# Représentation de connaissance par objets*

**Jérôme Euzenat**

*INRIA Rhône-Alpes,*
*655 avenue de l'Europe, 38330 Montbonnot Saint-Martin (France)*
*Jerome.Euzenat@inrialpes.fr*

RÉSUMÉ : *Les systèmes de représentation de connaissance sont utilisés pour modéliser symboliquement un domaine particulier. Certains d'entre eux utilisent la notion d'objet comme structure principale. On trace ici les traits principaux de tels systèmes, en évoquant les systèmes marquants. L'exposé approfondit ensuite un système particulier, TROEPS, abordant d'abord les problèmes que la conception de ce système cherche à résoudre. TROEPS est présenté en considérant les constructions et les mécanismes d'inférence qu'il met en œuvre.*
MOTS CLÉS : *Représentation de connaissance, classification, filtrage, type, WWW, hypertexte, logiques de descriptions, réseaux sémantiques, schémas, identité, nommage, inférence, évolution, spécialisation, points de vue, passerelles.*

Les objets sont utilisés non seulement pour la programmation mais aussi pour la représentation de connaissance. Cette perspective est présentée ici tout d'abord au travers de son histoire mais surtout au travers de ses traits particuliers. Ces derniers sont illustrés par la conception du système TROEPS et ses particularités. Cependant, plus que sur un langage précis, c'est sur la variété (le mot polymorphisme étant interdit ici) qu'insistera cette présentation. Pour certains problèmes posés par la conception de systèmes à objets, il existe plusieurs solutions. En ce qui concerne les représentations par objets, les solutions à retenir ne sont pas forcément celles des langages de programmation ou des bases de données. Ce point est en particulier discuté.

Le premier chapitre de [Stefik 1995] est un complément utile à cette présentation en ce qu'il met en perspective les systèmes de représentation de connaissance par objets au travers de leur utilisation dans une application.

## 1. INTRODUCTION

### 1.1 Motivations

Le but de la représentation de connaissance est de rendre compte d'un « domaine » particulier de telle sorte que cette représentation soit manipulable par une machine. Elle sert, en

* Version du chapitre homonyme de [Ducounau& 1998] (pp293-319) étendue et mise à jour (17/05/99 13:22) disponible sous `ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat98b.ps.gz`. D'abondantes références sont faites aux autres chapitres du livre.

particulier, dans les *systèmes à base de connaissance*. Ce but qui dirige une partie des travaux sur les représentations de connaissance ne peut être validé de manière théorique (le « domaine » étant inaccessible à la théorie). Ce but n'est pas non plus distinctif, il peut être revendiqué par d'autres disciplines (bases de données par exemple). Par contre, il est forcément trahi lorsque les objets sont utilisés pour se représenter eux-mêmes (si l'on peut dire que la pile — au sens informatique — implémentée en C++ se représente elle-même, il n'est plus possible d'affirmer quoi que ce soit d'intéressant sur cette représentation : la représentation est forcément correcte puisque l'objet représente ce qu'il est).

Les représentations de connaissance offrent donc des langages permettant de modéliser le « domaine ». Un panorama plus complet de ce type de langage peut être trouvé dans [Kayser 1997]. Les constructions de ce langage ayant un sens particulier, elles peuvent être manipulées par un ordinateur en respectant ce sens et, par conséquent, la cohérence interne de la modélisation du « domaine ». Ainsi, les employés d'une entreprise, les bureaux et les équipes pourront être représentés par des objets et jamais un bureau ne pourra être utilisé à la place d'une équipe. Dans les développements qui sont présentés ici, et dans d'autres [Ducournau 1998], le rapport entre ce qui est représenté et les expressions du langage de représentation de connaissance est analysé au moyen d'une sémantique du langage de représentation. Celle-ci permet de justifier la validité des opérations du système de représentation de connaissance par rapport au sens des expressions introduites dans le système.

## 1.2 Histoire

Historiquement, les modèles généraux de représentation de connaissance ont été nombreux. Par général, on entend un modèle de représentation qui peut être appliqué à toute sorte d'entités (objets physiques, idées, actions, relations…) par opposition à un modèle qui se concentre sur un domaine particulier (les relations temporelles, les tâches, les taxonomies biologiques [Lebbe 1998]). C'est à ce genre de modèles généraux qu'est consacré cette présentation.

L'un des premiers systèmes de représentation de connaissance générique est le modèle des *réseaux sémantiques* [Quillian 1968]. Il se caractérise par une organisation de la connaissance sous forme d'un graphe orienté dont les nœuds et les arcs sont étiquetés. Il est doté d'un mécanisme d'inférence : la propagation de marqueurs ("spreading activation", poussée à ses limites dans [Falhman 1979]) qui consiste à parcourir le graphe dans le sens des arcs à partir d'un nœud précis (ou de deux nœuds). Il permet donc formellement de déduire un ensemble de nœuds accessibles (clôture transitive) à partir d'un ou de plusieurs nœuds. Il est limité car le modèle ne distingue pas *a priori* d'étiquettes particulières alors qu'il serait utile de distinguer certains arcs (par exemple, les relations générique-spécifique) ou certains nœuds (en tant qu'individu ou espèce). Par ailleurs, la signification des arcs et des nœuds n'est pas précisée clairement [Woods 1975]. Par exemple, l'arc `marié-à` entre le nœud `homme` et le nœud `femme` signifie-t-il qu'un homme est marié à une femme, que tous les hommes sont mariés à toutes les femmes…? Bref, le principal grief à l'encontre des réseaux sémantiques est leur absence de sémantique justement.

Le modèle ultérieur est celui des *schémas* "frames" [Minsky 1974 ; Masini& 1989] introduit dans le contexte de la représentation de connaissance acquise par la vision. Ce modèle

organise la connaissance autour de la notion de schéma : un nom auquel est associé un ensemble d'attributs. Chaque attribut se voit associer à son tour un ensemble de *facettes* permettant de le caractériser. Cette notion de facette inspirée de la métaphore des différentes faces sous lesquelles il est possible de voir un objet a été utilisée à des fins multiples et variées : rendre compte de la position par rapport à un observateur, d'un rôle joué par l'objet ou de différentes acceptions du nom de l'attribut. Les facettes, à leur tour, contiennent des *valeurs*. La *mise en correspondance* (ou *appariement*, "matching") est le moyen d'inférence privilégié sur les schémas. Il consiste à comparer deux objets en fonction de leurs facettes et valeurs pour vérifier si elles correspondent ou non (si l'un est plus complet que l'autre ou si les deux peuvent être complétés de manière compatible). C'est un moyen très puissant pour comparer les objets. Par exemple, si un robot voit une voiture rouge garée devant lui et qu'on lui a demandé de se garer derrière une voiture rouge de marque Renault, il peut apparier les deux objets (car ils sont tous les deux rouges et garés, ce qu'il ne peut faire avec les voitures bleues) et poursuivre sur l'hypothèse que c'est derrière l'auto qu'il voit qu'il doit se garer.

Les modèles des schémas et des réseaux sémantiques ont été rationalisés dans un travail de longue haleine par Ronald Brachman (sous l'impulsion de William Woods [Brachman 1977, 1979, 1983, 1985]). Ce travail isole un certain nombre de problèmes de ces modèles qui doivent être impérativement tranchés. Dans sa thèse [Brachman 1977], ainsi que dans plusieurs articles publiés plus tardivement [Brachman 1983, 1985], Ronald Brachman énonce de précieux principes de bonne utilisation de ces systèmes (qui seront précisés plus loin) :

— ne pas considérer la même entité, tantôt comme un individu (une auto rouge particulière), tantôt comme un ensemble d'individus (la classe des autos rouges) ;
— ne pas confondre propriétés descriptives (les oiseaux doivent avoir des plumes) et propriétés typiques (typiquement les oiseaux volent) ;
— ne pas confondre définition (des facettes sur les attributs exprimant des conditions nécessaires et suffisantes pour appartenir à la classe), description (les conditions sont seulement nécessaires) et prototype (les valeurs données par les facettes sont uniquement typiques).

Cependant, dans le même temps, les développeurs continuent à implémenter de nouvelles idées. Ainsi, vers la fin des années 1970, des systèmes de représentation de connaissance par objets que nous qualifierons de "post-frames" voient le jour (KRL [Bobrow& 1977], FRL [Roberts& 1977], SRL [Wright& 1983]). Ces systèmes sont inspirés à la fois des schémas et des réseaux sémantiques mais aussi des langages de programmation par objets. Ils sont influencés par les travaux autour des systèmes de fenêtres des langages et machines LISP qui eux-mêmes le sont par SMALLTALK. Ils se caractérisent par l'adoption de la notion de classe (c'est-à-dire qu'ils distinguent les classes des instances) et d'un lien générique-spécifique fort (c'est-à-dire dont le sens est inclus dans le système), ainsi que par une rationalisation de l'utilisation des facettes. Ces dernières sont conservées en nombre restreint (mais parfois extensible) à des fins de typage ou d'inférence des valeurs d'attributs. De plus, la notion d'envoi de message est adoptée par certains et ils utilisent parfois un méta-modèle. Certains d'entre eux conservent cependant une notion de prototype. La programmation par réflexes a, par ailleurs, un certain succès. Ces systèmes ont connu une diffusion importante sous la forme de produits commerciaux qui ont été utilisés dans le monde entier (KNOWLEDGE CRAFT [Carnegie

group 1988], ART [Williams 1984], KEE [Filman 1988], NEXPERT OBJECT). Ils ont eu des successeurs en France sous la forme de prototypes de recherche (SHIRKA [Rechenmann 1985], OBJLOG [Dugerdil 1988]…) comme de produits commerciaux (KOOL [Albert 1984], SMECI [Ilog 1991], YAFOOL [Ducournau &1986, 1991]). Une comparaison de ces systèmes se trouve dans la table à la fin de la section 2. À côté, et parfois au sein, de ces systèmes pour lesquels la notion d'objets est centrale, se sont développés des systèmes permettant d'exploiter conjointement des règles et des objets. Ces systèmes sont soit des extensions de systèmes d'objets existant (par exemple OPUS [Atkinson& 1987]), soit des extensions de systèmes de règles existants (par exemple CLIPS).

Parallèlement, des recherches plus fondamentales ont été poursuivies. Dans la lignée des travaux de Brachman, mais renforcées par un courant formel (Levesque, puis bientôt Borgida et Nebel pour nommer les plus notables), les représentations de connaissance par objets évoluent vers une formalisation croissante qui conduit à définir la sémantique dénotationelle de ces langages [Ducournau 1998], à évaluer la décidabilité et la complexité de certains mécanismes de déduction et à concevoir des algorithmes corrects pour ces mécanismes. Un regret peut cependant être formulé : alors que de nombreux choix de langages sont possibles, les chercheurs se focaliseront sur un seul type de langages nommés maintenant logiques de descriptions [Napoli 1998]. Dans la même veine, mais plus directement inspiré des réseaux sémantiques, le modèle des *graphes conceptuels* défini par John Sowa [Sowa 1984] se voit doté d'une sémantique formelle [Chein& 1992].

### 1.3 Situation actuelle

Si la sémantique des langages est une préoccupation de certains développeurs de systèmes depuis très longtemps, elle a mis du temps à s'imposer en tant que telle. Ainsi, depuis les premiers travaux relativement peu formels de Ronald Brachman (voir ci-dessous) jusqu'aux logiques de descriptions [Napoli 1998], une lente évolution a pris place. Il y a d'abord des systèmes dont les développeurs ont conçu le comportement de la manière la plus raisonnée possible, puis des théories rendant compte de ces systèmes et mettant en évidence leurs déficiences. Par ailleurs, l'effort de formalisation a porté sur une simplification des systèmes existants. Ainsi, de nombreuses opérations et innovations (souvent très utiles) sont laissées dans l'ombre par le travail de formalisation. En conséquence, même les systèmes développés à partir des travaux théoriques offrent des traits qui sortent de la théorie et qui rendent le système utile. Il est donc important de s'intéresser à toutes les fonctions d'un système de représentation de connaissance et non uniquement à celles qui sont dûment formalisées. Au travers d'un exemple récent, les fonctionnalités et les problèmes des représentations de connaissance par objets seront décrites ici. Les aspects théoriques pourront être étudiés ailleurs [Euzenat 1998].

Actuellement les langages les mieux définis sont, sans conteste, les logiques de descriptions suivies par les graphes conceptuels. Mais les modèles qui ont été les plus utilisés sont les systèmes de représentation de connaissance "post-frames" offrant en général plus d'expressivité, mais dont il est difficile de cerner les aspects théoriques. Dans ce qui suit on va s'attacher à présenter ces systèmes. Il existe une manière de les présenter qui laisse une large place aux aspects d'implémentation [Karp 1993] : faut-il représenter les objets par des vecteurs

ou des listes de propriétés, faut-il conserver les classes lors de l'exécution et opérer une manipulation dynamique des objets et des classes ou compiler statiquement les accès aux objets et supprimer les classes lors de la compilation? Ici on s'attachera plutôt aux principes qui gouvernent les représentations de connaissance par objets, sachant qu'il est inutile de comparer les implémentations autrement qu'en comparant les performances (en temps et en occupation de la mémoire).

