permette la vérification de liens de spécialisation entre classes d'objets. La vérification est alors faite par décomposition des structures des classes selon les attributs qu'elles contiennent et par inférence de liens de sous-typage entre les attributs de même nom des deux classes. La recherche de liens est complète et facilitée par l'élaboration d'une structure particulière, le graphe des types d'un attribut *a*, qui est une visualisation de la base de connaissances projetée selon *a* ; le graphe des types permet d'apprécier les raffinements d'un attribut représentés par la relation de sous-typage, indépendamment des autres attributs ; il offre ainsi de nombreux avantages qui sont abordés dans la section 4. Parmi eux le processus de classification d'une classe par profil d'attributs [Finin& 86] est présenté ; ce mécanisme positionne une classe décrite par l'utilisateur dans une taxonomie de classes existante, ordonnée par la relation de spécialisation. Il revient donc à déterminer des liens de spécialisation, qui le seront par une interprétation de la spécialisation en terme de sous-typage.

Avant de conclure en section 6, les nombreuses perspectives de ce travail sont énoncées en section 5, et parmi elles nous retrouvons la formalisation de toutes les relations sémantiques que l'on retrouve dans les modèles de représentation par objets.

2 Modèle de représentation et système de types associé

Cette section décrit et illustre par des exemples le modèle à objet de représentation des connaissances, inspiré du modèle TROPES, qui sera utilisé tout au long de l'article. Les différents aspects de ce modèle seront expliqués et interprétés et un système de types sera enfin présenté de façon à dégager du langage les structures statiques intéressantes pour la mise en œuvre de processus de vérification de la cohérence d'une base de connaissances fondée sur la relation de spécialisation.

2.1 Entités de description du modèle

Le modèle considéré offre une approche classes/instances, dans le sens où une classe donne une structure commune à toutes ses instances. Une base de connaissances écrite dans ce modèle sera composée d'un ensemble de concepts, chacun étant représenté par un ensemble de classes **C** dénotant un ensemble d'instances (ou individus) **I**.

Les classes d'un concept sont réparties dans un nombre fini de *points de vue*. Un point de vue est une taxonomie de classes. Un objet pouvant être observé sous différents points de vue, il n'est plus pertinent de parler d'instance de classe, mais d'*instance de concept* ; une telle instance aura ainsi une décomposition sous chacun des points de vue, chaque décomposition étant rattachée à une et une seule classe du point de vue correspondant (figure 1). Par abus de langage, nous dirons que *I* est une instance de *C* (dans le point de vue *P*)si la décomposition de *I* selon *P* est rattachée à *C*. Par défaut, une instance est rattachée à la racine du concept.

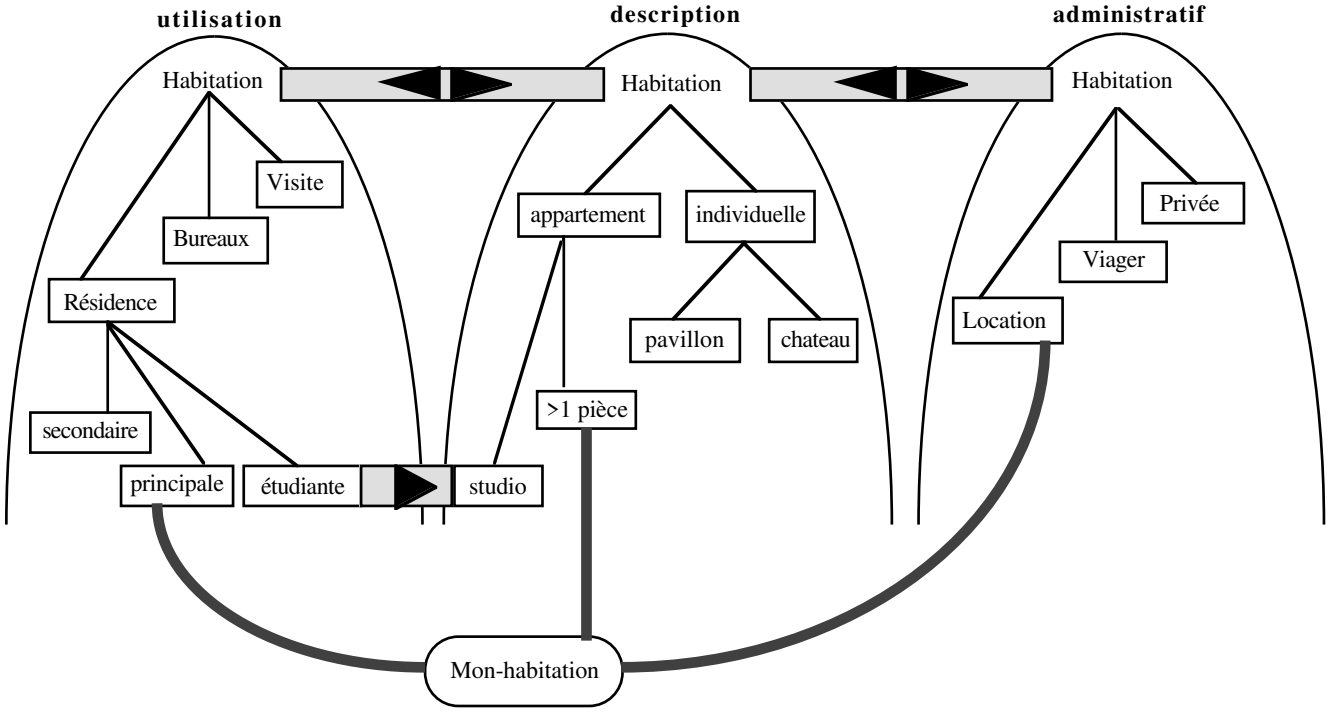

Figure 1. Le concept de nom `Habitation` comprend trois points de vue : `utilisation`, `description` et `administratif`. L'instance `mon-habitation` se décompose sous chacun des points de vue en se rattachant à une classe de chacun ; par exemple, du point de vue administratif, `mon-habitation` est une `location`.
Une passerelle bidirectionnelle existe entre la classe `Habitation` qui est la racine commune des trois points de vue. Une passerelle unidirectionnelle part de `Résidence-étudiante` pour aller à `studio`, signifiant que toute habitation qui est une résidence étudiante est un studio.

Les propriétés des instances sont spécifiées par des attributs. Si une instance donne une valeur à un attribut, cela signifie que la propriété représentée par l'attribut est admise et identifiée par cette valeur. Cette valeur doit appartenir au domaine défini pour l'attribut. Chaque concept d'une base de connaissances définit alors l'ensemble des attributs pertinents pour ses instances, en associant à chaque attribut son domaine minimal. Cette définition minimale d'un atribut dans le concept sera alors éventuellement augmentée d'autres restrictions au niveau des classes du concept. Ainsi, si une instance appartient à une classe sous un point de vue, alors les valeurs de ses attributs doivent appartenir aux domaines correspondants dans la définition de la classe.

Les définitions des attributs au niveau du concept sont minimales, dans le sens où seuls le type principal de l'attribut et le constructeur qui s'y applique sont donnés.

