Représentations à objets et raisonnement par classification en intelligence artificielle

Amedeo Napoli

Université de Nancy 1
UFR S.T.M.I.A.

CRIN–INRIA Lorraine
GFD Informatique

THESE

présentée et soutenue publiquement le ***31 Janvier 1992***

pour l'obtention du

Doctorat d'Etat ès Sciences Mathématiques
(Spécialité Informatique)

Représentations à objets et raisonnement par classification en intelligence artificielle

par

Amedeo NAPOLI

devant le jury composé de :

Président :	M. Jean-Pierre FINANCE
Rapporteurs :	M. Jacques FERBER
	Mme Odile FOUCAUT
Examinateurs :	M. Roland DUCOURNAU
	M. Jean-Paul HATON
	M. Claude LAURENÇO
	M. Gérald MASINI

Remerciements

Je remercie M. Jean-Pierre Finance qui a accepté de présider le jury de cette thèse, Odile Foucaut et Jacques Ferber qui en ont été les rapporteurs ; leurs suggestions et encouragements m'ont beaucoup aidé et rassuré. Je remercie Jean-Paul Haton pour avoir dirigé mon travail, pour son appui et ses conseils, pour sa disponibilité et pour la confiance qu'il m'a accordée. Je remercie Claude Laurenço, qui a dirigé la partie de mon travail relative à la chimie ; il m'a enseigné les rudiments de la synthèse organique, mais aussi beaucoup d'autres choses, scientifiques et non scientifiques. Je remercie Roland Ducournau, "maître ès objets", sa curiosité et sa vision des mots et des choses m'étonneront toujours ; ses réflexions et ses commentaires, y compris les commentaires musicaux, s'avèrent bien souvent de précieuses indications. Je remercie Gérald Masini, autre "maître ès objets" ; le temps passé ensemble à écrire et les nombreuses discussions, parfois tourmentées, que nous avons eues au sujet des lignes qui suivent et de toutes les autres, m'ont beaucoup apporté.

Il y a beaucoup d'autres personnes que je voudrais remercier,
elles sont ici ou là-bas,
je ne peux ni ne veux les nommer,
elles se reconnaîtront, n'est-ce pas ?
je les remercie de tout cœur pour tout ce qu'elles ont fait.

Lisez Moi

Les animaux se divisent en : a) appartenant à l'Empereur, b) embaumés, c) apprivoisés, d) cochons de lait, e) sirènes, f) fabuleux, g) chiens en liberté, h) inclus dans la présente classification, i) qui s'agitent comme des fous, j) innombrables, k) dessinés avec un pinceau très fin en poils de chameau, l) et cætera, *m) qui viennent de casser la cruche, n) qui de loin semblent des mouches.*

Cette phrase, empruntée à Michel Foucault dans *Les mots et les choses*[1], qu'il aurait lui-même empruntée à Jorge Luis Borges, et qui m'a été soufflée par Roland Ducournau, montre bien l'ampleur du problème de la classification. Il y a une infinité de façons de décrire une chose, comme il y a aussi une infinité de façons de la présenter.

L'être humain sait généralement comment réagir lorsqu'il perçoit une entité, animée ou non. Les informations sensorielles qui parviennent à un observateur lui permettent de reconnaître un lion ou un verre de bière et de réagir en conséquence, en utilisant la connaissance qu'il possède des entités qu'il perçoit, en accord avec les objectifs qu'il s'est fixé. Cette réaction fait suite à une série d'inférences, qui sont des conséquences de la classification de l'entité reconnue dans une catégorie déterminée. La reconnaissance complète la perception et permet d'appréhender globalement une entité, du point de vue de son statut et de son comportement. Elle rend également possible la prédiction de caractéristiques non observables à partir de celles qui ont été effectivement observées. Ainsi, si le lion a l'air belliqueux, il vaut mieux ne pas s'en approcher, si la bière a une couleur inhabituelle, il vaut mieux ne pas la boire. L'efficacité de ce type de raisonnement, qualifié de raisonnement par classification, suppose que les catégories sont ordonnées et dépend du niveau des catégories sur lesquelles il s'applique. L'appartenance à la catégorie des animaux autorise moins de déductions que l'appartenance à la catégorie des lions. Parallèlement, la réaction d'un observateur dépend du milieu dans lequel il est plongé, de ses connaissances sur le monde observé et de ses expériences passées. En particulier, plus l'ensemble de connaissances dans lequel puise l'observateur est riche, plus le raisonnement relève de la remémoration et moins il relève de l'improvisation.

Le terme "classification" recouvre plusieurs significations selon le contexte dans lequel il est utilisé. Le sens qui lui est accordé en analyse de données, mais aussi dans le dictionnaire, est celui de distribution en classes ou catégories : étant donné un ensemble d'objets, trouver une répartition en classes de ces objets en regroupant ceux qui possèdent des caractéristiques et un comportement similaires. S'appuyant

1. Michel Foucault, *Les mots et les choses*, Gallimard, Paris, 1966.

sur un anglicisme, le sens qui lui est communément accordé en intelligence artificielle (IA) est celui de classement : trouver la classe d'appartenance d'un objet. Dans la suite, le terme "classification" a une connotation liée au raisonnement, donc liée au sens IA du terme, tandis que le terme "catégorisation" désigne l'acte de construction d'un ensemble de classes contenant des individus. Ces précisions étant données, cette thèse peut être présentée comme une étude des phénomènes de classification et de catégorisation pour la représentation de connaissances. Sont examinées en détail les techniques de construction de hiérarchies d'objets et les techniques de raisonnement associées. La classification et la catégorisation s'avèrent être bien adaptées à la gestion d'univers où les objets sont assimilables à des graphes et à la résolution de problèmes sur de tels univers.

Le chapitre 1 introduit brièvement les principes sur lesquels sont conçus les systèmes à bases de connaissances classiques, ainsi que les bases du raisonnement par classification. Représenter des connaissances pour un système informatique consiste à élaborer des structures de données appropriées au stockage et à la manipulation d'informations. En s'appuyant sur ces informations, le système tente de résoudre des problèmes relatifs à un domaine étudié donné, à la façon d'un être humain. Il existe deux modèles principaux pour décrire une catégorie d'objets : le modèle classique des conditions nécessaires et suffisantes et celui plus moderne de la théorie du prototype. Ces catégories sont alors naturellement organisées en hiérarchies, sur lesquelles opère le raisonnement par classification. Ce type de raisonnement est l'un de ceux employés par l'être humain lorsque ce dernier veut prendre en compte et manipuler des situations typiques, par exemple. De ce point de vue, le raisonnement par classification cherche à expliquer une situation nouvelle en la comparant avec une situation connue, ou encore à reconnaître un objet en appariant ses caractéristiques avec celles de catégories d'objets connues, afin de découvrir la catégorie à laquelle l'objet pourrait se rattacher. Le chapitre 1 se termine par un regard vers certaines techniques de représentation à base d'objets qui relèvent de l'intelligence artificielle distribuée.

Le chapitre 2 définit les représentations à objets, qui sont des descendants de formalismes de représentations comme les réseaux sémantiques, les langages de frames et les langages hybrides. A l'instar des systèmes à subsomption, les langages de la famille KL-ONE, avec lesquels elles partagent de nombreuses caractéristiques, les représentations à objets ne sont pas que de simples langages à objets dédiés à la représentation des connaissances, car le raisonnement par classification y tient une place prépondérante. Les connaissances relatives à un domaine d'application sont décrites par des objets qui se présentent comme des collections de propriétés et qui sont organisés en hiérarchies. Les principales opérations effectuées sur une hiérarchie sont la mise en place ou la suppression d'un objet dans la hiérarchie, l'accès aux propriétés d'un objet, ainsi que la recherche d'objets vérifiant certaines contraintes. Des extensions comme la réflexivité du noyau de la représentation, les objets composites, les objets temporels et les perspectives sont également discutées. En particulier, les hiérarchies représentant des connaissances divisent et ordonnent les objets, mais souvent du seul point de vue de l'héritage. Les niveaux de la hiérarchie correspondent à des niveaux de spécialisations. Or, il y a autant de catégorisations, donc de classifications possibles, qu'il y a d'interprétations des relations d'ordre partiel existant entre les objets clas-

sés. Savoir construire des catégorisations croisées reflétant plusieurs points de vue est fondamental pour pouvoir donner toute sa force au raisonnement par classification.

Le chapitre 3 donne le détail du raisonnement par classification dans une représentation à objets. Ce dernier joue un rôle primordial dans la recherche, la gestion et la maintenance des informations, ainsi que comme méthode de résolution de problèmes. Dans une représentation à objets, les connaissances sont partiellement ordonnées en une hiérarchie d'héritage et la valeur associée à une propriété pour une entité donnée se déduit en examinant les ascendants de l'entité dans la hiérarchie. L'héritage est un mécanisme de partage de connaissances et de partage de code, mais il ne permet pas de produire de nouvelles connaissances à partir de connaissances déjà présentes. Il ne peut donc pas être l'unique mécanisme d'inférence dans une représentation à objets. La relation de subsomption généralise la relation d'héritage et pallie certaines de ses limitations en tant que mécanisme d'inférence. En toute généralité, un concept subsume un autre concept si le premier est nécessairement plus général que le second. C'est sur la subsomption que repose alors l'arrangement des concepts dans une hiérarchie, et, par suite, la catégorisation et le raisonnement par classification. C'est entre autre l'un des principaux emprunts faits par les représentations à objets aux systèmes à subsomption, dont une brève présentation termine le chapitre.

Dans le chapitre 4 sont abordés les problèmes de catégorisation selon des points de vue numériques et conceptuels. Dans les deux cas, la catégorisation sert à interpréter de grands ensembles de données non structurés, respectivement numériques et symboliques. Puisque les classes produites correspondent à des concepts, la catégorisation fait partie intégrante des techniques d'apprentissage et d'acquisition de connaissances. Les systèmes de formation incrémentielle de hiérarchie de concepts en fournissent un exemple typique, où catégorisation et classification sont intimement associées : la tâche des ces systèmes consiste à construire une hiérarchie de concepts regroupant des données hétérogènes en classifiant ces données une par une. La fin du chapitre est d'ailleurs dévolue à l'étude des analogies existant entre la classification et la formation de hiérarchies, incrémentielle ou non, ainsi qu'entre le raisonnement par classification et le raisonnement par cas.

Le chapitre 5 décrit la partie expérimentale de cette thèse, la construction d'un système de conception de plans de synthèse de molécules organiques. Le système a pour but de simuler le raisonnement du chimiste durant l'élaboration du plan de synthèse d'une molécule. Cette opération consiste à préparer une molécule, dite molécule cible, à partir de réactifs de départ, qui sont en général des molécules facilement accessibles. Dans le modèle présenté, une synthèse est considérée comme un système temporel qui évolue d'un état initial vers un état final en passant par une suite d'états intermédiaires. Chaque état est décrit par un réacteur qui se compose d'un ensemble d'objets primaires, les atomes et les liaisons, et d'un ensemble d'objets secondaires déterminé par la connexion des liaisons, les molécules qui entrent en jeu dans la synthèse. A un instant donné, un état correspond à la distribution des liaisons faisant suite à l'action d'une réaction appliquée au réacteur existant au temps précédent. Les états sont donc ordonnés dans le temps et ils matérialisent des objectifs devant être atteints au cours de la synthèse. En adoptant un mode de résolution de problèmes de synthèse

rétrosynthétique, l'état initial est constitué d'une molécule cible et l'état final d'un ensemble de réactifs de départ connus et disponibles. Le système est implanté sous la forme d'une représentation à objets et le raisonnement employé pour résoudre les problèmes de synthèse est le raisonnement par classification. Certains outils associés aux représentations à objets, qui sont décrits de façon théorique dans les premiers chapitres, trouvent un terrain d'application privilégié. C'est notamment le cas pour le raisonnement par classification qui porte simultanément sur la relation d'héritage et la relation de composition, et pour les objets temporels qui entrent en jeu dans la représentation de plans de synthèse.

Enfin, le chapitre 6 clôt la thèse en présentant un bilan de la contribution apportée par ce travail aux études menées sur la classification et la catégorisation en représentation des connaissances, et en indiquant les perspectives de continuation de ce travail.

Table des matières

1
Objets et représentation de connaissances

En toute généralité, représenter des connaissances pour un système informatique "intelligent" consiste à élaborer des structures de données appropriées au stockage et à la manipulation d'informations, qui portent sur des objets du monde réel, leur environnement et les façons de les manipuler. En s'appuyant sur ces informations, le système est censé résoudre des problèmes relatifs au domaine étudié, éventuellement à la façon d'un être humain. Ces considérations expliquent pourquoi il existe des liens assez étroits entre la représentation des connaissances et des disciplines comme la philosophie, la psychologie et la linguistique.

1.1 Représentation de connaissances

1.1.1 La modélisation des connaissances

Le cycle abstraction-inférence-interprétation

Nous supposons que, pour résoudre un problème, un être humain raisonne sur des concepts abstraits qui modélisent les objets de l'univers du problème, en tire des conclusions qu'il interprète ensuite dans cet univers [Chouraqui *et al.*, 1985]. La reproduction d'un tel comportement, lorsqu'elle est effectuée par un programme, peut être vue comme un processus qui comporte quatre étapes principales (cf Fig. 1.1). L'étape *d'abstraction* est associée à la perception d'une entité et à sa description sous la forme d'une "structure mentale". Cette structure est ensuite représentée en machine sous la forme d'un ensemble de phrases ou expressions symboliques bien formées d'un langage de représentation. Parmi ces phrases, certaines sont permanentes et d'autres sont temporaires. Les premières sont celles qui sont en rapport avec la connaissance dont dispose le système sur le domaine étudié, et peuvent être considérées comme un modèle utilisable du domaine. Les secondes sont celles qui sont en rapport avec un problème particulier. Les phrases temporaires sont ensuite combinées avec les phrases permanentes, ce qui a pour conséquence de produire de nouvelles phrases, qui sont interprétées en termes d'objets du domaine. Ces nouvelles phrases doivent (en principe) apporter des éléments de solution au problème considéré.

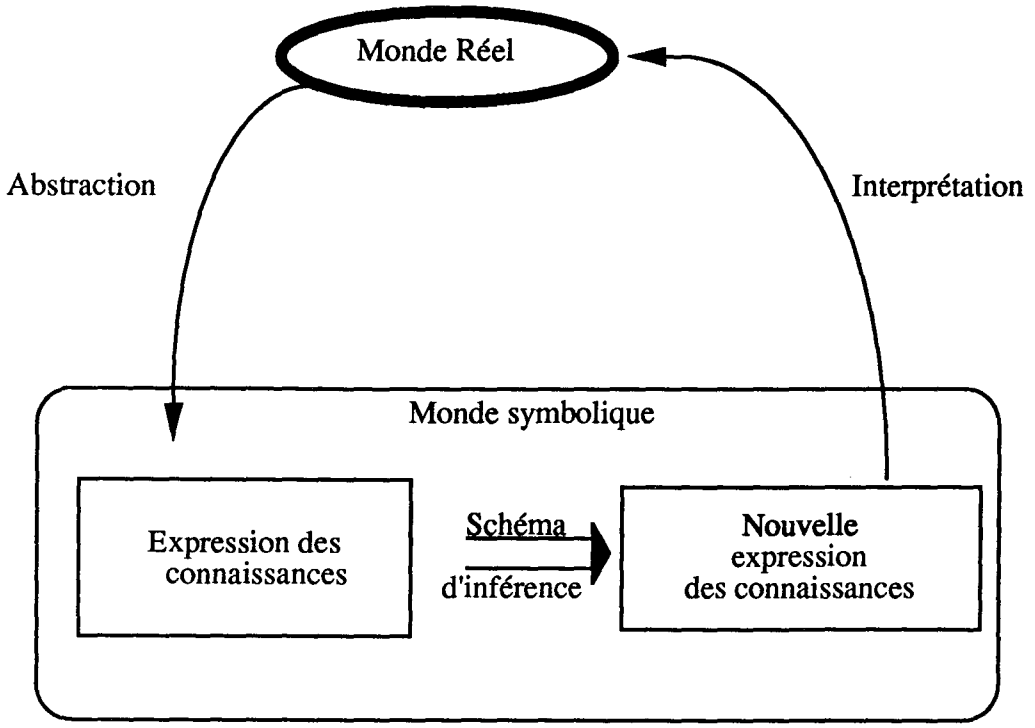


Figure 1.1. Le *cycle cognitif artificiel* : abstraction, représentation, inférences et interprétation, d'après [Chouraqui *et al.*, 1985].

L'intension et l'extension d'un concept

Un concept représente une entité, une action ou un état qui existe dans un univers. Depuis G. Frege [Frege, 1893] [Frege, 1971], un concept peut être vu comme une fonction définie sur un domaine de référence, qui prend ses valeurs dans l'ensemble {*vrai*, *faux*}. Cette fonction a pour but de discriminer les objets auxquels s'applique le concept - la fonction prend alors la valeur *vrai* et ces objets, appelés *référents*, *tombent* sous le concept ou sont *recouverts* par lui - de ceux auxquels il ne s'applique pas. L'*extension* d'un concept est l'ensemble des objets qui sont recouverts par le concept : l'extension du concept *Hirondelle* est composée par l'énumération de toutes les hirondelles. Le terme *intension* est d'utilisation plus récente et reprend l'idée plus ancienne de *compréhension* d'un concept [Arnauld and Nicolle, 1662], qui désigne l'ensemble des caractères définissant un concept, ou, dans certains cas encore, l'ensemble des conditions nécessaires et suffisantes devant être vérifiées par un objet pour être recouvert par le concept[1]. Par exemple, l'intension du concept *Hirondelle* contient les caractères propres aux hirondelles comme la couleur de leur plumage, la forme de leur queue, leur attitude en vol, etc., mais aussi les caractères propres aux oiseaux, comme le fait de porter des plumes et de pondre des œufs, et les caractères propres

1. Il existe des interprétations plus actuelles de la notion d'intension, qui ne seront pas détaillées ici. Il est possible de consulter à ce sujet [Carnap, 1958], [Desclés, 1986] et [Desclés and Kanellos, 1991].

aux animaux en général. En particulier, l'intension de *Hirondelle* contient l'intension des concepts que *Hirondelle* "comprend", c'est-à-dire l'intension de concepts comme *Oiseau* et *Animal* : les hirondelles possèdent les caractéristiques associées aux animaux, plus celles qui sont associées aux oiseaux, plus celles qui leur sont propres. De façon duale, l'extension de *Animal* contient celle de *Oiseau* qui contient à son tour celle de *Hirondelle* : l'énumération de tous les animaux contient l'énumération de tous les oiseaux, qui contient l'énumération de toutes les hirondelles.

Comme il est généralement impossible d'énumérer tous les référents d'un concept, c'est plutôt l'intension des concepts qui est décrite et manipulée dans les applications [Beech, 1988], sauf dans des cas très particuliers où il est possible de travailler avec une extension, comme en chimie par exemple, avec les 103 atomes (actuels) de la classification périodique des éléments.

A propos de structures et de relations

Un concept n'est généralement pas une entité isolée, il existe dans un certain environnement et il entretient des relations avec d'autres concepts. Le modèle *relationnel* et le modèle *structurel*[2] sont les deux manières principales d'appréhender un concept [Brodie *et al.*, 1984]. Le premier privilégie la description de l'ensemble des relations dans lesquelles le concept intervient ; la connaissance disponible sur le concept est répartie dans chaque relation. Le second met l'accent sur le concept en tant qu'objet ; la connaissance disponible sur le concept est alors regroupée au sein d'une entité autonome. Le modèle relationnel recouvre entre autres les bases de données relationnelles[3] et les formalismes "logiques", comme la logique des prédicats et les règles de production. Le modèle structurel, quant à lui, recouvre l'ensemble des formalismes de représentation fondés sur la notion d'objet, en particulier, les classes, les frames et les acteurs.

Si la description d'un ensemble de concepts est schématisée par un tableau à double entrée, où les entrées des lignes correspondent aux concepts et celles des colonnes aux attributs de ces concepts, alors le modèle structurel est obtenu en lisant le tableau ligne par ligne, le modèle relationnel en lisant le tableau colonne par colonne (cf. Fig. 1.2).

1.1.2 Systèmes à base de connaissances

Représenter = stocker + raisonner

Un *système à base de connaissances* est un programme capable d'accomplir une tâche "intelligente". Dans son architecture classique, un tel système comprend une base de connaissances relatives à un domaine d'application et un programme qui manipule cette base, recherche les informations et "raisonne" pour résoudre un problème donné.

2. Les termes *prédicatif* pour relationnel, et *conceptuel* pour structurel, sont également employés [Ducournau, 1989b].

3. Une base de données est un formalisme de représentation qui renferme de la "connaissance statique", par opposition à la "connaissance dynamique et auto-évolutive" contenue dans une base de connaissances [Brodie *et al.*, 1984] [Bdbc, 1987] [Brachman, 1988].

Objets \ Attributs	a-un-bec	a-des-plumes	pond-des-oeufs	a-des-dents	apte-au-vol	a-des-poils	allaite-ses-petits	apte-à-la-nage	a-des-ailes
moineau	1	1	1	0	1	0	0	0	1
chien	0	0	0	1	0	1	1	1	0
manchot	1	1	1	0	0	0	0	1	1
vache	0	0	0	1	0	1	1	0	0
kiwi	1	1	1	0	0	0	0	0	1
autruche	1	1	1	0	0	0	0	0	1
ornithorynque	1	0	1	0	0	1	1	1	0
brochet	0	0	1	1	0	0	0	1	0
baleine	0	0	0	0	0	0	1	1	0

L'objet moineau possède les attributs a-des-plumes, a-un-bec, pond-des-oeufs, apte-au-vol, a-des-ailes.

Les formules prédicatives apte-au-vol(moineau) ou apte-à-la-nage(manchot) sont vraies, les formules apte-au-vol(manchot) ou pond-des-oeufs(vache) sont fausses.

Figure 1.2. Un tableau caractéristiques-individus à valeurs booléennes. La lecture en ligne correspond au modèle structurel objet-attributs. La lecture en colonne correspond au modèle relationnel relations-individus.

La base de connaissances contient des faits ou données brutes caractérisant les objets du domaine considéré, des règles permettant de manipuler ces faits, ainsi que des heuristiques et des stratégies de raisonnement exprimant la façon de se servir des règles. Les opérations de manipulation des connaissances, recherche et raisonnement, sont indissociables de la représentation elle-même. Représenter des connaissances consiste donc à trouver des structures de données appropriées au stockage et à la manipulation d'informations [Kayser, 1984] [Levesque, 1986a].

Un système à base de connaissances se caractérise par sa capacité à utiliser explicitement des connaissances qui sont stockées dans le but d'accomplir une certaine tâche,

de raisonner pour résoudre un problème, d'analyser et d'exploiter son environnement. La connaissance est alors considérée comme une *entité calculable*, ce qui ne va pas sans poser des problèmes pratiques et philosophiques [Brachman, 1990] :

- Sous quelle forme doivent s'exprimer les connaissances ?
- Comment un mécanisme de raisonnement peut-il exploiter au mieux un ensemble de connaissances forcément limité, et comment peut-il tirer de cet ensemble toutes les informations implicites qu'il renferme ?
- Comment les connaissances produites influent sur le comportement du système ?
- Comment raisonner avec des informations incomplètes ou bruitées ?
- Comment faire aboutir un raisonnement alors que l'éventail des possibilités de recherche d'une solution est virtuellement infini ?

Le concepteur d'une base de connaissances doit choisir les unités de connaissance à représenter, les décrire, les organiser et leur imposer des contraintes de façon à ce que la base construite soit le reflet de l'univers modélisé [Sowa, 1984]. Il est alors possible de faire référence à la façon de faire des êtres humains lorsqu'ils s'adonnent au même genre de tâche. Dans ce cadre, l'hypothèse selon laquelle l'être humain construit et utilise des modèles mentaux de la réalité dans ses raisonnements permet de mettre en lumière les rapports qui existent entre la représentation des connaissances et d'autres disciplines faisant partie pour la plupart des sciences humaines [Johnson-Laird, 1983] :

- Psychologie, en rapport avec la théorie des modèles mentaux, la perception et la catégorisation : comment sont représentés les concepts dans l'esprit humain, quelle place y occupent-ils, et comment affectent-ils le comportement de l'être humain ?
- Linguistique, en rapport avec la représentation de l'intension et de l'extension d'un concept (cf. Fig. 1.3) : quelle est la relation entre un objet, le terme qui le dénote et le modèle mental qui décrit l'objet ? Quelles sont les règles de syntaxe et de sémantique qui régissent la construction de phrases modélisant le résultat de perceptions effectuées sur l'univers considéré ?
- Philosophie, en rapport avec les diverses façons de raisonner : quel sens donner aux unités de connaissances présentes dans une base et, parallèlement, comment sont utilisés les modèles mentaux dans le raisonnement, et comment ce raisonnement est-il lié à la logique formelle ?

A un niveau plus pratique, il reste à intégrer dans un programme la représentation d'un modèle mental. Pour cela, il faut choisir des langages et des outils qui sont adaptés à la représentation des modèles et à la gestion des communications entre les modèles et l'extérieur. Les formalismes de représentation des connaissances qui existent prennent en compte plus ou moins bien les problèmes évoqués ci-dessus [Barr *et al.*, 1981]. La plupart de ces formalismes sont déclaratifs, les connaissances expriment ce que

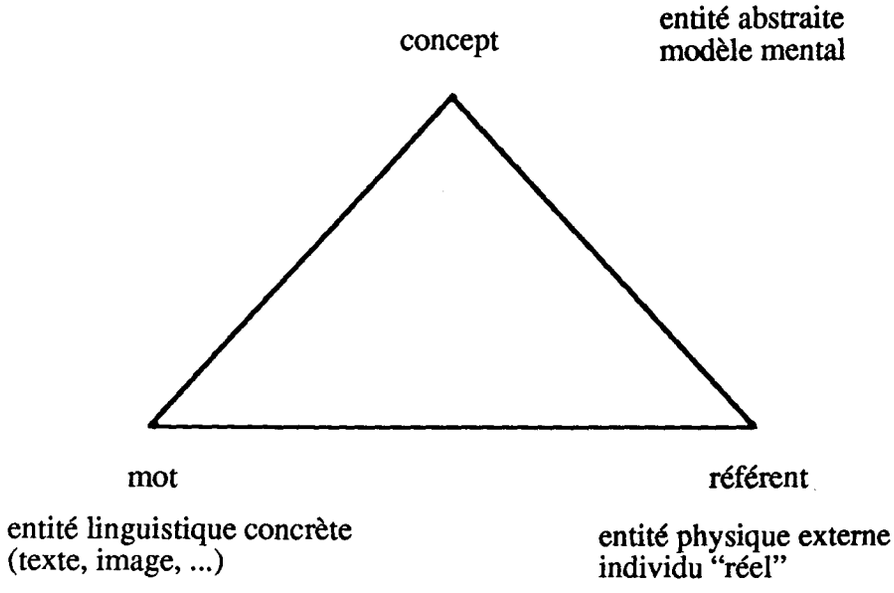


Figure 1.3. L'objet, le mot et le sens (d'après [Ogden and Richards, 1923], repris et adapté dans [Sowa, 1984] et dans [Regoczei and Hirst, 1990]).

l'on sait, le "quoi" du monde étudié, par opposition à un formalisme procédural, où est explicité le "comment" des choses. Dans la suite, les problèmes de représentation sont abordés du point de vue des formalismes où la modélisation des concepts est structurelle, essentiellement classes et frames.

Soulignons, pour terminer, l'importance croissante du couplage bases de données – bases de connaissances [Bdbc, 1987]. Certaines techniques développées dans le contexte des bases de données sont utilisées pour gérer certains aspects des bases de connaissances, en particulier lorsque la taille des bases de connaissances dépasse un certain niveau. Des exemples d'intégration des deux univers peuvent être trouvés dans les rapports décrivant l'actuel projet CYC, pour lequel est incorporé toute connaissance jugée utile ou pertinente dans la vie quotidienne [Lenat and Guha, 1990].

Quelques modes de raisonnement

Comme les êtres humains, mais au contraire des objets mathématiques, les objets du quotidien obéissent rarement à des lois rigoureuses. Intégrer les différentes natures de ces objets dans un formalisme de représentation pose de nombreux problèmes difficiles à résoudre [Kayser, 1984] : représentation de modalités (statut et degré de vérité des informations), représentation de connaissances typiques et exceptionnelles, représentation de connaissances incomplètes, évolutives, interdépendantes, etc.. Pour prendre en compte les natures diverses et variées des connaissances, de nombreux formalismes de représentation et de raisonnement ont été mis au point. Au côté des traditionnels raisonnements déductif, inductif et par analogie, les raisonnements

non monotone, abductif, qualitatif, par classification, par cas, temporel et probabiliste, voient leur importance augmenter [Haton *et al.*, 1991]. L'étude du raisonnement non monotone est primordiale, dans la mesure où le raisonnement de bon sens est non monotone par essence [Reiter, 1987] : de nouvelles connaissances ou perceptions viennent sans arrêt remettre en cause des hypothèses valides jusque là. Le raisonnement abductif recherche justement des hypothèses pour expliquer une situation, en essayant de relier les caractéristiques de la situation à ce qui pourrait l'avoir causé [Marquis, 1991]. Le raisonnement qualitatif s'attache à expliquer symboliquement les mesures numériques décrivant le fonctionnement d'un système physique [Qrps, 1984]. Le raisonnement par cas consiste à tirer profit d'un catalogue d'expériences passées et mémorisées : réutiliser une expérience passée au présent, l'adapter, puis la mémoriser, pour la réutiliser dans des circonstances analogues dans le futur [Slade, 1991].