Les logiques de descriptions sont présentées plus en détail ailleurs [Napoli 1998], ainsi que les travaux en cours pour décrire de manière formelle ces systèmes de représentation de connaissance par objets [Ducournau 1998, Euzenat 1998]. Leurs principes sont d'abord rapidement brossés (§2) avant de focaliser sur un système particulier : TROEPS. TROEPS sera présenté tout d'abord au travers des choix faits dans certaines alternatives de modélisation (§0) avant de décrire son langage de description (§4), les inférences possibles (§5) et ses interfaces (§6).

## 2. PRINCIPES DES REPRÉSENTATIONS DE CONNAISSANCE PAR OBJETS

Il n'y a pas, à proprement parler, de principes uniques, généraux et inflexibles des représentations de connaissance par objets. On peut cependant tracer à gros traits ce qu'est un système typique. Tout d'abord, la fonction d'un tel système est :

— de stocker et d'organiser la connaissance autour de la notion d'objet ;
— de fournir des services inférentiels de bas niveau destinés à compléter l'information disponible.

La présentation des principes sera donc organisée suivant ces deux axes.

### 2.1 Structure

La structure sur laquelle se fonde un système de représentation de connaissance par objets est très proche de celle d'un langage de programmation par objets [Masini& 1998, Rousseau 1998]. À l'instar des langages "post-frames", les objets sont décrits à l'aide de classes organisées dans une hiérarchie de spécialisation. La connaissance est organisée autour de la notion d'objet. Elle est ici brièvement rappelée.

Les objets sont des ensembles de couples attributs-valeurs associés à un identifiant (voir Figure 1). En général cette identification se fait à l'aide d'un nom (que celui-ci soit extérieur aux couples attributs-valeurs ou la valeur d'un attribut particulier). La valeur d'un attribut peut soit être un objet, soit être une valeur d'un type primitif du langage (par exemple, une chaîne de caractères ou un entier). Elle peut être connue ou pas. Souvent les attributs peuvent contenir une collection (par exemple, un ensemble) de valeurs. Il est très important de définir le sens de telles collections, par exemple le fait que l'attribut `membres` de la classe `projet` soit un ensemble peut signifier qu'il s'agit :

— d'un ensemble fermé d'objets (par exemple l'ensemble des `membres` est exactement ceux connus du système) ;
— d'un ensemble ouvert d'objets (l'ensemble des `membres` connus du système est inclus dans l'ensemble des `membres` réels) ;

— d'un ensemble d'objets possibles (on peut alors parler du membre d'un `projet` et celui-ci pourra être l'un des `membres` connus du système).

La première approche est souvent utilisée dans les représentations "post-frames" (inspirée des langages de programmation), la seconde dans les logiques de descriptions et la dernière dans les systèmes plus proches de l'esprit des schémas.

Les objets sont regroupés en *classes* qui, à l'instar de celles des langages de programmation par objets [Masini& 1998], permettent de regrouper des traits communs à ces objets. Contrairement au modèle original des schémas, seules les classes associent des facettes (ou descripteurs) aux attributs et seul un ensemble restreint de descripteurs est autorisé et clairement identifié (même si celui-ci est parfois extensible par la programmation — SRL [Wright& 1983], YAFOOL [Ducournau& 1986, 1991]). Dans les systèmes les plus restreints, elles servent principalement à deux objectifs : préciser les valeurs d'attributs admissibles dans une classe (*typage*) et déclarer des mécanismes capables de déduire la valeur d'un attribut manquant (*inférence*). Les systèmes les plus riches offrent un ensemble plus complet de facettes permettant de spécifier des comportements lorsqu'une valeur est écrite ou lue (« programmation dirigée par les accès » [Stefik& 1986]), de gérer des relations entre objets (réciproque par exemple [Ducournau 1991, Fornarino& 1990, Wright& 1983, Albert 1984]) ou d'assurer la liaison avec le monde extérieur (une interface graphique par exemple [Bobrow& 1981, Ilog 1991, Ducournau 1991]). On se limitera ici à ce qui concerne l'aspect représentation de connaissance : typage et inférence.

```
<employé
  attributs = {
    <nom type = chaine; constructeur = un;>,
    <prénom type = chaine; constructeur = un;>,
    <date-naissance type = date; constructeur = un;>,
    <diplômes type = diplôme; constructeur = liste;>,
    <projet type = projet; constructeur = un;>,
    <chef type = employé; constructeur = un;>,
    <subordonnés type = employé; constructeur = ensemble;>,
    <bureau type = bureau; constructeur = un;>,
    <salaire  type = entier;
              constructeur = un;
              intervalles = { [5000 20000], [23000 23000] };>,
    <régime type = chaine; constructeur = un;>};>

<chercheur
  sorte-de = employé;
  attributs = {
    <chef type = { cadre };>};>

<chercheur
  nom       = "Corrine";
  salaire   = 13000;
  chef      = <employé Pierre>;>
```

Figure 1 : Une classe, une sous-classe et un objet.

Les facettes de typage permettent de restreindre les valeurs possibles d'un attribut en précisant les valeurs admissibles par une énumération (`domaine`) ou une réunion d'intervalles (`intervalles`), introduisant des exceptions (`sauf`) ou restreignant la cardinalité dans le cas de collections (`cardinal`). Lorsque la valeur d'un attribut peut être un autre objet, les restrictions

peuvent se faire soit en précisant les classes auxquelles l'objet doit appartenir (`type`) soit grâce à un filtre permettant de n'accepter que les objets qui satisfont ce filtre (`filtre` — voir ci-dessous). La Figure 1 présente deux définitions de classes.

Les classes sont organisées par la relation de *spécialisation* (parfois nommée relation d'héritage ou de généricité). La représentation de connaissance par objets privilégie donc au moins ce lien particulier entre les objets. La spécialisation est une relation d'ordre. Dans certains systèmes, le graphe de la réduction transitive de cette relation peut être restreint à un arbre, contraint à être connexe ou à disposer d'une unique source (racine dans le cas d'un arbre). La sémantique de cette relation est celle de l'inclusion ensembliste : les individus appartenant à l'interprétation d'une classe doivent appartenir à celle de ses super-classes. On dispose donc d'un ensemble de sous-ensembles imbriqués (voir Figure 2).

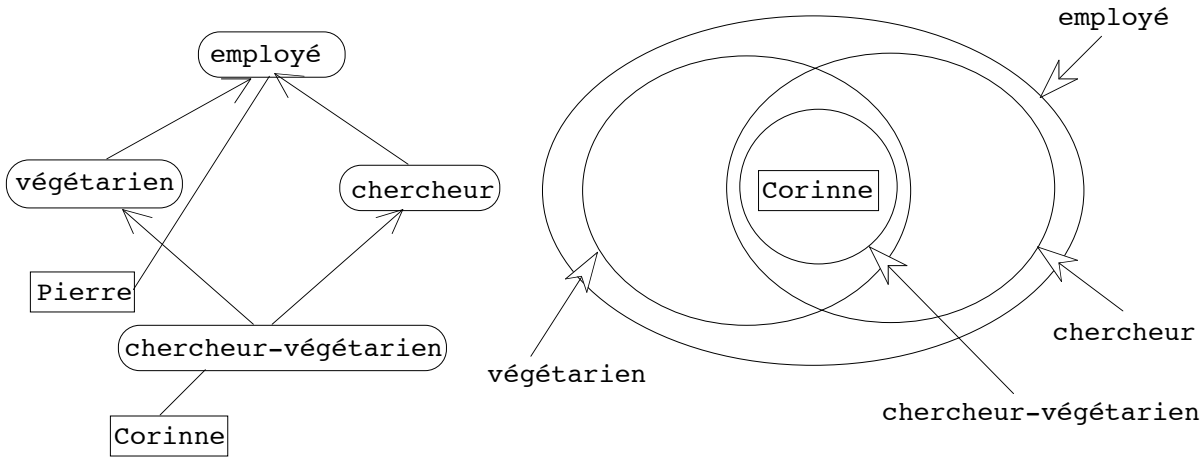


Figure 2 : Signification des relations entre classes. Les boîtes rectangulaires représentent les instances, les boîtes arrondies les classes. Les traits sans flèche représentent l'instanciation et les flèches vides la spécialisation. Le dessin de droite est une représentation ensembliste de la situation. Il faut noter que Pierre n'y figure pas car si l'on sait qu'il appartient à l'ensemble employé, on ignore s'il fait partie d'un de ses sous-ensembles.

De cette interprétation de la relation de spécialisation, tout découle :

— Plus une classe est spécifique, plus les domaines de ses attributs doivent être restreints ; si une classe est sous-classe de deux classes, les domaines de ses attributs sont forcément inclus dans (ou égaux à) l'intersection des domaines de ces deux classes. Ceci permet de faciliter l'expression du domaine de valeur des attributs (que l'on qualifiera d'*effectif*) en ne précisant dans les sous-classes que les restrictions par rapport à la super-classe (on parlera alors de domaine *exprimé*). Par exemple, s'il y a une contrainte sur le salaire des chercheurs, celle-ci pèse aussi sur le salaire des chercheurs végétariens : elle fait partie du domaine effectif de l'attribut salaire de la classe `chercheur-végétarien` mais il est inutile de l'exprimer.

— Si un objet est instance d'une classe, il l'est aussi de toutes ses super-classes. Si, par exemple, une base de connaissance décrit la classe des comptables, représentant les employés qui sont des comptables, et la classe des agents administratifs, représentant les employés qui sont agents administratifs, et que la première est sous-classe de la seconde, alors il sera impossible d'être un objet de la classe comptable sans être un agent administratif.

Par contre, et c'est l'un des enseignements de Brachman, il n'est pas question d'introduire des exceptions dans les domaines de classes car cela permettrait inévitablement de définir « un éléphant comme un singe qui ne grimpe pas aux arbres » [Brachman 1985].

Dans ce cadre (celui de la structure) certains problèmes n'apparaissent pas. C'est le cas des problèmes de conflits, dus à la *multi-généralisation* — aussi nommé héritage multiple ou multi-spécialisation —, c'est-à-dire le fait d'avoir plusieurs super-classes incomparables par la relation de spécialisation qui sont très discutés dans le monde de la programmation par objets [Ducournau& 1995]. C'est aussi le cas des problèmes de non monotonie traités par l'intelligence artificielle. Ainsi, si l'utilisateur exprime que les `végétariens` sont (ont pour valeur de l'attribut `moyens`) `pauvres`, que les `chercheurs` sont `riches` et qu'il existe des individus à la fois `chercheurs` et `végétariens`, il s'est tout simplement trompé quelque part.

## 2.2 Inférence

Les représentations de connaissance par objets ne proposent pas forcément de messages ni de méthodes. Lorsqu'ils en offrent[1], leur fonctionnement est le même que dans les langages de programmation par objets [Masini& 1998].

Un des traits qui différencie les langages de programmation des représentations de connaissance par objets est l'existence de mécanismes d'inférence permettant de calculer une valeur pour un attribut lorsque celle-ci n'est pas connue, c'est-à-dire lorsqu'elle n'est pas présente dans l'objet. Ces mécanismes sont attachés aux classes au moyen de facettes d'inférence. Les moyens d'obtenir cette valeur peuvent être très nombreux et toutes les méthodes de l'informatique (fonctions, procédures, règles, tâches, contraintes…) peuvent être mises à contribution. On se bornera ici à évoquer des mécanismes qui s'intègrent bien dans le contexte de la représentation de connaissance par objets. Indépendamment du calcul de la valeur, il est important de déterminer en fonction de quoi celle-ci est calculée. Une des règles en vigueur, idéalement, dans les représentations de connaissance par objets est que la procédure se borne à calculer cette valeur (elle ne fait pas d'effets de bord) et n'est pas influencée par autre chose que par l'objet lui-même (et indirectement par les objets auxquels il fait référence). Cette fonction, au sens informatique, s'interprète comme une fonction, au sens mathématique, de l'ensemble des objets vers le domaine de valeur de l'attribut. L'absence d'effets de bord est cependant très difficile à vérifier et, bien qu'une bonne discipline de conception dans un modèle à objets doive s'y plier, elle peut facilement être mise en défaut.

Afin spécifier les paramètres des mécanismes d'inférence, on utilise des chemins à partir de l'objet sur lequel il porte. Un *chemin* commence par l'objet lui-même (correspondant au "self" des langages de programmation [Masini& 1998]) et se poursuit par une suite de noms d'attributs séparés par des *indicateurs*. Dans TROEPS, l'objet lui-même est dénoté par le symbole « `!` » et les noms d'attributs sont séparés par des « `.` » ou des « `:` » en cas d'attributs multi-valués dont on va faire l'union des valeurs [Gensel 1998]. Ainsi, `!.chef.subordonnés` désigne la liste des subordonnés de son chef et `!.chef.subordonnés:chef` est un ensemble

[1] C'est introduire le LOO dans la bergerie (contribution conjointe de Gilles Bisson — CNRS — et de François Rechenmann — INRIA).

qui contient pour unique valeur `!.chef` puisqu'il s'agit de l'union des chefs qu'ont les subordonnés de son chef.