Exemple :

```
(Concept Habitation
   (PointDeVue Utilisation Description
   Administration)
   (attributs
   surface un entier ;
   propriétaire un Personne ;
   locataire un Personne ;
   prix-location un  entier ;
   charges un entier ;
   Date-construction un Date ;
   surf-jardin un entier ;
   nb-pièces un entier ;
   pièces liste chaîne ;
   nb-niveaux un entier ;
   chauffage ens chaîne ;
   exposition ens chaîne)
   (Instances H-12 H-45 H-42 H-74)
)
```

Les classes d'un point de vue reprennent un sous-ensemble des attributs du concept en raffinant leurs domaines de définition. Une définition de classe dans un point de vue est ainsi constituée d'une liste de définitions d'attributs. Une définition d'attribut est donnée par un

ensemble de descripteurs valués. Si D est le symbole d'une description d'attribut dans une classe, on a :

```
D = DA*
DA =  un TP | liste TP | ens TP |
      intervalle [val1 .. val2]+ |
      domaine val+ | sauf val+ |
      valeur val | card [Ent1..Ent2]
      inf proc | défaut val
TP =  type-base | (Classe, PtDeVue)+ |
      Concept
```

— où proc représente une procédure de calcul de valeur pour l'attribut, les valeurs spécifiées pour les descripteurs ont pour type le type principal et les bornes de cardinalité doivent être deux entiers. Les types de bases sont entier, chaîne, réel, booléens, voire par extension des types abstraits définis par le concepteur d'une base.

Exemple.

```
(Classe habitation
   (PointDeVue description)
   (Attributs
   surface ;
   nb-pièces ;
   pièces domaine "chambre" "cuisine "SàM"
   "WC" "SdB" "grenier" "cave" "salon" "hall"
   "buanderie" "boudoir" "salle-de-jeu" "bureau"
   "chambre-ami" "chambre-enfant" "donjon" ;
   chauffage domaine "électrique" "mazout"
   "sans" "collectif" "gaz" "solaire" ;
   exposition domaine "Nord" "Sud" "Est"
   "Ouest" )
)
(Classe Appartement
   (PointDeVue description)
   (sorte-de Habitation)
   (Attributs
   surface intervalle [9 .. 500] ;
   étage intervalle [0 .. 150] ;
   pièces sauf "donjon" ;
   surf-jardin valeur 0)
)
```

2.2. La relation de spécialisation

Les classes, dans chaque point de vue, sont organisées en une taxonomie ordonnée par la *relation de spécialisation* : la taxonomie des classes est alors appelée *graphe de spécialisation* propre au point de vue (figure 1). Plus on a d'informations précises sur une classe, c'est-à-dire plus ses attributs sont raffinés et/ou plus elle en comporte, plus cette classe est spécifique. Cette constatation permet de comprendre la relation de spécialisation entre deux classes d'un même point de vue : C_1 est une spécialisation de C_2 dans le point de vue P, noté $C_1 \leq_{sp} C_2$ si C_1 est définie à partir de C_2, à laquelle éventuellement elle rajoute des attributs, et raffine les attributs existants dans C_2 en leur associant des domaines de valeurs plus restreints. Par conséquent, une instance de C_1 est nécessairement une instance de C_2.

Les informations contenues explicitement dans une classe ne représentent qu'une partie de toutes les informations disponibles pour cette classe. En effet, une classe hérite les propriétés de sa super-classe, qui elle même hérite celles de sa super-classe... Ces propriétés sont les définitions d'attributs. Pour que la définition d'un attribut *a* dans une classe *C* soit complète, il s'agit de considérer tous les descripteurs de *a* existant dans les super-classes de *C*. Cette récupération fournit un ensemble de définitions partielles de *a*, c'est-à-dire un ensemble de descripteurs qui représente ainsi la description complète de *a* dans *C*. La récupération des définitions partielles de *a* dans les super-classes de *C* est réalisée par mécanisme d'héritage, c'est-à-dire par un parcours du graphe de spécialisation durant lequel toutes les propriétés qu'hérite *C* sont retenues.

Définition. Soient C_1 et C_2 deux classes d'un même point de vue *P*. L_1 (resp. L_2) est la liste des instances dont la projection sur *P* appartient à C_1 (resp. C_2). A_1 (resp. A_2) est la liste des définitions d'attributs de C_1 (resp. C_2). Si *a* est une définition d'attribut dans une classe, on note $\uparrow a$ sa définition complète obtenue par héritage. Alors :

$C_1 \leq_{sp} C_2$ <=>
$L_1 \subseteq L_2$
et $\forall\, a \in A_2, \exists\, a' \in A_1$ (avec nom(a) = nom(a'))
tel que **fdom**($\uparrow a'$) $\subseteq$ **fdom**($\uparrow a$)

— où **fdom** : *D -> ensemble de valeurs* associe à une description d'attribut l'ensemble des valeurs du monde possibles pour cet attribut compte-tenu de cette description.

Seuls les descripteurs d'ordre statique sont pris en compte dans la définition structurelle de la relation de spécialisation, c'est-à-dire par la fonction **fdom**. Ainsi, le descripteur **défaut** est ignoré, car la valeur qu'il peut avoir n'a aucune incidence sur la relation de spécialisation : la valeur par défaut pour un attribut d'une classe peut être différente de la valeur par défaut du même attribut dans une de ses sous-classes ou dans sa super-classe. Comme le défaut, les méthodes d'inférence (descripteur **inf**), elles ne participent pas à la définition du domaine de valeurs acceptables pour un attribut, mais sont utilisées pour calculer la valeur de l'attribut auquel elles sont associées, dans le cas d'une instance particulière.

2.3. Les passerelles

Les éventuels liens sémantiques entre deux points de vue doivent être considérés : il serait trop

restrictif de considérer que les points de vue sont complètement indépendants. Ainsi, le modèle permet de définir des liens d'inclusion ensembliste entre deux classes de deux points de vue différents. Un tel lien est une *passerelle*. Une passerelle unidirectionnelle de source (C_1, P_1) et de destination (C_2, P_2) indique que toute instance de la classe C_1 dans le point de vue P_1 doit automatiquement être une instance de la classe C_2 (ou d'une de ses sous-classes) dans le point de vue P_2[1]. Une passerelle bidirectionnelle entre deux classes C_1 et C_2 de deux points de vue différents indique que toute instance de C_1 est instance de C_2 et vice-versa.

Il faut noter qu'il existe automatiquement une passerelle bidirectionnelle entre les racines des points de vue : ces derniers ont un sommet commun, la classe racine du concept (figure 1).

Définition. Soient C_1 une classe du point de vue P_1 et C_2 une classe du point de vue P_2. L_1 (resp. L_2) est la liste des instances dont la projection sur P_1 (resp. P_2) appartient à C_1 (resp. C_2). A_1 (resp. A_2) est la liste des définitions d'attributs de C_1 (resp. C_2). Alors :

$$
\begin{aligned}
(C_1, P_1) \leq_{\text{pass}} (C_2, P_2) \\
\quad <=> \; & L_1 \subseteq L_2 \\
\quad \text{et} \quad & \forall\, a \in A_1, \text{ tel que } a' \in A_2 \\
& \text{et } \text{nom}(a) = \text{nom}(a') \\
& \mathbf{fdom}(\uparrow a) \subseteq \mathbf{fdom}(\uparrow a')
\end{aligned}
$$

Il est important de noter l'importance des descriptions de domaines des attributs dans la définition structurelle de la relation de spécialisation. De même, les domaines associés aux attributs jouent aussi un rôle au niveau de l'acceptabilité statique des passerelles ; en effet, une passerelle < (C_1, P_1) ; (C_2, P_2) > est acceptable si et seulement si les attributs communs à C_1 et C_2 ont une spécification dans C_1 qui est une restriction de la spécification correspondante dans C_2.