Proche du raisonnement par cas, le raisonnement par classification permet de gérer des connaissances par l'intermédiaire de catégories organisées en hiérarchies. L'être humain a d'autant plus de mal à manipuler des faits et à raisonner que le contexte dans lequel sont effectuées ces opérations est abstrait. Pour découvrir la solution d'un problème, il a plutôt tendance à raisonner d'abord sur un cas particulier, puis à essayer de tirer des généralisations du résultat obtenu [Cherniak, 1984]. Penser un problème en termes d'exemples permet de choisir un niveau d'abstraction où les concepts sont plus faciles à manipuler. Toutefois, l'exemple choisi doit être un bon exemple : il doit simplifier le raisonnement, être suffisamment spécifique pour pouvoir être manipulé aisément, mais aussi être suffisamment général pour pouvoir être adapté à une majorité de cas. Plutôt qu'une spécialisation particulière, un exemple doit donc être un élément représentatif ou typique d'une catégorie [Cohen and Murphy, 1984]. Ce type de raisonnement, fortement lié aux notions de catégorie et de niveau d'abstraction, est celui qui est étudié en détail dans la suite.

1.2 Connaître, c'est décrire pour retrouver

Ce paragraphe[4] introduit la problématique de la catégorisation. Cette introduction a pour support le livre de G. Kleiber, *La sémantique du prototype* [Kleiber, 1990]. Elle montre la catégorisation sous un jour plutôt psycho-philo-linguistique. Le chapitre 4 donne une vision plus informatique de la catégorisation. Il la traite du point de vue numérique de la classification automatique et du point de vue conceptuel de la construction incrémentielle de hiérarchies de concepts.

1.2.1 La catégorisation

La catégorisation est essentielle parce qu'elle est une des façons principales de stocker des expériences, des situations et, plus généralement, toutes sortes d'objets ou d'éléments vécus [Lakoff, 1987]. Cette opération mentale, qui consiste à ranger ensemble des "choses" *a priori* différentes, c'est-à-dire à découvrir les régularités qui

4. L'expression qui donne son titre au paragraphe est de Gaston Bachelard [Bachelard, 1973].

existent entre ces choses, se retrouve dans toute nos activités, qu'elles soient de pensée, de perception, d'action, de communication, de compréhension, etc. [Cauzinille-Marmèche *et al.*, 1990]. L'explication d'une perception passe généralement par une classification/catégorisation qui apparaît comme une reconnaissance de l'espèce de la chose, de sa "catégorie". Catégorisation et catégories sont des éléments fondamentaux de notre organisation de l'expérience. Sur un tel constat s'appuient la théorie des modèles mentaux [Johnson-Laird, 1983], celle des systèmes de frames [Minsky, 1975], et celle des scripts [Schank, 1982].

Sans catégorisation, sans la capacité à dépasser le niveau de l'entité individuelle, le monde serait "plat" et toute entité perçue de quelque façon que ce soit resterait unique. Les choses seraient ainsi très difficiles à mémoriser et donc à réutiliser [Smith and Medin, 1981]. La pensée humaine a avant tout affaire à des catégories et non à des éléments particuliers (prépondérance de l'intension sur l'extension). Il est donc essentiel pour modéliser toute approche du raisonnement de savoir comment s'effectue la catégorisation et de comprendre le processus qui fait que telle chose particulière est rangée avec telle autre. En d'autres termes, il faut savoir répondre à deux questions fondamentales : quels sont les critères qui décident de l'appartenance d'un membre à une catégorie et comment s'effectue un regroupement.

Il existe deux types de réponses, qui proviennent de deux courants différents et que nous qualifions de courants *dur* et *mou*[5]. Pour le premier, la catégorisation s'opère sur la base de propriétés communes, les membres d'une catégorie présentent des traits ou caractéristiques identiques. Un objet particulier est une tortue s'il possède les caractéristiques qui définissent le concept de *Tortue* : l'objet est un animal qui est théoriquement moins rapide qu'un lièvre, qui a de petites pattes, une carapace, une tête sympathique, une chair qui fait rêver les gourmets, etc.. Les objets sont alors regroupés sur la base d'un certain nombre de propriétés partagées qui sont communément culturellement admises, ce qui n'est peut-être pas le cas de la description qui a été donnée de la tortue. Pour le courant mou, la catégorisation s'appuie sur la *théorie du prototype*, qui rompt avec la thèse aristotélicienne précédente : l'appartenance d'un membre à une catégorie ne se fait plus sur la base de propriétés communes partagées par tous les membres, propriétés jouant le rôle de conditions nécessaires et suffisantes d'appartenance à la catégorie, mais sur une certaine similitude avec un élément jugé représentatif de la catégorie, appelé *prototype*. Le processus de catégorisation consiste alors à exhiber des prototypes et établir des similitudes entre prototypes et nouveaux membres [Cauzinille-Marmèche *et al.*, 1990] [Rips, 1989].

Vu le rôle central joué par la catégorisation dans l'activité mentale humaine, une théorie qui explique au mieux la catégorisation est peut-être en mesure d'expliquer les processus de raisonnements humains, au moins une bonne partie d'entre eux. C'est sur une telle théorie qu'il faut alors s'appuyer pour construire un système qui ait des chances d'être "intelligent".

5. Ces deux courants sont respectivement appelés les courants *objectiviste* et *expérientialiste* dans [Kleiber, 1990].

Le modèle des conditions nécessaires et suffisantes

Pour décider de l'appartenance d'un x à la catégorie des chats, il suffit de vérifier que x possède les attributs qui constituent le dénominateur commun de la catégorie, autrement dit x doit posséder les propriétés communes à tous les chats, être un animal, miauler, aimer la chaleur, aimer dormir, etc.. S'il s'avère qu'une des propriétés n'est pas détenue par le candidat, celui-ci ne peut pas être considéré comme un chat et il est donc rejeté. Ce type de catégorisation s'appuie sur un modèle de conditions nécessaires et suffisantes qui s'expriment sous la forme des propriétés communes à tous les individus faisant partie de la catégorie : si un x est un chat alors x est un animal, x miaule, x aime la chaleur et aime dormir, et réciproquement, si x vérifie ces conditions, alors x est un chat.

Dans ce type de catégorisation :

- les catégories sont clairement délimitées ;
- l'appartenance d'un individu à une catégorie est "booléenne", par opposition à "modale" qui exprimerait une certaine gradation dans l'appartenance : un individu est ou n'est pas membre de la catégorie, selon qu'il vérifie ou non les caractéristiques de la catégorie ;
- tous les membres d'une catégorie ont le même statut puisqu'ils ont les mêmes propriétés, et ils sont traités de la même façon.

Dans cette approche de la catégorisation, un objet appartient à une catégorie s'il vérifie les conditions nécessaires et suffisantes associées à la catégorie. Ces conditions décrivent l'intension du concept représenté par la catégorie et, parallèlement, elles déterminent l'extension du concept. Intension et extension sont clairement définies et, par conséquent, une catégorie a des limites bien précises. Toutefois, cette approche de la catégorisation ne s'applique pas forcément à toutes les catégories. Par exemple, l'extension d'un concept comme *Oiseau* est difficilement représentable par un ensemble de conditions nécessaires et suffisantes, en particulier, à cause de l'autruche et du manchot. Le modèle des conditions nécessaires et suffisantes ne permet pas de traiter convenablement les exceptions[6]. De plus, il omet de rendre compte du statut privilégié dont jouissent certains membres dans une catégorie.

La théorie du prototype

Il ressort de nombreuses expériences que les catégories semblent codées dans l'esprit non pas sous forme de listes énumérant les individus membres de la catégorie, pas davantage sous forme de listes énumérant les critères nécessaires et suffisants qui détermineraient l'appartenance à cette catégorie, mais plutôt sous la forme d'un prototype représentant le membre type de la catégorie considérée. Le codage le plus économique, sur le plan cognitif, pour représenter une catégorie, est bel et bien une image concrète du membre moyen de cette catégorie [Rosch, 1977].

6. Y compris les exceptions qui confirment les règles [Brachman, 1985].

La théorie du prototype, issue principalement des travaux de E. Rosch [Rosch and Mervis, 1975] [Rosch *et al.*, 1976], rompt radicalement avec l'approche classique "aristotélicienne" de la catégorisation, qui s'appuie sur le modèle des conditions nécessaires et suffisantes. Un *prototype* peut être vu comme le meilleur exemplaire d'une catégorie, meilleur représentant ou instance du concept représenté par la catégorie. La nouveauté est de taille, puisque l'existence dans une catégorie de membres qui sont "meilleurs" que d'autres est admise. Ainsi, la pomme, l'orange ou la banane sont de "meilleurs" fruits que l'olive ou la tomate, sans qu'il soit question de goût, bien sûr... La notion de prototype est ainsi fortement liée aux individus qui constituent la catégorie. Le consensus qui existe dans une communauté pour appréhender les choses est de première importance dans cette théorie. Un prototype n'est vraiment considéré comme le meilleur exemplaire d'une catégorie que s'il est le plus connu, ou bien s'il apparaît comme l'exemplaire le plus fréquemment cité comme tel [Dubois, 1986].

Cette approche de la catégorisation s'appuie sur les principes suivants :

- L'appartenance d'un individu à une catégorie se fait sur la base d'un appariement avec un prototype. L'appariement complet n'est pas nécessaire.
- Le degré d'appartenance d'un individu à la catégorie correspond à son degré de similitude avec le prototype. Réciproquement, le degré de représentativité d'un individu, sa potentialité à être un prototype, correspond à son degré d'appartenance à la catégorie.
- Les membres d'une catégorie n'ont pas tous exactement le même statut et les mêmes propriétés, mais ils partagent un certain air de famille, une ressemblance, qui permet de les regrouper.
- Les limites d'une catégorie sont floues : un individu peut appartenir à la catégorie "plus" qu'un autre[7].

Les points précédents suggèrent que les individus ne sont pas tous équivalents et que leur degré d'appartenance à une catégorie varie. Les catégories ne sont plus structurées comme dans le modèle des conditions nécessaires et suffisantes : le (ou les) prototype devient une "unité centrale" autour de laquelle s'organise la catégorie. Certains individus ont un degré de représentativité très faible, olive ou tomate pour fruit, autruche ou manchot pour oiseau, d'autres, un degré de représentativité moyen, framboise pour fruit, canari pour oiseau, d'autres encore, un très haut degré de représentativité, pomme ou orange pour fruit, moineau pour oiseau. Décider l'appartenance d'un objet à une catégorie n'est plus d'ordre "booléen" mais modal, et se mesure selon une certaine échelle. Voici un exemple d'une telle gradation dans l'appartenance à la catégorie des oiseaux :

1. Un moineau est un oiseau *(vrai)*.
2. Un poussin est un oiseau *(moins vrai que (1))*.

7. Il y a le parti et les sympathisants du parti.

3. Un manchot est un oiseau *(moins vrai que (2)).*

4. Une chauve-souris est un oiseau *(faux ou vrai de très loin).*

5. Un éléphant est un oiseau *(complètement faux).*

Si le degré de représentativité est équivalent à un degré d'appartenance, et si les limites d'une catégorie sont floues, alors le regroupement des individus dans une catégorie ne se fait plus sur la base de conditions nécessaires et suffisantes d'appartenance à la catégorie. Puisque les membres d'une catégorie ne partagent pas tous les mêmes caractéristiques, cette approche permet de prendre en compte plus facilement les individus exceptionnels. La relation qui unit les membres d'une catégorie est alors qualifiée d'*air de famille* [Rosch and Mervis, 1975]. La figure 1.4 montre un ensemble de caractéristiques associées aux oiseaux, caractéristiques qui ne sont pas toutes partagées par les individus décrits[8].

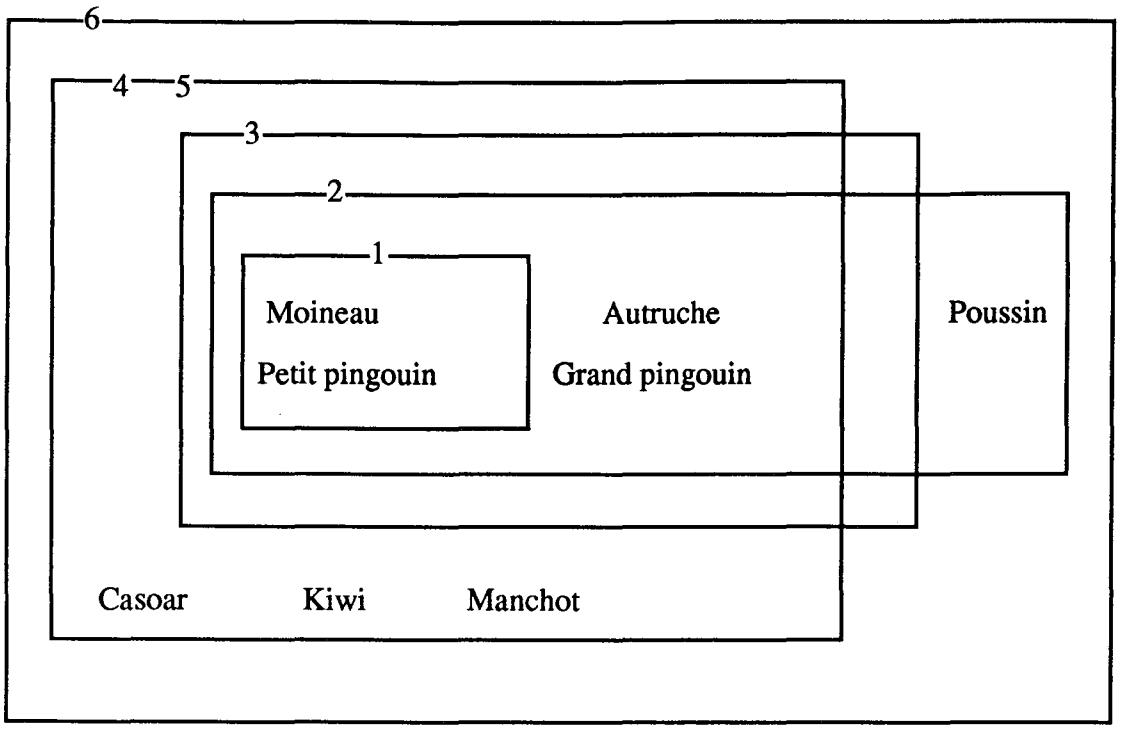
L'appariement est l'opération qui, par opposition à la vérification de conditions nécessaires et suffisantes, est à la base de la catégorisation[9]. Le prototype fonctionne alors comme un point de référence et les individus sont classés dans une catégorie en fonction de leur similitude globale avec le prototype. En particulier, les membres "prototypiques", qui peuvent être des prototypes, ou qui en sont très proches, ont des qualités particulières : ils servent de références, ils sont catégorisés plus vite que les autres membres, ils sont appris plus vite par les enfants, petits et grands, et ils sont généralement mentionnés en premier lorsque les membres de la catégorie sont énumérés [Cauzinille-Marmèche *et al.*, 1990]. Les propriétés des membres prototypiques sont "typiques", au sens où elles sont caractéristiques de la catégorie. L'existence de ces propriétés fait que les membres partagent un certain air de famille, sans que la présence de toutes les propriétés soit obligatoire chez les membres de la catégorie. La fréquence d'apparition d'une propriété est quelquefois un critère de mesure de typicalité d'une propriété[10]. La figure 1.4 montre d'ailleurs bien que certaines propriétés des oiseaux sont plus centrales ou plus fréquentes que d'autres.

La théorie du prototype a été revue et corrigée par ses fondateurs mêmes, pour essayer de rendre compte de concepts complexes comme les concepts composés, la mer bleue ou la lune jaune par exemple [Smith *et al.*, 1988]. La "déviation moderne" de la théorie ne nous concerne pas directement dans ce qui suit, aussi nous renvoyons à [Mervis and Rosch, 1981] et à [Kleiber, 1990] pour des précisions supplémentaires.

8. Dans son ouvrage *La vie des oiseaux*, Jean Dorst note que : *En dépit de variations étendues, les oiseaux présentent une remarquable unité architecturale, étant tous bâtis sur le même "modèle". Cette unité dans la diversité provient du fait qu'ils sont avant tout construits en vue du déplacement dans l'air* [Dorst, 1971].

9. Cette remarque se vérifie également au niveau de systèmes de représentation étudiés dans la suite (cf. § 2.1.4 et § 3.3.1).

10. Des critères de même type sont utilisés en classification automatique et en catégorisation conceptuelle (cf. § 4.1 et § 4.2.1).



1 : apte au vol
2 : couvert de plumes (non dégénérées)
3 : a des ailes (non atrophiées)
4 : non domestique
5 : pond des oeufs
6 : a un bec

Figure 1.4. Une autre façon d'imager *tous les oiseaux volent sauf l'autruche*, ou tous les oiseaux ne partagent pas les mêmes caractéristiques.

La catégorisation verticale

Le niveau de description des catégories qui vient d'être présenté est qualifié de *dimension horizontale*, car il traite de l'organisation interne de la catégorie. La *dimension verticale* traite de l'organisation des catégories les unes par rapport aux autres, organisation externe ou encore classification des catégories [Rosch *et al.*, 1976] : si un individu peut être décrit de multiples façons, il peut aussi être présenté de multiples façons.

L'étude de la dimension horizontale tente de justifier le choix d'une catégorie C par rapport aux catégories qui ne conviennent pas à un individu donné I. Parallèlement, l'autre question qui se pose est : pourquoi I est-il un C plutôt qu'un $Cprim$, alors que les deux catégories conviennent ? En particulier, une même entité peut être dénommée ou encore catégorisée de différentes façons : les deux phrases "l'animal dans le champ" et "la vache dans le pré" peuvent dénoter la même entité. Il ne s'agit pas de

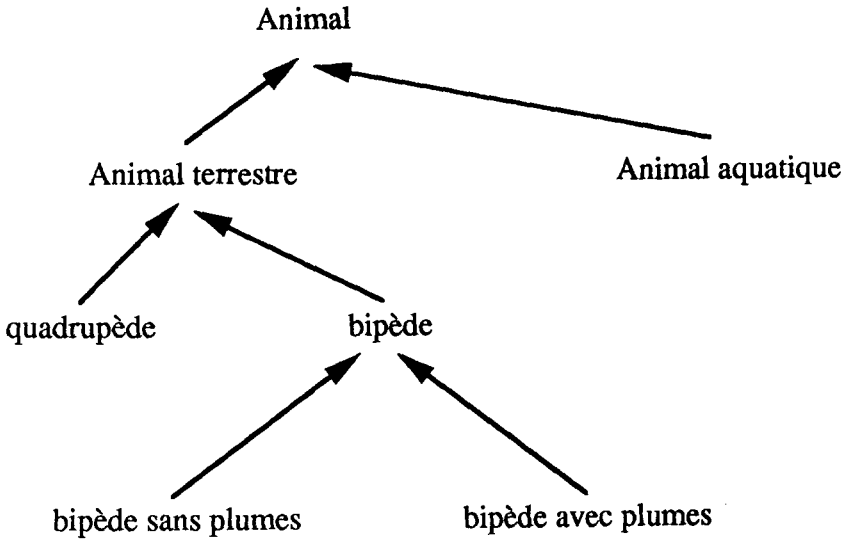


Figure 1.5. Une catégorisation verticale qui donne une vision relativement inhabituelle et pourtant très ancienne de la politique (d'après Platon).

synonymie, mais de niveau de généralité des termes qui dénotent des entités, d'où la "dimension verticale". Voici par exemple la division mise en place par Platon lorsqu'il cherche à déterminer l'objet de la politique. Le politique est un homme qui gouverne les troupeaux. Mais quels troupeaux ? Il y a des troupeaux aquatiques et des troupeaux terrestres. Nous pouvons éliminer les premiers. Mais parmi les troupeaux terrestres, il y a les troupeaux de quadrupèdes et de bipèdes. Le politique gouverne les bipèdes mais il y a des bipèdes à plumes et des bipèdes sans plume. La politique se définit donc rigoureusement comme "l'art de gouverner les bipèdes sans plumes" (cf. Fig. 1.5).

L'être humain organise naturellement les catégories sous la forme de *hiérarchies*, où des catégories plus générales dominent des catégories plus spécifiques. Dans les classifications (ou catégorisations verticales) botaniques, zoologiques ou minérales, se distinguent généralement six grands niveaux statiques :

- Le *règne* est animal, végétal, ou minéral, et il représente la division la plus large du monde naturel.
- L'*embranchement* est la grande division du règne, essentiellement pour les règnes végétal et animal. Un embranchement peut éventuellement avoir des sous-embranchements, l'embranchement des cordés a pour sous-embranchement les vertébrés par exemple.
- La *classe* regroupe des individus, êtres, objets, faits, en nombre indéterminé, et possédant un ou plusieurs caractères communs. Ainsi, la classe des oiseaux et celle des mammifères font partie du sous-embranchement des vertébrés.
- L'*ordre* est la première subdivision de la classe.

- La *famille* regroupe des genres ayant des traits généraux communs.
- Le *genre* regroupe des individus qui sont morphologiquement semblables.
- L'*espèce* regroupe des catégories d'êtres vivants, végétaux ou animaux, qui sont caractérisés par des formes bien définies et qui constituent un type héréditaire de fécondité illimitée par croisement (espèces naturelles).

L'espèce Chien domestique fait partie du genre *Canis*, famille des Canidés, ordre des carnivores, classe des mammifères, sous-embranchements des Vertébrés, embranchement des Cordés. Un des principes essentiels sur lesquels s'appuie ce type de classification est d'admettre que dans l'ascendance d'une espèce E doit nécessairement exister une catégorie F dont la scission en deux a engendré l'espèce E et une espèce sœur *Eprim*. Les extensions des espèces E et *Eprim* sont incluses dans celle de la catégorie F qui est parfaitement définie. Le même processus se réitère s'il y a lieu sur F.

Voici un texte qui montre comment s'expliquait l'ordre de la nature au siècle dernier[11], mais aussi la difficulté qu'il y a à dresser des catégorisations verticales statiques ayant rapport à des éléments naturels : *Les individus sont les supports, non seulement des caractères de l'"Espèce", mais de tous les autres. Comme représentants du "Genre", ils ont certains détails d'une structure définie et spécifique, identiques à ceux que possèdent les représentants d'autres Espèces. Comme représentants de la "Famille", ils ont une figure définie et expriment par des formes semblables à celles des représentants d'autres Genres, un modèle spécifique distinct. Comme représentants de l'"Ordre", ils se placent à un rang défini, quand on les compare avec les représentants d'autres Familles. Comme représentants de la "Classe", ils manifestent le plan de structure de leur Embranchement à l'aide de moyens spéciaux et suivant des voies spéciales. Comme représentants de l'"Embranchement", les individus sont tous organisés d'après un plan distinct qui diffère du plan des autres Embranchements.*

Les classifications statiques d'éléments naturels furent au départ très "artificielles", car sujettes au préjugé empiriste selon lequel les caractères les plus visibles, les plus manifestes, sont les plus importants. Or, une classification naturelle doit être plutôt fondée sur les caractères biologiques profonds, ce qui la rend tributaire de l'avancement des connaissances anatomiques et physiologiques. Il est maintenant communément acquis que la baleine partage "plus de" propriétés avec le chien qu'avec la truite. Le mérite des classifications naturelles modernes est de mettre en lumière les caractères biologiquement importants, en explicitant des caractères dominateurs et des caractères subordonnés. L'importance d'un caractère se mesure à son retentissement sur l'ensemble de l'organisme vivant. Les classifications biologiques actuelles sont donc plutôt fondées sur l'étude des corrélations entre caractères, sur la recherche d'un ordre défini sur les caractères des vivants qui soit naturellement fondé, plutôt que sur un ordre arbitraire empirique fondé sur l'observation de coïncidences.

Sur un plan plus strictement linguistique, une division en cinq niveaux est présentée dans [Kleiber, 1990] : dans le même ordre décroissant que ci-dessus, le règne, la forme

11. Ce texte est attribué au naturaliste Louis Agassiz dans [Cuvillier, 1954].

de vie, le genre, l'espèce et la variété. Le règne a déjà été présenté ; la forme de vie regroupe des taxons comme arbre, fleur, légume, animal ; le genre regroupe des taxons comme chêne, bouleau, ou peuplier pour la forme de vie arbre, poireau et pomme de terre pour la forme de vie légume, le genre est fréquemment employé pour dénoter dans le langage une catégorie, et il fait probablement partie des premiers taxons appris par les enfants ; l'espèce regroupe des particularités comme chêne vert ou pomme de terre à peau rouge ; la variété enfin définit le dernier niveau de classification, le chêne vert méditerranéen ou la pomme de terre Rosa y appartiennent.

D'un point de vue linguistique toujours, d'autres classifications, à trois niveaux cette fois, ont été proposées pour expliquer l'utilisation de termes ayant des niveaux de précisions différents dans des phrases exprimant les mêmes idées [Rosch *et al.*, 1976] : ainsi "un chien sur la pelouse" et "un arbre sur la pelouse", ne contiennent pas des termes de même niveau de généralité, "chien" est un genre dans le découpage linguistique, alors qu'arbre est une forme de vie. La dimension verticale à trois niveaux considère le niveau *superordonné*, qui contient animal, fruit, meuble, le niveau *de base*, qui constitue le niveau de dénotation privilégié et qui contient chien, pomme, chaise, le niveau *subordonné*, qui est le plus précis et qui contient épagneul, pomme golden et chaise longue. La distribution des entités selon ces trois niveaux est variable et dépend du type des entités, ainsi que de l'expérience du sujet qui catégorise. Le niveau de base est celui qui est utilisé pour décrire de façon courante l'environnement. Les catégories qui appartiennent à ce niveau regroupent des individus qui possèdent un nombre significatif de caractéristiques propres à la catégorie, qui ont une forme et un comportement similaires, et qui sont identifiés à partir de la forme moyenne des membres de la catégorie. Cette dernière précision montre comment la théorie du prototype intervient également dans la dimension verticale de la catégorisation.

La remémoration

La description des catégories a été traitée dans les paragraphes précédents. Dans ce qui suit, sont examinées les manières de manipuler des catégories et de retrouver la "bonne" catégorie dans une situation donnée.

Selon certaines théories de psychologie, le raisonnement humain s'appuie sur l'utilisation de modèles mentaux qui simulent, à la manière des prototypes, des situations du monde réel [Johnson-Laird, 1983]. Ces modèles sont recherchés, examinés, puis utilisés pour résoudre un problème particulier. Ainsi, pour analyser une situation nouvelle, un être humain choisit le modèle qui lui semble le plus *proche* de cette situation, et le modifie éventuellement pour le faire correspondre à la situation. Il est d'autant plus facile de comprendre une situation que cette dernière est proche d'une situation analogue déjà vécue. L'analyse d'une situation dépend donc étroitement de l'expérience personnelle, de la connaissance de base disponible [Schank, 1982]. Par exemple, un individu qui entre dans une maison prédit la présence d'une pièce délimitée par quatre murs, un plafond et un plancher, car le modèle mental de l'objet "pièce" est généralement mémorisé sous cette forme. Le concepteur de système à base de connaissances est confronté au même genre de problème : comment organiser des connaissances pour les réutiliser dans des situations similaires ?

Les premières études portant sur la réutilisation dynamique de modèles dans le cadre de la représentation des connaissances sont attribuées à M. Minsky et R.C. Schank. L'un et l'autre ont défini des formalismes généraux de représentation qui se démarquent de la logique classique, et qui essaient de rendre compte de l'aspect factuel et procédural du raisonnement de sens commun. Le premier a introduit le formalisme des *frames*[12] qui représentent des *prototypes d'objets*, comparables à des formulaires comportant des cases à remplir et des cases remplies par défaut [Minsky, 1975]. Le second s'est attaché à décrire des *prototypes de comportements*, sous la forme de *scripts*, et il s'en est servi pour l'étude du langage naturel et la compréhension d'histoires [Schank, 1982]. Le formalisme des frames et celui des scripts proposent une organisation de la mémoire telle que l'accès à des connaissances pertinentes soit le plus efficace possible. Les connaissances sont organisées comme le sont, ou sont supposées l'être, nos propres expériences : les situations génériques sont mémorisées sous la forme de modèles génériques ou prototypes ; les situations particulières sont représentées par des modèles qui dérivent des modèles génériques. Les modèles servent à décrire mais aussi à calculer, au sens où ils fournissent les éléments nécessaires à l'aboutissement d'un raisonnement.

Le raisonnement proprement dit repose sur le *principe de remémoration* : étant donné un réseau de modèles représentant un domaine d'application, il s'agit de trouver un modèle pouvant *expliquer* la situation courante. Cette recherche s'appuie sur un processus d'appariement et comprend généralement trois étapes [Kuipers, 1975] :

- Choix d'un modèle sur la base d'heuristiques ou d'informations disponibles sur la situation courante.
- Mise en correspondance du modèle avec la description de la situation courante.
- Production d'une explication découlant de l'interprétation de l'appariement ou de l'échec de l'appariement.

Le raisonnement se poursuit en choisissant un nouveau modèle pouvant décrire l'évolution de la situation. Lorsqu'aucun modèle n'est disponible pour s'apparier à la situation courante, il s'agit d'une nouvelle expérience. Il est alors nécessaire de construire un nouveau modèle mémorisant les caractéristiques de cette nouvelle expérience. Retrouver le bon modèle au bon moment dans une mémoire qui évolue constamment, permet de comprendre le domaine étudié, mais aussi de raisonner, c'est-à-dire d'établir une suite cohérente d'inférences pour résoudre un problème donné.