Les facettes d'inférence de valeurs d'attributs que l'on trouve communément sont :

*valeur par défaut* qui permet de fixer une valeur donnée en l'absence d'autres valeurs (ce mécanisme, lorsqu'il est activé, retourne toujours une valeur) ;

*passage de valeur* qui permet de contraindre la valeur à être celle d'un autre attribut ;

*héritage latéral* qui permet d'hériter la valeur du même attribut d'un objet référencé par l'objet courant ;

*filtrage* qui permet de donner pour valeur d'un attribut l'ensemble des objets qui satisfont certaines contraintes ;

*attachement procédural* qui permet d'exécuter une fonction (prenant pour paramètre des valeurs d'autres attributs de l'objet) retournant la valeur demandée.

Ces facettes seront plus détaillées dans le cadre de TROEPS (voir §5).

```
<employé
  attributs = {
    <salaire-de-base type = entier; constructeur = un;>,
    <heures-supplémentaires type = entier; constructeur = un;>,
    <salaire
      methode = !.salaire-de-base
                + !.salaire-horaire * !.heures-supplémentaires * 1.5;>,…};>

<cadre
  sorte-de = employe;
  attributs = {
    <caisse-de-retraite défaut = “caisse-des-cadres”;>,
    <salaire valeur = !.salaire-de-base;>,
    <subordonnés
      filtre = <employé
                 attributs = {
                   <projet valeur =
                     <projet
                       attributs = {<chef valeur = !;>};>;>};>;>,…};>
```

Figure 3 : Mécanismes d'inférence. La méthode pour obtenir le salaire dans la classe `employé` consiste à additionner le salaire de base (`!` représente l'objet lui-même et `!.salaire-de-base` la valeur de son attribut `salaire-de-base`) à une fois et demi le salaire horaire multiplié par le nombre d'heures supplémentaire. Dans la classe `cadre`, cette méthode n'est plus utilisée : le `salaire` est le `salaire-de-base`. Le filtre permettant d'obtenir la liste des `subordonnés` consiste à rechercher parmi les `employés` ceux dont l'attribut `chef` a pour valeur le `cadre` lui-même.

À l'opposé des facettes de typage, les facettes d'inférence sont héritées par défaut, c'est-à-dire qu'elles ne fournissent pas la « véritable » valeur de l'attribut recherché mais permettent de suggérer une valeur possible. Par défaut, signifie « si l'on n'a pas d'autres valeurs candidates » : ainsi, si une valeur vient à être affectée à l'objet, la valeur par défaut n'est plus considérée. Cette interprétation pourrait être remplacée par d'autres : valeur la plus probable ou la plus typique. Toutes ces interprétations s'appliquent pas de manière brutale : la valeur la plus probable pour une classe n'est pas forcément la plus probable pour ses sous-classes par exemple. C'est pourquoi les facettes d'inférence ne sont utilisées que s'il n'existe aucune méthode applicable définie dans une classe plus spécifique. Par exemple, s'il existe une méthode pour calculer la paye du mois en fonction du `salaire-de-base` et des `heures-supplémentaires` pour la classe `employé`, cette méthode pourra être masquée au profit d'une

méthode plus rapide (la paye est le `salaire-de-base`, sans tenir compte des `heures-supplémentaires`) pour les `cadres`.

Par conséquent, l'héritage des mécanismes d'inférence n'est pas absolu : il peut poser des problèmes. Il est, en particulier, lié au phénomène de *non monotonie* : l'ajout de connaissance ne préserve pas nécessairement celle issue des anciennes inférences. Par exemple, dans le cas de la Figure 3, si `Pierre` est un `employé`, un salaire particulier est déduit, alors que s'il est un `cadre`, un autre salaire qui peut être différent du précédent est déductible. En cas de multi-généralisation, l'héritage par défaut peut aussi poser un problème puisque plusieurs méthodes peuvent être applicables car aucune n'est en relation de spécificité avec les autres. Ce type de fonctionnement a donné lieu à toute une littérature sur les « conflits de valeurs » [Ducournau& 1995] ou exceptions et est l'une des causes de la création de logiques non monotones [Etherington 1986] et de théories de l'héritage non monotone [Touretzky 86, Simonet 1994].

Il ne faut pas confondre les mécanismes d'inférence des représentations de connaissance par objets et des mécanismes permettant de compléter la base de connaissance de manière implicite (pour éviter à l'utilisateur de le faire puisqu'il est possible de calculer ce qui manque, par exemple, lorsque la `caisse-de-retraite` est calculée — de manière absolue — automatiquement en fonction du statut de la personne). Pour simplifier, une inférence de connaissance implicite est une inférence réalisée à la création de l'objet : si la valeur n'a pas été explicitement fournie c'est qu'elle est implicite ; les inférences des représentations de connaissance par objets sont généralement réalisées lors de l'accès à la valeur : si elle est toujours inconnue, alors il est possible de rechercher une valeur par des moyens valides ou hypothétiques. Mark Stefik [1995] introduit deux distinctions explicite/implicite et primitive/dérivée pour bien distinguer ces deux dimensions. À notre connaissance, il n'existe aucun système de représentation de connaissance par objets utilisant cette notion de connaissance implicite, mais les utilisateurs utilisent souvent l'héritage par défaut pour inférer de la connaissance implicite. Utiliser le même mécanisme pour deux fonctions différentes est cependant problématique.

À cet ensemble de mécanismes s'ajoutent des mécanismes plus rares. C'est par exemple le cas de la classification où une instance est comparée aux contraintes posées sur les classes d'une taxonomies pour déterminer à quelles classes l'instance peut appartenir. Elle est utilisée dans CLASSIC [Granger 1988], SHIRKA [Rechenmann& 1988], FROME [Dekker 1994] ou dans TROEPS (voir §5).

### 2.3 Variations

Le tableau ci-dessous donne un aperçu de la variabilité des systèmes de représentation de connaissance par objets que l'on peut rencontrer dans la littérature (certains de ces systèmes sont opérationnels, d'autres restent à l'état de prototypes). Il ne prétend pas à l'exhaustivité (ni dans les systèmes, ni dans les critères) ; une vue plus apocalyptique de cette variabilité peut être trouvée dans [Karp 1993].

Les systèmes sont divisés par des lignes horizontales qui identifient des groupes. Les trois premières lignes, formant le premier groupe, permettent de contraster la variabilité avec des systèmes concurrents moyens — langages de programmation à objets (LPO), logiques de descriptions (LD), bases de données à objets (BDO). Le second groupe contient certains des systèmes "post-frame" américains : SRL [Wright& 1983], KRL [Bobrow& 1977] et FRL [Roberts& 1977]. Le troisième groupe contient les systèmes commerciaux américains ART [Williams 1984] et KEE [Filman 1988]. Puis viennent en quatrième trois systèmes commerciaux français SMECI [Ilog 1991], KOOL [Albert 1984] et YAFOOL [Ducournau &1986, 1991] et en cinquième des systèmes académiques français : SHIRKA [Rechenmann 1985], OBJLOG [Dugerdil 1988], FROME [Dekker 1994] et TROEPS [Sherpa 1995]. Le dernier sera beaucoup plus détaillé dans la suite.

| \ caractère / système | message | classe/inst. | méta-modèle | réflexes | multi-géné. | multi-inst. | types | attach. proc. | filtres | variables | défaut | héritage lat. | symétrique | contraintes | classification | catégorisation | règles | points de vue | commercial |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPO | • | • | • | / | • | | | | | | / | | | | | | | | • |
| LD | | • | | | • | • | | | • | • | | / | / | + | • | • | + | | |
| BDO | • | • | | + | • | | • | | • | | | | | + | | | + | / | • |
| SRL | | | | • | • | | | | • | | • | • | • | | | | | | |
| KRL | | • | | | | | • | | • | • | • | / | / | / | | | | | |
| FRL | | • | | • | | | | • | | | • | | | / | | | | | |
| ART | | / | • | | • | • | | • | | | • | | • | | | | • | | • |
| KEE | • | • | | | • | | • | • | | | • | | | | | | • | | • |
| SMECI | • | • | | • | | | • | • | | | | • | • | | | | • | | • |
| KOOL | • | • | • | • | | | • | • | | | • | • | • | | | | • | | • |
| YAFOOL | • | / | / | • | • | | • | • | / | • | • | • | • | + | | | + | | / |
| SHIRKA | | • | • | • | • | | • | • | • | • | • | | | / | • | + | + | | |
| OBJLOG | • | • | / | • | • | | • | • | | • | • | • | | / | | | • | • | |
| (F)ROME | • | • | | • | • | | • | • | • | | • | | | / | • | | + | • | |
| TROEPS | | • | | | | • | • | • | • | • | • | / | / | • | • | + | | • | |

Tableau 1 : Comparaison des caractéristiques d'un ensemble de systèmes de représentation de connaissance par objets (•= présent ; /=sous une forme dégradée ; +=extension).

Les différents critères de comparaison des langages et systèmes reprennent les notions présentées plus haut.

— Le premier groupe concerne les aspects généraux des langages — voir §2.1 — : y a-t-il des messages? y a-t-il une distinction entre classe et instance? y a-t-il un méta-modèle? y a-t-il la possibilité de faire de la programmation par réflexes?

— Le second ensemble considère les caractéristiques graphiques de la spécialisation et de l'instanciation (simple ou multiple).

— Le groupe suivant évalue si les attributs sont typés ou non.

— Le quatrième groupe énumère différents types de facettes d'inférence possible (attachement procédural, filtre, passage de valeurs, valeurs par défaut, héritage latéral, attributs symétriques et contraintes — voir §2.2).
— Le pénultième groupe considère des mécanismes plus sophistiqués d'inférence non forcément liés à des facettes (classification, catégorisation, règles de production — voir §5) et la notion de points de vue — voir §4.3.
— En dernier lieu on a noté les systèmes connaissant ou ayant connu une diffusion commerciale.

## 3. LES PROBLÈMES APPROCHÉS PAR TROEPS

Comme on peut le voir ci-dessus, et à l'opposé des logiques de descriptions [Napoli 1998], les représentations de connaissance par objets offrent une grande latitude dans la conception du langage. La suite de l'exposé décrit plus en détail le système TROEPS. TROEPS [Mariño& 1990, Sherpa 1995] est le successeur du langage SHIRKA (développé par l'équipe de François Rechenmann, à Grenoble, vers 1985 [Rechenmann 1985]). TROEPS permet de se faire une idée des principales possibilités à considérer. L'évolution de SHIRKA vers TROEPS est exposée plus en détail dans [Euzenat& 1995]. La présente section étudie quatre grands problèmes des représentations de connaissance qui furent à l'origine de la conception de TROEPS et les solutions que l'on tente d'y apporter.

### 3.1 Problème d'évolution

Un problème posé par la structure des représentations de connaissance par objets (et des langages de programmation par objets) est que les objets attachés à une classe ne peuvent pas en changer. Pourquoi changer de classe? Parce qu'une personne considérée tout d'abord comme un étudiant n'est pas fondamentalement un étudiant, c'est d'abord une personne (en effet, elle peut cesser d'être un étudiant sans cesser d'exister mais elle ne peut cesser d'être une personne). Ce problème relève de la différence entre « être » et « être vu comme ». Il est intéressant de pouvoir considérer la même personne comme un étudiant en maîtrise de japonais lorsque l'on a acquis plus d'informations. Il serait intéressant de considérer cette même personne plus tard comme un employé. De même, la personne en question peut aussi être un footballeur. Il est donc nécessaire de pouvoir la considérer comme tel. Sous l'aspect de ce que l'on appelle la migration d'instances, il y a donc trois problèmes distincts :
— un problème d'*enrichissement* de la connaissance sur la personne (ou *complétion*), qui apparaît lorsque l'on dispose de connaissance plus spécifique sur la personne et qui nécessite un affinement de la représentation,
— un problème d'*évolution* de la personne (ou *mise à jour*), qui apparaît lorsque la personne change et qui nécessite une modification de la représentation, et
— un problème de *point de vue* sur la personne (ou *perspective*), qui apparaît lorsque la personne peut être vue sous plusieurs aspects et qui nécessite un changement de représentation.

Seul le dernier point peut être mécaniquement résolu dans les systèmes de représentation de connaissance classiques par l'utilisation de la multi-instanciation ou de la multi-généralisation. Cependant, la classe `étudiant` devra être spécialisée en `étudiant-footballeur`, `étudiant-gymnaste`... puis `étudiant-footballeur en japonais`, `étudiant-footballeur en malgache`... ce qui n'est pas très satisfaisant [Goodwin 1979]. D'autres mécanismes particulièrement *ad hoc* (destruction puis recréation de l'objet) permettent de traiter les deux autres problèmes [Banerjee& 1987, Nguyen& 1989]. Ils sont particulièrement utilisés lors de la classification.