2.4. Les types.

Au niveau vérification statique, la relation de spécialisation, tout comme la relation d'ordre entre les classes source et destination d'une passerelle, nécessite la mise en œuvre de la fonction **fdom** ainsi que le calcul de l'inclusion entre deux ensembles.

Tout ensemble ne pouvant pas toujours être représenté de façon simple, il s'agit de déterminer une forme commune d'expression de ces ensembles sur laquelle pourra être calculée la relation d'inclusion. En effet, les descripteurs permettant d'exprimer le domaine de valeurs associé à un attribut sont variés ; réunis, ils peuvent présenter des redondances ou des incohérences qui ne sont pas visibles à première vue, d'autant plus que la description d'un attribut dans une classe est éparpillée entre cette classe et ses super-classes.

La forme commune d'expression d'un ensemble de valeurs associé à un attribut sera une *expression de type*, et la description d'un attribut d'une classe sous cette forme sera le *type de l'attribut* dans cette classe.

2.4.1. Normalisation et types d'un attribut

La définition d'un attribut dans une classe C résulte de la combinaison des définitions associées à cet attribut dans les super-classes de C. En particulier, le domaine de valeurs de cet attribut est défini par toutes les restrictions héritées des super-classes. Un même domaine de valeurs peut ainsi être défini de plusieurs façons, en combinant des restrictions de domaine en différents nœuds du graphe de spécialisation. Pour comparer les domaines de valeurs associés à un attribut dans des classes différentes, il importe donc de disposer d'une forme normalisée pour sa définition. Cette forme normalisée correspond au type de cet attribut (figure 2).

Le type d'un attribut est défini par son domaine de valeurs entièrement spécifié. Le type d'un attribut dans une classe C_1 est donc déterminé par toutes les spécifications du domaine de cet attribut qui apparaissent dans C_1 et dans les super-classes de C_1.

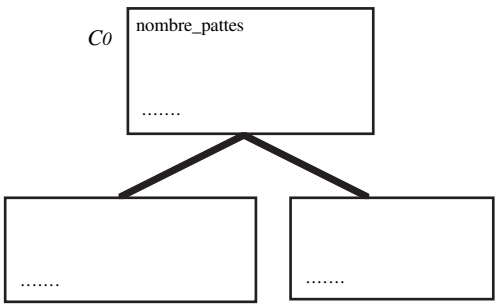

Figure 2 : Nécessité de la normalisation des définitions d'attributs. Les définitions de `nombre_pattes` dans C_1 et C_2 sont équivalentes, même si elles sont exprimées différemment. Afin d'obtenir le type de ces différentes définitions, il convient de récupérer toute l'information dans la (les) super-classe(s) ; l'équivalence entre les deux descriptions ne pourra être déduite que si l'on convient d'une forme unique d'expression.

Les spécifications du domaine d'un attribut dépendent de la nature de cet attribut. Le domaine d'un attribut peut être concret ou abstrait. Un

[1] Dans le modèle TROPES, la source d'une passerelle est un ensemble de couples (classe,Point de Vue) auxquels une instance doit appartenir pour appartenir à la destination.

domaine concret est spécifié par application des opérateurs de construction de domaines concrets (intersection, union, énumération...) sur un domaine primitif (réels, entiers, chaînes...). La spécification de ce domaine peut être complétée par des contraintes qui utilisent les opérateurs d'égalité, de différence et de comparaison. Un domaine abstrait correspond à un sous-ensemble des instances d'un concept. Dans la spécification d'un domaine abstrait, un nom de classe couplé avec un nom de point de vue dénote l'ensemble des instances de cette classe considérée sous ce point de vue.

Les domaines associés à un même attribut dans des classes différentes peuvent être comparés à l'aide des types associés respectivement à ces définitions d'attribut. La relation de sous-typage (notée $\leq_\tau$) est ainsi équivalente à la relation d'inclusion portant sur les domaines des attributs. Un type t_1 est sous-type d'un type t_2, si et seulement si t_1 correspond à un domaine inclus dans le domaine auquel correspond t_2.

Notons que dans le cas où le domaine de l'attribut est un ensemble d'instances d'une classe C d'un point de vue P de la base, le type de cet attribut est un *type-classe*. Le type d'une classe C dans un point de vue P sera un type d'enregistrement des types que C définit pour les attributs de P ; si C ne définit pas un attribut a du concept, cela revient à lui associer le type initial donné dans la définition du concept. En effet, cet attribut a n'est plus un critère de vérification pour l'instanciation ou pour la spécialisation.
En conséquence, si le domaine de valeurs d'un attribut est l'ensemble des instances d'un n-uplet de classes de différentes points de vue, c'est-à-dire $(P_1; C_1) \times_c ... \times_c (P_n ; C_n)$, le type associé sera un enregistrement, résultat de la combinaison $\times_c$ des différents types de classes : c'est un *type combiné*. Ce produit a pour résultat l'enregistrement des types de tous les attributs des deux classes : lorsque les deux classes ont un attribut commun, le type rendu pour cet attribut est l'intersection de ses types dans les deux classes ; lorsqu'un attribut a n'apparaît que dans une des deux classes C, le type associé à a est tout simplement le sien dans C.

Exemple :
(nous supposons pour la simplicité d'écriture que tous les attributs de C_1 et C_2 sont explicitement donnés dans leur définition et ainsi que le mécanisme d'héritage n'a pas à être mis en œuvre)

$$(C_1 ; P_1 ; S_1 ; LS_1 ; ((a : t_{a1}) (b : t_{b1}) (c : t_{c1})) ; LI_1)$$
$$\times_c$$
$$(C_2 ; P_2 ; S_2 ; LS_2 ; ((a : t_{a2}) (b : t_{b2}) (d : t_{d2})) ; LI_2)$$
$$=$$
$$((a : t_{a1} \cap t_{a2}) (b : t_{b1} \cap t_{b2}) (c : t_{c1}) (d : t_{d2}))$$

2.4.2. Le système de types

Voici ci-dessous le système de types considéré pour l'ensemble des domaines de valeurs qui peuvent être exprimés dans le modèle :

```
type      ==  Wrong                |
              <CT ; Tens ; card ; val>

CT        ==  un TP                |
              liste TP             |
              ens TP

TP        ==  TB                   |
              tuple

TB        ==  entier               |
              réel                 |
              chaîne               |
              booléen

tuple     ==  []                   |
              [ (TA; )*  TA ]

TA        ==  (nom : type)

Tens      ==  { val* }+            |
              { val* }-            |
              [val1..val2]*

card      ==  [Ent1..Ent2]
```

La plus petite unité de type construit est celle d'un attribut à partir duquel s'exprime un type de classe ou un type combiné. Le *type* d'un attribut est un quadruplé comprenant le type principal (ex. ***liste*** *entier*), une représentation explicite du domaine de valeurs de l'attribut, une contrainte de cardinalité, et enfin la donnée d'une possible valeur imposée pour l'attribut.
— le type principal d'un attribut est soit un type de base, qui ne requiert aucune représentation particulière, soit un type classe ou un type combiné, tous deux représentés par un *tuple*. Un tuple est un type d'enregistrement de couples (nom : type) des attributs présents dans la classe ou dans la combinaison typée.
— le domaine de valeurs de l'attribut *Tens* peut être énuméré, représenté par une liste d'intervalles dans le cas d'un type principal ordonné, ou bien il est possible d'énumérer son complémentaire. Les valeurs énumérées et les bornes des intervalles doivent appartenir au type principal.
— la cardinalité d'un attribut *card*, dans le cas où le type principal est une liste, est un intervalle dont les bornes représentent les cardinalités minimum et maximum de la liste de valeurs. Ces bornes doivent être des entiers positifs, `Ent1` $\leq$ `Ent2`, où `Ent2` peut prendre comme valeur le symbole ∞.
— la valeur imposée pour un attribut d'une classe, *val*, est une valeur que doit toujours prendre cet attribut pour une instance de la classe : c'est une

valeur du type principal, c'est-à-dire une valeur de base où une instance.
En cas d'incohérence de type (comme par exemple un type vide), celui-ci est dénoté par le mot-clé **wrong** de façon à rendre l'incohérence détectable et à éviter par exemple d'éventuels tests d'appartenance d'une valeur à ce type.