1.2.2 Les représentations hiérarchiques

Les frames et les scripts sont des formalismes de représentation particulièrement bien adaptés à la description de modèles mentaux. Dans la suite, nous nous intéressons essentiellement aux langages permettant d'implanter de tels formalismes, les *systèmes*

12. Terme anglo-saxon qui signifie cadre, trame ou encore schéma, que nous préférons utiliser tel quel.

à héritage [Masini *et al.*, 1989], et les *systèmes à subsomption* [Nebel, 1990]. Ces deux types de systèmes permettent de représenter des connaissances sous la forme de hiérarchies d'objets. Chaque objet dénote un concept, et il est défini comme une collection de propriétés structurelles et procédurales décrivant son état et les opérations qu'il est capable d'exécuter. Les objets sont organisés par niveaux d'abstraction et ils partagent des propriétés par l'intermédiaire des relations d'héritage ou de subsomption, qui sont des relations d'ordre partiel. Un objet *hérite* les propriétés des objets qui sont plus généraux que lui, ses ascendants, tandis que ses propriétés sont héritées par les objets qui sont plus spécifiques que lui, ses descendants. L'héritage est dit *simple* si un objet ne peut avoir qu'un seul ascendant, sinon il est dit *multiple*. La hiérarchie obtenue est aussi appelée *graphe d'héritage*. Dans le cadre des systèmes à héritage, les formalismes des classes et des frames sont habituellement employés pour représenter des hiérarchies d'objets.

Les classes et les langages de classes

Une classe décrit une famille d'objets appelés instances à l'aide d'un ensemble de variables et de méthodes qui représentent respectivement la structure et le comportement des instances. Elle peut être considérée comme un modèle ou un plan à partir duquel sont construites les instances. Toute instance est liée à la classe dont elle dérive, et elle est caractérisée par un état constitué par un ensemble de valeurs correspondant aux variables déclarées dans la classe. Appliquer une opération à une instance consiste à lui envoyer un message contenant le nom de la méthode devant être activée et des arguments.

Une classe dérive d'une ou de plusieurs classes appelées ses *superclasses*, dont elle hérite les variables et les méthodes. Aux caractéristiques de la classe "s'ajoutent" les caractéristiques des superclasses et, de façon transitive, les caractéristiques des superclasses des superclasses, et ainsi de suite. Du point de vue de l'implantation, l'héritage des variables est en général *statique* : les variables héritées sont recopiées dans la structure représentant la classe. En revanche, l'héritage des méthodes est *dynamique* : les méthodes héritées ne sont pas recopiées et la méthode à appliquer lors d'un envoi de message est recherchée dynamiquement dans le graphe d'héritage. Les *langages de classes* permettent de modéliser l'univers d'une application à l'aide d'une hiérarchie de classes. Smalltalk-80 [Goldberg and Robson, 1983], C++ [Stroustrup, 1989] et Eiffel [Meyer, 1990] en sont des représentants typiques.

Les classes et le modèle des conditions nécessaires et suffisantes présentent une certaine analogie. Une instance ne peut être créée que si sa classe existe, et elle possède obligatoirement toutes les caractéristiques de sa classe. Une classe est donc censée regrouper de manière exhaustive les conditions nécessaires et suffisantes d'appartenance d'une instance à la catégorie qu'elle représente. La définition de la classe précède la création de toute instance et chaque requête adressée à une instance nécessite une référence à la classe. Les extensions des catégories décrites par des classes organisées en hiérarchies d'héritage peuvent être assimilées à une suite d'ensembles emboîtés. La relation d'instanciation matérialise la relation d'appartenance d'un élément à un ensemble ; la relation d'héritage matérialise la relation d'inclusion qui unit une sous-

classe à sa classe. Toutefois, une classe est un moule, un générateur d'instances. Elle s'apparente plutôt à la représentation d'un type abstrait de données, et elle ne doit donc en aucun cas être assimilée à l'ensemble de ses instances.

Le modèle des conditions nécessaires et suffisantes est très rigide et ne s'avère un bon outil de catégorisation que lorsque les concepts ont une définition rigoureuse. En particulier, il ne permet pas de résoudre de manière satisfaisante les problèmes liés à la modélisation des concepts du monde réel, comme la gestion d'individus exceptionnels (ils ne font pas partie de la catégorie, mais ils n'existent pas sans elle), ou la représentation de connaissances incomplètes (il est très difficile de dresser *a priori* la liste des caractéristiques essentielles d'un concept). Les considérations sur l'adéquation du modèle des conditions nécessaires et suffisantes à la représentation des connaissances s'appliquent tout particulièrement aux langages de classes.

Les prototypes, les frames et les langages de frames

A l'opposé du modèle des conditions nécessaires et suffisantes, une catégorie d'individus peut être appréhendée à partir de certains d'entre eux, considérés comme des prototypes [Cohen and Murphy, 1984]. Un prototype représente une connaissance par défaut et possède les propriétés les plus fréquemment rencontrées chez les membres de la catégorie. La théorie du prototype permet d'appréhender plus facilement certains problèmes fondamentaux comme la construction incrémentielle de descriptions, la représentation de traits typiques et, parallèlement, celle d'individus exceptionnels, la représentation d'univers d'objets pour lesquels les connaissances sont incomplètes et/ou évolutives. Un prototype sert de référence pour reconnaître, analyser et finalement catégoriser des objets. Les propriétés exceptionnelles d'un individu ne peuvent perturber la structure de la catégorie, puisque tous les membres ne possèdent pas tous exactement les mêmes caractéristiques. Le raisonnement est alors effectué à l'aide de ces prototypes, même si, quelquefois, il s'avère être une simplification abusive de la réalité [Kayser, 1985].

Il existe plusieurs familles de langages qui permettent d'appréhender un univers d'objets en s'appuyant sur la théorie du prototype. Les plus communes sont les langages de frames [Masini *et al.*, 1989] et les langages d'acteurs [Agha, 1986]. Le langage SELF [Ungar and Smith, 1991] fait figure d'original, puisqu'il est fondé sur la théorie des prototypes, sans pouvoir être qualifié de langage de frames ou de langage d'acteurs. Dans l'immédiat, seuls les langages de frames nous intéressent.

Un frame est une structure composé d'attributs décrivant un concept et ses différentes propriétés. Un attribut est à son tour décrit par des facettes qui spécifient la nature de l'attribut et le comportement qui lui est associé : les facettes déclaratives définissent le type, la valeur et la valeur par défaut d'un attribut, tandis que les facettes procédurales introduisent des procédures appelées réflexes, qui peuvent être vues comme la représentation de procédures de raisonnement locales. Comme les classes, les frames sont organisés par niveaux de généralité, en une hiérarchie d'héritage. Un frame est une spécialisation d'un ou de plusieurs superframes, et il est créé par *copie différentielle* : il ne possède en propre que les propriétés, données et procédures, qui

diffèrent de ses superframes, les autres propriétés étant partagées avec les superframes. L'héritage de toute propriété, attribut ou réflexe, est dynamique. De plus, tout frame est potentiellement un prototype et peut donc être spécialisé à son tour. Les représentants d'un frame, équivalents des instances d'une classe, sont également créés par copie différentielle, ce qui n'est pas le cas pour les instances d'une classe.

Dans un langage de classes, la programmation se fait par envoi de messages. Dans un langage de frames, la *programmation dirigée par les accès* est le mode de programmation courant. Elle consiste à accéder en lecture ou écriture aux attributs des objets, ce qui entraîne le déclenchement de réflexes. Une requête adressée à un individu se traduit d'abord par une référence à la description de l'individu, suivie éventuellement d'une référence au prototype [Lieberman, 1986]. Ainsi, ne pas connaître exactement les valeurs de toutes ses propriétés n'empêche pas la définition d'un individu qui peut être complétée ultérieurement. Parallèlement, toute modification dans un frame est implicitement répercutée à tous les objets qui dérivent du frame, ces derniers ne possédant que virtuellement les propriétés qu'ils partagent avec lui. La prise en compte de telles modifications pour mettre à jour les instances d'une classe nécessite des traitements auxiliaires. Par ailleurs, le raisonnement par classification, qui n'est pas utilisé dans le modèle des classes, est une des principales méthodes d'inférence dans le modèle des frames.

Les représentations à objets

Les langages de classes et les langages de frames ont de nombreux points communs mais ils ne sont pas destinés aux mêmes usages. Les premiers sont des langages de programmation alors que les seconds sont des langages de représentation. La représentation de catégories avec des classes relève du modèle des conditions nécessaires et suffisantes, et, de ce fait, se révèle moins pratique pour prendre en compte les caractéristiques des connaissances du monde réel. Les frames, quant à eux, permettent d'appréhender un univers d'une façon qui est plus proche de celle qui est employée par l'être humain. La description d'un univers s'affine au fur et à mesure qu'augmente l'expérience que l'on a de cet univers. Par exemple, pour un enfant, la famille des éléphants se résume dans un premier temps au seul sujet qu'il a rencontré et qui devient le prototype de la famille. Plus l'enfant grandit, plus la famille des éléphants se peuple des sujets que l'enfant rencontre ultérieurement. Chaque nouveau sujet complète la catégorie des éléphants et précise la description du prototype. Ce n'est qu'après observation d'une certaine quantité de sujets que naît une abstraction "complète" du concept éléphant, qui pourrait alors être éventuellement décrit par un ensemble de conditions nécessaires et suffisantes. Ainsi, les frames sont souvent préférés en représentation des connaissances. Toutefois, une part importante des connaissances relatives à un domaine se présente sous forme de règles de diagnostic, de règles de manipulation ou règles de décision, qu'il est facile de représenter à l'aide d'un formalisme comme celui des règles de production par exemple. Plutôt que de se cantonner à un seul formalisme de représentation, les langages à objets dédiés à la représentation des connaissances ont évolué vers une famille de langages dits *hybrides*, qui intègrent généralement les trois styles de représentation, classes, frames et règles [Masini *et al.*, 1989].

En première approximation, nous définissons une *représentation à objets* comme un environnement de programmation où :

- la modélisation des connaissances est hiérarchique : les concepts du domaine étudié sont décrits par des catégories organisées en une hiérarchie, selon leur niveau d'abstraction ;
- un langage de représentation permet de traduire la modélisation des connaissances sous la forme d'une hiérarchie d'héritage où les unités de connaissance sont des objets munis de propriétés ;
- des outils spécialisés gèrent les objets, l'accès à leurs propriétés, ainsi que les opérations de filtrage et de classification.

Une représentation à objets peut avoir pour support un langage de classes, un langage de frames, ou encore un langage hybride. Dans la suite, la troisième possibilité est supposée être la règle par défaut : les objets sont des frames munis de méthodes, à l'image des objets de YAFOOL [Ducournau, 1989b].

Les systèmes à subsomption

Un certain courant de recherche sur la représentation des connaissances s'est créé à partir des idées qui sont à la base de KL-ONE [Brachman and Schmolze, 1985]. Il a donné naissance à une famille de systèmes de représentation appelés dans la suite les *systèmes à subsomption*. BACK [Nebel, 1990], CLASSIC [Brachman *et al.*, 1991] et LOOM [MacGregor, 1988], en sont des représentants typiques. Les systèmes à subsomption reposent sur les principes suivants [MacGregor, 1991] :

- Les entités d'un domaine d'application sont représentées par des objets généralement appelés concepts. L'organisation des concepts s'appuie sur la relation de subsomption : un concept C subsume un concept D si tout individu décrit par D est nécessairement décrit par C. Les concepts sont partiellement ordonnés en une hiérarchie semblable à une hiérarchie d'héritage.
- Un *classificur* gère la hiérarchie des concepts et trouve la place des nouveaux concepts. L'insertion des nouveaux concepts dans la hiérarchie peut provoquer un ensemble d'inférences.
- Il existe un niveau de langage pour décrire les concepts ($TBox$) et un niveau de langage pour déclarer des faits dans lesquels interviennent ces concepts ($ABox$).

Un concept se présente comme une conjonction de rôles (ou attributs) qui expriment les relations que le concept entretient avec les autres concepts. Chaque rôle peut se voir associer des restrictions à l'instar des facettes associées à un attribut dans un langage de frames. Les concepts se divisent en concepts primitifs et définis. Les concepts primitifs dénotent des catégories naturelles comme les personnes, les animaux, les plantes, etc.. Leur description est incomplètement spécifiée et ne contient que des

conditions nécessaires : il n'est pas possible d'affirmer qu'un individu est un représentant d'un concept primitif en examinant ses seuls rôles. Les concepts primitifs servent à construire les concepts définis dont la description est complètement spécifiée ; les rôles expriment cette fois des conditions nécessaires et suffisantes. L'ensemble des concepts est organisé en une hiérarchie où les concepts généraux partagent leurs caractéristiques avec les concepts spécifiques qu'ils subsument. La gestion de la hiérarchie des concepts est entièrement prise en charge par le classifieur. Pour augmenter l'efficacité du classifieur, il est envisagé d'utiliser une mémoire de faits qui contient des connaissances par défaut qui ne sont quelquefois qu'une approximation de la situation en cours d'étude mais qui ont l'avantage d'être directement disponibles, sans calculs [Levesque, 1986b] [Etherington *et al.*, 1989].

La recherche sur les systèmes à subsomption a actuellement un rôle prépondérant dans les études sur la représentation des connaissances. Une partie des techniques de classification et de description des concepts qui ont été développées pour ces systèmes ont été reprises dans le cadre des représentations à objets. L'existence d'une composante de représentation procédurale sous forme de réflexes et de méthodes dans les représentations à objets distinguent ces dernières des systèmes à subsomption. Les systèmes à subsomption apparaissent comme un formalisme de représentation à base d'objets, dont le modèle est le calcul des prédicats du premier ordre, l'accent étant mis sur la sémantique de la représentation. Les représentations à objets ont comme modèle les langages de frames, et l'accent est mis sur la représentation des actions et des comportements.

1.2.3 Le raisonnement par classification dans une représentation hiérarchique

Comme toute base de connaissances, une représentation à objets ne devient véritablement opérationnelle que si une part importante des connaissances du domaine d'application est représentée. Il est alors nécessaire de disposer d'outils efficaces et fiables pour acquérir de nouvelles informations, rechercher des informations, les mettre à jour et maintenir la cohérence de la hiérarchie. Même un utilisateur familier du système peut introduire des incohérences, des erreurs et des redondances lorsqu'il opère une modification dans la hiérarchie. Le raisonnement par classification permet de résoudre une partie de ces problèmes, en particulier, la recherche d'informations et l'acquisition de nouvelles connaissances sur la base des informations disponibles [MacGregor and Burstein, 1991]. Il est naturellement associé aux représentations à objets, puisque sa tâche consiste à gérer une hiérarchie d'objets ordonnés par une relation d'ordre partiel.

Le raisonnement par classification cherche à mettre en évidence les dépendances qui existent entre des objets, pour un ordre partiel noté $\preceq$ dans la suite. Ces dépendances se traduisent généralement par un partage de propriétés. Ainsi, un sous-ensemble $[P_1 \ \ldots \ P_K]$ de propriétés associées à un objet X est *compatible* avec un sous-ensemble $[P_{\sigma(1)} \ \ldots \ P_{\sigma(K)}]$ de propriétés associées à un objet O, pour l'ordre partiel $\preceq$, si :

- σ est une permutation sur l'ensemble des indices [1 ... K] : pour chaque propriété P_i de X, il existe une et une seule propriété $P_{\sigma(i)}$ de O qui s'identifie à P_i, pour i variant de 1 à K.
- Si P_i et $P_{\sigma(i)}$ s'identifient, alors les contraintes[13] associées à P_i dans X sont plus spécifiques que les contraintes imposées sur $P_{\sigma(i)}$ dans O, pour l'ordre partiel $\preceq$ et pour i variant de 1 à K.

Ceci étant, un objet X et un objet O peuvent alors se trouver dans l'une des trois situations suivantes :

- $X \preceq O$: X est inférieur à O, ou encore plus spécifique que O pour l'ordre partiel $\preceq$; parallèlement, O est supérieur à X ou encore plus général que X pour l'ordre partiel $\preceq$; un sous-ensemble des propriétés détenues par X est compatible avec l'ensemble des propriétés de O.
- $O \preceq X$: les rôles de X et O sont inversés dans l'inéquation précédente.
- X et O ne sont pas comparables pour l'ordre partiel $\preceq$. Ce phénomène se produit essentiellement dans les trois cas suivants :
  - Les objets X et O n'ont aucune propriété commune.
  - La compatibilité des contraintes n'est pas vérifiée : les contraintes associées aux propriétés de X ne sont pas toutes inférieures pour l'ordre partiel $\preceq$ aux contraintes imposées sur les propriétés correspondantes dans O par exemple.
  - Les objets X et O possèdent chacun des propriétés qui ne sont pas détenues par l'autre, qui ne sont donc pas comparables pour l'ordre partiel $\preceq$.

La classification d'un objet X divise l'ensemble des objets de la hiérarchie en trois familles, celle des objets avec lesquels X est comparable, celle des objets avec lesquels X est incomparable, et celle des objets pour lesquels la comparaison n'est pas décidable sans informations supplémentaires. Le raisonnement par classification s'appuie sur l'*hypothèse du monde ouvert* : O et X ne sont déclarés incomparables que si une incompatibilité a été effectivement détectée, mais pas s'il manque des informations pour décider de la compatibilité ou de l'incompatibilité.

La mise en œuvre du raisonnement par classification s'apparente à un cycle qui dérive des trois étapes caractérisant le principe de remémoration [Kaczmarek *et al.*, 1986] :

- Création d'un nouvel objet X en s'appuyant sur un résultat produit par le système ou sur des informations externes.
- Mise en place de l'objet X dans la hiérarchie, en accord avec l'ordre partiel $\preceq$ existant : X a pour ascendants directs les objets les plus spécifiques parmi ceux qui sont plus généraux que X ; X a pour descendants immédiats les objets les plus généraux parmi ceux qui sont plus spécifiques que X.

13. Les "contraintes" portent généralement sur le type et les valeurs des propriétés.

- Propagation des modifications : la mise en place de X provoque des modifications dans la hiérarchie, ce qui déclenche en cascade de nouvelles inférences. A la suite de ces inférences, de nouveaux objets sont généralement produits, ce qui ramène le cycle à sa première étape. Le cycle se poursuit tant que le but recherché n'est pas atteint ou que des nouvelles informations sont disponibles.

Ce cycle peut aussi être utilisé pour vérifier la cohérence de la hiérarchie en contrôlant que chaque objet est correctement placé. Dans ce cas, le raisonnement par classification est appliqué à un objet déjà présent afin de *réajuster* éventuellement sa position dans la hiérarchie. Le maintien de la cohérence de la hiérarchie peut être assuré en continu de cette façon, si les objets interdépendants sont prévenus des changements qui surviennent dans les objets dont ils dépendent.

Dans le cycle qui vient d'être présenté, il n'est fait aucune hypothèse sur la sémantique de la relation d'ordre partiel $\preceq$ qui organise les objets en hiérarchie. Dans les chapitres qui suivent, sont donnés des exemples de raisonnement par classification où la relation d'ordre partiel employée est la relation d'héritage (cf. § 3.3.2 et § 3.3.3), la relation de composition (cf. § 3.4.1 et § 5.4), et enfin la relation de subsomption (cf. § 3.5).

1.2.4 La classification heuristique et la résolution de problèmes

Ce paragraphe présente une variation sur la forme du raisonnement par classification, tirée des travaux de W.J. Clancey [Clancey, 1985]. Ce dernier a analysé les caractéristiques et les capacités de démonstration d'un certain nombre de systèmes de résolution de problèmes destinés à des tâches diverses comme le diagnostic, la planification, etc., indépendamment des formalismes de représentation utilisés. La stratégie communément employée par ces systèmes repose sur un raisonnement qualifié de *classification heuristique*. La recherche d'une solution consiste à établir une association heuristique entre l'objet du problème et une unité de connaissance, présente dans la base de connaissances du système. Ces unités sont considérées comme des solutions préétablies (de problèmes standard) et elles sont partiellement ordonnées. Que ces unités soient assimilées à des solutions ne signifie pas que la base contient toutes les solutions aux divers problèmes qui pourraient être posés au système, mais simplement que la démarche de résolution consiste à apparier l'objet du problème à une unité connue, plutôt que de construire une solution de toutes pièces.

Le processus de classification heuristique se compose de trois étapes principales (cf. Fig. 1.6) :

- L'*abstraction de données* consiste à transformer les données décrivant le problème en éléments de solution. Cette étape se révèle inutile lors de la résolution de problèmes simples car les données s'apparient directement avec les caractéristiques de certaines solutions. Dans les problèmes plus complexes, les éléments de solutions doivent être inférés à partir des données par déclenchement de règles,

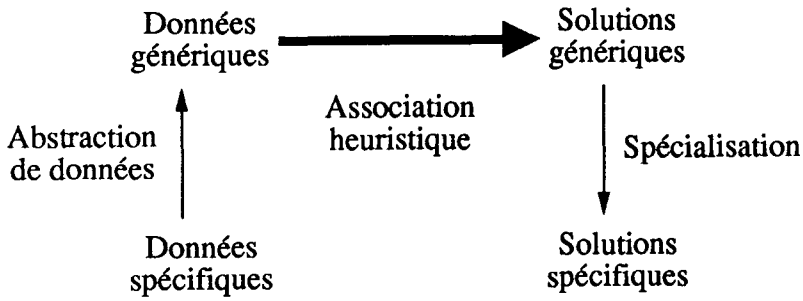


Figure 1.6. Les trois étapes principales de la classification heuristique.

par exemple. Ainsi, lors de l'élaboration d'un diagnostic médical, une donnée numérique résultant d'une analyse est interprétée de façon qualitative pour donner une appréciation du fonctionnement d'un organe.

- L'*appariement heuristique* associe, sur la base d'heuristiques, les données résultant de la première étape à une hypothèse générique, décrite par une unité de la base : les informations qualitatives obtenues lors de l'étape d'abstraction de données sont mises en relation avec les caractéristiques d'une maladie. Une telle association relève de l'expertise et repose sur des heuristiques.

- Lors de la *spécialisation de l'hypothèse retenue*, les données relatives au problème sont utilisées pour affiner l'hypothèse : certains résultats d'analyse permettent de cerner plus précisément, à l'intérieur de la classe de maladies trouvée à l'étape précédente, l'affection particulière du patient.

Ces trois étapes peuvent être considérées suivant des ordres différents. La recherche d'une solution est *dirigée par les données* lorsque les solutions potentielles sont exhibées sur la base des données disponibles. Elle est *dirigée par les solutions* lorsque le système tente de confirmer une hypothèse, comme dans la recherche d'une explication ou de la cause d'un événement.

La stratégie d'un système de résolution de problèmes relève de la classification heuristique lorsqu'elle consiste à mettre en relation des données relatives à un problème et des solutions préétablies dans une base de connaissances. Cette stratégie peut aussi être vue comme une composition de tâches génériques [Bylander and Mittal, 1986] [Chandrasekaran and Goel, 1988] : l'abstraction de données correspond à une recherche d'informations, l'appariement heuristique à une suggestion d'hypothèses devant être confirmées ou rejetées. Cette dernière tâche s'apparente au parcours d'un arbre de décision et peut être effectuée par un *filtrage structuré* [Bylander *et al.*, 1989]. Les informations positives recueillies au cours de l'abstraction de données sont regroupées en conjonctions de descripteurs qui correspondent à des prémisses de règles simples modélisant le domaine considéré. Les règles sélectionnées sont déclenchées dans un certain ordre, et elles produisent de nouvelles conjonctions de descripteurs, ce qui

entraîne de nouveaux déclenchements de règles, et ainsi de suite, jusqu'à ce qu'une décision soit obtenue.

La classification heuristique est une stratégie bien adaptée à la résolution de problèmes simples et bien formalisés, de diagnostic ou de planification. En revanche, elle ne permet pas de découvrir la solution d'un problème nécessitant une étape de construction de la solution comme dans les problèmes de catégorisation. Notons que le mécanisme de classification employé dans le langage OBJLOG relève de la classification heuristique car il s'appuie sur un ensemble de règles qui décrivent les modalités de classification d'un objet dans le graphe d'héritage [Dugerdil, 1988].

1.3 Futurs

L'approche déclarative en représentation des connaissances préconise l'emploi d'un programme central et général, qui essaie de résoudre un problème en utilisant une base de connaissances séparée [Laurière, 1982]. Toute l'expertise nécessaire à la résolution du problème est rassemblée en un même lieu, à savoir la base de connaissances. Cette vision de la représentation des connaissances et de la résolution de problèmes est souvent contestée, en particulier par les tenants de l'intelligence artificielle distribuée [Ferber and Briot, 1988] [Ferber and Ghallab, 1988] [Bond and Gasser, 1988], qui proposent de faire coopérer des spécialistes détenant chacun une expertise restreinte, mais très spécifique. D'une façon générale, les classes et les frames sont des entités autonomes pouvant représenter des processus concurrents et être adaptés à ce type de programmation répartie. Toutefois, le modèle objet le mieux adapté à l'intelligence artificielle distribuée est celui des acteurs, modèle inventé par Carl Hewitt [Hewitt *et al.*, 1973] [Hewitt, 1977], et amélioré par G. Agha [Agha, 1986].

Un acteur peut être vu comme un expert qui vit en société et qui communique avec d'autres experts, ses semblables, pour résoudre des problèmes. Chaque expert peut à son tour se décomposer en une société d'experts plus élémentaires. Ce modèle de représentation des connaissances est décentralisé et distribué (ou réparti). Les connaissances, les modalités de raisonnement et le comportement des individus, sont diffusés parmi les acteurs, qui effectuent des tâches en parallèle, de manière indépendante. La seule faculté que partagent tous les acteurs est la possibilité de communiquer les uns avec les autres. A la façon d'une classe ou d'un frame, un acteur regroupe au sein d'une même entité un ensemble limité de connaissances et le moyen de les exploiter. Les connaissances associées à l'acteur se composent de données locales, appelées aussi *accointances*, qui sont les autres acteurs que l'acteur connaît directement. Ces données présentent une certaine analogie avec les variables d'instance d'une classe. Le comportement fait référence aux actions que l'acteur entreprend au cours de son existence, en fonction des événements qui surviennent. Le comportement est défini par un script qui filtre les messages reçus par l'acteur et active la partie appropriée du comportement, lorsque le message est compris. Le script est comparable à un ensemble de méthodes qui modélisent les actions que l'acteur est capable d'accomplir. Un acteur est censé être une entité autonome, qui détient toute la connaissance requise pour effectuer des tâches spécialisées. De ce point de vue, il peut être assimilé

à une *machine* dont le fonctionnement est régi par son script. Cette machine peut éventuellement posséder plusieurs processeurs lui permettant d'exécuter des tâches en parallèle. L'interface d'un acteur définit le contrat que l'acteur a passé avec le monde extérieur. En particulier, l'acteur est seul responsable de la manière dont il remplit son contrat. A l'origine, la notion d'héritage n'existe pas dans le modèle acteur. Toutefois, un acteur ne travaille jamais seul ; il fait partie d'une société et il communique avec d'autres acteurs pour résoudre un problème donné.

Les futurs systèmes à bases de connaissances seront des systèmes répartis, fortement évolutifs, dans lesquels il sera facile d'incorporer de nouvelles entités, la prise de décision y sera décentralisée afin d'éviter les goulots d'étranglement. Ces systèmes seront fondés sur l'hypothèse du monde ouvert, car il n'est pas raisonnable de penser représenter à l'avance toutes les connaissances caractérisant un domaine particulier. La résolution d'un problème, l'accomplissement d'une tâche en général seront des phénomènes coopératifs, s'appuyant sur des négociations entre acteurs. Puisque chaque acteur n'a qu'une connaissance partielle du système dans son entier, il ne peut avoir qu'une influence limitée et il doit négocier avec les autres acteurs pour que le système évolue convenablement. En ce sens, les systèmes ouverts sont des précurseurs des futurs systèmes à bases de connaissances [Hewitt and Jong, 1982] [Hewitt, 1985]. Le parallélisme y est intrinsèque, la connaissance est distribuée parmi les acteurs, les communications sont asynchrones, chaque acteur ne détient que des informations locales, et son comportement est indépendant de celui des autres acteurs. En revanche, le comportement global du système reste encore difficile à appréhender.

2
Les représentations à objets

Dans ce chapitre, nous nous proposons d'examiner les principales caractéristiques des langages de programmation qui sont relativement bien adaptés à la représentation des hiérarchies d'objets introduites dans le chapitre précédent. En particulier, les langages de frames permettent de construire et de gérer des représentations à objets où les connaissances relatives à un domaine sont décrites par une hiérarchie de frames [Masini *et al.*, 1989]. Les principales opérations effectuées sur la hiérarchie sont, comme dans une base de données, la mise en place ou la suppression d'un frame, l'accès en lecture ou en écriture aux caractéristiques d'un frame et la recherche d'informations sous certaines contraintes [Hull and King, 1987]. Dans la suite, nous présentons d'abord le noyau minimal d'un langage de frames, puis les extensions qui peuvent y être apportées. Le vocabulaire utilisé est principalement emprunté au langage de frames formel présenté dans [Masini *et al.*, 1989], à Shirka [Rechenmann, 1988] et enfin à YAFOOL [Ducournau, 1989b].

2.1 Un modèle de langage de frames

2.1.1 Anatomie d'un frame

Un frame est une structure à trois niveaux, *frame-attribut-facette*, utilisée pour représenter un concept et ses différentes propriétés. Un attribut, appelé aussi propriété, est décrit par des facettes qui spécifient la nature de l'attribut et le comportement qui lui est associé. Les facettes *déclaratives* décrivent la valeur de l'attribut tandis que les facettes *procédurales* introduisent des réflexes qui se déclenchent lors des accès à l'attribut. Les réflexes se divisent en réflexes en lecture et en écriture ; les seconds se subdivisent à leur tour en réflexes *a priori* ou *a posteriori*, selon qu'ils sont activés avant ou après la modification de la valeur de l'attribut auquel ils sont associés. Toute action sur la valeur d'un attribut résulte d'un accès à cet attribut. Ce style de programmation est appelé *programmation dirigée par les accès* [Stefik *et al.*, 1986].