TROEPS reprend ce problème à son origine. Il contraint à distinguer entre l'aspect *ontologique* des classes et leur aspect *taxonomique*. Par ontologique on entend ce qui est lié à l'essence même de la notion représentée et par taxonomique ce qui est lié à une classification des objets dépendant d'un point de vue extérieur. Il sépare alors la notion de concept (ontologique, voir §4.1) de la notion de classe (taxonomique, voir §4.4). Ainsi, un étudiant est défini ontologiquement en tant que personne et non plus en tant qu'étudiant. Cependant, il peut exister une taxonomie professionnelle permettant de classer cette personne parmi les étudiants, puis les étudiants en japonais. Ainsi, le problème de complétion est résolu. Le problème de perspective connaît aussi un début de solution : dés lors que les objets sont créés hors de leur classe, il est possible de définir plusieurs taxonomies (par exemple, profession, sport) sur un même concept (celui des personnes). Un même objet peut alors être attaché à plusieurs classes pourvu qu'elles appartiennent à des taxonomies différentes.

### 3.2 Problème d'interprétation des classes et de la spécialisation

Un problème important signalé par William Woods [Woods 1975] est celui de l'interprétation des classes : que signifie une classe, un objet, un lien entre classes ou objets? Ce problème est en partie tranché par la distinction classe-instance, mais pas complètement.

Les classes peuvent être interprétées de manière soit *descriptive*, soit *définitionnelle* — aussi qualifiée de prescriptive. Une classe descriptive introduit les conditions nécessaires pour qu'un objet soit une instance de la classe, alors qu'une classe définitionnelle décrit des conditions nécessaires et suffisantes. Ceci est crucial pour la classification (voir §5.2) : dans un système définitionnel, en général, la classification détermine (lorsque les objets ont des valeurs pour tous leurs attributs) les classes auxquelles un objet appartient, alors que, dans un système descriptif, elle détermine les classes auxquelles un objet *peut* appartenir. Dans les logiques de descriptions [Napoli 1998], la distinction entre concept primitif et concept défini, tout d'abord fondée sur le fait que certains concepts sont définis en fonction d'autres alors que les concepts primitifs ne peuvent être définis, correspond sémantiquement à la distinction entre classe descriptive et définitionelle. À l'opposé, le caractère descriptif ou définitionnel des langages de programmation par objets n'a pas été clairement établi (voir, par exemple la discussion dans [Davis 1987]) : même si les classes sont considérées comme descriptives, dès qu'un mécanisme de classification automatique est introduit, il repose souvent sur une interprétation définitionnelle.

Il y a deux intérêts à disposer de classes descriptives.

— Tout d'abord, il y a beaucoup de classes descriptives à modéliser (soit parce qu'il n'existe pas de conditions suffisantes, soit parce que l'on n'est pas capable de les exprimer) et il faut donc avoir les moyens de le faire.
— Ensuite, tout résultat d'inférence valide dans un monde descriptif est valide dans un monde définitionnel. Les inférences pourront donc y être faites en toute sécurité.

En effet, l'interprétation descriptive est plus faible que l'interprétation définitionnelle : le résultat des inférences est plus restreint même si plus de notions peuvent être représentées. Ainsi, si une tasse est caractérisée comme un contenant avec une anse, alors, d'un point de vue définitionnel, il est possible d'inférer qu'un contenant avec une anse est une tasse. D'un point de vue descriptif, il est seulement possible d'inférer que celui-ci peut être une tasse, mais est-ce bien une tasse?

TROEPS, issu de SHIRKA, s'inscrit dans une tradition héritée des langages de programmation par objets dans laquelle les classes sont descriptives (il est possible de décrire deux classes ayant les mêmes propriétés et une instance d'une de ces classes ne sera pas instance de l'autre). La popularité des langages de programmation par objets atteste que cette sémantique est bien comprise des utilisateurs actuels des objets. Par ailleurs, il y a de nombreux « domaines » dans lesquels il est possible d'exprimer des conditions nécessaires mais très difficile d'exprimer des condition suffisantes [Lebbe 1998]. Pour ces raisons, TROEPS est, pour l'instant, restreint à une interprétation descriptive des classes[2].

### 3.3 Problème de nommage et d'identité

L'attachement d'un objet à plusieurs classes incomparables pose un problème connu dans les systèmes à multi-généralisation : le conflit de nom [Ducournau& 1995]. Ce dernier s'énonce comme suit : si deux attributs différents peuvent être nommés de la même manière dans deux classes différentes et que l'objet appartient à ces deux classes, il y a conflit. Il faut noter que le problème des conflits de nom est un problème à double tranchant : soit l'on considère qu'un attribut ne peut être nommé qu'une fois dans une base et on se prive de la possibilité d'utiliser son nom dans des classes incomparables dont on sait qu'elles ne partageront jamais une instance (on se prémunit par avance de « trop de conflits ») ; soit on décide que deux attributs nommés de la même façon, introduits dans deux classes incomparables, ne sont pas les mêmes attributs et alors on multiplie les attributs (on passe à côté de conflits réels) [Dekker 1994].

Le problème est résolu dans TROEPS en ramenant la définition des attributs et leur nommage au niveau du concept[3]. Ainsi, toutes les classes de ce concept doivent utiliser le nom défini au niveau du concept pour nommer l'attribut et ce nom ne peut nommer un autre attribut. Par exemple, le concept `projet` et le concept `employé` peuvent avoir un attribut nommé `nom`, il n'y a aucune ambiguïté sur le fait qu'il ne s'agit pas du même attribut. Par contre, dans la classe `chercheur` et dans la classe `administrateur`, l'attribut `nom` est le même et n'a rien à voir avec celui d'un `projet`.

Au problème des noms s'ajoute le problème d'*identité* des objets. L'identité est aussi du ressort de l'aspect ontologique, c'est-à-dire du concept. Les objets de TROEPS sont nommés au

[2] Il est possible d'envisager des classes définitionnelles sous certains points de vue.
[3] Il est possible d'envisager un système de synonymie pour utiliser des noms différents.

niveau du concept et ce dernier se charge de l'identité des objets, c'est-à-dire qu'il garantit que deux objets différents ne peuvent être identifiés de la même manière et qu'il se charge de retrouver l'objet si l'identificateur est valide. Ceci permet de nouveau de se libérer des conflits de noms au sein d'un même concept et de donner le même nom à deux objets différents, pourvu qu'ils soient dans deux concepts différents. Les bases de données à objets ont promu la notion d'identifiant d'objet par opposition à la *clé* qui permettait de trouver les *n*-uplets dans les bases relationnelles. Dans TROEPS, la décision d'identifier les objets par une clé a été prise afin de disposer de noms plus naturels pour l'utilisateur que les identifiants engendrés par le système ou qu'un nom dont l'utilisateur avait du mal à gérer l'unicité. Les objets sont donc nommés par l'intermédiaire d'un sous-ensemble de leurs attributs (valides pour tout le concept et pouvant varier d'un concept à l'autre)[4].

### 3.4 Problème d'inférences

Un dernier problème délicat est celui du statut des mécanismes d'inférence permettant de calculer une valeur d'attribut non disponible dans l'objet lui-même. Traditionnellement, dans les systèmes de représentation de connaissance par objets, les mécanismes d'inférences sont attachés aux classes et permettent d'obtenir une valeur pour les attributs des objets appartenant à la e. Par ailleurs, ces mécanismes sont « hérités par défaut », c'est-à-dire que le mécanisme calculant une valeur dans la classe la plus spécifique est réputé fournir la meilleure valeur. Dans les systèmes n'autorisant pas la multi-généralisation, il est possible de définir un ordre stable sur les valeurs obtenues. Mais un autre problème apparaît pour la classification : a-t-on le droit d'utiliser les mécanismes d'inférence pour déterminer une valeur qui va être ensuite utilisée lors de la classification? Le mécanisme ne sera pas forcément adapté (car on ne sait pas si l'instance en cours de classification appartient à la classe) ni le plus adapté (car peut-être appartient-elle à une classe plus spécifique qui possède un mécanisme plus adapté). Sur ce s on peut consulter l'important travail de Lenneke Dekker [Dekker 1994] qui conduit à distinguer les ensembles d'attributs définissant (au sens de « définitionnels pour ») une classe des autres (descriptifs).

Pour résoudre ce problème, François Rechenmann [Rechenmann 1993a] propose la notion de *détachement procédural*. Le détachement procédural, à l'instar de l'attachement procédural, permet d'invoquer des procédures auxquelles sont passés des arguments définis à partir des attributs de l'objet et qui retournent la valeur d'un attribut particulier. Il est possible de proposer plusieurs méthodes dans un concept pour calculer le même attribut (à partir d'arguments différents par exemple), mais ces procédures doivent retourner la même valeur. Une attitude médiane adoptée dans TROEPS fait coexister les deux approches :

- Le détachement procédural retourne des valeurs qui sont bien les valeurs implicites des attributs.
- Des mécanismes d'inférence hérités par défaut sont associés aux classes. Ils retournent des valeurs hypothétiques qui ne sont pas utilisés lors de la classifications et sont activés par une primitive spécifique (`tr-guess-value`).

[4] Il est possible d'envisager un système de clés secondaires pour utiliser des clés différentes.

# 4. DESCRIPTION DE TROEPS

TROEPS [Mariño& 1990, Sherpa 1995] est donc un système de représentation de connaissance par objets développé par le projet Sherpa de l'INRIA Rhône-Alpes. TROEPS (prononcez trop') fut désigné sous le nom de Tropes de 1990 à 1997. Le nom a été changé en TROEPS pour des raisons juridiques. Le logiciel, ainsi que sa documentation, sont disponibles sur le serveur ftp de l'INRIA Rhône-Alpes (`ftp.inrialpes.fr`) dans le catalogue `/pub/sherpa/logiciels` (voir aussi `http://co4.inrialpes.fr`).

Par rapports à d'autres systèmes de représentation de connaissance par objets, TROEPS peut se caractériser par quatre points :

— l'ensemble des objets est partitionné en concepts ;
— un concept peut être vu sous plusieurs points de vue ;
— chaque point de vue détermine une hiérarchie de classes ;
— les rapports d'inclusion entre classes de différents points de vue peuvent être exprimés à l'aide de passerelles entre ces classes.

Dans ce contexte, un objet est instance d'un seul concept, est visible sous plusieurs points de vues où il est attaché à une seule classe plus spécifique.

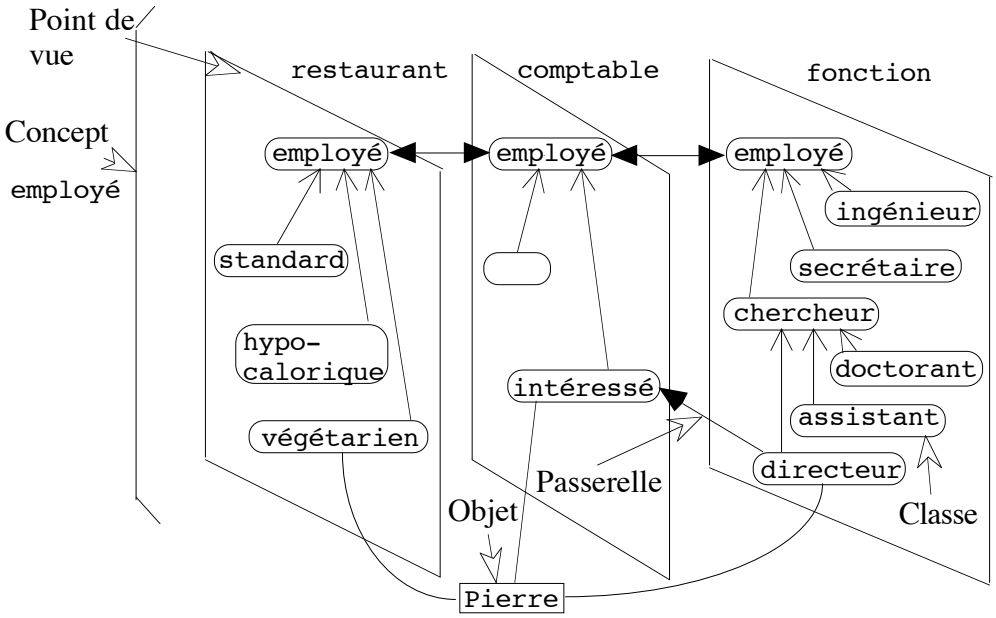


Figure 4 : Les différentes sortes d'entités de TROEPS. Le concept `employé` peut être vu sous les points de vue `restaurant`, `comptable` et `fonction`. Chacun d'entre eux détermine une hiérarchie de classes dont la racine est nommée `employé`. Par exemple, sous le point de vue `fonction`, l'ensemble des employés est décomposé selon leur fonction. L'objet `Pierre` est attaché à la classe `végétarien` sous le point de vue `restaurant`, à la classe `intéressé` sous le point de vue `comptable` et à la classe `directeur de recherche` sous le point de vue `fonction`.