Exemple : le type de `nombre_pattes` dans les classes C_1 et C_2 de la figure 2 est le suivant :

```
nombre_pattes :
      < entier;
        [0..0] [2..2] [4..4] [6..6] [8..8];
        – ; – >
```

Le typage des entités est effectué par des fonctions de récupération, normalisation et réduction dont les sémantiques précises sont définies par des règles d'inférence.

Ainsi défini, ce système apporte une structure commune à l'expression des domaines de valeurs des attributs. Cette forme normale va permettre la définition d'une relation de sous-typage traduisant la relation d'inclusion des domaines de valeurs représentés par les types, de façon à mettre en œuvre la fonction **fdom** et permettre ainsi la vérification structurelle de la relation de spécialisation et des passerelles.

3 Relation de sous-typage et graphe des types d'un attribut

Il s'agit de définir la relation de sous-typage, à la fois dans le cas de comparaisons entre attributs et entre classes. Comme cela a déjà été précisé, la relation de sous-typage à définir doit être équivalente à la relation d'inclusion de domaines.

3.1. Relation de sous-typage entre types d'un attribut

La relation de sous-typage entre types d'un attribut doit prendre en compte toutes les relations existant entre les composants correspondants de l'expression de type : les types principaux doivent appartenir à la relation de sous-typage, les domaines explicites doivent appartenir à la relation d'inclusion, de même pour les intervalles de cardinalité, et les valeurs imposées, si elles existent toutes deux, doivent être les mêmes.

Soient $T_1 = (\text{nom} : < t_1 ; d_1 ; c_1 ; v_1 >)$ et $T_2 = (\text{nom} : < t_2 ; d_2 ; c_2 ; v_2 >)$ deux types d'un même attribut observé dans différentes classes. La relation de sous-typage entre ces types se définit comme suit :

$T_1 \leq_\tau T_2 \iff t_1 \leq_\tau t_2$
et $d_1 \subseteq d_2$ et $c_1 \subseteq c_2$
et $v_1 = v_2$ ou $v_2 =$ **void**.

Il n'y a pas de relation de sous-typage entre des types de base.

Le calcul ou la vérification d'un lien de sous-typage entre types d'un attribut peut mener au calcul ou à la vérification de liens de sous-typage entre types d'enregistrement (record). Il est donc nécessaire de définir la relation à ce niveau.

3.2. Relation de sous-typage entre record

Un type de classe ou un type combiné est un enregistrement de types d'attributs. Ainsi, la relation de sous-typage entre deux records résulte de la conjonction des liens de sous-typage entre les types des attributs correspondants [Cardelli& 91].

Soient $T_1 = [\ a_{11} \ldots a_{1n}\ ]$ et $T_2 = [\ a_{21} \ldots a_{2p}\ ]$ deux records. La relation de sous-typage entre de tels types s'énonce comme suit :

$T_1 \leq_\tau T_2 \iff$
- $p \leq n$

et
- $\forall\ a_{2i} = (\text{nom} : < t_{2i} ; d_{2i} ; c_{2i} ; v_{2i} >)$,
  $\exists\ a_{1j} = (\text{nom} : < t_{1j} ; d_{1j} ; c_{1j} ; v_{1j} >)$,
  tel que $a_{1j} \leq_\tau a_{2i}$

et
- $\forall\ a_{ki} = (\text{nom} : T_{ki}) \in T_k$,
  $\nexists\ a_{kj} = (\text{nom} : T_{kj}) \in T_k, T_{ki} \neq T_{kj}$.

Le typage et la recherche de liens de sous-typage dans un modèle similaire à celui présenté ici ne posent pas forcément les mêmes problèmes que dans le cadre des langages de programmation, en particulier du fait que les classes manipulées en représentation des connaissances, contrairement à celles des langages de programmation, sont essentiellement descriptives. En effet, elles n'ont pas de composante comportementale (procédurale) : le "comportement" des objets est dicté par la sémantique même du modèle de représentation.

3.3. Le problème de la récursivité

Cette définition, reliée à la précédente, fait apparaître le problème de la récursivité. En effet, le typage d'un attribut nécessite parfois le typage d'une classe (du fait des objets complexes) et réciproquement. Ce problème peut être partiellement évité en assimilant la relation de sous-typage entre types combinés à la relation de spécialisation (éventuellement combinée) entre classes. Quoi qu'il en soit, cette solution n'est pas entièrement satisfaisante car, comme le montre la suite, le problème de la récursivité est plus profond.

La recherche interactive de liens de sous-typage entre schémas tels que ci-dessous n'est pas réellement un problème puisque l'interprétation de ces schémas — indépendante de leur structure —

est parfaitement connue par l'utilisateur. Cependant, le typage automatique du schéma `Enfant` n'est pas évident puisqu'il présente une auto-référence. De plus, le système ne peut pas déterminer automatiquement si le type `Enfant` est sous-type du type `Personne`, c'est-à-dire que le système ne peut pas déterminer un lien de spécialisation entre ces deux classes à moins qu'il ne soit donné explicitement.

Exemple:

```
(classe Personne
      (PointDeVue Vie-courante)
      (sorte-de Objet)
      (Attributs
      nom un chaîne ;
      âge un entier ;
      amis liste-de Personne))

(classe Enfant
      (PointDeVue Vie-courante)
      (sorte-de Personne)
      (Attributs
      jouets liste-de chaîne
             dom poupée meccano ballon ;
      amis       liste-de Enfant))
```

Ce problème peut être énoncé de façon plus générale : quel lien de sous-typage existe entre les types $T_1 = [a : T_1]$ et $T_2 = [a : T_2]$? C'est le problème bien connu des types récursifs polymorphes dont l'étude n'est pas le but de cet article. Pour plus d'informations, [Amadio& 91] aborde ce large sujet dans le cadre des langages de programmation. Notons que ce problème est aussi présent dans les langages terminologiques : *les cycles terminologiques* posent les mêmes problèmes de non-calculabilité de la relation de subsomption. Dans ce cadre, des études sont aussi menées actuellement pour pallier ce handicap (sémantique des points fixes, sémantique descriptive...). Pourtant, aucune des solutions actuelles n'étant satisfaisante, le problème est généralement contourné en interdisant ces définitions cycliques [Nebel 91].

Par la suite, afin d'éviter le problème de la dépendance récursive entre types combinés et types d'attributs, la relation de sous-typage entre types combinés et la relation de spécialisation observée du point de vue structurel seront confondues. En effet, il est facile de voir que $C1 \leq_{sp} C2 \Rightarrow C1 \leq_{\tau} C2$, c'est-à-dire qu'un lien de sous-typage peut être déduit d'un lien de spécialisation. Cependant, le contraire et faux, donc en confondant sous-typage et spécialisation, un degré d'information est perdu. Ainsi, la relation générale de sous-typage reste actuellement incomplète.