Les facettes de typage et de restriction de type

Les facettes déclaratives se divisent en facettes de typage, de restriction de type, de cardinalité et de valeur. Les facettes de typage `$un`[1] et `$liste-de` définissent le type d'un attribut, respectivement monovalué et multivalué[2]. Une restriction de type est indiquée par la facette `$intervalle` pour un type numérique et `$domaine` pour un type donné en extension. Les langages de frames sont généralement à typage dynamique : même si le type d'un attribut est déclaré statiquement, la validité de la valeur d'un attribut n'est vérifiée qu'à l'exécution. Les facettes de cardinalité `$min` et `$max` précisent respectivement le nombre minimal et maximal de valeurs que peut prendre un attribut.

La figure 2.1 montre la représentation des frames `Point` et `Fenetre`, définis grâce à la primitive `defmodele` comme des spécialisations du frame le plus général `Objet-ideal`. Un point est composé d'une abscisse et d'une ordonnée, l'abscisse pouvant prendre des valeurs entières entre 0 et 1000, l'ordonnée des valeurs entières entre 0 et 800. Une fenêtre située sur un écran est définie par une origine, qui est le point situé en haut et à gauche, et par un coin, qui est le point situé en bas et à droite. La donnée de l'origine est obligatoire, comme l'indiquent les facettes de cardinalité alors que celle du coin ne l'est pas.

Les facettes de valeurs

Les facettes `$valeur` et `$defaut` donnent respectivement la valeur courante, appelée *valeur fixe*, et la valeur par défaut de l'attribut. Par opposition à une valeur fixe, une valeur par défaut est "faiblement" couplée avec un attribut : elle est héritée par les spécialisations du frame et n'est prise en compte qu'en l'absence de toute autre information. Ainsi, pour créer une fenêtre particulière, seule son origine doit être connue et donnée explicitement par l'utilisateur. Le coin d'une fenêtre est par défaut le point `Bas-droit`, qui est supposé avoir pour coordonnées (1000 800), mais l'utilisateur a la possibilité de choisir n'importe quel autre point qui ne soit pas inférieur, pour la relation d'ordre définie sur les points, à l'origine de la fenêtre (l'origine d'une fenêtre est supposée être le point supérieur gauche, le coin le point inférieur droit).

Les relations

Une *relation* permet de créer une dépendance entre deux frames. La relation et sa relation inverse sont définies grâce à deux liens orientés, réciproques l'un de l'autre, chaque lien étant représenté par un attribut muni d'une facette de typage et d'une facette `$lien-inverse`. Le frame "de départ", auquel appartient l'attribut définissant la relation, s'appelle le *domaine* de la relation, tandis que le frame "d'arrivée", dans

1. Par convention, le nom de chaque facette commence par le caractère `$`.
2. La facette `$liste-de` n'admet généralement comme argument qu'un type unique, tous les objets de la liste sont donc censés avoir le même type.

```
(defmodele Point
  (sorte-de
    ($valeur Objet-ideal))
  (abscisse
    ($un entier)
    ($intervalle [0 1000]))
  (ordonnee
    ($un entier)
    ($intervalle [0 800]))
  (region
    ($liste-de Fenetre)
    ($lien-inverse origine)))

(defmodele Fenetre
  (sorte-de
    ($valeur Objet-ideal))
  (origine
    ($un Point)
    ($min 1)
    ($max 1)
    ($lien-inverse region)
    ($si-ajout dessiner-cadre)
    ($si-enleve effacer-cadre))
  (coin
    ($un Point)
    ($defaut Bas-droit)
    ($si-possible inferieur-a-origine)
    ($si-enleve effacer-cadre))
  (hauteur
    ($un entier)
    ($si-possible nil)
    ($si-besoin +hauteur))
  (largeur
    ($un entier)
    ($si-possible nil)
    ($si-besoin +largeur))
  (etiquette
    ($un chaine)
    ($si-besoin demander)
    ($si-ajout afficher-etiquette)))
```

Figure 2.1. Les frames représentant les modèles `Point` et `Fenetre`.

lequel la relation prend ses valeurs, s'appelle *co-domaine* de la relation[3]. Deux attributs définissant une relation sont symétriques : chaque fois que la valeur d'un des attributs est modifiée, celle de l'attribut avec lequel il est en relation l'est automatiquement [Napoli and Masini, 1989]. Par exemple, lorsqu'une fenêtre `F1` est créée, la donnée de l'origine de `F1`, notée `O-F1`, provoque la mise en place de la valeur `F1` dans l'attribut `region` associé au point `O-F1`. Réciproquement, donner la valeur `F1` à l'attribut `region` du point noté `P1` provoque la mise en place de la valeur `P1` dans l'attribut `origine` associé à l'objet décrivant `F1`, et par conséquent la création de la fenêtre.

Dans certains cas, il peut être utile de déclarer que deux attributs ont la même valeur, pour exprimer par exemple qu'un frère et une sœur ont les mêmes parents, qu'une épouse a les mêmes enfants que son mari. Ce type de connaissance est représenté comme un cas particulier de relation, en remplaçant la facette `$lien-inverse` par la facette `$meme-que`. Cette dernière facette est suivie d'une valeur ou bien d'une liste de valeurs et joue un rôle similaire à la facette `$lien-inverse`, en contrôlant que la valeur des deux attributs en relation évolue de la même façon. Une telle facette existe en FRL, où elle est notée `@` [Roberts and Goldstein, 1977], et dans certains systèmes à subsomption sous la forme d'un opérateur de restriction sur la valeur des rôles (cf. § 3.5). La valeur d'un attribut pointée par une facette `$meme-que` peut elle-même pointer vers un autre attribut et ainsi de suite. Cette façon de faire permet donc de définir un frame relativement à d'autres frames, augmentant ainsi les possibilités de partage d'informations et facilitant les mises à jour.

Un réflexe en lecture

Le réflexe si-besoin est le réflexe en lecture standard. Il est introduit par la facette `$si-besoin`[4] et sert à calculer la valeur de l'attribut auquel il est associé, lorsqu'il existe une méthode de calcul pour le faire. Il est déclenché à la lecture de la valeur de l'attribut, lorsqu'aucune valeur n'est présente. Par exemple, les valeurs des attributs `hauteur` et `largeur` sont calculées grâce aux réflexes supposés définis par ailleurs `+hauteur` et `+largeur`, uniquement par nécessité.

Les réflexes a priori et a posteriori en écriture

Le réflexe *a priori* standard est introduit par la facette `$si-possible`. Il sert à vérifier la validité d'une valeur à mettre en place. Il est est généralement à valeur booléenne et interdit toute écriture s'il retourne faux. Un tel réflexe est surtout utilisé pour implanter des contraintes qui ne peuvent pas être exprimées simplement à l'aide d'une facette de typage car elles nécessitent un calcul ou bien mettent en jeu les valeurs de plusieurs attributs[5]. Le réflexe si-possible associé à l'attribut `coin` vérifie

3. En anglais, les termes *domain* et *range* sont généralement utilisés pour dénoter respectivement domaine et co-domaine.

4. Par convention, les réflexes portent le même nom que les facettes qui les déclarent.

5. Un réflexe si-possible peut éventuellement être utilisé pour convertir une valeur ou vérifier le type d'objets faisant partie de listes d'objets hétérogènes.

que le point donné en valeur a une abscisse et une ordonnée qui sont supérieures, pour la relation d'ordre définie sur les points, à celles du point donné comme origine.

Les facettes `$si-ajout` et `$si-enleve` introduisent les deux réflexes *a posteriori* standard. Un réflexe si-ajout est activé après une première affectation ou après une modification de valeur, un réflexe si-enlève l'est en cas de suppression de valeur. Par exemple, une fois que l'origine d'une fenêtre particulière est connue, le réflexe `dessiner-cadre` est activé pour dessiner le cadre de la fenêtre. Si cette origine est supprimée, le réflexe `effacer-cadre` est activé pour effacer cette fenêtre. De tels réflexes sont généralement utilisés pour propager des valeurs en cas de modification de l'environnement. Ils peuvent également servir à tracer l'historique de la valeur d'un attribut en mémorisant dans une facette ou une variable spéciale toute modification de valeur.

2.1.2 Spécialisation et partage de propriétés

L'ensemble des frames est organisé en une *hiérarchie* ou *graphe d'héritage*, dans lequel un frame est une spécialisation d'un ou de plusieurs superframes, dont il hérite les couples attribut-facette. L'héritage est dit multiple si un frame peut avoir plusieurs superframes, et c'est généralement le cas, sinon il est dit simple. Un frame hérite les caractéristiques de ses ancêtres selon un ordre donné par la *liste de priorité* du frame. En héritage simple, la liste de priorité n'est autre que la fermeture transitive de la relation `sorte-de`. En héritage multiple, cette fermeture transitive n'est plus un ordre total et il faut avoir recours à des stratégies de linéarisation qui s'appuient sur des critères comme l'ordre et la multiplicité de la relation d'héritage, ainsi que sur la modularité du graphe d'héritage [Ducournau and Habib, 1989].

L'héritage des couples attribut-facette est dynamique : les couples hérités ne sont pas recopiés dans le frame mais sont consultés dans les superframes, lorsque c'est nécessaire. L'héritage dynamique favorise la représentation de connaissances incomplètes et celle d'objets dont le comportement et la structure évoluent au cours du temps. Toute modification de couple attribut-facette dans l'un des superframes se répercute automatiquement aux spécialisations qui héritent le couple. Lorsque le partage de propriétés est statique, comme dans certains langages de classes, il est nécessaire de prévoir la mise à jour des descendants des objets modifiés.

Enrichissement, masquage et sémantique des facettes déclaratives

Un frame est spécialisé par *enrichissement* lorsqu'il est doté de nouveaux couples attribut-facette. Si l'attribut d'un nouveau couple est déjà défini dans un des superframes, la nouvelle facette complète la liste des facettes existantes. Par exemple, le point `Origine-horloge` de coordonnées (900 0) est la valeur par défaut donnée à l'attribut `origine` du frame `Horloge`, sous-frame de `Fenetre` (Fig. 2.2). De même, la valeur de l'attribut `etiquette` est fixée dans le frame `Horloge` alors qu'elle ne l'est pas dans le frame `Fenetre`.

Un frame est spécialisé par *substitution* lorsque certaines facettes des couples

```
(defmodele Horloge
  (sorte-de
    ($valeur Fenetre))
  (origine
    ($defaut Origine-horloge)
    ($si-ajout afficher-heure)
    ($si-enleve effacer-heure))
  (coin
    ($defaut Coin-horloge)
    ($si-enleve effacer-heure))
  (etiquette
    ($valeur "*Horloge*")))
```

Figure 2.2. Le frame `Horloge` est un sous-frame de `Fenetre`. Par défaut, une horloge s'affiche dans le coin supérieur droit de l'écran.

attribut-facette hérités sont *masquées*, la valeur de la facette du couple héritée est redéfinie. Ainsi, le coin par défaut `Bas-droit` de coordonnées (1000 800), déclaré par la facette `$defaut` associée à l'attribut `coin` de `Fenetre` est masquée dans `Horloge` par le point `Coin-horloge`, de coordonnées (1000 100).

Le masquage ne porte que sur les facettes déclaratives et sur la facette `$si-besoin`. Voici les règles élémentaires que nous proposons et qui sont étendues au paragraphe 3.3.3 :

- Facettes `$un` et `$liste-de` : un type simple comme entier, chaîne, booléen, etc., ne peut pas être spécialisé ; un co-domaine défini par un objet O du graphe peut être spécialisé par un des descendants de O.
- Facettes `$domaine` et `$intervalle` : les domaines ou intervalles de valeurs sont emboîtés, les domaines et intervalles spécialisant étant inclus dans les domaines et intervalles spécialisés.
- Facettes `$min` et `$max` : comme les intervalles de valeurs, les intervalles de cardinalités sont emboîtés.
- Facette `$defaut` : une valeur par défaut peut être masquée à n'importe quel niveau de la hiérarchie d'héritage. Il est implicite que les contraintes portant sur le type et les restrictions de type sont respectées par une valeur par défaut comme par une valeur fixe.
- Facette `$valeur` : une valeur fixe élémentaire (`$un`) ne peut pas être masquée ; une valeur fixe multiple (`$liste-de` pour un attribut multivalué) peut éventuellement être complétée si la cardinalité maximale n'est pas atteinte.

- Facette `$si-besoin` : puisqu'un réflexe si-besoin implante une méthode de calcul, il peut être redéfini chaque fois que cela est nécessaire et/ou souhaitable.
- Facette `$si-possible`, `$si-ajout` et `$si-enleve` : comme nous le verrons au paragraphe 2.1.3, les réflexes introduits par ces trois facettes ne se masquent pas mais se cumulent. Par exemple, les réflexes si-ajout et si-enlève associés à `origine` dans `Horloge` "s'ajoutent" à ceux qui sont définis dans le frame `Fenetre`.

Ces règles sont comparables à celles qui sont utilisées en OBJLOG [Dugerdil, 1988] et en Shirka [Rechenmann, 1988]. Toutefois, en Shirka, une valeur fixe ne peut pas être redéfinie, ni même complétée, dans le cas où la cardinalité maximale n'est pas atteinte.

La position du concepteur de YAFOOL est encore autre [Ducournau, 1989b] (position réactualisée dans [Ducournau, 1992]). Le langage ne possède que la seule facette de valeur `value` et la facette `si-besoin`. La facette `value` joue le même rôle que la facette `$defaut` : elle introduit une valeur qui peut être masquée à n'importe quel niveau de la hiérarchie d'héritage. Pour indiquer qu'une valeur est fixe, il suffit d'utiliser la facette de typage `un` pour introduire la valeur fixe désirée.

Définition de représentants et contraintes globales

Les frames `Fenetre` et `Horloge` sont des modèles génériques qui servent à engendrer des *représentants* par l'intermédiaire d'une primitive qui est appelée ici `creation`. Par analogie avec les classes, les représentants sont quelquefois nommés instances. La structure d'un représentant est calquée sur celles des frames qui lui servent de modèles. Des valeurs spécifiques sont éventuellement données aux attributs du représentant, sinon ces valeurs sont recherchées ou calculées, lorsque c'est nécessaire, et s'il existe les moyens de le faire. Par exemple, la création pour une station de travail d'un environnement comportant une fenêtre `console`, une horloge et deux fenêtres de travail, étiquetées `editeur` et `lisp`, s'effectue de la façon suivante :

```
(creation Fenetre
          (origine (creation Point (abscisse 0) (ordonnee 0)))
          (coin (creation Point (abscisse 400) (ordonnee 100)))
          (etiquette "console"))

(creation Horloge)

(creation Fenetre
          (origine (creation Point (abscisse 0) (ordonnee 200)))
          (coin (creation Point (abscisse 450) (ordonnee 800)))
          (etiquette "editeur"))

(creation Fenetre
          (origine (creation Point (abscisse 450) (ordonnee 200)))
```

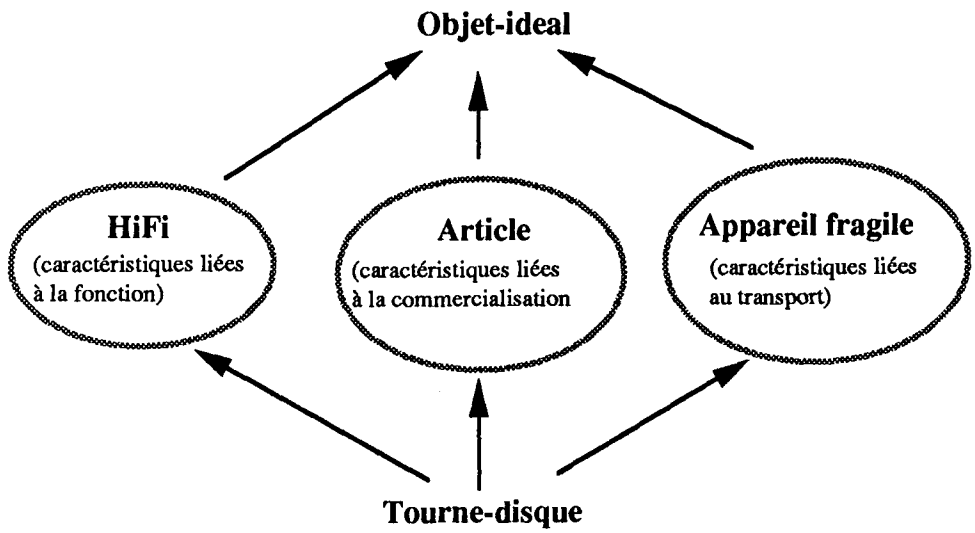


Figure 2.3. Modules dans un graphe d'héritage : l'objet `Tourne-disque` peut être considéré du point de vue commun à tous les articles pour ses références de base, du point de vue des appareils haute fidélité pour ses caractéristiques électriques, du point de vue des appareils fragiles pour son transport, etc. (d'après [Masini *et al.*, 1989]).

```
(etiquette "lisp"))
```

Il est possible de définir des contraintes dites *globales*, qui sont vérifiées à la création d'un représentant, en associant un réflexe si-possible à l'attribut `sorte-de`. En Shirka, un prédicat global est activé lors de la création de chaque instance, pour vérifier que l'instance respecte bien les conditions exprimées par le prédicat [Rechenmann, 1988]. Il est également nécessaire d'avoir recours à un prédicat global lorsqu'une contrainte globale porte sur plusieurs attributs et qu'elle ne peut être attachée en propre à l'un d'entre eux.

Les objets primitifs et les modules du graphe d'héritage

Un des critères sur lequel est fondé le calcul de la liste de priorité d'un objet dans un graphe d'héritage est la préservation des modules du graphe dans cette liste [Ducournau and Habib, 1989]. Un *module* est un sous-graphe d'héritage "autonome" qui décrit un point de vue sur l'univers modélisé. La figure 2.3 montre un exemple de décomposition en modules d'un graphe d'héritage. Un module a un plus grand et un plus petit élément pour l'ordre défini par l'héritage ; tout chemin dans le graphe d'héritage reliant un objet du module à un objet extérieur au module ne peut sortir du module qu'en passant par le plus grand ou le plus petit objet du module. La liste de priorité d'un objet quelconque du graphe est alors composée de sous-listes qui correspondent chacune à une liste de priorité "locale" à un module du graphe. Le graphe d'héritage dans son entier ou un objet seul sont des exemples de modules

triviaux. L'existence de modules non triviaux, comme ceux de la figure 2.3, montre qu'il est parfois possible de considérer des objets selon plusieurs points de vue.

Pour déclarer explicitement l'existence de modules indépendants dans un graphe d'héritage, nous introduisons la primitive `defprimitif`, qui définit un objet *primitif*, racine d'un module. A l'instar des concepts primitifs de systèmes comme KL-ONE [Brachman and Schmolze, 1985], les objets primitifs servent à dénoter des catégories très générales d'objets, catégories naturelles comme être humain, personne, animal, plante, etc.. Les propriétés des objets primitifs expriment des conditions nécessaires qui doivent être vérifiées par tout individu qui fait partie du module. Les objets primitifs déterminent une partition naturelle du graphe d'héritage en modules : un objet primitif peut être spécialisé par un objet primitif et les ancêtres primitifs d'un objet O définissent autant de points de vue sur O. Le choix du statut d'un objet, primitif ou non, est laissé à l'appréciation de l'utilisateur, qui est responsable du découpage de l'univers décrit en catégories d'objets.

Les unités de type `Basic` en KRL sont équivalentes à des objets primitifs [Bobrow and Winograd, 1977]. Elles représentent des concepts généraux et déterminent une partition des connaissances en arbres d'héritage simples dont elles sont les racines. Les unités de type `Abstract` sont d'un niveau de généralité comparable à celui des unités de type `Basic`, mais elles sont plutôt utilisées pour factoriser des informations à la manière de classes abstraites, sans déterminer aucune partition de l'univers représenté.

Ensembles d'objets dans le graphe d'héritage

Dans ce paragraphe, nous présentons une vision ensembliste des objets d'un graphe d'héritage, inspirée de [Chabre, 1988], et nous définissons les opérations particulières qui s'appliquent à de tels ensembles d'objets[6].

Le lien `sorte-de` relie un objet O à ses ascendants directs ou pères, qui sont les objets du graphe d'un niveau de généralité immédiatement supérieur, que O spécialise. Le lien `specialisations` relie O aux objets qui spécialisent directement O, objets qui sont d'un degré de généralité immédiatement inférieur. Les liens `sorte-de` et `specialisations` sont des attributs particuliers gérés par le système. Les ascendants d'un objet sont les objets qui font partie de la fermeture transitive du lien `sorte-de`, ses descendants sont ceux qui font partie de la fermeture transitive du lien `specialisations`.

Soit $\mathcal{S}$ un ensemble d'objets situés dans un graphe d'héritage. Par définition, les *maximaux* de $\mathcal{S}$ sont les objets qui ne possèdent aucun ascendant dans $\mathcal{S}$, les *minimaux* de $\mathcal{S}$ sont les objets qui ne possèdent aucun descendant dans $\mathcal{S}$. L'ensemble des maximaux des objets $\{I\ J\ D\ G\ H\}$, situés dans le graphe d'héritage formel de la figure 2.4, est $\{D\ H\}$, celui de ses minimaux est $\{I\ J\ H\}$.

Tout objet est racine d'un sous-graphe d'héritage, éventuellement réduit à un élément unique qui est l'objet lui-même. Il est possible d'associer à chaque objet O du graphe l'ensemble de ses descendants et d'identifier cet ensemble à O. Par extension, un

6. Une approche différente est présentée dans [McAllester and Zabih, 1986] où sont définies des "classes booléennes" et les opérations booléennes associées.

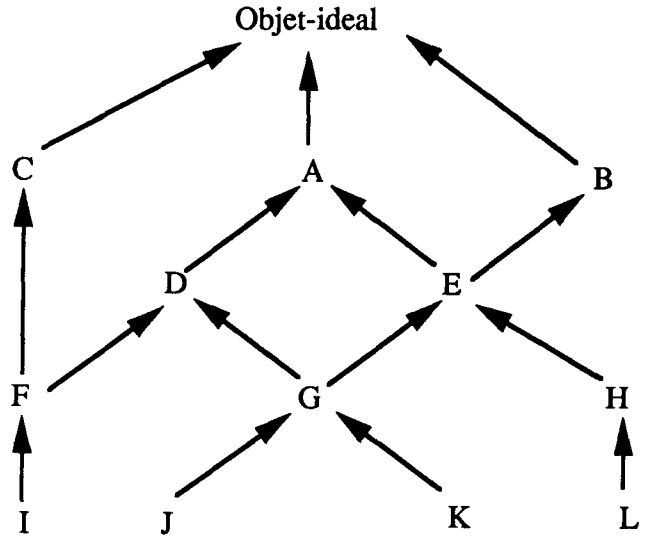


Figure 2.4. Un graphe d'héritage formel.

ensemble d'objets peut être considéré comme un ensemble de maximaux : l'ensemble $\{D\ H\}$ dénote l'ensemble $\{I\ F\ J\ G\ K\ D\ H\ L\}$.

Un ensemble $\mathcal{S}$ est alors inclus dans un ensemble $\mathcal{T}$ si les maximaux de $\mathcal{S}$ sont inclus dans $\mathcal{T}$: les maximaux de $\mathcal{S}$ sont des descendants des maximaux de $\mathcal{T}$, l'ensemble $\{F\ H\}$ est inclus dans $\{A\}$.

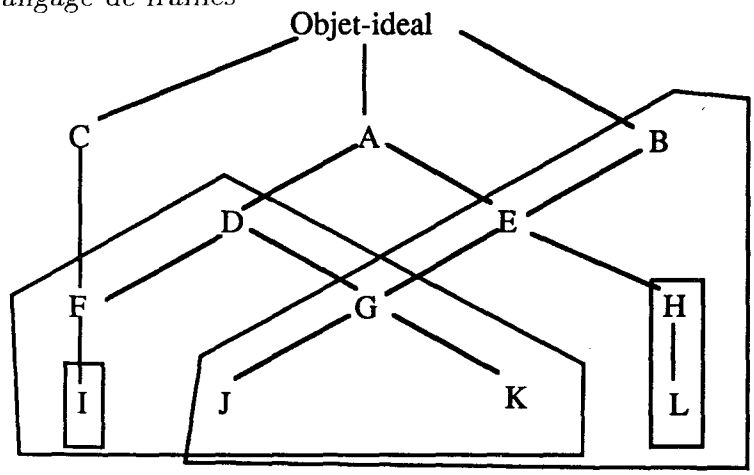
La réunion de deux ensembles d'objets se définit comme la réunion des maximaux de chaque ensemble : la réunion de l'ensemble $\{I\ B\}$ et de l'ensemble $\{D\ H\}$ est l'ensemble $\{D\ B\}$ (cf. Fig. 2.5).

L'intersection de deux ensembles d'objets se définit comme l'ensemble des descendants communs maximaux des maximaux de chaque ensemble. Ainsi, l'intersection de l'ensemble $\{A\}$ et de l'ensemble $\{C\ B\}$ est l'ensemble $\{F\ E\}$. Le complémentaire d'un ensemble $\mathcal{S}$ dans un ensemble $\mathcal{T}$ se définit comme l'ensemble des maximaux des objets de $\mathcal{T}$ n'ayant aucun descendant dans $\mathcal{S}$. Ainsi, le complémentaire de $\{C\}$ dans l'ensemble $\{Objet\text{-}ideal\}$ se réduit à $\{B\}$ et vaut $\{G\}$ dans $\{D\}$.

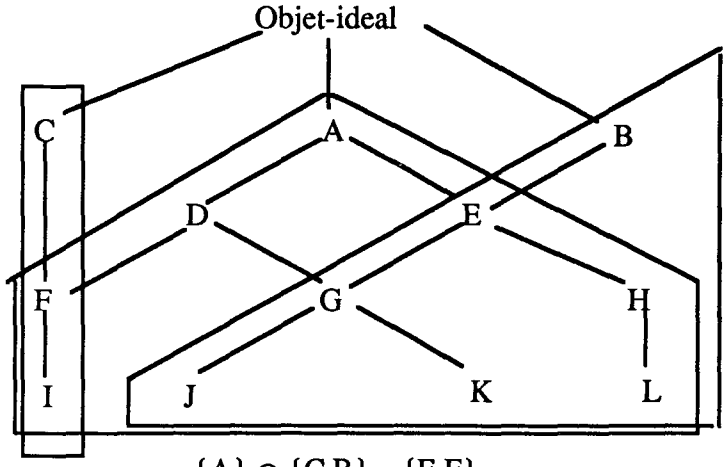
Fixer, posséder et hériter une propriété

La définition d'objets maximaux permet de différencier les expressions "fixer", "posséder" et "hériter" une propriété[7]. Un objet *fixe* une propriété si elle n'est pas héritée mais effectivement définie dans la description de l'objet, ou encore si elle est masquée dans la description de l'objet (la propriété est héritable mais redéfinie). Un objet *possède* une propriété s'il la fixe ou s'il l'hérite.

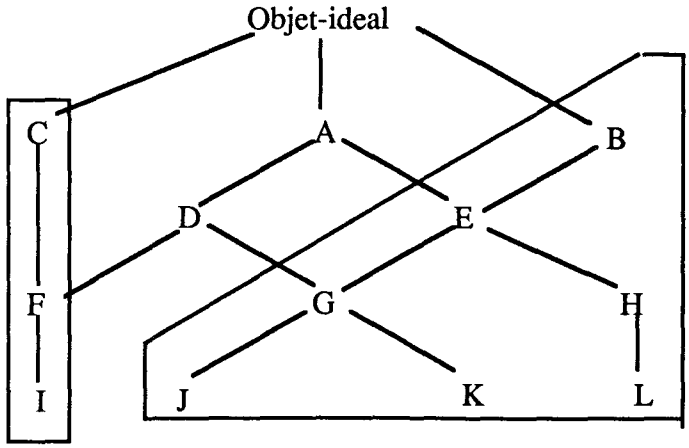
7. *Hériter n'est pas posséder* dans [Ducournau and Habib, 1991], où les termes employés sont respectivement "posséder", "avoir" et "hériter".



{I B} ∪ {D H} = {B D}



{A} ∩ {C B} = {E F}



Complémentaire de {C} dans {Objet-ideal} = {B}

Figure 2.5. Opérations ensemblistes dans un graphe d'héritage formel : réunion, intersection et passage au complémentaire.

Considérons un objet O faisant partie d'un ensemble $\mathcal{S}$ d'objets situés dans un graphe d'héritage. L'objet O est dit *maximal pour la propriété* P dans $\mathcal{S}$ si O est maximal parmi les objets qui fixent P dans $\mathcal{S}$. Ainsi, O peut être maximal et fixer P, ou encore fixer P sans qu'aucun de ses ascendants ne fixe P dans $\mathcal{S}$. Parallèlement, O est dit *minimal pour la propriété* P dans $\mathcal{S}$ si O est minimal parmi les objets qui fixent P dans $\mathcal{S}$.

En Smalltalk-80 par exemple, une classe est maximale pour chacune de ses variables d'instances. Elle n'est minimale que dans le cas où elle ne possède aucune sous-classe, car les variables d'instances font partie de la structure des sous-classes. En revanche, rien de tel ne peut être affirmé pour un frame, puisque ses propriétés peuvent être soit fixées soit héritées dynamiquement.

Représentation d'intensions et d'extensions

Une classe ou un prototype permettent d'appréhender une famille d'objets en définissant l'intension des objets de la famille, intension que nous considérons comme le dénominateur commun à tous les objets de la famille (oublions temporairement les différences qui existent entre les approches classe et prototype exposées au chapitre précédent). Dans la plupart des cas courants, il est assez facile de définir une intension, aussi minimale soit-elle. Pourtant, il est des cas où l'extension d'un concept est parfaitement connue, mais où l'intension du concept est plus délicate à définir. Un exemple de ce paradoxe est l'étoile du soir et du matin de G. Frege [Frege, 1971], deux intensions possibles pour un seul référent, en l'occurrence la planète Vénus.