La présentation de TROEPS suit les différentes étapes de la conception d'une base de connaissance à objets (voir aussi [Dao 1998]) :

1. inventaire des concepts et de leurs instances (concepts et clés) ;
2. inventaire des relations entre concepts (attributs composés) ;
3. inventaire des autres attributs ;
4. inventaires des points de vue sur les concepts (et attributs visibles) ;
5. construction des taxonomies dans les points de vue (et contraintes sur les attributs) ;

6. inventaire des relations entre points de vue (passerelles).

Bien entendu, cet ordre n'est pas immuable d'autant plus qu'il est possible d'ajouter à tout instant de nouveaux attributs et points de vue.

## 4.1 Concepts

Un des principes nouveaux de TROEPS est le *concept* (à l'état embryonnaire dans SHIRKA sous le nom de *famille*). L'ensemble des objets présents dans une base TROEPS est partitionné en un ensemble de concepts (un objet fait partie d'un et un seul concept). Le concept rassemble l'aspect ontologique des objets, c'est-à-dire qu'il définit :

- la structure des objets : les attributs qu'il peut avoir sont décrits et typés sous forme d'attributs de concepts (voir §4.2),
- l'identité et l'intégrité des objets : un sous-ensemble de ces attributs forme la clé qui doit être unique et qui permet d'identifier l'objet (le système garantit cette unicité comme il garantit que cette clé ne sera pas modifiée lors de la vie de l'objet),
- l'espace des noms d'attributs, d'objets et de points de vue,
- des mécanismes d'inférence qui seront développés au §5.

Ainsi, un `employé` est doté des attributs `nom`, `prénom`, `date-naissance`, `salaire`, `projet`, `diplômes`, etc. Il est identifié par ses `nom`, `prénom` et `date-naissance`.

```
<employé
  cle = {
    <nom type = chaine; constructeur = un;>,
    <prénom type = chaine; constructeur = liste;>,
    <date-naissance type = date; constructeur = un;>};
  attributs = {
    <salaire type = entier; constructeur = un;>,
    <diplômes type = diplôme; constructeur = liste;>,
    <projet type = projet; constructeur = un;>,
    <chef type = employé; constructeur = un;>,
    <subordonnés type = employé; constructeur = ensemble;>,
    <bureau type = bureau; constructeur = un;>};
  contraintes = { !.chef = !.projet.chef };>
```

Figure 5 : Description du concept `employé`. Elle contient trois parties : les deux premières définissent les attributs dont le premier sous-ensemble est la clé (ici `nom`, `prénom` et `date-naissance`) ; la partie suivante décrit les contraintes portant sur les attributs (pouvant lier plusieurs attributs).

## 4.2 Attributs de concepts et types

L'ensemble des *attributs* applicables aux instances du concept sont définis dans celui-ci. Il est possible d'ajouter des attributs de concepts à n'importe quel moment. Les attributs définis dans le concept sont applicables à n'importe quel objet du concept même si celui-ci est attaché à des classes qui ne les mentionnent pas. Chaque attribut est décrit au niveau du concept par son type et son constructeur.

Le *type* d'un attribut est un concept ou un type externe au système TROEPS. Il n'y a pas, dans TROEPS, de type primitif : tous les types sont externes et sont déclarés au système par un prédicat d'appartenance, un prédicat d'égalité entre deux valeurs, une fonction de lecture et

d'écriture et, si le type est ordonné ou discret, un prédicat d'ordre ou une fonction successeur. Ceci permet à l'utilisateur d'importer n'importe quel type (comme entier mais aussi date ou séquence nucléotidique) dans le système tout en voyant ce type pris en compte par des mécanismes tels que la vérification des types des attributs ou la classification des objets (voir §5.2).

Le *constructeur* d'un attribut permet de préciser si la valeur doit être une valeur unique ou une collection de valeurs (un ensemble ou une liste par exemple).

Ainsi, les quatre premiers attributs de la Figure 5 sont typés respectivement par une chaîne de caractères, une liste de chaînes de caractères, une date et un entier. Les deux attributs suivants sont typés respectivement par un objet du concept `projet` et une liste d'objets du concept `diplôme`.

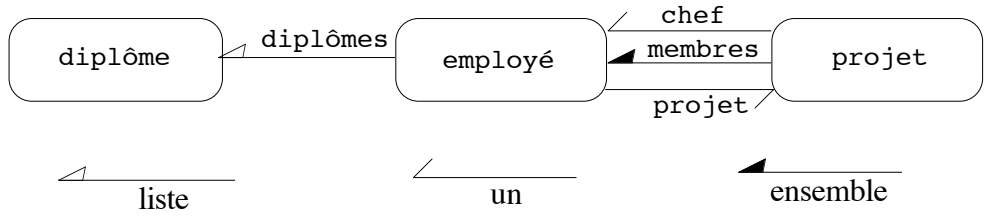


Figure 6 : Relations entre concepts.

Des *contraintes* peuvent aussi être attachées aux concepts de manière à exprimer les relations entre attributs. Ces relations sont exprimées par un prédicat (ici =) entre des objets atteignables à partir de l'objet courant par le moyen de chemins (voir §2.2). Ainsi, les facettes `herit-by` et `lien-inverse` de YAFOOL [Vismara& 1998] peuvent-elles être exprimées par les contraintes :

```
chef herit-by projet      <=> !.chef = !.projet.chef
époux lien-inverse épouse <=> !.époux.épouse = !
```

### 4.3 Point de vue

Un *point de vue* est une structure destinée essentiellement à accueillir une taxonomie de classes sur les objets du concept. Il décrit donc cette taxonomie en commençant par sa classe racine puis en y ajoutant des sous-classes. Les taxonomies sont de strictes hiérarchies (c'est-à-dire qu'elles n'autorisent pas la multi-généralisation). La présence de plusieurs taxonomies dans un même concept rend compte plus simplement de nombreuses situations dans lesquelles les langages de programmation par objets utilisent la multi-généralisation (voir Figure 7).

Les attributs pertinents pour un point de vue sont ceux dont il est fait mention au sein des classes du point de vue. Un objet est attaché à exactement une classe par point de vue. Par conséquent, les taxonomies sont supposées *exclusives* (c'est-à-dire qu'un objet ne peut appartenir à des classes incomparables) et tous les objets du concept appartiennent à la racine de chaque point de vue, ce qui justifie que toutes les racines portent le nom du concept.

La notion de point de vue rappelle celle de vue dans les bases de données relationnelles (reprises actuellement dans le cadre des bases de données à objets [Libourel 1998]). En effet, les deux notions permettent d'identifier un ensemble d'attributs pertinents et de masquer les autres. Elle sont cependant différentes sous deux aspects :

— Les *vues* des bases de données permettent d'agréger plusieurs relations dans une nouvelle relation (la vue), ce que TROEPS ne pourrait permettre qu'au prix de l'utilisation additionnelle de contraintes.
— Les points de vue de TROEPS permettent de disposer d'une taxonomie différente sous chaque point de vue, ce que les bases de données n'autorisent pas. Cette dernière caractéristique permet de traiter de nombreuses applications qu'il est difficile de mettre en œuvre autrement.

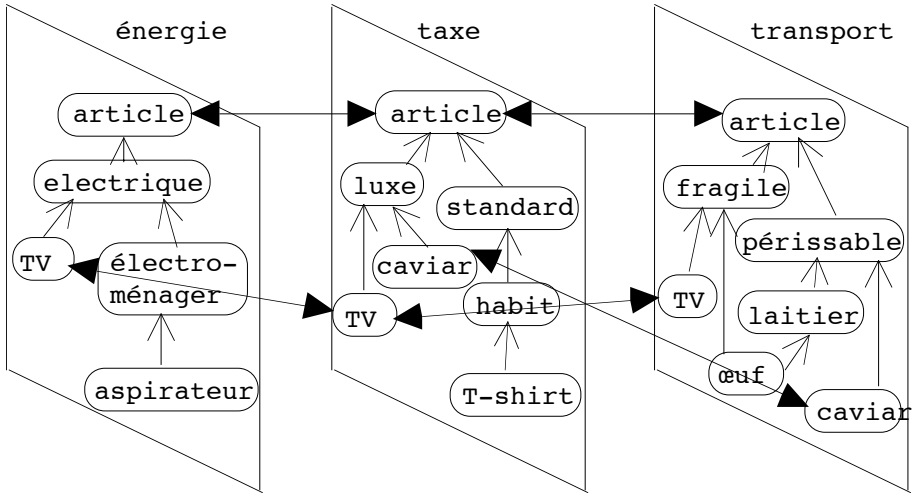


Figure 7 : L'exemple 2.5 (héritage multiple) de [Masini& 1989] repris en TROEPS. Comme on le voit sur cette figure, il reste un cas de multi-généralisation qui ne se traite pas avec des points de vue.

Les langages de programmation par objets utilisent des classes purement ontologiques alors que les logiques de description manipulent des classes taxonomiques. Cependant, ROME [Carré& 1988, Dugerdil 1988] fut le premier système à séparer explicitement les deux aspects. Un objet est alors instance d'une *classe d'instanciation* (rassemblant la description ontologique de l'objet) et permet de la classer sous des *classes de représentation* (correspondant plus à l'aspect taxonomique). Mais ROME ne sépare pas la base en points de vue différents : il n'y a qu'une taxonomie. Par contre, VIEWS [Davis 1987] offre la possibilité d'organiser la connaissance sous différents points de vue, mais la notion de taxonomie n'est pas centrale dans ce modèle. Par ailleurs, la partie ontologique est présentée comme du « platonisme inutile », c'est-à-dire que l'auteur s'oppose à la possibilité de décrire *a priori* les caractéristiques des objets. FRAMETALK [Rathke 1993] est une implémentation de schémas en CLOS qui offre une notion similaire aux points de vue [Rathke& 1993].

## 4.4 Classes et attributs de classes

Les *classes* décrivent un ensemble d'objets particuliers. Pour cela, elles définissent les attributs pertinents pour ces objets et un ensemble de restrictions sur les types des attributs. Cela ne signifie pas que les attributs non pertinents ne sont pas associés aux objets, mais que la classe est indépendante de ces attributs (par exemple, sous le point de vue `restaurant`, l'attribut `projet` n'est pas pertinent, ce qui ne signifie pas qu'un employé ne fasse pas aussi partie d'un projet).

Les classes sont définies en référence à leur super-classe. Toute classe peut préciser les types des attributs introduisant des *attributs de classes*. Ceux-ci sont décrits par un ensemble de descripteurs (correspondant aux facettes des représentations de connaissance par objets). Les descripteurs de TROEPS sont :

`classes` (pour les objets) qui introduit un ensemble de classes auxquelles les valeurs de l'attribut doivent appartenir (ces classes obéissent à la syntaxe `<classe>@<point de vue>`) ;

`intervalles` (pour les types ordonnés) qui introduit une réunion d'intervalles de valeurs acceptables ;

`domaine` qui introduit un ensemble de valeurs acceptables ;

`sauf` qui introduit un ensemble de valeurs non acceptables pour l'attribut ;

`cardinal` (pour les collections) qui introduit un intervalle de cardinaux acceptables pour la valeur.

```
<employé
  attributs = {
    <nom>,
    <prénom>,
    <date-naissance>,
    <poste intervalles = {[1 266]};>};>,

<cadre
  sorte-de = employé@fonctionnel;
  attributs = {
    <diplômes cardinal = [2 +inf[;>,
    <chef type = { cadre@fonction };>,
    <poste    intervalles = {[1 72]};
              sauf = {1, 42, 53, 54};>,
    <salaire intervalles = { [10000 20000] };>};>
```

Figure 8 : Description de classes en TROEPS. La première classe, `employé`, n'est pas le concept décrit à la Figure 5, mais la racine du point de vue `Fonction`. C'est pour cela qu'elle n'a pas de super-classe (`sorte-de`).

La grammaire des expressions de type pour les attributs de classe s'exprime comme ci-dessous (les $i$ sont des entier, les $v_i$ des valeurs, les $c_i$ des classes sous le format `<classe>@<point de vue>`) :

$$\left\{\begin{array}{c} domaine = \{v_1,\ldots v_n\}; \\ sauf = \{v_1,\ldots v_n\}; \\ intervalles = \{[v_1\ v_1'],\ldots[v_n\ v_n']\}; \\ type = \{c_1,\ldots c_n\}; \\ cardinal = [r\ r']; \end{array}\right\}*$$

Les autres descripteurs sont destinés aux inférences et sont décrits ci-dessous (voir §5). Des contraintes sont par ailleurs associées aux classes que leurs instances doivent respecter.

## 4.5 Passerelles

Les *passerelles* permettent d'exprimer des contraintes entre les classes de points de vue différents. Ainsi, si les points de vue sont théoriquement indépendants, il est possible que l'appartenance à une classe sous un point de vue implique l'appartenance à une autre classe sous

un autre point de vue. Par exemple, la passerelle entre `directeur` et `intéressé` dans la Figure 4 indique que les employés qui sont directeurs de recherche sont forcément intéressés aux bénéfices.