Le type d'une classe sera ainsi le nom de cette classe, un type combiné sera la liste des noms des classes mises en jeu couplées avec les points de vue correspondants. La relation de sous-typage sera confondue avec la relation de spécialisation éventuellement combinée en cas de produit.

3.4. Graphe des types d'un attribut

Le calcul du type d'un attribut dans chaque classe du concept permet la construction du *graphe des types* de cet attribut dans le concept.

3.4.1. Structure du graphe des types d'un attribut

En observant un attribut dans tous les points de vue d'un concept, de nombreuses spécifications de cet attribut apparaissent, différentes mais souvent reliées par la relation d'inclusion des domaines de valeurs. Afin de mettre en évidence ces liens, il s'agit de retrouver chacune de ces spécifications puis de calculer leur forme normale. Ainsi, certaines spécifications vont apparaître comme étant équivalentes, et d'autres vont pouvoir être liées par le sous-typage. Ce sont ces résultats qui vont former le graphe des types de l'attribut observé (figure 3). À chaque nœud du graphe est associé un type de l'attribut et la liste des couples *(classe, point de vue)* donnant ce type à l'attribut. Un arc du type *t1* vers le type *t2* signifie que *t1* est un sous-type de *t2*, c'est-à-dire que $t1 \leq_{\tau} t2$. Le nombre d'arcs dans le graphe des types est minimal, du fait de la propriété de transitivité de la relation de sous-typage : le graphe des types est la fermeture transitive et reflexive du graphe issu de la relation de sous-typage. Le type du nœud maximal est le type ***Top*** = (*nom* : < *t* ; **{}-** ; *card* ; — >), qui représente le domaine de toutes les valeurs possibles pour l'attribut. Notons que *t* peut être un type simple, record ou construit par **liste** ou **ens**, et il doit être le maximum des types principaux observés pour l'attribut qui est lui même un type (par exemple, si le type principal de chaque nœud est *entier*, ***Top*** référencera le type *entier*) ; *card* vaut, par défaut, **—** pour un attribut de type simple et **]-∞..+∞[** pour un type construit. Le type ***Top*** permet à la fois au graphe des types d'être fermé, et force tous les attributs de nom *na* dans le concept à avoir un type principal inférieur ou égal à ***Top*** selon la relation de sous-typage. C'est-à-dire que tous les types d'un attribut de même nom dans un concept seront sous-types du type principal, ce qui évite certains conflits de noms.

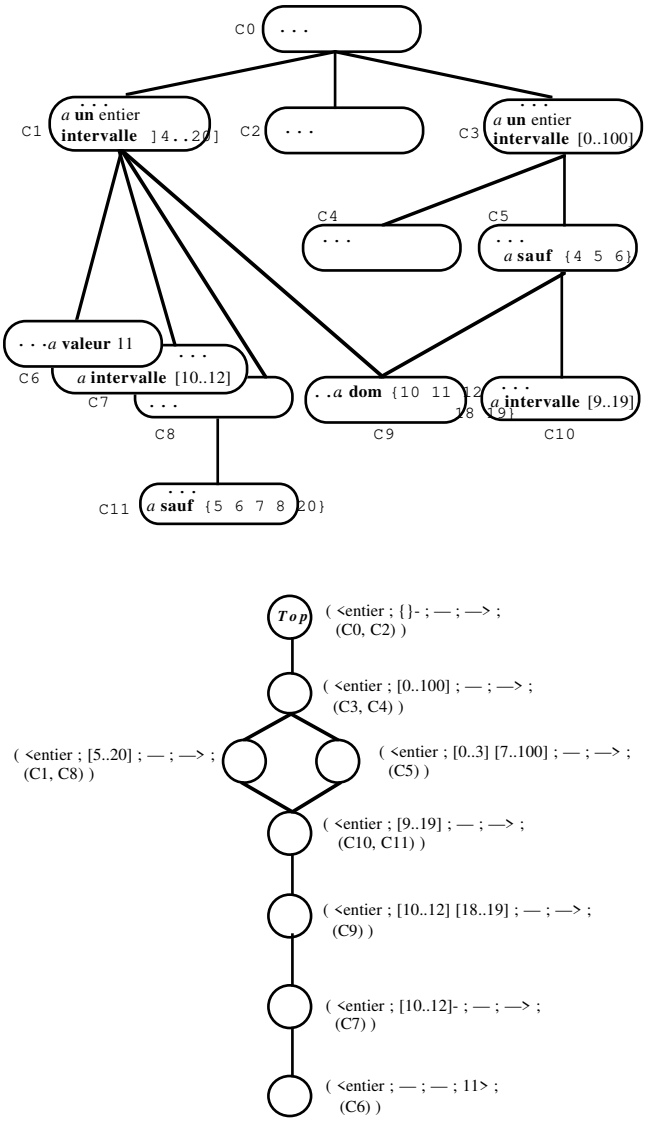

Figure 3. Graphe de spécialisation d'un point de vue et graphe des types de l'attribut *a* visible sous le point de vue. Pour plus de lisibilité, seules les spécifications de l'attribut *a* sont données dans le graphe de spécialisation.

Voyons maintenant la définition générale d'un graphe des types pour un attribut observé dans l'ensemble des points de vue d'un concept.

Soit $\mathcal{C}$ l'ensemble des définitions du concept considéré. Soit *a* le nom de l'attribut.

Soit $\mathcal{G}a = \mathcal{N}a \cup \mathcal{A}a$ le graphe des types de *a* dans $\mathcal{C}$ ($\mathcal{N}a$ sont les nœuds et $\mathcal{A}a$ les arcs de $\mathcal{G}a$).

• $\mathbf{Top} \in \mathcal{N}a$

• $\forall\ N1 \in \mathcal{N}a, N1 = (ST1\ ;\ LNC1)$

$ST1 = (a : T1)$

et $\forall NC \in LNC1$,

$C = (NC\ ;\ PV\ ;\ nom1\ ;\ Lnom2\ ;\ LAexp\ ;\ Linst)$ est un schéma de $\mathcal{C}$,

$\exists\ Aexp \in LAexp, Aexp = a\ Cexp$,

FN o **FR** $(Aexp) = T1$.

et $\forall\ N2 \in \mathcal{N}a, N2 = (ST2\ ;\ LNC2)$,

$\forall\ NC \in LNC1, NC \notin LNC2$ et $ST1 \neq ST2$.

• $\forall\ N1 \neq \mathbf{Top} \in \mathcal{N}a, \exists\ N2 \in \mathcal{N}a$ tel que $(N1, N2) \in \mathcal{A}a$

et $\forall\ N1 \neq \mathbf{Top} \in \mathcal{N}a$, $(N1, \mathbf{Top}) \in \mathcal{A}a^*$ où $\mathcal{A}a^*$ est la fermeture transitive de $\mathcal{A}a$.

• $\forall\ (N1, N2) \in \mathcal{A}a, N1 = (ST1\ ;\ LNC1) \in \mathcal{N}a$,

$N2 = (ST2\ ;\ LNC2) \in \mathcal{N}a$,

$ST1 = (a : T1), ST2 = (a : T2)$

$T1 \leq_\tau T2$.