Voici un autre exemple, tiré de l'univers des objets de la chimie, qui peut se généraliser à tout autre domaine. La famille des halogènes comporte quatre atomes, le fluor, le chlore, le brome et l'iode, qui définissent l'extension du concept *halogène*. Ces atomes font partie de la même colonne dans la table de Mendeleev et ont un certain nombre de propriétés communes, une valence égale à 1 par exemple. L'intension du concept *halogène* peut alors être définie d'au moins deux façons possibles :

- Un halogène est un atome particulier, donc le concept halogène est décrit comme une spécialisation de l'objet `Atome` (cf. § 5.3.1), où est indiqué l'intervalle de valeurs, en l'occurrence `[9 17 35 53]`, auquel doit appartenir le numéro atomique d'un atome de la famille. Les atomes de fluor, de chlore, de brome et d'iode sont des spécialisations du concept halogène.

- Un halogène est un atome qui est soit un fluor, un chlore, un brome ou une iode. Le "superatome" halogène est donc une spécialisation de chacun de ces atomes, puisqu'il partage indifféremment les propriétés de chacuns des halogènes et qu'il peut leur être substitué.

Ces deux descriptions sont radicalement différentes. Toutefois, dans le second cas, il serait plus exact de préciser que les liens de spécialisation existant entre halogène et les atomes fluor, chlore, brome et iode sont de type disjonctif et non de type conjonctif : un halogène est un fluor, *ou* un chlore, *ou* un brome, *ou* une iode, mais n'est pas un fluor,

et un chlore, *et* un brome, *et* une iode. Ce paradoxe se retrouve chaque fois lorsqu'il est nécessaire de définir un objet qui décrive à la fois l'ensemble et l'élément générique de l'ensemble [Winograd, 1978]. Nous proposons au chapitre 5 une solution partielle pour la représentation du superatome halogène qui dérive du premier cas, car nous ne disposons d'aucun moyen nous permettant de décrire des disjonctions (cf. page 177).

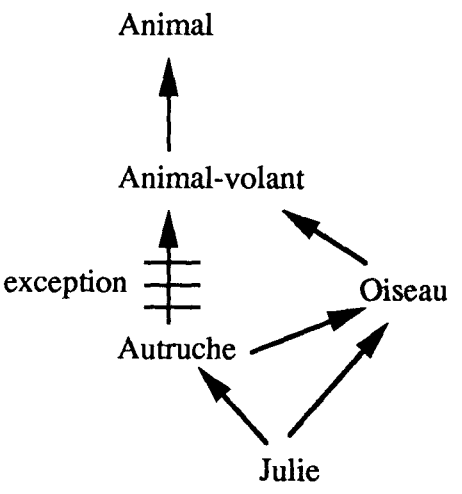
Les exceptions à l'héritage

Un frame ou une classe appartenant à une hiérarchie d'héritage dénotent une certaine catégorie d'individus. Les propriétés qui sont associées à ces objets générateurs sont *obligatoirement* héritées par leurs spécialisations, à moins qu'elles ne soient masquées. Cependant, il est difficile, voire impossible, de définir un objet ne décrivant que des propriétés valides pour *tous* les individus de la catégorie étudiée : toute généralisation est simplificatrice et comporte inévitablement des cas particuliers [Brachman, 1985]. Pour marquer le caractère exceptionnel de certains individus, il est nécessaire de masquer les valeurs de certaines des propriétés héritées. Ainsi, dans un modèle de classes, une sous-classe particulière doit être créée à cet effet, même pour un seul individu ou une seule propriété exceptionnels.

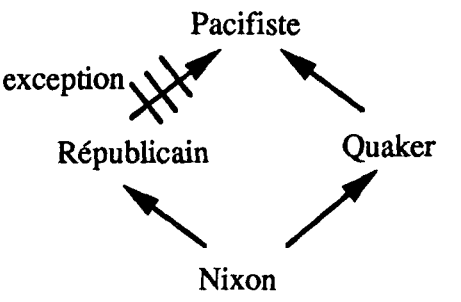
Lorsqu'un individu possède simultanément plusieurs propriétés exceptionnelles, certains systèmes de représentation autorisent la création d'un arc d'exception entre cet individu et l'objet dont il hérite, pour indiquer que le premier n'hérite pas des caractéristiques du second. Cette façon de faire est préférable au masquage de chacune des propriétés remises en cause, qui va à l'encontre de la réutilisation : chaque fois qu'une propriété est ajoutée ou modifiée dans les ascendants d'une exception, il faut vérifier sa compatibilité avec l'exception ; le maintien de la cohérence du graphe d'héritage devient très difficile à assurer et pratiquement impossible à automatiser, faute d'indication explicite sur les incompatibilités existant entre les objets liés hiérarchiquement.

Les exceptions font perdre sa transitivité à la relation d'héritage et conduisent à un raisonnement non monotone. Pour un objet, l'existence simultanée de plusieurs de chemins dans le graphe d'héritage lui permettant d'hériter et de ne pas hériter une même propriété produit des contradictions qu'il faut résoudre. Comme dans le cas de l'héritage sans exceptions, une partie des contradictions se traite grâce au masquage, en particulier celles qui reposent un arc de transitivité, direct ou indirect, de la relation d'héritage [Ducournau and Habib, 1989]. Un arc de transitivité est un arc reliant deux objets comparables mais non consécutifs pour l'ordre induit par la relation d'héritage. Un arc de transitivité est redondant puisqu'il exprime un héritage déjà défini par ailleurs, mais il n'est pas forcément contradictoire. Par exemple, l'arc `Julie→Oiseau` est redondant car il double le chemin `Julie→Autruche→Oiseau` ; il est contradictoire car il fait hériter `Julie` des propriétés de `Animal-volant` alors qu'il existe un arc exceptionnel entre `Autruche` et `Animal-volant` (Fig. 2.6.1). Dans un tel cas, le masquage s'applique : `Julie` hérite d'abord des propriétés de `Autruche` qui masquent celles de `Animal-volant`.

Les contradictions *pures* sont celles qui ne peuvent pas être traitées grâce au mas-



1 : Contradiction résolue par masquage : les propriétés de Autruche masquent celles de Animal-volant



2 : Contradiction pure
Nixon est quaker et républicain, est-il Pacifiste ?

Figure 2.6. Exceptions à l'héritage et contradictions.

quage. La figure 2.6.2 en montre un exemple célèbre : `Nixon` est défini comme étant simultanément `Quaker` et `Républicain`, ce qui provoque l'existence de deux chemins d'héritage, entre `Nixon` et `Pacifiste`, l'un permettant d'hériter des propriétés de `Pacifiste` et l'autre pas. Il n'existe aucun moyen de résoudre ce genre de contradiction sans introduire des critères supplémentaires permettant de trancher en faveur de l'un ou l'autre cas.

La gestion des exceptions à l'héritage reste un problème ouvert. Aucune solution complètement satisfaisante n'a encore été proposée, malgré des recherches actives dans le domaine [Etherington, 1987] [Shastri, 1989] [Garlatti, 1990]. Dans leur approche, J.F. Horty, R.H. Thomason et D.S. Touretzky ont mis au point un ensemble de règles d'inférences qui permettent de déduire des informations à partir d'un réseau où les nœuds sont connectés entre eux par des liens positifs, ou négatifs s'ils traduisent une exception [Horty *et al.*, 1990] [Horty, 1991]. Cette approche peut être qualifiée de déclarative au sens où, comme en logique, des faits sont déclarés et représentés dans un réseau, les nouveaux faits étant déduits des faits disponibles à l'aide de règles d'inférences.

Une place à part doit être faite à l'approche employée pour traiter les exceptions en YAFOOL [Ducournau and Habib, 1991]. Les exceptions s'expriment grâce à une facette `sauf`, qui est associée au lien d'héritage. Une autruche est un oiseau mais n'est pas un animal volant :

```
(defmodele Autruche
  (est-un (value Oiseau Animal)
          (sauf Animal-volant)))
```

Certaines contradictions, comme celle de la figure 2.6.1, sont résolues par masquage : `Autruche` est plus spécifique qu'`Oiseau` pour `Julie`, la propriété *ne pas être*

un Animal-volant associée à `Autruche` masque la propriété *être un Animal-volant* possédée par `Oiseau` : `Julie` n'hérite pas de `Animal-volant`. Les propriétés associées au frame `Animal-volant` et à ses ascendants ne sont alors pas héritées par le frame `Autruche`. Toutefois, une autruche étant malgré tout un animal, le frame `Autruche` doit hériter du frame `Animal`, comme le précise la facette `value` du lien `est-un`. Les contradictions pures sont traitées en donnant la priorité à l'héritage : puisqu'il existe, le chemin du graphe d'héritage qui permet à l'objet `Nixon` d'hériter des propriétés associées à `Pacifiste`, l'emporte sur le chemin `Nixon`→`Republicain`↛`Pacifiste`.

2.1.3 La programmation dirigée par les accès

Contrairement aux classes, les frames n'ont pas de comportement propre et sont manipulés par des *primitives*. Cependant, des procédures peuvent être attachées localement aux attributs pour définir des méthodes de calcul, de gestion ou d'utilisation de ces derniers. Ces procédures traduisent des raisonnements locaux et décentralisés et se déclenchent lors des accès aux attributs. Il existe trois modalités d'accès principales à la valeur d'un attribut :

- La *lecture* retourne la valeur de l'attribut (primitive `lire`).
- L'*écriture* met en place ou encore ajoute une valeur à l'attribut (primitives `ecrire` et `ajouter`).
- La *suppression* efface la valeur de l'attribut ou supprime une valeur dans la liste de valeurs associée à l'attribut (primitives `detruire` et `enlever`).

Les accès standard en lecture

L'accès en lecture permet d'obtenir la valeur d'un attribut. Il s'appuie sur un parcours dynamique de la hiérarchie d'héritage, qui peut être effectué selon plusieurs stratégies et qui dépend d'une priorité accordée aux facettes de valeur. Une lecture doit retourner en premier la valeur fixe si elle existe, sinon la valeur calculée par un réflexe si-besoin si le réflexe existe, et en dernier recours la valeur par défaut. Ainsi, la facette `$valeur` prime sur `$si-besoin`, qui prime sur `$defaut`. Cet ordre n'est pas toujours respecté : une lecture standard en FRL se fait sans tenir compte de la facette `$si-besoin` [Roberts and Goldstein, 1977] ; la priorité standard des facettes dans la première version de MERING est donnée par la liste `$valeur`, `$defaut` et `$si-besoin`, mais cette priorité peut être redéfinie [Ferber, 1983].

Pour notre part, l'accès standard en lecture est la lecture en N, qui est composée de trois lectures successives en I. Pour un objet, un attribut et une facette donnés, une lecture en I consiste à remonter la hiérarchie d'héritage de l'objet, jusqu'à ce que le couple attribut-facette ait été rencontré. La lecture en N commence par une lecture en I pour la facette `$valeur`, qui est suivie, si aucune valeur n'est délivrée, d'une lecture en I pour la facette `$si-besoin`, suivie elle-même, si aucune valeur n'est produite, d'une lecture en I pour la facette `$defaut`. Le résultat donné par la lecture en N est

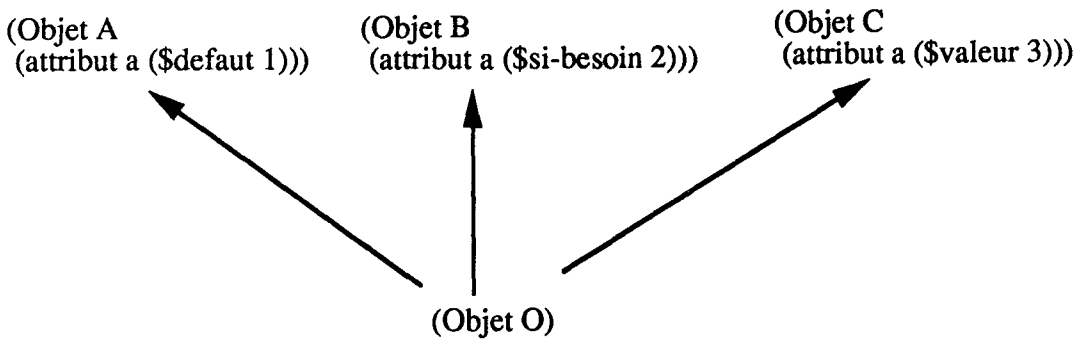


Figure 2.7. Un cas où la lecture en Z ne retourne pas la valeur fixe.

toujours conforme à la priorité définie sur les facettes, quelles que soient les positions relatives des facettes dans la hiérarchie d'héritage de l'objet.

En YAFOOL, la lecture standard est la lecture en Z, qui consiste à consulter les facettes `value` (pour `$valeur`) et `si-besoin`, dans l'ordre, pour chacun des ascendants de l'objet considéré jusqu'à ce qu'une valeur soit retournée. Le résultat fourni par la lecture en Z n'est pas toujours conforme à la priorité définie sur les facettes : la lecture en Z privilégie l'ordre induit par la relation d'héritage, au détriment de la priorité définie sur les facettes, qui n'est prise en compte que localement à un ascendant de l'objet considéré. Un résultat incorrect peut alors être fourni : une facette `$defaut` est héritée avant une facette `$valeur` ou `$si-besoin`, ou bien une facette `$si-besoin` est héritée avant une facette `$valeur` (cf. Fig.2.7). Toutefois, il est possible de redéfinir le mode de lecture standard grâce à la facette `type-de-recherche`, dont la valeur, qui est Z par défaut, peut être I ou N[8]. Comme dans la plupart des langages de frames, l'utilisateur dispose de plusieurs primitives de lecture qui lui permettent de choisir la stratégie la mieux adaptée à chaque opération de lecture, y compris des primitives qui ne déclenchent pas les réflexes.

Nous proposons au chapitre 3 une série de règles qui garantit le bon respect de la priorité définie sur les facettes (cf. page 109). Si la hiérarchie d'héritage est construite en obéissant à ces règles, alors les lectures en N et Z sont équivalentes.

Les accès standard en écriture

L'écriture de la valeur d'un attribut se déroule en trois temps :

- vérifications *a priori* : les contraintes spécifiées par les facettes de typage, les facettes de restriction de type, les facettes de cardinalité et les réflexes si-possible doivent être vérifiées, des plus générales aux plus particulières. Si une des contraintes n'est pas satisfaite, l'écriture n'est pas effectuée,

8. Un mode de lecture mixte I-Z, I pour la facette `$valeur` et Z pour les facettes `$si-besoin` et `$defaut` dans l'ordre, est présenté dans [Ducournau, 1992].

- écriture : la valeur est mise en place dans la facette `$valeur` de l'attribut considéré.
- actions *a posteriori* et propagation des modifications : mise à jour des valeurs des attributs qui sont en relation, et déclenchement des réflexes si-ajout des plus généraux aux plus spécifiques, sans remettre en cause l'écriture.

Aucune opération n'est effectuée si la valeur à mettre en place est égale à l'ancienne valeur. Cette restriction est destinée à éviter les boucles infinies d'appels de réflexes *a posteriori*, en particulier dans la gestion des relations [Napoli and Masini, 1989].

La suppression d'une valeur se déroule en deux temps : effacement de la valeur et activation des réflexes si-enlève dans l'ordre ascendant de la hiérarchie d'héritage, c'est-à-dire des plus particuliers aux plus généraux.

Le réflexe si-besoin et la cohérence d'une hiérarchie de frames

Le calcul d'une valeur à l'aide d'un réflexe si-besoin pose deux problèmes particuliers : la détection de boucles infinies d'appels de réflexes si-besoin et le maintien de la cohérence des valeurs d'attributs interdépendants.

Le premier problème se pose lorsque la valeur d'un attribut *a1* est calculée par un réflexe si-besoin, qui utilise la valeur d'un attribut *a2*, qui elle-même est calculée par un réflexe si-besoin pour lequel la valeur de *a1* est nécessaire... Les langages YAFOOL et OBJLOG disposent d'un mécanisme permettant de détecter dynamiquement une éventuelle boucle infinie d'appels de réflexes si-besoin. Notons que ces problèmes de circularité se posent aussi avec des méthodes. Des attributs comme *a1* et *a2*, dont les valeurs sont interdépendantes, sont appelés attributs *adjacents*.

Certains attributs adjacents ont des valeurs qui évoluent en parallèle : les modifications apportées à l'une des valeurs sont automatiquement reportées sur l'autre grâce au mécanisme des relations (introduites par les facettes `$lien-inverse` ou `$meme-que`). Il est des cas où deux attributs faisant partie du même frame sont adjacents, la valeur de l'un étant calculée en fonction de la valeur de l'autre grâce à un réflexe si-besoin. Par exemple, la valeur des attributs `hauteur` et `largeur` du frame `Fenetre` dépend des valeurs des attributs `origine` et `coin`. Lorsque ces deux dernières changent, les deux premières doivent être modifiées en conséquence. Il est alors nécessaire de résoudre le problème suivant : faut-il ou ne faut-il pas affecter une valeur produite par un réflexe si-besoin ?

Trois types de solutions sont envisageables :

- Ne jamais écrire la valeur calculée, comme en OBJLOG : les valeurs d'attributs adjacents calculées par un réflexe si-besoin sont toujours cohérentes, puisqu'elles sont effectivement calculées chaque fois qu'elles sont consultées. En contrepartie, chaque consultation demande un surcroît de temps de calcul.
- Écrire systématiquement la valeur calculée, comme en YAFOOL : la mise à jour des valeurs d'attributs interdépendants doit être programmée explicitement, par l'intermédiaire de réflexes si-ajout et si-enlève. En YAFOOL, il est possible de

redéfinir le mode de lecture standard, Z ou N, sans pour autant remettre en cause l'écriture systématique. Pour contourner cela, il faut soit utiliser un échappement dans le corps du réflexe si-besoin, soit remplacer le réflexe par une méthode qui calcule la valeur de l'attribut à chaque accès (cf. § 2.2.5).

- Associer au langage un mécanisme de maintien de la cohérence des valeurs d'attributs, comme en Shirka [Euzénat, 1990], en KEE [Filman, 1988] ou en KRS [Van Marcke, 1986] : une valeur calculée est mise en place et, lorsqu'elle est modifiée, les valeurs qui en dépendent sont marquées de manière à être actualisées à leur prochaine utilisation. Si l'automatisation du maintien de la cohérence offre un meilleur confort de programmation et une plus grande sécurité, elle ne doit en aucun cas faire baisser les performances du système dans son entier.

Protection de données et encapsulation

Généralement, dans un modèle de classes, les communications avec les instances se font par l'intermédiaire d'une interface qui cache l'organisation interne des instances, assurant ainsi l'abstraction de données. En Smalltalk-80 par exemple, la valeur d'une variable d'instance n'est accessible que par une méthode spécialement définie pour cela. En revanche, comme dans tout modèle fondé sur les prototypes [Chambers *et al.*, 1991], aucune donnée associée à un frame n'est *a priori* privée. Les réflexes standard ne donnent généralement pas la possibilité d'interdire l'accès en lecture à la valeur d'un attribut. Notons cependant que la première version de MERING disposait d'un réflexe si-lu, associé à une facette de même nom, qui permettait de réglementer et éventuellement interdire l'accès en lecture à la valeur d'un attribut [Ferber, 1983].

En revanche, les réflexes si-possible peuvent interdire l'accès en écriture à la valeur d'un attribut. Par exemple, le réflexe si-possible associé aux attributs `hauteur` et `largeur` dans le frame `Fenetre` retourne systématiquement faux (`nil`), empêchant toute écriture et conférant ainsi à ces deux attributs un statut analogue à celui d'une donnée privée. Il est possible d'accéder en écriture à ces attributs en utilisant une primitive qui ne déclenche pas le réflexe si-possible. Comme pour l'accès en lecture, ils existe des primitives d'accès en écriture qui ne déclenchent pas les réflexes (Fig. 2.8).

Le cas du langage de prototypes SELF est particulier. Il n'est possible d'accéder à une variable, appelée *slot*, que par l'intermédiaire d'un envoi de message, aussi bien en lecture qu'en écriture [Ungar and Smith, 1991]. Comme dans les langages de l'"école scandinave" [Masini *et al.*, 1989], il est possible de spécifier pour le *slot* un statut public ou privé en lecture et/ou écriture. Il existe ainsi quatre possibilités d'accès à un slot. Le statut public en lecture et écriture correspond au statut par défaut, qui est surtout utilisé en phase de développement de programme.

La sémantique des réflexes

Ce paragraphe a pour but de clarifier la sémantique de l'ordre du déclenchement des réflexes. Une bonne maîtrise des réflexes est de première importance en programmation dirigée par les accès. Les réflexes servent principalement à calculer des valeurs

Lecture directe de la valeur de l'attribut
`fget frame attribut facette`
Variantes :
Lecture en I pour une facette : `fget-I`
Lecture en N avec ou sans réflexe si-besoin : `fget-N`
Lecture en Z avec ou sans réflexe si-besoin : `fget-Z`

Ecriture directe de valeur dans attribut
(avec écrasement de l'ancienne valeur)
Variantes :
avec ou sans déclenchement des réflexes si-possible et si-ajout
`fput frame attribut facette valeur`
Adjonction directe d'une valeur dans une liste
Variantes :
avec ou sans déclenchement des réflexes si-possible et si-ajout
`fadd frame attribut valeur` :

Suppression directe de valeur dans facette
Variantes :
avec ou sans déclenchement des réflexes si-enlève
`frem frame attribut facette valeur`

Figure 2.8. Les principales primitives d'accès à la valeur d'un attribut.

d'attributs et à propager par effets de bord les modifications qui en découlent [Chein and Cogis, 1989]. Pour l'écriture comme pour la lecture, fixer la sémantique du déclenchement des réflexes est arbitraire. Toute sémantique est concevable à condition de respecter une certaine cohérence. Des études, proches de celle qui suit, ont été réalisées pour LOOPS [Bobrow and Stefik, 1983] (justification de l'ordre du déclenchement des actions associées à des valeurs actives emboîtées), pour TOPIC [Reimer and Hahn, 1983] [Reimer and Hahn, 1985] (approche formelle de la sémantique d'un modèle de données fondés sur des frames) et enfin pour BETA [Østerbye, 1988] (mise en œuvre de la programmation dirigée par les accès par le biais d'objets actifs).

Dans notre cas, la sémantique qui régit le déclenchement des réflexes est empruntée à YAFOOL [Ducournau, 1989b]. Elle suppose que les réflexes sont du même niveau de généralité que les frames auxquels ils sont associés et qu'ils sont déclenchés dans un environnement adéquat, c'est-à-dire que les opérations attachées aux informations les plus spécifiques s'exécutent dans un environnement où les informations les plus générales sont à jour.

Ainsi, pour un frame donné, les réflexes si-possible se déclenchent dans l'ordre inverse de la liste de priorité : la validité d'une contrainte spécifique n'est testée que si les contraintes qui sont plus générales sont satisfaites. Comme les types associés aux

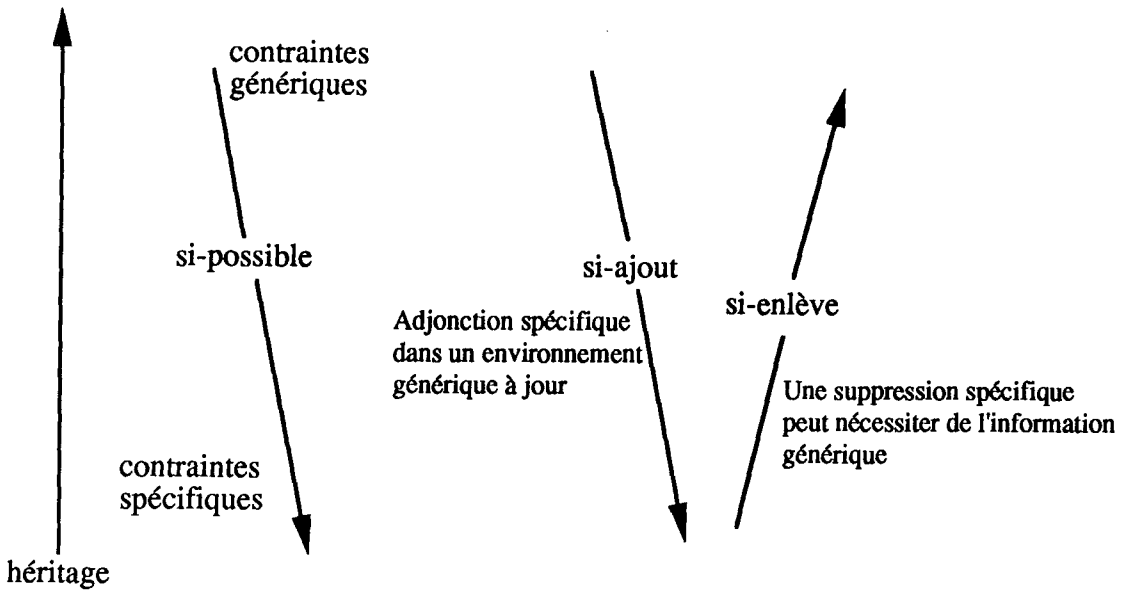


Figure 2.9. Le sens du déclenchement des réflexes.

attributs, les réflexes si-possible testent des contraintes qui sont "emboîtées".

Les réflexes si-ajout et si-enlève servent à propager des actions résultant de modifications de valeurs. Puisqu'un réflexe si-ajout se déclenche à la suite d'une adjonction de valeur, il ne peut modifier l'environnement à un niveau donné que si les niveaux supérieurs sont à jour. Par suite, les réflexes généraux, qui mettent à jour les informations les plus générales, sont déclenchés avant les réflexes spécifiques. Pour l'objet particulier `objet` et l'attribut `attribut`, l'événement (`déclenchement F1 attribut si-ajout`), déclenchement du réflexe si-ajout associé à `attribut` dans l'objet `F1`, précède l'événement (`déclenchement F2 attribut si-ajout`) si `F2` précède `F1` dans la liste de priorité de `objet` (cf. Fig. 2.9). Ainsi, lorsque le frame `Horloge`, sous-frame de `Fenetre`, est créé (cf. Figures 2.1 et 2.2), les réflexes si-ajout `dessiner-cadre` et `afficher-heure` sont déclenchés dans cet ordre, le cadre de l'horloge étant affiché avant l'heure elle-même.

Pour les réflexes si-enlève, le sens de déclenchement est, par symétrie, inversé. Puisqu'un réflexe si-enlève se déclenche à la suite d'une suppression de valeur, il peut avoir besoin, pour modifier l'environnement, d'informations qui se trouvent éventuellement dans les niveaux supérieurs, qui doivent donc être supprimées ultérieurement s'il y a lieu. Ainsi, la mise à jour de l'environnement est commencée par les réflexes spécifiques et achevée par les réflexes généraux. Pour l'objet `objet` et l'attribut `attribut`, l'événement (`déclenchement F1 attribut si-enlève`), déclenchement du réflexe si-enlève associé à `attribut` dans l'objet `F1`, précède l'événement (`déclenchement F2 attribut si-enlève`) si `F1` précède `F2` dans la liste de priorité de `objet`. Ainsi, lorsque l'horloge est effacée à la suite de la suppression de la valeur d'`origine` ou de `coin`, les

réflexes si-enlève `effacer-heure` et `effacer-cadre` sont déclenchés dans cet ordre, l'heure étant effacée avant le cadre.

Notons en particulier que le masquage des réflexes autres que le réflexe si-besoin n'a pas de sens. Dans certains cas, il pourrait être judicieux d'autoriser l'utilisateur à définir son propre mode de déclenchement des réflexes, comme dans les Flavors [Moon, 1986]. Une certaine analogie peut être faite avec les constructeurs et les destructeurs du langage C++ [Stroustrup, 1989]. Les constructeurs servent à créer les instances d'une classe, les destructeurs à les supprimer. Par exemple, l'allocation dynamique de place mémoire et l'initialisation des instances est assurée par un constructeur, la suppression et la récupération de la place mémoire occupée par les instances devenues obsolètes est assurée par un destructeur. Lorsqu'une classe et une de ses sous-classes sont munies chacune d'un constructeur et d'un destructeur, le constructeur de la classe est activé avant celui de la sous-classe, le destructeur de la sous-classe est activé avant celui de la classe.

2.1.4 Le filtrage

Le filtrage est un mécanisme qui permet de retrouver des objets ou des termes qui s'apparient avec un *modèle* ou *filtre* donné. Dans un contexte logique, un filtre est un terme particulier qui contient un certain nombre de constantes et de variables, décrivant les contraintes qui doivent être satisfaites par les termes recherchés. Mettre en correspondance un terme et le filtre consiste à déterminer, s'il existe, un ensemble de substitutions liant les variables du filtre avec des valeurs détenues par le terme de telle façon que le filtre et le terme soient "identiques" après substitution. Lorsque toutes les variables sont liées, le filtre et le terme sont structurellement isomorphes et l'appariement est dit *complet*, sinon il est dit *partiel*.

```
F = (soigne (agent x) (patient y))
A1 = (soigne (agent Esculape) (patient Moliere))
A2 = (soigne (agent Esculape) (moyen bistouri))
```

Dans les expressions précédentes, le filtre `F`, qui contient les variables `x` et `y`, s'apparie complètement avec la donnée `A1` via les substitutions `x = Esculape` et `y = Moliere`. En revanche, `F` ne s'apparie que partiellement avec la donnée `A2` via la substitution `x = Esculape`. Le résultat obtenu met en valeur la partie commune[9] de `A2` et de `F`.