### 4.6 Objets

Les *objets* sont semblables aux instances des langages de programmation par objets. Ce sont des ensembles de couples attributs-valeurs. Il sont identifiés par le concept dont ils sont instance et un sous-ensemble de leurs attributs (défini pour chaque concept) formant la clé. Par ailleurs, les objets de TROEPS, s'ils appartiennent à un et un seul concept, sont attachés à une (et une seule) classe par point de vue. Les contraintes qui pèsent sur un objet sont donc celles du concept et celles qui concernent toutes les classes auxquelles l'objet appartient.

Les objets peuvent être manipulés pour changer leurs valeurs d'attributs ou leurs classes d'appartenance.

## 5. TROEPS : INFÉRENCE

Parmi les services que peut offrir une représentation de connaissance, on s'attachera à l'aspect complétion uniquement. La complétion permet de compléter la base de connaissance automatiquement de manière à ce qu'elle soit plus fidèle au « domaine » modélisé.

Il existe deux types principaux de complétion dans les représentations de connaissance par objets : l'inférence de valeur qui permet d'obtenir une valeur pour un attribut et l'inférence de position qui permet de déterminer une place possible (pour un objet ou une classe) dans la hiérarchie des classes. Ces deux aspects sont bien entendu étroitement liés puisque l'inférence de position se fait sur la foi des valeurs d'attributs alors que l'obtention de valeurs d'attributs dépend de mécanismes attachés à la classe.

Par ailleurs, il existe deux statuts pour ces complétions : soit elles sont considérées comme *valides* (c'est-à-dire qu'elles sont toujours vraies), soit elles sont considérées comme des *hypothèses* (voir les discussions sur les valeurs par défaut au §2.2).

Dans TROEPS, le rapport entre les différents mécanismes d'inférence est clarifié par deux propositions : (1) les inférences valides sont définies dans les concepts (ontologiques) et les inférences hypothétiques le sont dans les classes ; (2) seules les inférences valides — donc indépendantes de la classe — peuvent être utilisées lors de la classification — et les classes d'attachements sont utilisées pour les inférences hypothétiques. Il y a donc une stricte hiérarchie des mécanismes d'inférence lié à l'aspect ontologique/taxonomique du couple concept/classes.

### 5.1 Inférence de valeurs

Les inférences de valeurs doivent donc être distinguées suivant qu'elles sont nécessaires à l'objet (on parle aussi de connaissance implicite) ou qu'elles sont des hypothèses sur la valeur de (plus ou moins étayées en fonction de la situation de l'objet). Les premières sont décrites au niveau du concept au titre des aspects nécessaires des objets, alors que les autres sont attachées aux classes.

Dans les deux classes d'inférence, les paramètres nécessaires aux méthodes sont spécifiés par des chemins qui permettent d'indiquer n'importe quel objet atteignable à partir de l'objet de départ (voir §2.2).

Les inférences attachées aux concepts ressortent de l'idée de détachement procédural (voir §3.4). Les inférences attachées aux classes sont plus diversifiées ; elles sont introduites à l'aide des descripteurs suivants :

`défaut` spécifie une valeur prise par défaut (c'est-à-dire si aucune valeur n'est connue) pour l'objet ;

`affecte` spécifie un chemin par lequel la valeur de l'attribut peut être atteinte ;

`méthodes` spécifie un ensemble de méthodes qui peuvent être utilisées pour calculer une valeur (à noter que, contrairement à ce qui se passe pour les concepts, les valeurs retournées ne sont qu'hypothétiques et peuvent parfaitement être différentes d'une méthode à une autre) ;

`filtres` (pour les collections) spécifie un filtre (voir plus bas) dont l'ensemble des valeurs le satisfaisant formera la valeur de l'attribut. Les chemins sont aussi utilisés pour spécifier des contraintes sur le filtre.

Ces valeurs hypothétiques ne sont soumises à l'utilisateur de la base que si cela est explicitement demandé (elles ne se substituent pas automatiquement aux valeurs d'attributs). La primitive invoquant ces valeurs hypothétiques retourne l'ensemble des valeurs obtenues. En effet, l'appartenance d'un objet à plusieurs classes peut conduire à l'existence d'une certaine valeur sous un point de vue et une autre sous un autre point de vue. Par exemple, sous une interprétation probabiliste, ces deux valeurs peuvent être aussi probables l'une que l'autre.

Les méthodes d'inférences, définies dans les classes, sont héritées d'une classe vers l'autre. Mais, alors que l'héritage des descripteurs de types est strict (voir §2.1), celui des mécanismes d'inférence se fait par défaut (voir §2.2). C'est-à-dire qu'un mécanisme capable de retourner une valeur hypothétique est utile tant qu'il n'y en a pas, dans les sous-classes, qui soit capable de retourner une valeur. Ainsi, pour obtenir la valeur d'un attribut d'un objet, TROEPS cherche les mécanismes susceptibles de l'inférer dans ses classes d'attachement puis, tant qu'aucune valeur n'a été trouvée dans un point de vue, dans les super-classes. Si une autre valeur d'attribut est nécessaire, comme paramètre d'un mécanisme d'inférence (obtenu par un chemin), elle est inférée de la même manière.

## 5.2 Classification

L'inférence de position pour un objet est réalisé par le mécanisme de *classification* — plus précisement de classification d'instance. Le mécanisme de classification disponible en TROEPS est particulier, car il manipule des classes descriptives (c'est-à-dire constituées de conditions nécessaires mais non suffisantes, voir §3.2). Classer une instance $i$ sous une classe $c$, c'est trouver les sous-classes de $c$ auxquelles $i$ peut appartenir compte tenu des valeurs des attributs de $i$. Une partition des classes en trois ensembles est alors obtenue ($\tau_{f,c}$ est le type de l'attribut $f$ dans la classe $c$, valeur?($i$,$f$) est la valeur de l'attribut $f$ pour l'instance $i$ et $\iota$ dénote la valeur inconnue) :

*Possibles* : si toutes les valeurs d'attributs de l'instance satisfont les contraintes de la classe : $\forall f \in \text{attributs}(c)$, $\text{valeur?}(i,f) \in \tau_{f,c}$,

*Inconnues* : si aucune valeur d'attribut de l'instance ne viole une contrainte de la classe mais certaines valeurs d'attributs sont inconnues : $\forall f \in \text{attributs}(c)$, $\text{valeur?}(i,f) \in \tau_{f,c} \cup \{\iota\}$,

*Impossibles* : s'il existe une valeur d'attribut de l'instance qui viole une contrainte de la classe : $\exists f \in \text{attributs}(c)$, $\text{valeur?}(i,f) \notin \tau_{f,c} \cup \{\iota\}$.

La classification de TROEPS possède un certain nombre de traits notables :

— Elle prend en compte l'incomplétude des objets (le classement d'instances incomplètes apparaît dans l'ensemble *Inconnues*).

— Parallèlement aux logiques de descriptions [Napoli 1998] qui ne classifient pas dans les concepts primitifs (correspondant à l'interprétation descriptive des classes) — sauf cas très particuliers [Ducournau 1998] —, et comme SHIRKA, TROEPS dispose d'un mécanisme de classification agissant uniquement dans des concepts primitifs. Cependant, la classification est ici restreinte aux individus, elle correspond donc à une opération beaucoup plus simple que la classification dans les logiques de descriptions qui classe des concepts (se rapprochant plus des classes).

— L'algorithme prend en compte récursivement la « classifiabilité » des objets placés en valeurs d'attributs dans la classe caractérisant le type de l'attribut, c'est-à-dire la possibilité pour un objet valeur d'attribut d'être classé dans les classes définissant le type de l'attribut. Ainsi, si un `ingénieur-informaticien` est un `employé` avec un diplôme d'ingénieur travaillant dans un `projet-informatique`, la classification, pour classer un `employé` comme `ingénieur-informaticien`, doit classer la valeur de son attribut `projet` comme `projet-informatique`.

— L'algorithme prend en compte les points de vue et les passerelles. Il opère la classification simultanément dans chaque point de vue. Les passerelles interfèrent dans ce processus en introduisant des contraintes sur les classes d'appartenance.

La classification de classes, autre inférence de position, bien que possible et spécifiée, n'est pas encore disponible dans TROEPS.

### 5.3 Filtres

Une classe est une description d'un ensemble d'objets. Ainsi, tout objet satisfaisant la description d'une classe peut y être attaché, mais rien ne l'y contraint. En particulier, il peut exister deux classes incomparables pour la spécialisation dont la description est identique. Ces descriptions constituent donc des conditions nécessaires mais non suffisantes à l'appartenance à la classe.

Les filtres sont formés des mêmes éléments que les classes mais constituent des conditions nécessaires et suffisantes à l'appartenance. Ils sont définis à partir de plusieurs classes appartenant à des points de vue différents auxquels sont ajoutées des contraintes. Ils ne peuvent alors filtrer (et se voir attacher) que les objets attachés aux classes à partir desquelles ils sont définis. La Figure 3 présente ainsi un filtre permettant d'obtenir l'ensemble des subordonnés d'un employé en recherchant ceux qui l'ont pour chef.

Les filtres sont utilisés pour deux usages : (1) la recherche d'un ensemble d'instances d'une classe satisfaisant un certain nombre de critères et, en particulier, (2) l'inférence de valeur (associée au descripteur filtre décrit plus haut). Dans le premier cas, les filtres sont alors créés et manipulés individuellement, l'ensemble des objets filtrés, alors attachés au filtre, peut être manipulé en tant que tel. C'est ainsi que sont construits les filtres dans l'interface d'interrogation de TROEPS (voir Figure 9). Dans le second cas, la valeur retournée (qui est toujours un ensemble d'objets) est considérée comme la valeur (hypothétique) de l'attribut.

### 5.4 Catégorisation

L'objet de la *catégorisation* est l'obtention d'une taxonomie de classes (décrites à l'aide de leurs descripteurs) à partir d'un ensemble d'objets. On peut trouver ailleurs des techniques permettant cela [Fisher& 1986]. Nous présentons seulement l'intérêt de TROEPS dans ce contexte.

TROEPS offre plusieurs avantages pour la catégorisation. Les objets étant définis indépendamment de leurs classes, il est possible de disposer des objets puis des classes. De plus, les points de vue étant indépendants, il est possible de construire plusieurs taxonomies avec des méthodes différentes ou des paramètres différents de la même méthode et de les comparer. À cette fin, le système T-TREE a été défini comme une extension de TROEPS [Euzenat 1993]. Celui-ci implémente un algorithme simple (de type « plus proches voisins ») et construit des taxonomies à partir des valeurs numériques des objets.

Pour aller au delà des valeurs numériques, il est nécessaire de définir une métrique sur les objets qui permette de les comparer entre eux. C'est l'objet des travaux de Petko Valtchev [Valtchev& 1996] qui tirent parti du système de types de TROEPS. Il est en effet possible de définir une distance associée aux types de valeurs. Cette distance est prolongée en une dissimilarité sur les objets fondée uniquement sur la structure des taxonomies existantes. Ainsi, deux objets appartenant à deux classes ayant une super-classe commune seront plus proches entre elles que d'une classe plus éloignée dans la taxonomie. Ces distances ne sont utilisées qu'une seule fois dans le processus de construction de taxonomies : une fois que les objets sont organisés en une (ou plusieurs) taxonomies, il est possible de catégoriser les objets qui y font référence en utilisant la métrique définie à partir de cette dernière taxonomie.

À des fins de comparaisons de diverses taxonomies, nous avons aussi développé une méthode d'inférence de passerelles qui permet d'obtenir toutes les passerelles possibles entre un ensemble de taxonomies (source) et une autre taxonomie (cible) [Euzenat 1993]. Cet algorithme, utilisé à l'origine sur les ensembles d'objets attachés aux classes, peut maintenant utiliser les descriptions des classes elles-mêmes.

## 6. INTERFACE

TROEPS a été développé indépendamment des aspects interface mais dispose maintenant d'une interface intégrée dans le "World-wide web" (Web). Celle-ci est brièvement présentée ici.

## 6.1 Motivation

Les concepteurs de bases de connaissance ont une expérience importante en ce qui concerne l'acquisition et l'organisation de connaissance, ce qui ne doit pas être négligé, lorsqu'il s'agit d'organiser sur un support informatique la connaissance d'un « domaine » particulier. Une base de connaissance est exploitable aisément pour engendrer automatiquement un ensemble d'hyper-documents dont la cohérence est assurée par le système de représentation de connaissance. Par la suite, ces sites peuvent utiliser les capacités inférentielles des systèmes de représentation de connaissance afin d'interroger directement la base et non sa représentation graphique (ou textuelle, comme c'est le cas d'un site Web traditionnel). La puissance des requêtes de type classification ou filtrage pourra alors être mise en œuvre afin d'obtenir des informations plus précises.

D'un autre côté, les bases de connaissance sont loin de se suffire à elles-mêmes. Depuis longtemps, on a pu remarquer le parti qui pouvait être tiré de leur liaison avec un système hypertexte, à la fois pour documenter les choix de conception ou pour illustrer le contenu à l'aide de la littérature sur le « domaine ». La conviction sous-tendant cette approche est qu'il est impossible de consulter une base de connaissance sans disposer d'un minimum d'aide technique.