3.4.2. Calcul du graphe des types d'un attribut

Le graphe des types est calculé, point de vue après point de vue, des feuilles du graphe de spécialisation de chaque point de vue vers son sommet. Le principe du calcul du graphe dans un point de vue *PV* est le suivant :

— si l'attribut *a* n'existe pas (ou n'est pas hérité) dans la classe *C*, alors toutes les super-classes de *C* sont marquées *a*-vide. Ainsi, l'algorithme ne prend pas en compte pas les classes marquées *a*-vide qui apparaissent dans le nœud du type ***Top***.

— si l'attribut *a* existe (ou est hérité) dans la classe *C*, l'algorithme recherche sa spécification complète par **FR**[2] et normalise le résultat par **FN** afin d'obtenir le type *T* de *a* dans *C*. Si le type *T* existe déjà dans un nœud du graphe des types, alors *C* est rajoutée dans la liste de classes *LNC* du nœud ; sinon, un nouveau nœud ((*a* : *T*) ; (*C*)) est créé puis les arcs de sous-typage entre ce nouveau nœud et les existants sont calculés (c'est-à-dire que le nœud est inséré dans le graphe des types selon la relation de sous-typage), et le nombre d'arcs est minimisé par fermeture transitive.

A ce stade, le traitement de la classe *C* est terminé, *C* est marquée «examinée» ; l'algorithme passe alors à la classe suivante, qui est soit une feuille (en priorité), soit une super-classe non marquée d'une classe déjà examinée. L'algorithme termine lorsqu'il atteint le sommet du graphe de spécialisation d'un point de vue.

Ce processus itéré sur tous les points de vue du concept fournit le graphe des types général pour un attribut dans un concept.

Pour tous les points de vue *PV* du concept $\mathcal{C}$, faire

• **Pour** chaque classe *C* non marquée dans le graphe de spécialisation du point de vue *PV* (parcours en largeur d'abord, de bas en haut), **faire** :

si *a* n'existe pas (ni hérité, ni modifié, ni défini) dans *C*,

alors marquer *a*-vide *C* et toutes ses super-classes.

$LNC(\mathbf{Top}) \leftarrow LNC(\mathbf{Top}) \cup C \cup Super\text{-}classes(C)$

sinon (*a* existe (hérité, modifié ou défini) dans *C* avec une spécification locale *Aexp*)

[2] Les fonctions **FR** et **FN** sont les deux fonctions principales de typage d'un attribut. Elles sont décrites sous forme de règles d'inférence au dessus d'un langage de représentation duquel ont été extraits les exemples de cet article. La grammaire du langage et les définitions de **FN** et **FR** sont dans [Capponi 93b].

T <— **FN** o **FR** ($Aexp$)
si $N = (T ; LNC)$ existe
alors $LNC(N)$ <— $LNC(N) \cup (C)$
sinon Insérer ($\mathcal{G}_a$, (T ; (C)))
marquer C "examinée"

L'insertion d'un nouveau nœud dans le graphe des types (Insérer($\mathcal{G}$, N)) revient à créer des arcs traduisant la relation de sous-typage entre le nouveau type t à insérer et tous les types du graphe auxquels t peut être comparé. Cette insertion doit être garante de la structure du graphe ; en particulier, elle ne doit pas introduire de circuit, ni d'arcs redondants en respectant la propriété de transitivité de la relation de sous-typage.

4 Exploitation

Les intérêts et possibilités d'exploitation des types et de la structure de graphe pour représenter les raffinements d'un attributs sont multiples et sont abordés dans ce paragraphe, de façon non exhaustive.

4.1. Intérêts principaux du graphe des types

Un grand intérêt du graphe des types d'un attribut a observé sous un seul point de vue est qu'il est compatible avec les différentes positions de a dans le graphe de spécialisation du point de vue considéré (ceci est dû au fait que la relation de spécialisation implique la relation de sous-typage, par définition même de la relation de sous-typage) :

$$\forall (t ; lnc) \in \mathcal{N}_{a,PV}, \forall (t'' ; lnc') \in \mathcal{N}_{a,PV},$$
$$\forall C \in lnc, \forall C' \in lnc', C \leq_{sp} C' \Rightarrow t \leq_{\tau} t'$$

De façon similaire, le graphe des types général d'un attribut contient aussi les informations structurelles sur la relation induite par les passerelles, formalisée par la relation de sous-typage entre attributs communs des classes source et destination.

Par ailleurs, le nombre de nœuds du graphe des types d'un attribut considéré sous un point de vue est inférieur ou égal au nombre de nœuds (classes) du graphe de spécialisation de ce point de vue. En effet, plusieurs classes peuvent donner le même type à un même attribut, mais il n'y a pas plus de types d'un même attribut d'un point de vue que de classes dans ce point de vue. C'est pour cette raison que le graphe des types d'un attribut dans un point de vue n'est pas plongé dans le graphe de spécialisation. Cette constatation met en évidence le fait que tout algorithme contenant un travail de consultation ou de modification des domaines des attributs nécessitant un parcours de graphe, est plus efficace lorsque le graphe de spécialisation est projeté selon les attributs considérés afin d'effectuer les parcours sur les graphes des types.

L'intérêt principal du graphe des types d'un attribut a observé sous un point de vue PV réside donc dans le fait qu'il représente une réduction de la projection du graphe de spécialisation de PV selon a. En effet, la projection du graphe de spécialisation selon un attribut a est isomorphe à ce graphe ; la projection est ensuite réduite par considération d'éventuels types égaux dans des nœuds différents.

Le graphe des types d'un attribut rend alors efficace plusieurs mécanismes intégrés dans un modèle de connaissances par objets, en augmentant la rapidité de parcours, en diminuant le nombre de comparaisons, et en rendant ces comparaisons plus rapides puisqu'effectuées sur des formes normales. Ainsi, les mécanismes de résolution de requêtes, de vérifications de types, de classification de classes (§4.5) voire même de classification d'instances sont améliorés. Pour plus de détails sur la complexité des algorithmes considérés, [Capponi 94] décrit les calculs de complexité et réalise les comparaisons nécessaires.

De façon plus générale, tous les processus de vérification de cohérence structurelle d'une base de connaissances sont simplifiés et améliorés ; en effet, ces processus sont activés suite à la modification de la structure d'une base (ajout, suppression et modification de classe ou de points de vue), et dépendent entre autres de la relation de spécialisation et de la relation induite par les passerelles, projetées sur les attributs et vérifiables structurellement par la relation de sous-typage [Capponi& 93].

4.2. Interactivité

Lors de l'utilisation et la manipulation de bases de connaissances, l'accent est de plus en plus mis sur la puissance d'interaction d'un système [Brachman 92].

Un avantage appréciable du graphe des types est la qualité interactive qu'il offre. En effet, un utilisateur peut apprécier sous une seule structure tous les raffinements d'un même attribut observé sous un point de vue ou de façon générale, sans avoir à se préoccuper ou à subir les autres attributs de la base. Le graphe des types apporte une nouvelle vision du graphe de spécialisation d'un point de vue ou du concept tout entier, ciblée vers un attribut particulier.

Cette structure s'est avérée particulièrement nécessaire lors de la création d'une nouvelle classe, qui se décompose en une succession de définitions d'attributs : la tâche de prise de connaissance des multiples définitions d'un attribut est ainsi facilitée pour l'utilisateur, compte-tenu de l'aspect synthétique du graphe des types d'un attribut.