L'exemple précédent porte sur des formules du calcul des prédicats. Dans le contexte de la programmation par objets, le filtrage sert à déterminer la valeur d'un attribut ou encore à rechercher un ensemble d'objets vérifiant des contraintes données. Toutefois, il n'est généralement pas possible dans un objet d'exprimer que la valeur d'un attribut est une variable dont il faut trouver la valeur. Les "variables du filtre" sont

9. L'appariement partiel est utilisé entre autres en apprentissage symbolique pour généraliser plusieurs descriptions et procéder à un raisonnement inductif ou analogique [Kodratoff, 1986], ainsi qu'en interprétation de données textuelles, visuelles ou autres [Miezitis, 1988], [Gomez and Segami, 1989].

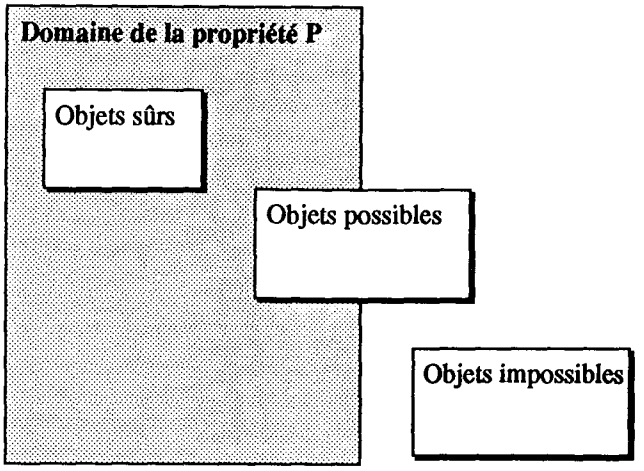


Figure 2.10. Les objets sûrs, possibles et impossibles sont déterminés selon la position de l'intersection de l'ensemble de valeurs D de la propriété P détenue par le filtre, avec l'ensemble de valeurs de la propriété P associé à chaque objet considéré.

donc remplacées par des contraintes portant sur le type et la valeur des attributs du filtre, qui doivent être vérifiées par les objets recherchés. Ainsi, le filtrage peut être vu comme un mécanisme permettant de passer d'une intension à une extension, autrement dit d'une conjonction de propriétés, le filtre (l'intension), à un ensemble d'objets donné en extension qui s'apparient avec le filtre (l'extension). Il sert en particulier de mécanisme de base pour le raisonnement par classification présenté au chapitre suivant.

Le filtrage s'effectue avec une logique à trois valeurs, vrai, faux et inconnu. La dernière valeur correspond à l'*hypothèse du monde ouvert* : un objet ne vérifiant pas toutes les conditions d'un filtre n'est pas rejeté s'il ne contient aucune information incompatible avec celles détenues par le filtre.

Considérons par exemple un filtre défini par la seule propriété P ayant D comme ensemble de valeurs. Les objets du graphe d'héritage se répartissent alors en trois catégories (Fig. 2.10). Les objets *sûrs* et *possibles* possèdent la propriété P, avec un ensemble de valeurs égal ou inclus dans D pour les premiers, d'intersection non vide avec D pour les seconds. Dans les autres cas, les objets ainsi que leurs descendants, sont dits *impossibles* : dans une hiérarchie bien formée, l'ensemble des valeurs associé à une propriété doit vérifier les contraintes qui sont définies plus haut dans la hiérarchie, en particulier un domaine spécialisant est inclus dans le domaine qu'il spécialise (cf. § 2.1.2). Ainsi, les descendants d'objets impossibles sont impossibles, les descendants d'objets possibles sont possibles mais peuvent devenir sûrs ou impossibles en cas de redéfinition de domaine.

Plusieurs filtres peuvent être imbriqués dans la mesure où un filtre peut lui-même comporter un attribut dont la valeur est calculée à l'aide d'un filtre et ainsi de suite.

Plusieurs stratégies de filtrage sont alors envisageables. Par exemple, les propriétés dont la valeur n'est pas calculée à l'aide d'un filtre sont considérées les premières ou encore les propriétés sont triées selon un degré d'importance et cet ordre doit être respecté pendant le filtrage.

Le filtrage peut être effectué avec des procédures comme en KRL ou FRL, ou encore avec une facette spéciale comme en Shirka. En KRL, une procédure de filtrage permet de comparer deux unités ou de sélectionner, parmi un ensemble d'unités donné, celles qui satisfont certains critères [Bobrow and Winograd, 1977]. Un filtrage nécessitant une recherche dans plusieurs niveaux de la hiérarchie d'héritage est divisé en tâches, qui sont ordonnées dans un agenda. L'utilisateur peut éventuellement indiquer la profondeur maximale de recherche, la priorité ou encore la quantité de temps de calcul qu'il désire accorder à une tâche. Le filtrage en FRL n'a pas la puissance de celui de KRL et sert plutôt à déboguer les programmes. Une fonction spéciale permet de procéder à des mises en correspondance, qui doivent rester très simples [Roberts and Goldstein, 1977] : elle recherche les frames qui s'apparient à un modèle, dans lequel les éventuelles variables sont préfixées par un caractère ?.

En Shirka [Rechenmann, 1988], le filtrage permet de retrouver la valeur d'un attribut dont le co-domaine est défini par un objet du graphe d'héritage (cf. page 104 et figure 3.4). Il est effectué par l'intermédiaire de la facette `$sib-filtre`, qui constitue l'une des originalités du langage. Le mécanisme de filtrage est activé lors de l'accès en lecture. La priorité sur les facettes est alors redéfinie comme suit : `$valeur`, `$sib-filtre`, `$si-besoin`, `$defaut`. La facette `$sib-filtre` introduit un filtre qui est un frame particulier noté `FILTRE`, spécialisation d'un frame noté `SUPER`, dont les descendants et leurs représentants, c'est-à-dire l'ensemble associé à `SUPER` dans le graphe d'héritage, vont être filtrés. La structure de `FILTRE` est calquée sur celle de `SUPER`, `FILTRE` pouvant avoir des attributs supplémentaires, ou encore avoir des attributs hérités assortis de restrictions particulières. Au cours du filtrage, les représentants des frames appartenant à l'ensemble associé à `SUPER` sont comparés avec `FILTRE`, et les représentants qui s'apparient avec `FILTRE` sont collectés. Les possibilités du filtrage sont multiples : plusieurs filtres peuvent être emboîtés, de même que plusieurs filtres peuvent être associés à un attribut. Dans ce dernier cas, les filtres sont tous essayés en cascade, sauf si l'attribut est monovalué (facette `$un`) et qu'un des filtres a déjà délivré un résultat.

2.2 Un modèle de langage hybride

La complexité croissante des problèmes abordés en intelligence artificielle rend de plus en plus longues et difficiles la conception et la réalisation de systèmes modélisant des aspects du monde réel. La taille des programmes à mettre en œuvre et la diversité des données manipulées nécessitent l'emploi d'outils de génie logiciel, pour le débogage, la maintenance et le partage des ressources, d'outils pour représenter les connaissances, suivre et expliquer le raisonnement et d'interfaces graphiques, pour la gestion du dialogue homme-machine... Les langages *hybrides* sont conçus dans cette optique : mettre à la disposition du programmeur l'éventail des formalismes standard

de programmation et de représentation des connaissances, procédures, classes, frames, calcul des prédicats ou règles de production.

2.2.1 La cohabitation

Historiquement, il est possible de distinguer trois générations de systèmes à base de connaissances. Les premiers systèmes sont procéduraux, généralement écrits en LISP, le contrôle et les connaissances étant couplés dans un ensemble de fonctions.

La seconde génération, apparue dans les années soixante-dix avec les premiers systèmes experts, est fondée sur la séparation contrôle-méthodes d'inférences et connaissances. Depuis, les règles, quelle que soit leur forme (règles de production, clauses, etc.), sont devenues un formalisme standard et minimal de représentation. Parallèlement, les limitations de l'approche "règle" sont vite apparues : nécessité de structurer une base de règles dès qu'elle contient un nombre important de règles[10], difficulté de représenter des méta-connaissances ou des objets à structure complexe (comme des graphes par exemple). De plus, si le contrôle et les connaissances sont séparés, il subsiste encore un amalgame entre règles de contrôle et règles de description du domaine.

La troisième génération est issue du "modèle objet" des années quatre-vingts. Cette génération est celle des systèmes hybrides, ou encore "langages à objets dédiés à la représentation des connaissances", dans lesquels le modèle objet joue un rôle unificateur. Puisqu'il existe une infinité de façons de décrire une chose et une infinité de façons de la présenter ; puisqu'un formalisme de représentation unique offre autant d'avantages que d'inconvénients ; puisque la résolution d'un problème complexe fait généralement appel à divers types de raisonnement ; pourquoi ne pas utiliser plusieurs formalismes de représentation conjointement [Brachman, 1988] ? Un système hybride intègre donc plusieurs styles de représentation, objets et règles semblant être un noyau minimal, ainsi qu'un environnement de développement graphique. La puissance d'un tel système ne se mesure pas forcément au nombre de styles de représentation qu'il propose. Plus un système est sophistiqué, plus il devient complexe à comprendre, à développer et à utiliser. De plus, si une application se programme simplement avec des règles, l'utilisation d'un système hybride ne s'impose pas, elle est même déconseillée. En revanche, les différentes capacités de raisonnement offertes par un système hybride ont de plus en plus tendance à prendre de l'importance, en regard des nombreux problèmes de représentation non encore résolus [Brachman, 1990].

2.2.2 Système hybride et représentation à objets

Comme l'a mis en évidence la controverse déclaratif-procédural [Winograd, 1975], chaque formalisme de représentation est en fait bien adapté à la description de certains types de connaissances. Par exemple, les frames et les classes permettent de décrire des objets complexes et de les structurer ces objets en hiérarchies. Les formalismes issus du courant logique permettent, pour leur part, de décrire facilement des

10. Cinquante règles semble être un seuil à partir duquel il est nécessaire de définir des paquets pouvant être manipulés globalement.

faits et des assertions de type relationnel comme des règles de décision ou de diagnostic. Aucun formalisme de représentation n'est universel et ne possède simultanément toutes les qualités généralement souhaitées : expression naturelle et compréhensible des connaissances, recherche et manipulation efficace des informations et explication des raisonnements effectués [Fikes and Kehler, 1985]. Il est en particulier illusoire de prétendre qu'un formalisme est globalement meilleur qu'un autre, indépendamment de tout contexte [Niwa *et al.*, 1984]. Un "bon" langage de représentation peut se caractériser par sa capacité à fournir divers formalismes adaptés et/ou adaptables à des besoins divers. Un utilisateur peut alors choisir le formalisme qui lui semble idéal pour chacune des parties de son application.

Les langages hybrides sont à vocation multiple et permettent de répondre, au moins en partie, aux différents types de besoins évoqués précédemment, en intégrant plusieurs styles de programmation [Mettrey, 1987] [Tichy, 1987]. En règle générale, un langage hybride est d'abord un langage à objets (langage de classes ou langage de frames), construit à partir de LISP ou de PROLOG et dédié à la représentation des connaissances. Dans un cas, les classes sont enrichies de fonctionnalités empruntées aux frames, les variables d'instance des classes étant pourvues de facettes par exemple. Dans l'autre cas, les frames sont généralement dotés d'un comportement décrits par des méthodes. Puisqu'un langage hybride sert avant tout à représenter des connaissances hétérogènes, un moteur d'inférence gérant des clauses ou des règles de production est généralement intégré au langage. Dans le meilleur des cas, un utilisateur dispose de quatre styles de programmation[11] : programmation fonctionnelle, programmation objet, programmation dirigée par les accès et programmation logique. Une mention spéciale doit être faite au langage Common Lisp et à la couche objet CLOS qui lui est associée [Keene, 1989]. Les fonctionnalités de CLOS, qui dérivent pour la plupart de LOOPS [Bobrow and Stefik, 1983], CommonLoops [Bobrow *et al.*, 1986] et Flavors [Moon, 1986], placent le couple Common Lisp + CLOS comme un standard dans l'ensemble des langages hybrides de la "famille LISP", Common Lisp + CLOS se voulant être à l'intelligence artificielle ce que C++ est au génie logiciel. Toutefois, CLOS est un langage de classes et toutes les fonctionnalités des frames ne font pas partie des bibliothèques de base du langage [Veitch, 1991].

Un langage hybride et son environnement forment un tout appelé système hybride. Nous pouvons maintenant compléter la définition de la notion de *représentation à objets* donnée au chapitre 1 (cf. page 1.2.2) : une représentation à objets est un formalisme de représentation qui permet de décrire des connaissances sous la forme d'objets qui encapsulent données et procédures ; les objets sont organisés en une hiérarchie d'héritage ; à ce formalisme sont associés des outils qui gèrent les objets et la hiérarchie dont ils font partie, ainsi que des outils de filtrage et de classification. Une représentation à objets peut être implantée avec un langage de classes, un langage de frames ou avec un langage hybride, ce qui est pour nous le standard dans la suite. S'il s'avère nécessaire de distinguer les représentations à objets selon le type des objets

11. Nous ne parlerons pas dans cette étude de la programmation par contraintes [Guesguen *et al.*, 1987] [Guesguen, 1989], des combinaisons logique + contraintes [van Hentenryck, 1989] et logique + fonction [Aït-Kaci and Podelski, 1991].

qu'elles intègrent, nous dirons *représentation à classes* lorsque le langage sous-jacent est un langage de classes (CLOS par exemple), *représentation à frames* lorsque le langage de base est un langage de frames, (Shirka par exemple), et représentation à objets lorsqu'il n'y a pas lieu de différencier les deux types précédents.

2.2.3 L'environnement de programmation

L'environnement de programmation d'un système hybride comprend un ensemble d'outils qui permettent de créer, mettre au point, valider puis utiliser des systèmes à base de connaissances. Un tel environnement se compose idéalement de :

- une interface graphique multi-fenêtres pour visualiser la base de connaissances, en particulier le graphe d'héritage, les objets contenus dans ce graphe et les propriétés associées aux objets [Ducournau, 1989a],
- des outils pour maintenir la cohérence de la base de connaissances [Euzénat, 1990], pour visualiser les objets interconnectés et leurs relations, pour reconnaître et supprimer des connaissances redondantes ou obsolètes,
- des outils de mise au point et d'explication pour tracer une exécution, pour visualiser l'arbre des appels de fonctions, méthodes, réflexes, règles [Ducournau, 1989a],
- des outils pour gérer les différentes versions d'une base de connaissances et visualiser les transformations opérées sur les objets [Bobrow and Stefik, 1983] [Nguyen and Rieu, 1989].

L'environnement de développement et l'environnement d'utilisation d'un système sont généralement les mêmes, bien que les deux fonctionnalités ne nécessitent pas tout à fait les mêmes besoins [Mettrey, 1987]. L'environnement d'utilisation est plutôt dévolu à la gestion des ressources, temps de calcul, espace mémoire, etc., à la gestion du partage de la base de connaissances en cas de multi-utilisation. Il doit aussi faciliter l'accès aux fonctionnalités du système et assurer une portabilité maximale vers des applications externes écrites dans des langages différents ou vers des bases de données par exemple.

Notons, pour terminer, que la plupart des systèmes actuels ne sont pas utilisables directement par un expert d'un domaine d'application néophyte en programmation, qui désire construire un système à base de connaissances modélisant son domaine [Stefik *et al.*, 1983] [Kempf and Stelzner, 1987]. L'écriture des méthodes, réflexes et autres règles nécessite de savoir programmer, la plupart du temps en LISP ou en PROLOG, ce qui n'est pas à la portée d'un utilisateur inexpérimenté.

2.2.4 Un exemple

Dans la suite, nous nous intéressons plus particulièrement aux représentations à objets, où les objets sont des frames dotés de méthodes. La figure 2.11 montre ce que

```
(defmodele Fenetre
  (sorte-de
    ($valeur Objet-ideal))
  (origine
    ($un Point)
    ($min 1)
    ($max 1)
    ($lien-inverse region)
    ($si-ajout dessiner-cadre)
    ($si-enleve effacer-cadre))
  (coin
    ($un Point)
    ($defaut Bas-droit)
    ($si-possible inferieur-a-origine)
    ($si-enleve effacer-cadre))
  (hauteur
    ($un entier)
    ($si-possible nil)
    ($si-besoin +hauteur))
  (largeur
    ($un entier)
    ($si-possible nil)
    ($si-besoin +largeur))
  (etiquette
    ($un chaine)
    ($si-besoin demander)
    ($si-ajout afficher-etiquette))
  (ouvrir
    ($methode f-ouvrir))
  (activer
    ($methode f-activer))
  (deplacer
    ($methode f-deplacer))
  (fermer
    ($methode f-fermer)))
```

Figure 2.11. Description de l'objet `Fenetre`.

devient la définition de l'objet `Fenetre` (cf. Fig. 2.1), dans une telle représentation à objets. L'objet `Fenetre` est muni des méthodes `ouvrir` qui permet de créer une nouvelle fenêtre, `activer` qui rend la fenêtre active, `deplacer` qui sert à déplacer la fenêtre et `fermer` qui referme une fenêtre. Le nom de chaque méthode fait référence

à une fonction qui est appliquée à chaque représentant de l'objet `Fenetre` lors d'un envoi de message. La nouvelle description de l'objet `Fenetre` intègre un certain nombre de méthodes, mais les attributs déjà présents dans l'ancienne définition n'ont pas été modifiés. Cette nouvelle façon de décrire les objets de type `Fenetre` est plus naturelle et plus conforme au modèle objet : un objet réagit à la réception de message, il s'affiche, se déplace, se ferme, etc., il n'est plus nécessaire de passer par l'intermédiaire des attributs pour que certains comportements soient exécutés.

2.2.5 Méthodes et/ou si-besoin

La diversité des outils disponibles dans une représentation à objets impose des choix d'implantation qui ne sont pas toujours immédiats [Ducournau, 1989b] : dans quelles circonstances utiliser une fonction, une méthode, un réflexe ou une règle ? La méthode est l'unité de base modélisant une action s'appliquant à un objet. Toutefois, si la fonctionnalité décrite ne nécessite pas d'être différenciée selon les objets, aussi bien "horizontalement" que "verticalement", et si des critères liés à l'efficacité interviennent, il peut être judicieux d'implanter la fonctionnalité par une fonction[12], qui, à l'image d'une primitive, est valide pour l'ensemble des objets. Cela peut être le cas par exemple pour des fonctions qui effectuent des tris, seuls les opérateurs de comparaison étant alors détenus par les objets.

Si le choix entre méthode et fonction est relativement facile à déterminer, le choix entre méthode et réflexe si-besoin, dans le cadre du calcul de la valeur d'un attribut, doit être examiné avec plus d'attention. Voici quelques critères qui résument la plupart des situations pouvant survenir [Ducournau, 1989b] :

- Utilisation de données externes : lorsqu'un calcul nécessite des données externes, donc non accessibles directement dans l'objet concerné, il est alors préférable d'employer une méthode.

- Spécialisation des calculs : si la façon de calculer la valeur d'un attribut A dans un objet O est générale, notons-la M, et que dans les spécialisations de O, le calcul de A fait appel à M, alors il est souhaitable que M soit implantée par une méthode et que le calcul de la valeur de A dans les spécialisations de O fasse appel à M en tant que superméthode.

- Fréquences des accès et complexité du calcul : si la valeur d'un attribut est souvent demandée, que le calcul de cette valeur est complexe et qu'il n'est fait qu'une seule fois (pas de variations de la valeur dans le temps), alors il est souhaitable que le calcul soit effectué par un réflexe si-besoin et que la valeur soit écrite immédiatement.

- Maintien de cohérence : si la valeur d'un attribut est souvent modifiée, tout en dépendant de la valeur d'autres attributs qui évoluent en parallèle, alors une

12. Il faut remplacer fonction par clause si le langage sous-jacent est de type PROLOG, comme en OBJLOG par exemple.

méthode s'impose. Un tel calcul peut s'envisager avec un réflexe si-besoin lorsque deux attributs sont adjacents dans un même objet. Dans ce cas, il faut mettre en place des réflexes si-ajout et si-enlève pour maintenir la cohérence des valeurs interdépendantes.

Une véritable méthodologie de programmation avec des représentations à objets reste encore à développer. En tout état de cause, le programmeur a le plus grand intérêt à respecter la sémantique des outils disponibles et, en particulier, la sémantique de l'héritage, en évitant autant que possible de donner des noms identiques à des propriétés ayant des fonctionnalités différentes dans des objets différents, et en maîtrisant correctement le déclenchement des réflexes. La plupart des problèmes liés à la méthodologie de la programmation objet ont surtout été étudiés dans le cadre des langages de classes [Booch, 1990] [Rumbaugh *et al.*, 1991].

2.2.6 Règles et objets

Règles et objets sont deux formalismes de représentation complémentaires, mais leur cohabitation ne va pas sans poser quelques problèmes. Ainsi, comment se situent les règles par rapport aux objets, comment combiner l'envoi de message et le cycle classique d'un moteur d'inférences sélection-choix-déclenchement ? Dans la plupart des cas, les règles contiennent des variables qui font référence à des objets et à leurs attributs ; les conclusions des règles sont des actions qui peuvent être des envois de message, des accès à des attributs ou encore des références à d'autres règles ; l'ensemble des règles est structuré en une hiérarchie de paquets ou classes de règles, à l'image de l'ensemble des objets. Par ailleurs, les règles ou les paquets de règles peuvent être, comme une méthode, attachés à une classe, ou, comme un réflexe, attachés à un attribut, et sont ainsi déclenchés par envoi de message ou encore par accès à la valeur d'un attribut.

Parmi les différentes façons d'appréhender les règles dans un univers d'objets, il est possible de faire coexister côte à côte règles et objets, envoi de message et moteur d'inférences, ou alors, il est possible d'intégrer règles et objets en un seul formalisme, au niveau de la représentation et du contrôle.

La première approche est classique. Les règles et les objets ont des capacités de représentation duales, un formalisme compensant les faiblesses de l'autre. Les règles décrivent plutôt des heuristiques et des stratégies, les objets décrivent les entités référencées dans les règles [Allen. 1983] [Woods, 1983]. Le contrôle est centralisé et régi par un agenda, dans lequel les déclenchements de règles sont traités comme des tâches qui sont exécutées ou différées, selon le statut de la règle considérée [Aikins, 1983].

La seconde approche est moins classique. L'implantation des unités de représentation est unifiée : les règles sont organisées en paquets et elles sont représentées ou transformées sous forme d'objets, de méthodes ou encore de fonctions particulières [Ferber, 1989]. Définir des classes de règles permet d'organiser les règles en hiérarchie, ce qui est le point fort d'une telle approche, par opposition aux bases de règles non structurées. Lorsque de telles classes existent, des méthodes spéciales d'évaluation, équivalentes à de véritables moteurs d'inférences, peuvent leur être associées, ce qui

étend les possibilités d'activation, puisque chaque classe de règles peut avoir son propre moteur d'inférences [Fikes and Kehler, 1985] [Ferber, 1986] [Dugerdil, 1987b]. D'autres études sont faites actuellement sur la combinaison règles-objets selon cette approche [Franke, 1990] [Ibrahim and Cummins, 1990].

L'approche adoptée dans les systèmes à subsomption semble particulièrement prometteuse [Schild, 1989] [Yen *et al.*, 1991b] [Yen *et al.*, 1991a]. Une règle est vue comme un couple (condition,conclusion), où la partie condition est une conjonction de sous-conditions, exprimées par des rôles ou attributs, la partie conclusion regroupant un certain nombre d'actions devant être exécutées en cas de déclenchement de la règle. Deux programmes spécialisés permettent de gérer les règles, le premier est chargé d'insérer toute nouvelle règle dans la hiérarchie des règles, à l'image d'un classifieur (cf. 3.5), le second est chargé de sélectionner les règles candidates, en examinant la position des règles dans une hiérarchie spéciale de règles candidates. Cette approche est plus homogène que les précédentes, car les règles sont considérées et manipulées comme n'importe quel terme, les deux programmes principaux manipulant les règles étant équivalents au classifieur et au gestionnaire d'assertions pour les termes.

La cohabitation existe quelquefois là où on ne l'attend pas. Les fonctions génériques de CLOS [Keene, 1989] ont été utilisées dans CLASP, le système de règles de production associé au système à subsomption LOOM [MacGregor, 1988] (cf. 3.5.4). Un paquet de règles est implanté comme une fonction générique, dont les différentes méthodes correspondent à des règles spécifiques [Yen *et al.*, 1991a]. Le déclenchement d'une fonction générique nécessite d'ordonner les différentes méthodes associées à la fonction : de même, les règles d'un paquet sont ordonnées grâce à une relation de subsomption spécialement définie pour les règles.

2.3 Extensions

2.3.1 La réflexivité

Langages à objets et réflexivité

Un système est *réflexif* s'il possède une représentation de lui-même, appelée *auto-représentation*, avec laquelle il entretient un rapport de cause à effet [Maes, 1987]. Le comportement d'un système réflexif est donc toujours en accord avec son auto-représentation et réciproquement. Plus généralement, la capacité d'un système de représentation à raisonner sur lui-même est appelé *introspection* [Maes and Nardi, 1987] : tout système de représentation contenant des méta-connaissances est introspectif puisqu'elles lui permettent de raisonner sur ses propres connaissances.

Un système réflexif effectue deux types d'opérations : des opérations externes qui traitent des problèmes et des informations extérieurs, des opérations réflexives qui concernent le système lui-même et les actions qu'il effectue, comme pister et déboguer un calcul, visualiser les objets manipulés, etc., opérations qui pourraient être qualifiées d'internes. Le fait qu'un système soit réflexif n'a pas une influence directe sur les capacités de calcul du système, mais plutôt sur son organisation interne et sur

son interface avec l'extérieur, et garantit l'homogénéité et le bon fonctionnement du système [Ibrahim and Cummins, 1988].

Dans les langages à objets, le caractère réflexif s'est tout d'abord traduit par le fait qu'une classe contient une information décrivant la structure de ses instances [Cointe, 1988]. Puis, ce caractère a évolué dans le sens d'une uniformisation des concepts : *toutes les entités du langage sont des objets de première classe (elles peuvent recevoir des messages) et la seule action possible est l'envoi de message*, avec des langages comme Smalltalk-80 [Goldberg and Robson, 1983] [Briot and Cointe, 1989], ObjVlisp [Briot and Cointe, 1987] ou MERING [Ferber, 1989]. De même, pour favoriser l'imbrication des différents styles de programmation, le modèle d'un langage hybride est en général réflexif et uniforme. En particulier, toutes les entités utilisables, classes, méthodes, frames, attributs, réflexes, règles, sont représentées par des objets. Ces entités ont la même structure physique, elles peuvent donc communiquer facilement et être manipulées au moyen des mêmes structures de contrôle [Ibrahim and Cummins, 1990]. Ainsi, un réflexe peut activer un ensemble de règles en lui envoyant un message, une règle peut modifier la valeur d'un attribut, provoquant ainsi le déclenchement d'un réflexe, etc.. Cependant, ce type de réflexivité est le plus souvent limité : une classe peut contenir des informations portant à la fois sur le domaine d'application et sur le système lui-même.

La réflexivité telle qu'elle est conçue dans les langages 3-KRS [Maes, 1987] et KRS [Van Marcke, 1988] est plus rigoureuse et plus générale. La connaissance sur le domaine d'application est complètement séparée de l'auto-représentation du système. Tout objet possède un méta-objet qui détient de façon exhaustive les informations qui concernent l'objet et uniquement celles-là : implantation, héritage, instanciation, comportement, etc.. En particulier, la relation d'instanciation, instance-classe ou classe-métaclasse, n'est pas confondue avec celle qui lie l'objet et son méta-objet. Ainsi, en KRS, un concept de l'univers de référence est modélisé par un objet appelé aussi *concept*, dont le comportement et la structure sont décrits par un autre objet appelé *méta-concept*. Le système est homogène : tous les objets sont des concepts et tout concept est décrit par un méta-concept. La régression à l'infini est évitée grâce à une technique d'*évaluation paresseuse* inspirée du système ARLO [Haase, 1986] : un concept, donc un méta-concept, n'est créé que par nécessité. L'ensemble des méta-concepts forme l'auto-représentation du système. Les opérations appliquées à un concept sont de la responsabilité de son méta-concept, qui détient toutes les informations sur la structure et la façon de manipuler le concept. La gestion des calculs se fait grâce à des "tours" virtuellement infinies de méta-concepts, où chaque méta-concept est responsable du concept qu'il décrit [Wand and Friedman, 1987]. Un concept de bas niveau a un rapport direct avec le problème traité, tandis qu'un concept de niveau supérieur a un rapport avec les concepts de niveau inférieur auxquels il est lié causalement. L'implantation d'une tour virtuellement infinie de concepts est techniquement possible car, d'une part, une application spécifique n'utilise effectivement qu'un nombre fini de méta-concepts, d'autre part, un méta-concept n'est construit qu'en cas de besoin d'informations réflexives sur le concept qu'il décrit.