Par rapport aux bases de connaissance traditionnelles, l'utilisation du Web possède plusieurs avantages :

accessibilité : la base est accessible dans le monde entier sans nécessiter de logiciel particulier (ni cher, ni sophistiqué) ; le portage des logiciels de gestion de bases de connaissance n'est plus essentiel pour la diffusion de la base.

inter-opérabilité : la documentation de la base utilise un langage d'hypertexte standard : HTML (HyperText Markup Language, le langage de description de documents hypertexte utilisé par le Web). Elle peut faire référence à et être référencé par d'autres sites par ce moyen. Elle s'en trouve donc plus aisée à développer et à réutiliser. Le développement des aspects interface est réduit.

medium actif : mais surtout, la conviction d'un certain nombre de chercheurs [Gaines 1990, Rechenmann 1993b], dont nous sommes, est qu'une base de connaissance a au moins autant d'intérêt en tant qu'encyclopédie interactive et rigoureuse qu'en tant qu'outil de résolution de problème. C'est la notion de "knowledge medium" introduite par Mark Stefik [Stefik 1986].

L'omniprésence du Web est l'occasion de tester ces idées dans un contexte opérationnel.

## 6.2 Réalisation

Une interface hypertexte simple peut-être réalisée de manière immédiate pour une base d'objets. Cela consiste à exploiter le fonctionnement essentiellement client-serveur du protocole HTTP (HyperText Transfert Protocol) qu'utilise le Web. Ainsi, à chaque entité — concept, attribut, classe, objet… — est associé un identificateur (URL pour Uniform Resource Locator). Cet identificateur, c'est souhaitable, est fondé uniquement sur les moyens d'accès traditionnels

aux objets (les noms et les clés)[5]. À cet URL, chaque entité sait répondre sous la forme d'une page HTML la représentant. Bien entendu, cette représentation fait référence à d'autres entités avec lesquelles elle est en relation. Celles-ci sont présentes dans la page sous forme de liens (morceau de texte par lequel il est possible d'accéder à une autre page) permettant, lorsqu'on les active, de demander l'URL leur correspondant.

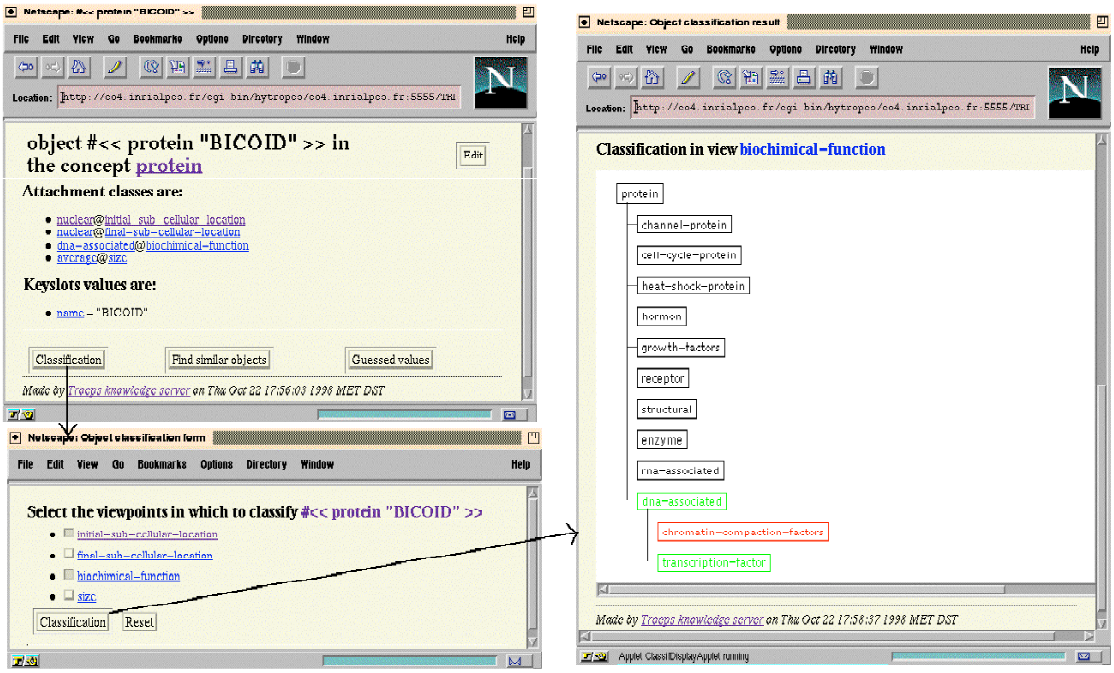


Figure 9 : Les pages correspondant à une instance d'une classe « protéine » et le résultat de la classification sur cette instance.

Ceci constitue un premier niveau d'interface dans lequel le schéma et les données de la base sont visibles et il est possible de naviguer dans la base. L'intégration de données externes qui peuvent contenir du texte, des images mais aussi des liens vers d'autres documents, d'autres bases voire même l'appel à des méthodes spécifiques se fait en spécifiant les documents à intégrer dans la représentation de chaque l'entité. Mais ceci n'est pas suffisant, l'apparence ainsi donnée aux objets n'est pas adaptée à une application particulière. Pour cela il est possible de réécrire complètement la fonction qui retourne la page correspondant à l'objet. Ceci permet d'invoquer des mécanismes (par exemple des "applets") avec des paramètres fonction de l'objet à représenter.

Les actions accessibles depuis TROEPS le sont aussi depuis la base. Ainsi, il est possible de définir un filtre qui, une fois activé, va retourner la liste des objets filtrés. Si le résultat n'est pas satisfaisant, il est possible de revenir sur la page correspondant au filtre, de la modifier et d'obtenir un nouveau résultat. De même, la classification est directement accessible à partir de l'interface : le résultat est fourni à l'aide de plusieurs taxonomies dont les classes apparaissent colorées différemment suivant la possibilité d'attacher l'objet à la classe.

Enfin, il possible d'éditer la base de connaissance à l'aide du même type d'interface (par exemple, pour effectivement attacher l'objet mais aussi pour créer, modifier ou détruire toute

[5] Il est possible de réaliser cela à l'aide de la syntaxe des formulaires.

sorte d'entité TROEPS). Cela pose de nouveaux problèmes (de concurence d'accès et de consistance) pris en compte par l'interface [Alemany 1998].

## 7. CONCLUSIONS

Les apports de la représentation de connaissance par objets sont à rechercher dans les deux termes utilisés :

— les principaux avantages des objets sont bien entendu exploités : concentration des informations concernant un individu au sein de l'objet, exploitation des rapports générique-spécifique ;

— ceux de la représentation de connaissance aussi : une sémantique (au moins dans l'intention) contraignant la cohérence de la représentation.

Mais la place des représentations de connaissance par objets est inconfortable entre les langages de programmation par objets d'une part et les bases de données à objets d'autre part. La sémantique des représentations de connaissance par objets est différente de celles des langages de programmation par objets, même s'il peut y avoir un noyau commun [Ducournau 1998] : les exigences des entités de programmation et de représentation n'ont pas de raison d'être les mêmes (les uns cherchent avant tout à créer un programme, les autres cherchent avant tout à représenter un « domaine » extérieur). Par rapport aux bases de données à objets, deux distinctions peuvent être faites :

— l'aspect système des bases de données à objets, c'est-à-dire le stockage des objets en mémoire secondaire (sur disque par exemple) et les contraintes que cela impose sur les modèles de données, n'est pas traité par les représentations de connaissance par objets ;

— même si le domaine des bases de données à objets s'étend tous les jours et menace d'englober à la fois les langages de programmation par objets et les représentations de connaissance par objets [Libourel 1998], les modèles de représentation de connaissance restent plus riches.

Mais il n'y a pas, *a priori*, de raison de penser que ces deux disciplines ne soient pas confondues un jour.

Au sein de la représentation de connaissance aussi, la situation des représentations de connaissance par objets est particulière. Elle apporte cependant une réponse à des besoins auxquels ne répondent pas les autres systèmes modernes de représentation de connaissance (graphes conceptuels et logiques de descriptions). En effet, la focalisation sur l'aspect formel de ces systèmes n'a pas encore permis d'atteindre une facilité d'utilisation nécessaire aux concepteurs de bases de connaissances. Par ailleurs, l'accent mis sur la définition des concepts et le calcul de subsomption comme unique mécanisme d'inférence hypothèque l'utilisabilité de ces systèmes : il existe trop de concepts dont il n'est pas possible de donner la définition (ou dont il n'est pas possible de donner la définition dans un langage restreint de manière à permettre la polynomialité du calcul de subsomption) [Doyle& 1991].

Bien que TROEPS ait été présenté plus en détail ici, il existe de nombreux systèmes de représentation de connaissance par objets (voir Tableau 1). En général, la syntaxe utilisée pour exprimer ou pour manipuler la connaissance n'est pas la même. Cependant, ces systèmes

partagent entre eux les mêmes concepts fondamentaux. Il est donc tentant de chercher à standardiser l'expression de ces représentations, comme il est tentant de le faire pour les objets de programmation. Ceci a été tenté diversement au travers des langages KIF (Knowledge Interchange Format) [Genesereth& 1992] et Generic Frame Protocol [Karp& 1995] (maintenant Open Knowledge Base Connectivity). Cependant, l'absence d'une compréhension profonde des différences entre langages fondée sur la sémantique formelle de ceux-ci (plutôt que sur des malentendus) rend illusoire l'obtention d'un résultat de qualité (outre qu'il risque de geler les recherches sur les modèles de représentation de connaissance [Ginsberg 1991]).

E des représentations de connaissance par objets dans le cadre d'applications très diverses est largement avéré (voir par exemple [Euzenat& 1995] pour SHIRKA). Ce succès devrait permettre la pérennité de ce type de systèmes de représentation de connaissance.

En dernier lieu, l'attention portée à la sémantique de la représentation est le trait à faire progresser au sein des recherches sur les représentations de connaissance par objets [Ducournau 1998, Euzenat 1998].

## 8. RÉFÉRENCES


[Alemany 1998] Christophe Alemany, Étude et réalisation d'une interface d'édition de bases de connaissances au travers du World Wide Web, Mémoire CNAM, Grenoble (FR), 1998 `ftp://ftp.inrialpes.fr/pub/sherpa/rapports/cnam-alemany.ps.gz`

[Albert 1984] Patrick Albert, KOOL at a glance, Actes 6th ECAI, Pisa (IT), p345, 1984

[Atkinson& 1987] R. Atkinson, J. Lausen, `Opus`: a smalltalk production system, Proc. 2nd OOPSLA, Orlando (FL US), pp377-387, 1987

[Banerjee& 1987] Jay Banerjee, Won Kim, Hyoung-Joo Kim, Henry Korth, Semantics and implementation of schema evolution in object-oriented databases, *SIGMOD records* 16(3):311-322, 1987

[Bobrow& 1987] Daniel Bobrow, Albert Collins (éds.), Representation and understanding: studies in cognitive science, New-York Academic Press, New-York (NY US), 1975

[Bobrow& 1977] Daniel Bobrow, Terry Winograd, An overview of KRL: knowledge representation language, *Cognitive science* 1(1):3-45 (rep. dans [Brachman& 1985], pp263-295, 1985), 1977

[Bobrow& 1981] Daniel Bobrow, Mark Stefik, The LOOPS manual, Memo KB-VLSI-81-13, Xerox PARC, Palo Alto (CA US), 1981

[Brachman 1977] Ronald Brachman, A structural paradigm for representing knowledge, Thèse, Harvard university, Cambridge (MA US), 1977

[Brachman 1979] Ronald Brachman, On the epistemological status of semantic networks, dans Nicholas Findler (éd.), Associatives networks: representation and use of knowledge by computers, Academic Press, New-York (NY US), pp3-50, (rep. dans [Brachman& 1985], pp191-215, 1985), 1979

[Brachman 1983] Ronald Brachman, What is-a is and isn't: an analysis of semantics links in semantic networks, *IEEE Computer* 16(10):30-36, 1983

[Brachman 1985] Ronald Brachman, "I lied about the trees" or, defaults and definitions in knowledge representation, *AI magazine* 6(3):80-93, 1985

[Brachman& 1985] Ronald Brachman, Hector Levesque, (éds.), Readings in knowledge representation, Morgan Kaufmann, Los Altos (CA US), 1985

[Carnegie group 1988] Carnegie group, Knowledge craft, version 3.2, CRL Technical Manual, Carnegie Group Inc., Pittsburgh (PA US), 1988

[Carré& 1988] Bernard Carré, Gérard Comyn, On multiple classification, points of view and object evolution, dans Jacques Demongeot, Thierry Hervé, Vincent Rialle, Christophe Roche (eds.), Artificial intelligence and cognitive sciences, Manchester University Press, Manchester (GB), pp49–62, 1988

[Chein& 1992] Michel Chein, Marie-Laure Mugnier, Conceptual graphs: fundamental notions, *Revue d'intelligence artificielle* 6(4):365-406, 1992