4.3. Comparaison de graphes de types : application à la comparaison de bases de connaissances

Deux bases de connaissances traitant un même domaine du monde peuvent être amenées à être comparées ou même fusionnées. Indépendamment des instances de deux bases de connaissances B_1 et B_2, les graphes des types des attributs apportent des facilités de comparaisons ou fusions de structures de bases de connaissances.

En effet, évaluer la similarité structurelle entre deux bases de connaissances décrites sous le même modèle se décompose automatiquement dans un premier temps à l'étude de similarités entre les attributs des bases — entités de plus faible granularité. Nous pouvons donc imaginer l'élaboration et la formalisation d'une nouvelle relation, *la relation de similarité*, établie entre deux attributs, formant alors une liaison pondérée par le *degré de similarité*. La détermination de liens de similarités entre deux attributs pourrait se faire au niveau des types de ces attributs, c'est-à-dire en étudiant les relations entre graphes de types (recherche de plongement de graphes, de recouvrements, d'équivalences, mises en évidence d'homomorphismes entre types d'attributs...) qui parfois s'avèrent sémantiquement intéressantes et révélatrices à un niveau plus concret.

La mise en œuvre de tels processus nécessite bien évidemment un formalisme supportant leurs preuves et expressions. L'explicitation des types est une approche naturelle compte-tenu de leur importance dans le modèle considéré où beaucoup de mécanismes sont basés sur les domaines de valeurs.

4.4. Graphe des types d'un attribut et hiérarchie de rôles

Il existe une similarité entre le graphe des types considéré dans ce modèle et la hiérarchie des rôles établie dans les langages terminologiques. En effet, la relation de subsomption [Brachman& 85] existe entre spécifications de rôles, ce qui permet d'établir des comparaisons entre rôles n'ayant pas nécessairement la même sémantique. Le graphe des types est quant à lui construit sur un attribut de nom donné, c'est-à-dire un attribut ayant une seule signification dans la base de connaissances.

Cette ressemblance est révélatrice dans le sens où elle montre que la subsomption est plus générale que la relation de sous-typage dont nous avons besoin pour vérifier ou déterminer des liens de spécialisation.

En effet, le graphe des types a été initialement introduit afin de mettre en œuvre la classification des classes (§4.5), c'est-à-dire la recherche de liens de spécialisation. L'approche utilisée étant la projection de la classification sur les types des attributs, la réunion des graphes des types n'est pas pertinente puisque nous considérons les attributs indépendamment les uns des autres.

Il est toutefois envisagé d'étendre le graphe des types d'un attribut à un graphe regroupant tous les types de tous les attributs de la base de connaissances, leur nom jouant un rôle secondaire. Cette extension a plusieurs motivations qui ne font pas partie du sujet de l'article, parmi elles la possibilité de mettre en évidence la redondance, la complémentarité et la contradiction entre deux attributs différents.

4.5. Application à la classification des classes

A partir d'un certain nombre de classes contenues dans un concept, la description et le positionnement d'une classe dans ce concept devient irréalisable sans assistance du système. Le rôle du processus de classification d'une classe est de déterminer automatiquement la position d'une description de classe dans un ou plusieurs points de vue du concept, en accord avec la relation de spécialisation telle qu'elle a été définie dans §2.2. La classification d'une classe se fera interactivement dans un seul point de vue à la fois, au fur et à mesure de sa description dans ce point de vue. Eventuellement, l'utilisateur pourra demander ultérieurement sa classification dans un autre point de vue.

$$\sup(a, C)_{PV} = \bigcup_{\substack{L \in LNC \\ (t, LNC) \in \mathcal{N}_{a,PV} \\ t \geq_\tau TC(a)}} L$$

$$\inf(a, C)_{PV} = \bigcup_{\substack{L \in LNC \\ (t, LNC) \in \mathcal{N}_{a,PV} \\ t \leq_\tau TC(a)}} L$$

Figure 4. $Tc(a)$ est le type de l'attribut a dans la classe C.

La classification d'une classe C revient dans un premier temps à la projection de la description de C selon les attributs qu'elle contient ; dans un second temps, le type de chaque attribut a de C, préalablement calculé, est inséré dans son graphe des types $\mathcal{G}_{a,PV}$, et la position de C relativement à a est récupérée dans ce graphe à partir des expressions de la figure 4.

La définition structurelle d'une classe C étant donnée par la spécification statique de tous ses attributs, la position de C dans le graphe de spécialisation d'un point de vue est calculée par intersection des positions relatives à chacun des attributs de C. Parmi ces solutions, certaines ne seront pas retenues, ne vérifiant pas la propriété d'héritage complet qui stipule qu'une classe doit contenir au moins tous les attributs de sa super-classe. Ainsi, le calcul de la position finale

possible d'une classe C est réalisé grâce aux expressions suivantes :

$$\sup(C)_{PV} = \left[\bigcap_{a \in A(C)} \sup(a, C)_{PV} \right] \cap \{ C' / A(C') \subseteq A(C) \}$$

$$\inf(C)_{PV} = \left[\bigcap_{a \in A(C)} \inf(a, C)_{PV} \right] \cap \{ C' / A(C) \subseteq A(C') \}$$

Où $A(C)$ est l'ensemble des attributs de C.

Ce résultat fournit donc la position possible de la classe, compte-tenu de sa description. Toutefois, $\sup(C)_{PV}$ peut contenir plusieurs super-classes non liées par la relation de spécialisation. Sans critère de choix, le système n'a aucun moyen théorique de préférer une super-classe à une autre : dans le cas de l'héritage simple tel que nous le supposons dans le modèle, ce choix doit être effectué par l'utilisateur. Quant à la décision concernant les sous-classes, elle ne peut pas non plus être prise automatiquement par le système qui dans un premier temps retient, sur demande de l'utilisateur, les sous-classes les plus générales non liées par la spécialisation, mais doit dans un second temps s'assurer que ces sous-classes n'ont pas, par conséquent, deux super-classes, ce qui peut arriver. Dans ce cas, c'est à l'utilisateur de décider quelle sera la super-classe pertinente.

Une étude plus approfondie de la classification des classes est fournie dans [Capponi& 93]. Une comparaison de l'algorithme de classification de classes avec les "classifieurs" des langages terminologiques [Brachman& 91], [Schmolze& 83] et les algorithmes de classification d'instances [Mariño& 90], est abordée dans [Capponi 93]. Une analyse plus précise est faite dans [Euzenat 93]. Ces comparaisons sont principalement axée vers les similitudes entre la relation de subsomption, O-subsomption [Napoli& 92] et spécialisation.

5 Réalisation et travaux futurs

Le système de types défini dans §2 et la relation de sous-typage explicitée dans §3 ont été mis en œuvre dans le modèle de représentation des connaissances par objets Shirka (chap.8 de [Masini& 89]), en Le-Lisp, et testé via l'algorithme de classification de classes utilisant la notion de types, sur des bases de connaissances contenant plus de 700 classes. L'interface développée parallèlement a montré le réel avantage interactif du graphe des types.

L'élaboration d'un système de types plus conséquent est en cours dans le modèle multi points de vue TROPES, sur lequel sont aussi en cours de réalisation les algorithmes de gestion de la dynamique des classes permettant d'assurer la cohérence d'une base de connaissances suite à des modifications d'ordre structurel.

D'autre part, les apports de la relation de sous-typage sont étudiés pour une extension à la formalisation de la relation de composition entre objets, largement présente dans le modèle TROPES.