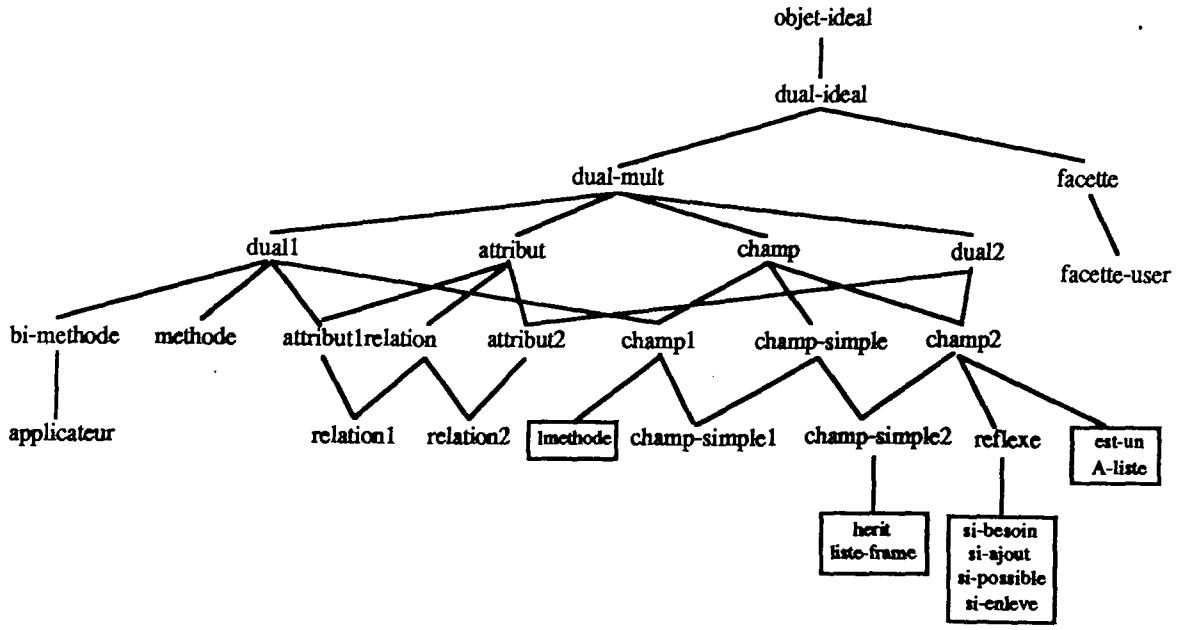


Figure 2.12. L'univers dual de YAFOOL.

L'univers dual de YAFOOL

Le langage de représentation YAFOOL est organisé en deux parties complémentaires appelées respectivement *noyau mou* et *noyau dur* [Ducournau, 1989b]. Le noyau mou définit la structure des objets de manière réflexive, indépendamment de leur implantation physique. Tout objet est constitué d'un ensemble de *slots* qui représentent aussi bien des attributs que des méthodes. Un *slot* est classiquement décrit par des facettes, qui ont une sémantique sensiblement équivalente à celle des facettes présentées au paragraphe 2.1.1. Attributs, facettes et méthodes sont des objets à part entière, décrits par des frames formant une hiérarchie d'héritage multiple de racine `dual-ideal`, descendant de l'objet le plus général, `objet-ideal` (Fig. 2.12). Le noyau dur, quant à lui, contient les primitives réalisant la manipulation physique des objets en fonction de leur structure logique, donnée par le noyau mou : gestion des mécanismes d'héritage, transmission de message, accès aux valeurs des attributs et déclenchement des réflexes.

Puisque les *slots* sont des objets, les opérations d'accès peuvent être définies comme des méthodes qui leur sont attachées. Ces opérations sont réalisées par des primitives de haut niveau appelées *applicateurs*. L'idée a été empruntée dans un premier temps à MERING [Ferber, 1983], puis généralisée : les applicateurs sont en réalité associés à des bi-méthodes car les opérations qu'ils effectuent dépendent à la fois des *slots* et des objets considérés. L'appel d'un applicateur correspond donc à un envoi de message où la sélection de la méthode à activer se fait grâce aux deux premiers arguments de l'appel, qui sont l'objet et le *slot* consultés. Par exemple, pour un objet O donné, l'effet de l'applicateur noté : : dépend de la nature du `slot` S : si S représente un sélecteur

de méthode, alors l'applicateur : : réalise un envoi de message dont le receveur est O, sinon, S représente un attribut et l'applicateur : : réalise un accès en lecture à la valeur de l'attribut S, qui est par défaut une lecture en Z avec activation éventuelle du réflexe si-besoin. Le principal intérêt des applicateurs est de permettre à l'utilisateur d'oublier les nombreuses primitives du noyau dur et de ne programmer qu'avec un ensemble limité d'instructions, ayant une syntaxe simple et uniforme.

La représentation des attributs

Un système réflexif est ouvert et extensible dans le sens où il peut être enrichi avec des utilitaires et des bibliothèques d'objets écrits dans le langage lui-même. Chaque utilisateur peut redéfinir les outils disponibles, et en créer de nouveaux selon ses besoins [Ibrahim and Cummins, 1988]. Le fait que toutes les entités du langage ait la même structure favorise la communication entre ces entités ainsi que leur manipulation : si tout est objet, une seule structure de contrôle suffit, en l'occurrence, l'envoi de message.

La figure 2.13 montre une représentation possible des attributs, inspirée du modèle `attribut` de YAFOOL. En disposant d'une telle représentation, il est possible de moduler le comportement des attributs en lecture et en écriture. Dans la suite, nous considérons trois caractéristiques modulables pour un attribut donné A, qui sont la gestion de l'historique des valeurs de A, le maintien de la cohérence des valeurs d'attributs adjacents à A et le mode de déclenchement des réflexes *a posteriori* associés à A.

La création d'un objet O provoque la création des objets qui représentent ses différentes parties, en l'occurrence ses attributs, ses méthodes et ses facettes. Ainsi, chaque attribut A de O possède une représentation qui est une spécialisation du modèle `Attribut` et qui est appelée *attribut-objet* A (l'attribut et l'attribut-objet portent le même nom) [Ferber, 1989]. La valeur d'un attribut A est modifiée chaque fois que A reçoit un message d'écriture, de sélecteur `ecrire`. Dans ce cas, la valeur de l'attribut `valeur` attachée à l'attribut-objet A, qui mémorise la valeur de l'attribut A, est modifiée. Les réflexes `+historique` se déclenchent alors pour compléter la valeur de l'attribut `historique` de l'attribut-objet A, qui conserve la trace de l'évolution de la valeur de l'attribut A. De la même façon, les attributs adjacents à A sont "prévenus" de toute modification dans la valeur de A par les réflexes *a posteriori* `prevenir-adjacents`, dont le rôle consiste à supprimer la valeur périmée des attributs adjacents, pour que leur nouvelle valeur soit effectivement calculée à la prochaine utilisation. Rappelons que la valeur d'un attribut adjacent à A est calculée par un réflexe si-besoin qui utilise la valeur de A. La mise en place de la valeur de l'attribut `adjacents` dans la représentation de A peut se faire soit explicitement par l'utilisateur, soit par une procédure qui analyse le code des réflexes si-besoin. Le mécanisme des dépendances en Smalltalk-80 fonctionne de façon similaire [Borning *et al.*, 1987] : un objet *O1*, qui dépend d'un objet *O2*, pour la valeur d'une variable d'instance par exemple, est informé de tout changement survenu dans l'objet *O2*, et *O1* est modifié s'il le faut.

Les attributs `fixants-si-ajout` et `fixants-si-enleve` mémorisent respectivement la liste des objets qui fixent dans la hiérarchie d'héritage un réflexe si-ajout

```
(defmodele Attribut
  (sorte-de
    ($valeur Objet-ideal))
  (valeur
    ($liste-de Objet-ideal)
    ($si-ajout (and (+historique) (prevenir-adjacents)))
    ($si-enleve (and (+historique) (prevenir-adjacents))))
  (historique
    ($liste-de Objet-ideal))
  (maximaux
    ($liste-de Objet-ideal))
  (adjacents
    ($liste-de Attribut))
  (fixants-si-ajout
    ($liste-de Objet-ideal)
    ($si-besoin (+fixants-si-ajout)))
  (mode-ajout
    ($liste-de Objet-ideal)
    ($defaut inverse-liste-de-priorite))
  (fixants-si-enleve
    ($liste-de Objet-ideal)
    ($si-besoin (+fixants-si-enleve)))
  (mode-enleve
    ($liste-de Objet-ideal)
    ($defaut liste-de-priorite))
  (lire
    ($methode lire))
  (ecrire
    ($methode ecrire))
  (ajouter
    ($methode ajouter))
  (supprimer
    ($methode supprimer)))
```

Figure 2.13. Une représentation simplifiée d'un modèle des attributs d'un objet.

et si-enlève pour l'attribut A. Pour redéfinir le mode de déclenchement des réflexes *a posteriori*, il faut modifier la valeur des attributs `mode-ajout` et `mode-enleve`, en indiquant explicitement la liste des objets et l'ordre dans lequel ils doivent être considérés lors du déclenchement des réflexes si-ajout et si-enlève. Par défaut, la valeur de `mode-ajout`, qui est associée au déclenchement des réflexes si-ajout, est égale à l'inverse de la liste de priorité, et celle de `mode-enleve`, qui régit le déclenchement des réflexes si-enlève, à la liste de priorité elle-même. Les valeurs de `mode-ajout` et `mode-enleve` peuvent alors être fournies par les listes réduites contenues dans `fixants-si-ajout` et `fixants-si-enleve` respectivement.

2.3.2 Les objets composites

La représentation d'objets composites

Un *objet composite* est formé par l'agrégation d'un ensemble d'objets, appelés ses *composants*, qui décrivent chacun une partie de l'objet composite. Les composants sont liés à l'objet composite par la relation *partie de*, l'objet composite à ses composants par la relation inverse *composé-de*. Bien que l'étude des objets composites semble être une préoccupation déjà ancienne dans le domaine des bases de données [Smith and Smith, 1977b] [Smith and Smith, 1977a] [Kim, 1990], seuls quelques langages à objets comme LOOPS [Stefik and Bobrow, 1986], YAFOOL et OBJLOG, offrent la possibilité de représenter explicitement de tels objets[13]. Un objet composite est alors considéré et manipulé globalement, comme un tout qui ne peut être dissocié de ses composants, des attributs spéciaux décrivant les composants et leurs connexions. Le système fournit des outils de création, de manipulation, de visualisation et de destruction de l'objet composite et de ses composants. En particulier, lors de la création d'un représentant de l'objet composite, tous les représentants des composants sont créés automatiquement, y compris les éventuels composants d'un composant, et les relations de composition entre objet composite et composants sont mises en place.

La figure 2.14 montre la représentation de l'objet composite `Fenetre`, défini à partir des objets `Ecran` et `Cadre`, une fenêtre étant composée d'un écran et du cadre qui l'entoure. La relation de composition est décrite par deux attributs, `compose-de` donne la liste ordonnée des composants et `lien-de-composition` répertorie les noms des liens sous lesquels sont connus les composants dans l'ordre de la liste des composants. Lorsqu'un représentant de l'objet composite `Fenetre` est créé, chaque composant est automatiquement créé et lié à l'objet dont il constitue une partie. Le principe de création s'applique récursivement aux composants qui sont eux-mêmes des objets composites. La création d'un représentant de l'objet composite `Fenetre` provoque donc la création d'un représentant de l'objet `Ecran`, celle d'un représentant de l'objet `Cadre`, ainsi que l'installation de liens de composition entre tous les représentants créés.

13. Il existe des extensions *ad hoc* comme celle réalisée pour Smalltalk-80 et présentée dans [Blake and Cook, 1987].

```
(defmodele Fenetre
  (sorte-de ($valeur Objet-ideal))
  (compose-de ($valeur Ecran Cadre))
  (lien-de-composition ($valeur l-ecran le-cadre)))

(defmodele Bordure
  (sorte-de ($valeur Objet-ideal))
  (compose-de ($valeur Bandeau Curseur))
  (lien-de-composition ($valeur le-bandeau le-curseur)))

(defmodele Curseur
  (sorte-de ($valeur Objet-ideal))
  (position ($un Point))
  (deplacer ($methode deplacer)))

(defmodele Fenetre-graphique
  (sorte-de ($valeur Fenetre))
  (compose-de ($valeur Bord-vg Bord-vd Bord-hb Bord-hh))
  (lien-de-composition ($valeur bvg bvd bhb bhh)))

(defmodele Fenetre-edition
  (sorte-de ($valeur Fenetre-graphique))
  (compose-de ($valeur Bordure-mixte))
  (lien-de-composition ($valeur bhb)))

(defmodele Bordure-mixte
  (sorte-de ($valeur Bordure))
  (compose-de ($valeur Zone-dialogue))
  (lien-de-composition ($valeur la-zone)))

(defmodele Fenetre-dessin
  (sorte-de ($valeur Fenetre-graphique))
  (compose-de ($valeur Palette-formes Palette-fonds))
  (lien-de-composition ($valeur bvg bhh)))

(defmodele Palette
  (sorte-de ($valeur Bordure))
  (compose-de ($valeur (10 Case)))
  (lien-de-composition ($valeur les-cases)))
```

Figure 2.14. Définition d'objets composites. A l'instar de YAFOOL, la relation de composition est décrite par les attributs `compose-de` et `lien-de-composition`. La partie composite de cette description de l'objet `Fenetre` peut très bien coexister avec la description donnée à la figure 2.11.

La spécialisation d'objets composites

Classiquement, un objet composite peut être spécialisé de deux façons, par spécialisation de composants déjà existants ou par adjonction de nouveaux composants. Ainsi, l'objet `Fenetre-graphique` est une spécialisation de l'objet `Fenetre`, à l'écran et au cadre s'ajoutent quatre objets de type `Bordure`, deux bords verticaux, un gauche (`bvg`) et un droit (`bvd`), deux bords horizontaux, un haut (`bhh`) et un bas (`bhb`). Les bords sont des spécialisations de l'objet composite `Bordure`, et la création de représentants de `Bordure` provoque la création de représentants des objets `Bandeau` et `Curseur`. Deux autres types de fenêtres particuliers ont été définis, `Fenetre-edition` et `Fenetre-dessin`. Dans le premier, le composant décrivant le bord horizontal bas, de type `Bord-hb`, est spécialisé en un objet de type `Bordure-mixte`. Dans le second cas, les composants décrivant les bords vertical gauche et horizontal haut sont spécialisés en objets de type `Palette`, ce dernier étant aussi un objet composite, puisqu'une palette est composée de 10 objets de type `Case`.

L'accès aux attributs d'un objet composite

Les liens de composition servent à nommer les connexions entre composite et composants, et permettent d'accéder aux différentes parties de l'objet composite en s'utilisant comme des attributs.

Supposons que, comme en YAFOOL, l'expression `(attribut objet)` retourne la valeur de l'attribut `attribut` pour l'objet `objet`. Dans ce cas, l'expression :

```
(position (le-curseur (bvg fenetre-edition-1)))
```

permet de connaître la position du curseur dans la bordure verticale gauche de la fenêtre d'édition `fenetre-edition-1`. La lecture de la valeur de l'attribut `position` emprunte un chemin qui passe par les liens de composition `bvg` et `le-curseur`, `bvg` permettant d'avoir accès à l'objet représentant la bordure gauche, `le-curseur` permettant d'avoir accès au curseur dans cette bordure.

Toutes les valeurs des liens de composition sont considérées comme des propriétés de l'objet composite et un objet composite partage les liens de composition de ses composants qui sont des objets composites[14]. En d'autres termes, l'objet `Fenetre-graphique` "connaît" les liens de compositions `le-bandeau` et `le-curseur`, comme s'ils faisaient partie de sa propre description. Pour faciliter l'écriture des accès aux attributs, il est alors possible d'oublier qu'un composant est lui-même un objet composite. Considérons par exemple les expressions suivantes :

```
(E1) (position (le-curseur (bvg fenetre-edition-1)))
(E2) (position (bvg fenetre-edition-1))
```

14. Ce principe a été qualifié de *programmation par métonymie* [Ducournau, 1989b]. Le petit Robert dit que la métonymie est un procédé de langage par lequel on exprime un concept au moyen d'un terme désignant un autre concept qui est uni au premier par une relation nécessaire : la cause pour l'effet, le contenant pour le contenu, le signe pour la chose signifiée, la partie pour le tout, etc..

```
(E3) (les-cases fenetre-dessin-2)
(E4) (les-cases (bvg fenetre-dessin-2))
(E5) (les-cases (bhh fenetre-dessin-2))
```

L'expression (E2) retourne la même valeur que l'expression (E1), la position du curseur dans la bordure verticale gauche de la fenêtre d'édition `fenetre-edition-1`. L'indication du lien `le-curseur` peut être omise, car il n'y a ici aucune ambiguïté. Toutefois, le lien `bvg` ne peut pas être omis, car il existe quatre composants de type `Bordure`, et sans information additionnelle, le système ne peut trancher en faveur de l'une ou l'autre bordure. Il se passe exactement le même phénomène avec l'expression (E3). Il faut alors résoudre des conflits de partage de propriétés qui sont du même ordre que les conflits d'héritage. En YAFOOL par exemple, la valeur retournée, lorsqu'apparaît un conflit de cette espèce, est la première valeur rencontrée en parcourant les liens de composition en profondeur d'abord. En l'occurrence, les expressions (E3) et (E4) retournent la même valeur, mais l'existence du conflit est signalée. Pour accéder aux cases de la bordure horizontale (`bhh`) de l'objet `fenetre-dessin-2`, il faut utiliser l'expression (E5).

Les liens de composition sont assimilables à des attributs, mais le pouvoir d'expression associé à la composition d'objets reste relativement limité. Lorsqu'un objet composite possède plusieurs exemplaires d'un même composant (une fenêtre est entourée de quatre bordures), il n'est en général pas possible de spécialiser les liens de composition en les différenciant, comme il est possible de différencier un rôle attaché à un concept en KL-ONE [Brachman and Schmolze, 1985]. En KL-ONE, une fenêtre pourrait posséder le rôle `bordure`, de cardinalité 4, qui, pour une fenêtre graphique, est spécialisé en un rôle `bordure-verticale` de cardinalité 2 et un rôle `bordure-horizontale` de cardinalité 2, chaque bordure horizontale et verticale pouvant être consultée séparément. Pour tenter de résoudre ce genre de problèmes, les objets composites de YAFOOL vont évoluer ; la figure 2.15 en montre quelques exemples.

Composition et partage de propriétés

Nous avons vu qu'un objet composite partageait certaines propriétés avec ses composants, comme les liens de composition de ses composants qui sont des objets composites. Ce partage de propriétés a une sémantique proche de celle de l'héritage ; il est d'ailleurs quelquefois appelé "héritage horizontal", par opposition à l'héritage entre un objet et ses spécialisations qui pourrait être qualifié de "vertical". Cependant, la relation qui traduit le partage de propriétés est "commutative" et n'est pas orientée comme l'est la relation d'héritage. Un composant partage des propriétés avec l'objet composite, la hauteur d'une bordure verticale est égale à la hauteur de la fenêtre bordée par exemple ; l'objet composite partage des propriétés avec l'un de ses composants, la position d'un texte dans une fenêtre d'édition est donnée par la position relative du curseur dans la bordure. Comme dans le cas de l'héritage, le partage de propriétés entre composants et composite peut conduire à des conflits analogues aux conflits d'héritage. Un premier type de conflit survient lorsque deux attributs de même nom mais de valeur différente existent dans deux composants : si une fenêtre est composée

```
(defmodele objet-composite
  (sorte-de ($valeur Objet-ideal))
  (composition (<lien-de-composition-1> <modele-1>)
               (<lien-de-composition-2> <modele-2>)
               ...))

(defmodele Fenetre
  (sorte-de ($valeur Objet-ideal))
  (composition (l-ecran Ecran)
               (le-cadre Cadre)))

(defmodele Fenetre-graphique
  (sorte-de ($valeur Fenetre))
  (composition (les-bordures (les-bh (bhb Bord-hb)
                                     (bhh Bord-hh))
                             (les-bv (bvg Bord-vg)
                                     (bvd Bord-vd)))))
```

Figure 2.15. Une nouvelle définition des objets composites en YAFOOL. La relation de composition est cette fois décrite par l'unique attribut `composition`.

d'un écran et d'un cadre qui ont chacun une couleur différente, quelle est la couleur de la fenêtre ? Un second type de conflit survient lorsque deux composants ont des composants de même nom : les bords vertical droit et horizontal haut d'une fenêtre de dessin sont chacun composés de cases, quelles sont les cases désignées par l'expression `(E3)` ?

L'implantation du partage de propriétés proposée dans les langages permettant de gérer des objets composites, YAFOOL et LOOPS entre autres, est restreinte : la recherche d'un lien de composition dans le graphe de la relation de composition se fait en profondeur d'abord, sans se préoccuper des conflits.

Un partage de propriétés plus strict, appelé *héritage sélectif*, existe en OBJLOG [Dugerdil, 1987a] [Dugerdil, 1991]. Un objet composite est une agrégation d'objets qui sont liés par une relation dite *structurelle*. Des objets en relation structurelle partagent des propriétés : il faut déclarer explicitement les propriétés partagées par l'intermédiaire de la relation structurelle. Aucun conflit n'est alors possible lors de la recherche des valeurs partagées, qui est effectuée grâce au mécanisme d'héritage sélectif. La figure 2.16 montre la représentation des objets composites `Fenetre` et `Bordure` en OBJLOG. L'aspect `statut`, équivalent d'une facette, associée à l'attribut `compose-de`, stipule que l'attribut décrit une relation structurelle ; l'aspect `heritage` spécifie que `Fenetre` partage la propriété `largeur` avec l'objet `Bas`, qui est de type `Bordure`. Ainsi, la valeur de `largeur` pour une instance de la classe `Fenetre` est

```
Fenetre
   sorte-de objet.nil
   <compose-de,nil>
      statut structurel
      domaine-ref <Ecran.Bas.Haut.Gauche.Droit.nil>
      heritage
         <<Bas,<largeur,nil>.nil>.nil>

Bordure
   sorte-de objet.nil
   <compose-de,nil>
      statut structurel
      domaine-ref <Bandeau.Curseur.nil>
   <largeur,nil>
      domaine <entier>

Bas
   sorte-de Bordure.nil
```

Figure 2.16. Les classes `Fenetre` et `Bas` sont en relation structurelle et elles partagent la propriété `largeur` (syntaxe OBJLOG).

recherchée dans l'instance de l'objet `Bas` qui lui est associé. Plus généralement, la valeur d'une propriété partagée est recherchée dans chaque objet partageant la propriété.

Nous présentons au chapitre 5 (cf. page 194) une façon non classique de gérer le partage de connaissances entre composants et objets composites.

Les nuances de la composition d'objet

La relation de composition ne doit pas être confondue avec celle d'héritage : une fenêtre graphique est composée de quatre bordures et d'un écran, mais l'objet `Fenetre` n'est pas une spécialisation des objets `Bordure` et `Ecran`, l'inverse n'étant pas vrai également. La plupart du temps, le comportement d'un objet composite ne se résume pas à la réunion des comportements associés à ses composants. En particulier, le déplacement d'une fenêtre ne doit pas être confondu avec le déplacement d'un curseur sur la bordure de cette fenêtre.

Les objets composites tels qu'ils ont été décrits sont construits à partir d'une relation prédéfinie *composé de*, de relation inverse *partie de*, qui peut être vue comme une relation d'ordre partiel :

- Elle est considéré comme non réflexive dans [Winston *et al.*, 1987] : l'objet A n'est

Relation	Exemple	Fonction	Equivalence	Séparabilité
composant-composite	poignée-tasse	+	-	+
membre-collection	arbre-forêt	-	-	+
portion-masse	grain-sel	-	+	+
matière-objet	cuir-veste	-	-	-
phase-processus	matin-journée	+	-	-
lieu-espace	oasis-désert	-	+	-

Figure 2.17. Les six grandes familles de relations de composition, d'après [Winston *et al.*, 1987]. Un "+" dans la rubrique **Fonction** indique que les parties sont dans une position spatiale ou temporelle spécifique qui est en relation avec leur fonctionnalité dans l'objet composite. Un "+" dans la rubrique **Equivalence** indique que les parties sont toutes de même type. Un "+" dans la rubrique **Séparabilité** indique que les parties sont physiquement séparables de l'objet composite.

pas composé de A. Pour notre part, nous la considérons comme étant réflexive (voir le paragraphe 3.4.1).

- Elle est anti-symétrique : si A a pour composant B, alors B n'a pas A pour composant.
- Elle est transitive : si A a pour composant B, et B a pour composant C, alors A a pour composant C.

La relation de composition possède de nombreuses nuances (cf. Fig. 2.17). Considérons par exemple la phrase suivante : *un chardonneret est un petit oiseau qui a des ailes ayant de jolies couleurs, qui niche souvent dans les arbres des forêts de la partie tempérée de l'hémisphère nord* . Cette phrase contient plusieurs exemples et contre-exemples de six grandes familles de relations de composition qui vont être examinées en détail.

La première famille fait référence à la forme de composition la plus courante, qui décrit le rapport d'un ensemble de parties à un tout. Cette relation est fonctionnelle, au sens où chaque partie a un rôle à jouer dans la composition, les parties ne sont pas toutes de même type, et elles ne sont pas séparables du tout sans que l'objet composite perde sa fonctionnalité. L'anse d'une tasse, les roues d'une voiture, les chapitres d'un livre, les pays du marché commun sont des exemples de ce type de composition. Il convient de distinguer "morceau" et partie, le premier n'étant pas fonctionnel, alors que la partie l'est : la roue d'une voiture, un morceau de la jante de cette roue.

Le second type de composition fait référence à l'appartenance d'un élément à une collection ou famille d'objets ou d'individus. Une classe regroupe un ensemble d'individus qui partagent un certain nombre de caractéristiques, une collection regroupe des individus qui sont "proches" dans le temps, dans l'espace ou encore dans la société. L'arbre fait partie de la forêt tandis que Georges fait partie du groupe communiste.

Le troisième type de composition fait référence à la relation portion-masse, ou sous-unité-unité, et sert de base aux calculs de conversion. Dans ce cas, toutes les parties sont identiques, comme les minutes d'une heure, les millimètres d'un mètre, etc..

Le quatrième type de composition fait référence à la relation matière-objet, et modélise le rapport qui existe entre un objet et ce dont il est fait, constituants et ingrédients. Le vin se fabrique à partir de raisin et contient de l'alcool. Le raisin est un ingrédient, car il est nécessaire à la fabrication du vin. L'alcool apparaît en cours de fabrication et il est un des constituants du vin, une des parties du produit final. Si un des constituants est supprimé, alors l'objet perd son identité, ainsi, H_2O sans H n'est plus de l'eau.

Les deux derniers types de composition font référence à la composition temporelle, les phases d'un processus, et spatiale, la situation géographique d'un endroit. Ainsi, les annonces font partie d'une partie de bridge, le matin, midi, l'après-midi et la soirée composent une journée, l'oasis est dans le désert, Crusnes est en Meurthe-et-Moselle. Une phase dans un processus et un endroit ne sont pas séparables du processus ou du lieu dont ils font partie.

Comme la relation de composition est souvent confondue avec la relation d'héritage, les six types de composition précédents sont souvent confondus avec les relations suivantes :

- Inclusion topologique : le vin est dans la bouteille.
- L'appartenance à une classe et la relation d'héritage.
- L'attribution : la couleur bleue de la mer.
- L'attachement : les boucles d'oreilles, la chaîne du chien.
- La possession : l'épouse de Barbe-bleue, l'argent de la vieille, le vélo de Poulidor.

La figure 2.18 récapitule ces relations sous la forme d'un arbre, qui montre les rapports existant entre les différentes nuances de la composition, les amies et les fausses amies.

Toutes les nuances de la composition sont des relations transitives, y compris les relations qui ne relèvent pas de la composition d'objets. Toutefois, "les amis de nos amis ne sont pas forcément nos amis", et la transitivité n'est pas toujours conservée lorsque des relations qui ne sont pas de même type sont combinées. Considérons par exemple les trois assertions suivantes :

(A1) Le bras d'Amedeo fait partie d'Amedeo.
(A2) Amedeo fait partie du CRIN.
(A3) Le bras d'Amedeo fait partie du CRIN.

La transitivité conduit à une inférence qui est relativement curieuse, sinon fausse, puisqu'un membre du CRIN ne peut être qu'une personne. En réalité la première relation de composition est de type partie-tout, la seconde de type membre-collection.

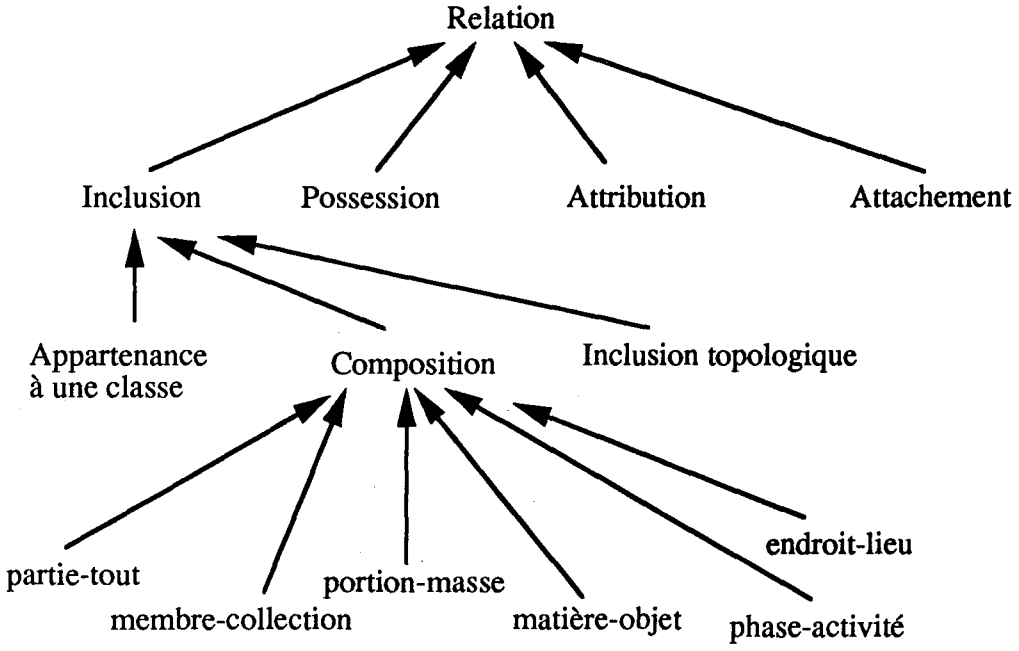


Figure 2.18. L'arbre des relations de composition, les relations de composition et relations "voisines", d'après [Winston *et al.*, 1987].