[Dao 1998] Michel Dao, Méthodes d'analyse et de conception par objets, dans [Ducournau& 1998], pp103-128, 1998

[Davis 1987] Harley Davis, VIEWS: muliple perspectives and structured objects in a knowledge representation language, Master thesis, MIT, Cambridge (MA US), 1987

[Dekker 1994] Lenneke Dekker, FROME: représentation multiple et classification d'objets avec points de vue, Thèse d'informatique, université des sciences et technolgie de Lille, Lille (FR), 1994

[Doyle& 1991] Jon Doyle, Ramesh Patil, Two theses of knowledge representations: language restrictions, taxonomic classification and the utility of representation services, *Artificial intelligence* 48(3):261-297, 1991

[Ducournau& 1986] Roland Ducournau, Joël Quinqueton, YAFOOL: encore un langage à objets à base de frames, Rapport technique 72, INRIA, Rocquencourt (FR), 1986

[Ducournau 1991] Roland Ducournau, Y3: YAFOOL, le langage à objets, Manuel de référence, Sema Group, Montrouge (FR), 1991

[ 95] Roland Ducournau, Michel Habib, Marianne Huchard, Marie-Laure Mugnier, Amedeo Napoli, Le point sur l'héritage multiple, *Techniques et science informatiques* 14(3):309-345, 1995

[Ducournau 1998] Roland Ducournau, La logique des objets: application à la classification incertaine, dans [Ducournau& 1998], pp351-380, 1998

[ Roland Ducournau, Jérôme Euzenat, Gérald Masini, Amedeo Napoli, Langages et modèles à objets: état des recherches et perspectives, INRIA, Rocquencourt (FR), 1998

[Dugerdil 1988] Philippe Dugerdil, Contribution à l'étude de la représentation de connaissances fondée sur les objets: le langage OBJLOG, Thèse, université d'Aix-Marseille, Luminy (FR), 1988

[Etherington 1986] David Etherington, Reasoning with incomplete information: investigation of non monotonic reasoning, Thèse, University of British Columbia, Vancouver (CA) (rev. reasoning with incomplete information, Pitman, London (GB), 1988), 1986

[Euzenat 1993] Jérôme Euzenat, Brief overview of T-TREE: the TROPES Taxonomy building Tool, dans Philip Smith, Clare Beghtol, Raya Fidel, Barbara Kwasnik (éds.), Avances in classification research 4 (proc. 4th ASIS SIG/CR classification research workshop, pp69-87, Columbus (OH US), pp, 28 octobre 1993), Learning information, Medford (NJ US), 1994 `ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat93c.ps.gz`

[Euzenat& 1995] Jérôme Euzenat, François Rechenmann, SHIRKA, 10 ans, c'est TROPES?, Actes 2ndes journées «langages et modèles à objets», Nancy (FR), pp13-34, (1-3 octobre) 1995
`ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat95f.ps.gz`

[ Jérôme Euzenat, Sémantique des représentations de connaissance, Polycopié, université Joseph-Fourier, Grenoble (FR), 1998

[Falhman 1979] Scott Falhman, NETL: A system for representing real world knowledge, The MIT press, Cambridge (MA US), 1979

[Filman 1988] Robert Filman, Reasoning with worlds and truth maintenance in a knowledge-based programming environment, *Communications of the ACM* 31(4):382-401, 1988

[Fisher& 1986] Douglas Fisher, Pat Langley, Conceptual Clustering and its Relation to Numerical Taxonomy, dans William Gale (éd.), Artificial Intelligence and Statistics, Addison Wesley, Reading (MA US), pp77-116, 1986

[Fornarino& 1990] Mireille Fornarino, Anne-Marie Pinna, Un modèle objet logique et relationel: le langage OTHELO, Thèse, université de Nice, Nice (FR), 1990

[Gaines 1990] Brian Gaines, Knowledge-support systems, *Knowledge based systems* 3(4):192-203, 1990

[Genesereth& 1992] Michael Genesereth, Richard Fikes, Knowledge interchange format, version 3.0 - reference manual, Rapport de recherche Logic-92-1, Stanford university, Stanford (CA US), 1992

[Gensel 1998] Jérôme Gensel, Objets et contraintes, dans [Ducournau& 1998], pp257-290, 1998

[Ginsberg 1991] Matthew Ginsberg, Knowledge interchange format: the KIF of death, *AI magazine* 12(3):57-63, 1991

[Goodwin 1979] James Goodwin, Taxonomic programming with KLONE, Rapport de recherche, Linköping university, Linköping (SE), 1979

[Granger 1988] Catherine Granger, An application of possibility theory to object recognition, *Fuzzy sets and systems* 28(3):351-362, 1988

[Ilog 1991] Ilog S.A., SMECI: manuel de référence, Ilog, Gentilly (FR), 1991

[Karp 1993] Peter Karp, The design space of frame knowledge representation systems, Technical note 520, SRI AI center, Menlo Park (CA US), 1993

[Karp& 1995] Peter Karp, Karen Myers, Thomas Gruber, The Generic Frame Protocol, Actes 14th IJCAI, Montréal (CA), pp768-774, 1995

[Kayser 1997] Daniel Kayser, La représentation de connaissances, Hermès, Paris (FR), 1997

[Lebbe 1998] Jacques Lebbe, Représentations par objets et classifications biologiques, dans [Ducournau& 1998], pp421-448, 1998

[Libourel 1998] Thérèse Libourel, Les systèmes de gestion de bases de données objet, dans [Ducournau& 1998], pp129-162, 1998

[Mariño& 1990] Olga Mariño, François Rechenmann, Patrice Uvietta, Multiple perspectives and classification mechanism in Object-oriented Representation, Actes 9th ECAI, Stockholm (SE), pp425-430, 1990

[Masini& 1989] Gérald Masini, Amedeo Napoli, Dominique Colnet, Daniel Léonard, Karl Tombre, Les langages à objets, InterÉditions, Paris (FR), (tr. ang. Object-oriented languages, Academic press, London (GB), 1991) 1989

[Masini& 1998] Gérald Masini, Karol Proch, Introduction à C++ et comparaison avec Eiffel, dans [Ducournau& 1998], pp73-102, 1998

[Minsky 1974] Marvin Minsky, A framework for representing knowledge, Rapport technique AI-memo 306, MIT, Cambridge (MA US), (rep. dans Patrick Winston (éd.), The psychology of computer vision, Mac Graw-Hill, New York (NY US), pp211-277, 1975 ; rep. dans [Brachman& 1985], pp245-262, 1985), 1974

[Napoli 1998] Amedeo Napoli, Une introduction aux logiques de descriptions, dans [Ducournau& 1998], pp321-349, 1998

[Nguyen& 1989] Gia Toan Nguyen, Dominique Rieu, Schema evolution in object-oriented database systems, *Data & Knowledge Engineering* 4(1):43-67, 1989

[Quillian 1968] M. Quillian, Semantic memory, dans Marvin Minsky (éd.), Semantic information processing, The MIT press, Cambridge (MA US), pp227-270, 1968

[Rathke 1993] Christian Rathke, Object-oriented programming and frame-based knowledge representation, Actes 5th. IEEE conference on tools with artificial intelligence, Boston (MA US), pp95-98, 1993

[Rathke& 1993] Christian Rathke, David Redmiles, Multiple representation perspectives for supporting explanation in context, Research report CU-CS-645, University of Colorado, Boulder (CO US), 1993

[Rechenmann 1985] François Rechenmann, SHIRKA: mécanismes d'inférence sur une base de connaissance centrée-objet, Actes 5ième RFIA, Grenoble (FR), pp1243-1254, (novembre) 1985

[Rechenmann 1993a] François Rechenmann, Integrating procedural and declarative knowledge in object-based knowledge models, Actes IEEE International Conference on Systems, Man and Cybernetics, Le Touquet (FR), pp98-101, 18-20 octobre 1993

[Rechenmann 1993b] François Rechenmann, Building and sharing large knowledge bases in molecular genetics, Actes 1st International Conference on Building and Sharing of Very

Large-Scale Knowledge Bases (KBKS), Tokyo (JP), pp291-301, (1-4 decembre) 1993 ftp://ftp.inrialpes.fr/pub/sherpa/publications/rechenmann93.ps.gz

[Rechenmann& 1988] François Rechenmann, Patrice Uvietta, Pierre Fontanille, SHIRKA, manuel d'utilisation, Rapport interne, IMAG, Grenoble (FR), 1988 (rev. 1989) ftp://ftp.inrialpes.fr/pub/sherpa/rapports/manuel-shirka.ps.gz

[Roberts& 1977] R. Roberts, Ira Goldstein, The FRL manual, Technical report AIM-409, MIT, Cambridge (MA US), 1977

[Rousseau 1998] Roger Rousseau, Eiffel: une approche globale pour programmer et réutiliser des composants logiciels, dans [Ducournau& 1998], pp35-72, 1998

[Sherpa 1995] Projet SHERPA, TROPES 1.0 reference manual, Rapport interne, INRIA Rhône-Alpes, Grenoble (FR), Juin 1995, 85p. (rev. TROEPS 1.2 reference manual, Juillet 1998, 145p.) ftp://ftp.inrialpes.fr/pub/sherpa/rapports/troeps-manual.ps.gz

[Sowa 1984] John Sowa, Conceptual structures: information processing in mind and machine, Addison-Wesley, Readings (MA US), 1984

[Stefik 1986] Mark Stefik, The next knowledge medium, *AI magazine* 7(1):34-46, 1986

[Stefik& 1986] Marc Stefik, Daniel Bobrow, Kenneth Kahn, Integrating access-oriented programming into a multiparadigm environment, *IEEE Software* 3(1):10-18, 1986

[Stefik 1995] Mark Stefik, Introduction to knowledge systems, Morgan Kauffman, San Francisco (CA US), 1995

[Valtchev& 1996] Petko Valtchev, Jérôme Euzenat, Classification of concepts through products of concepts and abstract data types, dans Edwin Diday, Yves Lechevalier, Otto Opitz (éds.), Ordinal and symbolic data analysis, Springer Verlag, Heildelberg (DE), pp3-12, 1996 ftp://ftp.inrialpes.fr/pub/sherpa/publications/valtchev96a.ps.gz

[Vismara& 1998] Philippe Vismara, Pierre Jambaud, Claude Laurenço, Joël Quinqueton, RéSyn: objets, classification et raisonnement distribué en chimie organique, dans [Ducournau& 1998], pp397-420, 1998

[Williams 1984] Chuck Williams, ART: The advanced reasoning tool, conceptual overview, Inference Corporation, Los Angeles (CA US), 1984

[Wright& 1983] J. Wright, Mark Fox, SRL/1.5 user manual, Carnegie-Mellon university, Pittsburg (PA US), 1983

[Woods 1975] William Woods, What's in a link: foundations for semantic networks, dans [Bobrow& 1975], pp35-82 (rep. dans [Brachman& 1985], pp217-241, 1985), 1975

### Syntaxe d'un objet

<objet> ::= '<'nom-d-objet
nom-attribut-1 '=' valeur-attribut-1 ';'… nom-attribut-n '=' valeur-attribut-n ';''>'

### Syntaxe d'une classe

<classe> ::= '<' nom-de-classe
'sorte-de' '=' nom-de-superclasse ';'
'attributs' '=' '{' <specif-d-attribut> '}' ';'
'contraintes' '=' '{' <specif-de-contrainte> '}' ';''>'

### Syntaxe d'une spécification d'attribut

<specif-d-attribut> ::= '<' nom-d-attribut <facettes>* '>'

<facette> ::= ('$un'|'$liste-de'|'$ensemble-de') <type>

| '$domaine' '{' <valeur>* '}'
| '$sauf' '{' <valeur>* '}'
| '$intervalle' (']'|'[') <valeur> <valeur> (']'|'[')
| '$card' (']'|'[') <entier> <entier> (']'|'[')

| '$defaut' <valeur>
| '$var<-' <nom>
| '$var-nom' <nom>
| '$var->' <nom>
| '$sib-exec' <classe>
| '$sib-filtre' <classe>

### Syntaxe d'une spécification de contrainte (non spécifiée)

nom-xxx est une chaîne de caractère;
<valeur> est une valeur d'un type ou un objet.

### Exemple:

```
< resistance-telle-marque
   sorte-de = resistance;
   attributs = {
      <lui-meme $var-nom lui>,
      <intensite $un reel>,
      <v-entrée $un reel $intervalle [0. 1000.]>,
      <v-sortie $un reel $intervalle [0. 1000.]>,
      <resistance $un entier $domaine {5 10 20 50}>,
      <modeles-nomenclature $liste-de chaine $sauf {"no-ref"}>,
```

```
   <c-entree $un composant
      $sib-filtre
         <composant
            attributs = {
               <connections $liste-de composant
                  $var<- lui>,
               <lui-meme $var-> c-sortie>};>;>};
contraintes = {
   v-entree - v-sortie = intensite * resistance;
   v-entree = c-entree.voltage;}; >
```