La fusion et la comparaison de bases de connaissances est aussi une perspective sérieusement envisagée, d'autant plus que le nombre d'applications existantes dans un même domaine va croissant, et ainsi des confrontations de modélisations peuvent, dans peu de temps, s'avérer utiles sinon nécessaires.

6 Conclusion

Les langages à objets en général utilisent les types comme formalisme, en partie du fait de la propriété commune aux objets et aux types qui est l'abstraction de données, et donc les similitudes existants entre ces notions. Les modèles de représentation des connaissances doivent intégrer de la même façon un formalisme adapté aux entités et aux processus définis. Par exemple, les langages terminologiques (LT) sont proches de la logique sur laquelle est fondée l'expression des termes. Pourtant, certains travaux dans le domaine des LT [Borgida 92] ont montré l'intérêt des types pour l'étude de propriétés inhérentes à un modèle de frames.

Pour répondre à un besoin d'intégration d'un formalisme adapté au modèle de représentation choisi, cet article a dégagé la notion de types implicitement contenus dans les objets manipulés. Ce travail a été réalisé sous un modèle multi-points de vue proche du modèle TROPES, dont les composantes principales de modélisation sont (§2.1) les concepts, les points de vue, les classes, les instances, les attributs de définition et d'inférence, la relation inter-points de vue (§2.3), et la relation de spécialisation (§2.2), critère principal de structuration hiérarchique d'une base.

Le développement d'un système de types (§2.4) parallèlement au langage de manipulation doit permettre la définition stricte et claire des entités et algorithmes existants dans le modèle. Les types sont d'autant plus adaptés qu'ils ne confèrent pas aux données une identité ; ils sont utilisés essentiellement pour mettre en œuvre des mécanismes de vérification de cohérence structurelle, pour accélérer des mécanismes d'inférence, mais ils ne sauraient être une structure de base pour la représentation et la manipulation des connaissances. Un type tel que nous l'entendons permet de représenter sous une forme normale des données dégagées de toute signification ; cette forme normale, doublée d'une nécessité de hiérarchisation inhérente au modèle, introduit la relation de sous-typage (§3) qui permet d'ordonner les données selon un critère formellement défini. Ce critère varie selon ce que

le sous-typage doit formaliser. Lorsqu'il s'agit de représenter la relation de spécialisation, le sous-typage doit traduire la relation d'inclusion ensembliste entre les domaines de valeurs des attributs qui définissent la structure des classes mises en relation. L'élaboration d'une relation de sous-typage en accord avec la relation de spécialisation permet alors d'observer le graphe de spécialisation d'un point de vue projeté et réduit selon un seul attribut du concept, sous une structure appelée le *graphe des types d'un attribut*, qui représente tous les raffinements de domaines d'un attribut dans le point de vue considéré. D'une manière générale, les notions de types et de sous-types seront étendues à la formalisation de tous les liens sémantiques entre entités relevant de la comparaison de domaines de valeurs et de structures de données élaborées. En particulier, la relation structurelle entre deux points de vue, appelée passerelle, a été formalisée par les types, et ce travail sera étendu à la relation de composition, de façon à mettre en œuvre des processus de validation et de vérification de cohérence structurelle complets.

Le graphe des types d'un attribut offre de nombreux avantages (§4) dont nous retiendrons principalement la possibilité d'une mise en œuvre formelle du mécanisme de classification de classes par décomposition de la classe selon ses attributs. Le graphe des types apporte, en outre, une assistance à l'utilisateur dans les phases de descriptions incrémentales des attributs, donc d'une classe, en donnant le maximum d'informations statiques sur les attributs et en évitant des définitions imprécises, redondantes ainsi que des conflits.

Une perspective intéressante et prometteuse du graphe des types est la possibilité d'activer des algorithmes de comparaison, de fusion, de recherches de plongements et de recouvrements entre graphes, afin d'étendre ces algorithmes existants et de les appliquer aux structures des bases de connaissances. Ces possibilités vont devenir en effet importantes dans divers domaines d'applications nécessitant des confrontations de modélisations fondées sur des structures formellement définies.

References

[Amadio& 91] R.M. Amadio, L. Cardelli, *Subtyping Recursive Types*, proc. of 18th POPL, Orlando (FL, US), 1991.

[Borgida 92] A. Borgida, *From type systems to knowledge representation: natural semantics specifications for description logics*, Int. Jour. of Intelligent and Cooperative Information Systems, vol. 1 n°1, pp 93-126, april 1992.

[Brachman& 91] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L. A. Resnick, A. Borgida, *Living with CLASSIC, When and How to Use a KL-ONE-Like Language*, Sowa (ed), Principles of Semantics Networks, Explorations in the Representation of Knowledge, Morgan Kaufmann Publisher, chap. 14, pp 401—456, 1991.

[Brachman& 85] R.J. Brachman, J.G. Schmolze, *An overview of the KL-ONE Knowledge Representation System*, Cognitive Science n°9, 1985.

[Brachman 92] R.J. Brachman, *Reducing CLASSIC to Practice : Knowledge Representation Theory Meets Reality*, proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge (MA, US), October 1992.

[Capponi 93] C. Capponi, *Classification des classes par les types*, actes de la journée Classification, RPO, La Grande-Motte (FR), pp 215—224, june 1993.

[Capponi 94] C. Capponi, *Exploitation des types dans un modèle de représentation à objets*, rapport de recherche INRIA Rhône-Alpes, à paraître, 1994.

[Capponi& 93] C. Capponi, M. Chaillot, *Construction incrémentale d'une base de classes correcte du point de vue des types*, actes de la journée Validation, JAVA, St-Raphaël (FR), march-april 1993.

[Cardelli& 91] L. Cardelli, J. C. Mitchell, *Operations on records*, Math. Struct. in Comp. Science, vol.1 n° 1, pp 3-48, March 1991.

[Euzenat 93] J. Euzenat, *Définition abstraite de la classification et son application aux taxonomies d'objets*, actes de RPO, La Grande-Motte (FR), pp 235—246,june 1993.

[Finin 86] T.W. Finin, *Interactive Classification : a Technique for Acquiring and Maintaining Knowledge Bases*, proceedings of the IEEE, vol 74, n°16, october 1986.

[Mariño& 90] O. Mariño, F. Rechenmann, P. Uvietta, *Multiple Perspectives and Classification Mechanism in Object-Oriented Representation*, Proceedings of the 9th ECAI, Stockolm (Sweden), august 1990.

[Masini& 89] G. Masini, A. Napoli, D. Colnet, D. Léonard, K. Tombre, *Les Langages à Objets*, InterEditions, collection Informatique Intelligence Artificielle, 1989.

[Napoli& 92] A. Napoli, R. Ducournau, *Subsumption in Object-Based Representations*, proc. of ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation, Pisa (Italy), may 1992.

[Nebel 91] B. Nebel, *Terminological Cycles*, Sowa (ed), Principles of Semantics Networks, Explorations in the Representation of Knowledge, Morgan Kaufmann Publisher, chap. 1, pp 331—361, 1991.

[Schmolze& 83] J.G. Schmolze, T.A. Lipkis, *Classification in the KL-ONE Knowledge Representation System,* proc. of the 8th IJCAI, Karlsruhe (Germany) 1983.

[Wegner& 88] P. Wegner, S.B. Zdonik, *Inheritance as an Incremental Modification Mechanism, or What Like Is and Isn't Like*, proc. of ECOOP'88, Lecture Notes in Computer Science, vol 322, pp 55—77, 1988.