Comme les deux relations de composition sont différentes, la transitivité n'est pas forcément respectée. Dans [Winston *et al.*, 1987], il est suggéré qu'une conclusion résultant de la transitivité de la relation de composition n'est valide que lorsque la conclusion décrit la composition la plus faible suivant l'échelle d'importance descendante : appartenance à une classe - composition - inclusion spatiale.

(B1) Les ailes sont une des parties des oiseaux.
(B2) Les oiseaux sont des animaux volants.
(B3) Les ailes sont une des parties des animaux volants.
(B4) Les ailes sont des animaux volants.

(C1) La roue fait partie du vélo.
(C2) Le vélo est dans le garage.
(C3) La roue est dans le garage.
(C4) La roue fait partie du garage.

La conclusion (B3) est valide car elle est modélise la transitivité de la composition en (B1) et de l'appartenance à une classe en (B2), la composition étant dominée par l'appartenance à une classe. Pour la même raison, la conclusion (B4) n'est pas valide. De même, la conclusion (C3), qui exprime une relation spatiale, est valide, car elle correspond à la transitivité de l'inclusion spatiale (C2) avec la composition (C1), tandis que la conclusion (C4) est erronée pour la même raison.

Il est important de disposer d'outils capables d'exprimer les diverses nuances de

la relation de composition. Les méthodes "d'instanciation" de OWL II permettent de saisir, dans une certaine mesure, la richesse de quelques nuances de la relation de composition [Martin, 1979]. Nous proposons au paragraphe 3.4.1 une méthode générale de description de la composition d'objets, qui peut être affinée et étendue pour prendre en compte, lorsque c'est nécessaire, les nuances de la composition d'objets qui viennent d'être présentées.

2.3.3 Les perspectives

Classiquement, concevoir une hiérarchie d'objets consiste à définir d'abord des concepts généraux puis à les spécialiser pour définir les concepts plus spécifiques qui en dérivent, et ainsi de suite. Le partage de connaissances est géré par un mécanisme généralement procédural, qui est chargé de régler les conflits d'héritage multiple qui surviennent [Ducournau and Habib, 1991]. Pour un objet donné, il y a un conflit de nom si des propriétés ayant une sémantique différente portent le même nom et sont héritables ; il y a un conflit de valeur si la valeur d'une propriété est différente selon la façon dont elle est obtenue ; plusieurs chemins permettent d'hériter la propriété ou bien les valeurs héritables ne se masquent pas l'une l'autre [Ducournau, 1992]. S'il est possible de définir de façon déclarative des *perspectives* ou *points de vue* différents sur un objet, alors un certain nombre de conflits d'héritage, essentiellement des conflits de noms, peuvent être évités. Dans ce cas, deux propriétés de même nom peuvent modéliser deux points de vue différents d'une seule et même propriété, la force d'un homme du point de vue musculaire, moral, ou intellectuelle, son habileté du point de vue professionnel ou du point de vue ludique, etc.. Les perspectives peuvent être vues comme une généralisation des objets composites [Stefik and Bobrow, 1986] : un homme peut être considéré biologiquement, du point de vue de ses parties, et donc comme une entité composée d'une tête, d'un corps et de membres, mais aussi du point de vue de son travail, de ses activités extra-professionnelles, familiales, etc..

Les perspectives en TROPES

Le système TROPES propose une des meilleures formalisations qui soient pour construire des perspectives différentes qui représentent un concept, pour gérer les communications et le partage de propriétés entre les perspectives [Mariño, 1989] [Mariño *et al.*, 1990]. Les objets de TROPES sont construits sur les mêmes bases que ceux de Shirka [Rechenmann, 1988] (cf. page 104). Le graphe d'héritage regroupe un ensemble de concepts qui forment une partition en classes (Shirka) de l'univers considéré. Au découpage du graphe en concepts disjoints s'ajoute le "découpage" d'un concept en un ensemble de perspectives qui modélisent chacune un point de vue sur le concept. Chaque perspective définit un "plan" et l'ensemble des perspectives pour un concept donné définit un ensemble de plans parallèles (Fig. 2.19).

La description des concepts du domaine étudié s'appuie sur l'hypothèse suivante : des classes d'objets organisées dans un graphe d'héritage sont non disjointes lorsqu'il est nécessaire de modéliser des vues ou perspectives différentes d'un même concept,

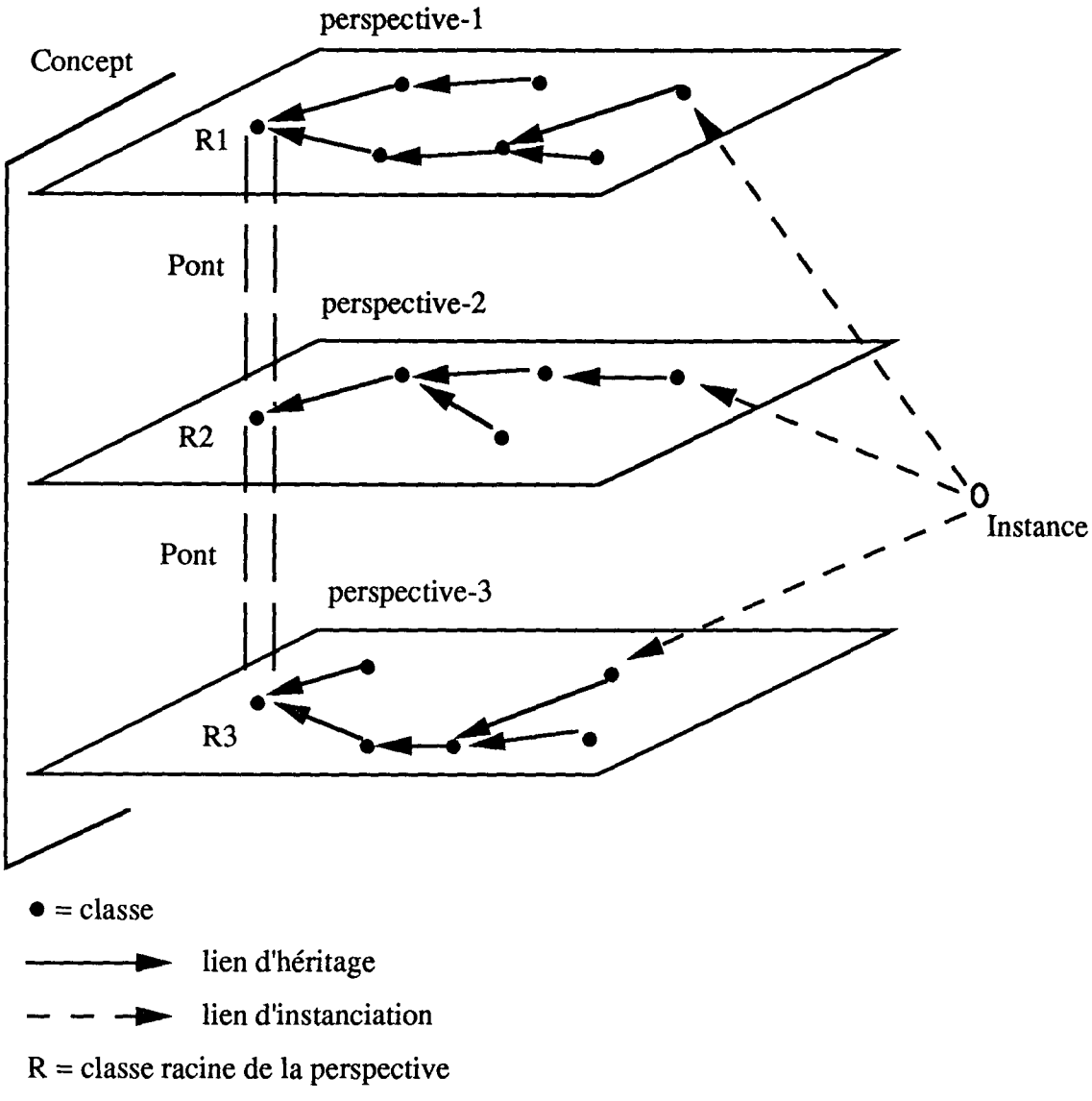


Figure 2.19. Perspectives et ponts en TROPES.

l'héritage multiple étant le seul mécanisme disponible. En TROPES au contraire, un concept est représenté par un ensemble de perspectives indépendantes, chaque perspective étant représentée par un arbre d'héritage de classes. Dans la suite, lorsqu'il n'y a aucune ambiguïté possible, nous assimilons perspective et arbre des classes définissant la perspective, concept et ensemble des perspectives décrivant le concept.

La racine de chaque perspective est une classe qui porte le même nom que le concept, et qui décrit tous les individus recouverts par le concept ou extension du concept. Si chaque racine de perspective a la même extension, son intension se compose d'une intension commune à toutes les racines, appelée *clé* du concept, et d'une intension

propre à la perspective considérée, indépendante des autres perspectives.

Pour un concept donné, donc pour un ensemble de perspectives, un attribut possède une sémantique unique : un attribut se rencontrant dans deux perspectives différentes doit avoir le même sens dans les deux perspectives, entre autre le même type, et il doit respecter les contraintes définies dans chaque perspective. Cette contrainte sur les attributs est très forte, et peut apparaître comme une limitation, en particulier par rapport aux perspectives de KRL, où un attribut peut avoir un type différent dans des perspectives différentes [Bobrow and Winograd, 1977]. Toutefois, deux attributs TROPES de même nom, mais définis dans des concepts différents, ont une portée donc une sémantique différente.

Pour une perspective donnée, un individu ou instance ne peut dériver que d'une seule classe, mais pour un concept donné, un individu peut dériver de plusieurs classes appartenant à des perspectives différentes. Ainsi, l'instanciation est "unique" par rapport à une perspective, mais elle est "multiple" par rapport au concept, donc à l'ensemble des perspectives. Puisque le concept auquel se rattache une instance est unique, et puisqu'un attribut a une sémantique unique pour un concept donné, aucun conflit de nom ne peut survenir.

Deux classes appartenant à deux perspectives différentes peuvent être en relation si elles ont la même extension. L'ensemble des classes qui sont en relation est appelé un *pont*. Les ponts sont "orthogonaux" à la dimension de l'héritage dans chaque perspective, mais la spécialisation des ponts suit celle des classes de chaque perspective (cf. Fig. 2.19). Puisqu'un attribut apparaissant dans deux classes d'un pont a la même sémantique dans chacune des classes, un individu est membre d'un pont s'il satisfait les contraintes associées à une seule des classes du pont. Par extension, si un individu est reconnu comme étant une instance d'une des classes d'un pont, il est considéré comme une instance des autres classes du pont, et des informations supplémentaires provenant de ces autres classes peuvent être inférées. L'apport des ponts relève donc plutôt du niveau du raisonnement par classification, qui permet de classer une instance en puisant de l'information dans les perspectives dont dépend l'instance. Les règles de classification d'une instance sont plus complètement expliquées au chapitre suivant, où une mention spéciale est faite sur le raisonnement par classification dans le cadre des représentations multi-perspectives (cf. page 107).

Autres approches

Les travaux traitant des perspectives restent peu nombreux. En KRL, les unités `Basic` partagent un univers en classes disjointes, tandis qu'une unité `Individual` décrit des individus qui dérivent d'une unité `Basic` unique, mais qui peuvent être vus selon différents points de vue exprimés par des unités de type `Specialization` (unités qui spécialisent des unités de type `Basic`), ou `Abstract` (unités qui sont des réservoirs d'information à la manière des classes abstraites) [Bobrow and Winograd, 1977]. Un individu donné peut posséder plusieurs attributs de même nom mais de sémantique différente, si ces attributs appartiennent à des perspectives différentes. L'âge d'un individu est un entier selon un certain point de vue et un symbole selon un autre point

de vue par exemple. C'est une des différences notables qui existe entre l'approche KRL et celle de TROPES.

Le système de traitement du langage naturel OWL [Martin, 1979] et le système de gestion de fichiers et de développement de programmes PIE [Goldstein and Bobrow, 1980] implantent des mécanismes de perspectives qui ont une optique proche de celle de KRL. Les perspectives de LOOPS, décrites dans [Stefik and Bobrow, 1986], semblent, comme d'autres caractéristiques du langage d'ailleurs, n'en être restées qu'au stade des spécifications. Il n'en est pas de même du système de représentation multi-perspectives VIEWS [Davis, 1987], première implantation complète du genre, avec lequel TROPES partage certaines idées.

Dans le langage ROME [Carré, 1989] [Carré and Geib, 1990], une instance dérive d'une classe unique dite classe d'instanciation mais peut partager des données avec plusieurs autres classes dites classes de représentation. Le point de vue d'une classe C sur un objet O est constitué par l'ensemble des classes de la hiérarchie de C, sous-classes et superclasses de C, qui sont des superclasses de la classe d'instanciation de O. Il est alors possible de rechercher des informations concernant une instance selon un certain point de vue. Le modèle de perspectives existant dans ROME est cependant plus limité que celui de TROPES, car il ne permet pas de décrire le parallélisme de perspectives et le mécanisme des ponts associés existant en TROPES.

Perspectives et conflits d'héritage

Le mécanisme des perspectives est puissant, mais il ne suffit pas à éliminer le recours à l'héritage multiple et à résoudre l'ensemble des conflits pouvant survenir. Considérons par exemple le graphe d'héritage multiple donné dans [Masini *et al.*, 1989] qui décrit un stock d'articles (cf. Fig. 2.20). En réalité, les articles peuvent se voir sous trois points de vue, celui de l'alimentation électrique, celui du produit de consommation, celui du transport, ce dernier point de vue se divisant lui-même en deux points de vue, objets fragiles et objets périssables pour le transport. Supposons que soient définies les perspectives `article-fragile` et `article-perissable`, et qu'il existe deux façons différentes, mais non incompatibles, de calculer le prix du transport d'un article, l'une pour les articles fragiles, l'autre pour les articles périssables. Les œufs étant des articles fragiles et périssables, comment doit-on fixer le prix de la boîte de six œufs ? Le résultat est différent selon que les œufs sont considérés comme des entités dérivant de `article-fragile` ou de `article-perissable`. Le mécanisme des perspectives ne permet pas, à lui seul, de régler ce conflit de valeur, et il faut avoir recours à une autre méthode, éventuellement de type algorithmique.

2.3.4 Les objets temporels

La formalisation du temps en intelligence artificielle a commencé dans les années soixante-dix. Les travaux réalisés ont porté sur la granularité du temps, sur les notions de causalité, de séquentialité, de durée et de continuité des intervalles de temps, sur la position d'intervalles de temps ou d'actions les uns par rapport aux autres [Allen,

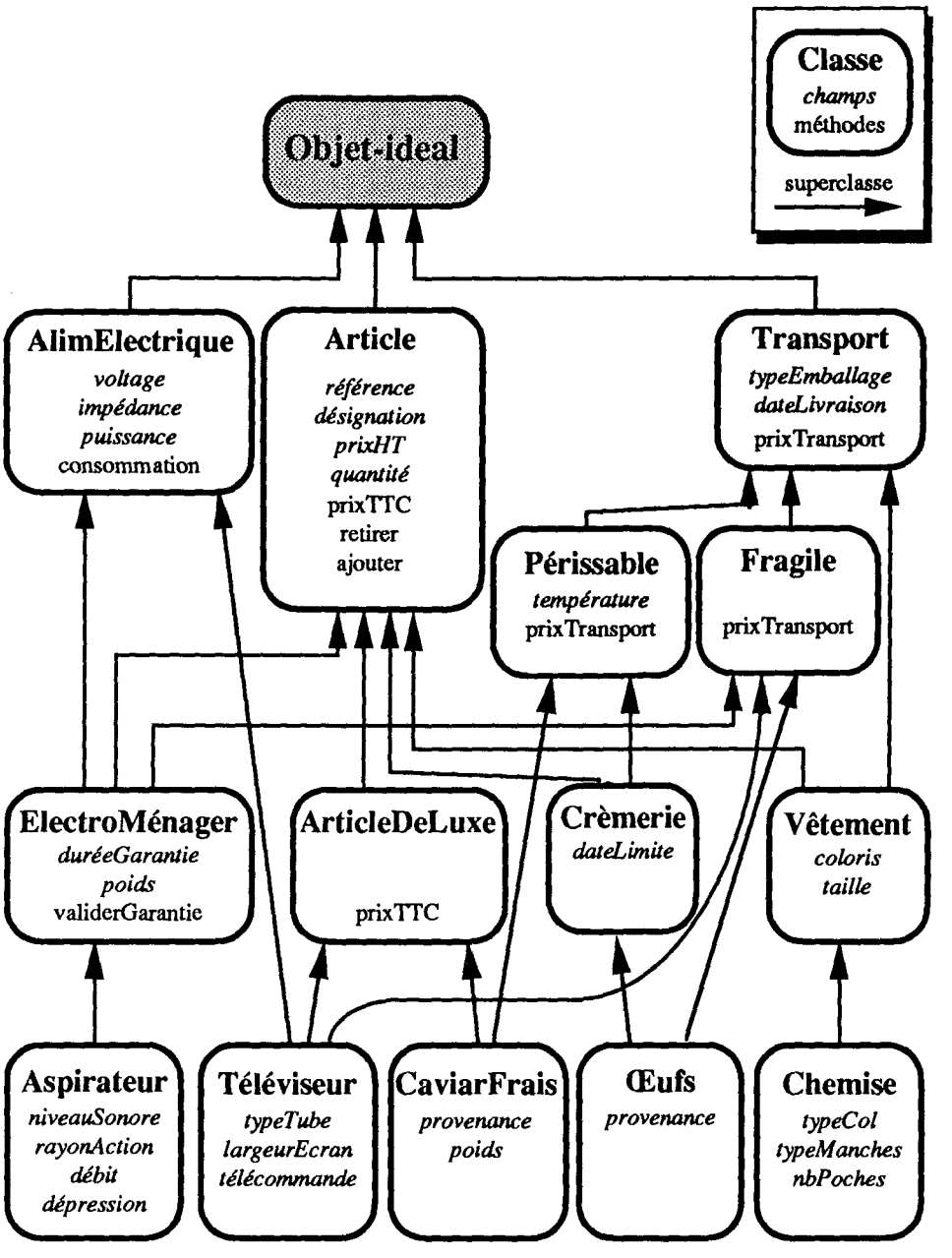


Figure 2.20. Un stock d'articles divers et variés, d'après [Masini *et al.*, 1989].

1984] [Audureau *et al.*, 1987] [Dean and McDermott, 1987] [Shoham, 1987] [Segev and Shoshani, 1987]. Le raisonnement a été essentiellement formalisé par des logiques temporelles permettant de gérer des actions et des plans d'actions temporels, ainsi que les relations temporelles existant entre des objets qui évoluent [Bestougeff and Ligozat, 1989]. Dans certains contextes, les relations sont considérées comme des objets, comme dans SRL [Sathi *et al.*, 1985] par exemple, mais les objets n'ont que rarement une

"facette" temporelle, si ce n'est en ART[15] [Clayton, 1984] et en YAFOOL [Ducournau, 1989b]. La formalisation des objets temporels que nous allons présenter est différente de celle des objets temporels de YAFOOL dont elle s'inspire. Des exemples détaillés des deux formalisations apparaissent au chapitre 5 (cf. § 5.4.2 et § 5.5.2).

Le mécanisme des *objets temporels* sert à décrire l'évolution des caractéristiques d'un objet en fonction des événements qui lui arrivent. Ces événements sont traduits par des modifications dynamiques apportées aux propriétés de l'objet à un instant donné. Le temps est pris en compte comme un paramètre associé aux événements qui décrivent l'évolution de l'univers [Delpech *et al.*, 1987] ; il est supposé évoluer de façon discrète et linéaire de 0 à l'infini, l'unité de temps étant quelconque, fraction de seconde, seconde, minute, heure, etc..

Tout objet temporel dépend d'un objet *atemporel*, ou objet non concerné par le temps, dont il hérite les caractéristiques. L'objet temporel partage certains attributs atemporels avec son ascendant non temporel, et fixe certains *attributs temporels*, attributs éventuellement hérités, donc redéfinis. La valeur d'un attribut temporel est constituée d'une suite de couples *(temps valeur)*. La suite des temps est lacunaire et ne contient que les temps où il y a modification effective de la valeur, puisque la valeur d'un attribut temporel peut être constante sur un intervalle de temps. Lorsqu'un objet varie au cours du temps, ce sont en réalité ses propriétés et leurs valeurs qui évoluent, et pour une propriété ne figurent que les instants où elle est modifiée.

Il est possible d'avoir un point de vue atemporel et un point de vue temporel sur une propriété, puisqu'une propriété atemporelle peut être redéfinie en une propriété temporelle. Un attribut représentant une telle propriété est dit *mixte*. L'existence de tels attributs autorise un partage maximal d'information entre objets temporels et atemporels. Il n'est pas nécessaire de créer, comme en YAFOOL par exemple, un nouvel objet à chaque pas de temps : à l'origine, un seul objet est créé, puis les propriétés de cet objet évoluent au cours du temps. La représentation du temps est donc associée en priorité aux propriétés plutôt qu'aux objets, favorisant ainsi le partage de connaissances.

Eléments d'implantation

Puisque la valeur d'un attribut temporel est une suite de couples *(temps valeur)*, les spécifications des accès en lecture et en écriture à un tel attribut sont redéfinies. Lors d'un accès en écriture, les données sont l'objet et l'attribut consultés, la valeur à écrire, et l'instant d'écriture. La donnée de l'instant d'écriture, qui est égale à un nombre entier d'unités de temps, est optionnelle. Par défaut, l'instant d'écriture est égal au temps courant, qui existe toujours, et qui est mémorisé par la variable globale `temps*`. Les accès en écriture sont régis précisément par les règles suivantes :

- Un accès en écriture est une adjonction ou une suppression de valeur. Les différents instants d'accès en écriture sont associés à autant d'événements concernant

15. Les points de vue (*viewpoints*) de ART permettent de représenter les changements d'états d'un processus temporel.

l'attribut et sont ordonnés de façon décroissante. Une valeur temporelle est donc une pile de couples *(temps valeur)*, et tout nouvel accès en écriture est fait au sommet de la pile constituant la valeur de l'attribut temporel.

- Les accès au passé sont contrôlés : comme dans la vie, il est interdit de modifier une valeur temporelle dans le passé, mais il est toujours possible d'aller la consulter...

- Le mode de déclenchement des réflexes si-possible, si-ajout et si-enlève est inchangé.

L'accès en lecture à la valeur d'un attribut s'appuie sur le principe suivant : les valeurs temporelles ont priorité sur les valeurs atemporelles ; une valeur devient temporelle dès qu'il existe un instant t où elle est modifiée. Lors d'une lecture, les données sont l'objet et l'attribut consultés, et éventuellement l'instant d'accès. Par défaut, l'instant de lecture est celui qui est donné par `temps*`, associé au dernier événement en date. Les modes de lecture standard I, N et Z restent inchangés, mais les spécifications suivantes leurs sont ajoutées :

- Les valeurs temporelles ont priorité sur les valeurs atemporelles pour un attribut mixte. Tant qu'aucun événement n'est survenu, la valeur d'un attribut mixte est par défaut sa valeur atemporelle (associée à l'objet atemporel dont dérive l'objet temporel considéré).

- Un accès est *au présent* lorsque le temps n'est pas indiqué. La valeur retournée est alors celle qui est associée au dernier événement concernant l'attribut, donc celle qui est associée au premier couple *(temps valeur)*, même si *temps* n'est pas égal à `temps*`.

- Un accès est *au passé* si le temps donné est inférieur à `temps*`. Si l'instant indiqué lors d'un accès au passé n'existe pas, aucun événement n'est venu troubler l'attribut à cet instant là, la valeur retournée est celle qui est associée au temps immédiatement inférieur. La valeur d'un attribut est supposée constante sur les intervalles de temps qui séparent chaque événement concernant l'attribut : si un attribut A est créé et initialisé au temps $t1$, et qu'il est modifié au temps $t2 \geq t1$, alors la valeur de A au temps $t \in [t1\ t2]$ est la valeur associée au temps $t1$.

- Un accès est *au futur* si le temps donné est supérieur à `temps*`. La valeur retournée est celle qui est associée au dernier événement.

- Afin que les valeurs calculées soient toujours cohérentes, qu'elles dépendent ou non d'autres valeurs, les réflexes si-besoin sont systématiquement déclenchés. Si l'écriture de la valeur produite par un si-besoin est systématique, l'écriture temporelle n'a lieu que si la valeur calculée est différente de la valeur au temps précédent.

Une autre formalisation des objets temporels, proche de celle qui vient d'être présentée, est concevable : tous les objets et toutes leurs propriétés sont temporels. Dans

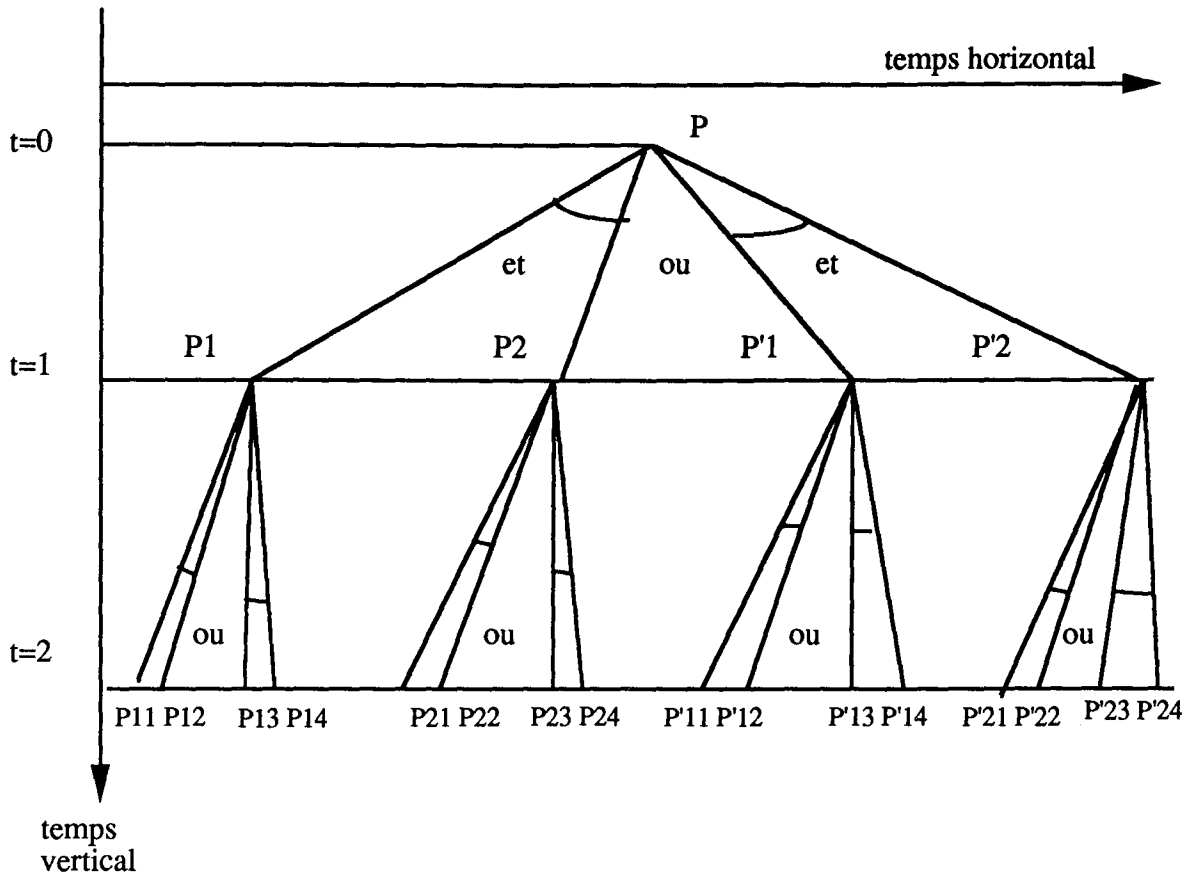


Figure 2.21. Perspectives temporelles.

ce cas, un attribut dont la valeur est permanente, et donc atemporelle, se voit associé le temps métaphorique `permanent*`. Les autres spécifications restent identiques aux précédentes.

Les perspectives temporelles ou arbres et-ou temporels

Un processus temporel peut, à l'image de n'importe quel concept, être considéré selon plusieurs perspectives temporelles. Supposons que l'évolution d'un processus temporel P soit modélisée par un arbre, où la profondeur associée à un niveau de l'arbre mesure le temps, et les nœuds associés à un niveau décrivent l'état du processus (cf. Fig. 2.21). Le processus P se divise au temps 1 en deux processus fils, *P1* et *P2*, qui eux-mêmes se divisent au temps 2 en quatre processus fils, respectivement *P11* et *P12* pour *P1*, *P21* et *P22* pour *P2*. Cet "arbre temporel" décrit la marche du processus P dans le temps. Chaque nœud de l'arbre est un nœud *et*, le temps n'étant pris en compte que selon une dimension unique.

Supposons maintenant qu'il existe une alternative (*P1'*,*P2'*) à l'évolution de P en deux sous-processus (*P1*,*P2*). Pour modéliser l'alternative, il est nécessaire d'insérer

dans l'arbre temporel des nœuds *ou*. Pour représenter le fait que *P1'* est une alternative de *P1* et que *P2'* en est une de *P2*, nous définissons une nouvelle dimension temporelle "orthogonale" à la dimension temporelle précédente : la première dimension temporelle, ou temps *vertical*, décrit la marche normale du temps et permet de modéliser le cours d'un processus et sa durée de vie ; la seconde dimension temporelle, ou temps *horizontal*, décrit des alternatives temporelles ou faits simultanés, en définissant un découpage à l'intérieur d'une unité de temps donnée. Par extension, le temps horizontal permet de décrire plusieurs actions qui se déroulent en parallèle au cours d'une même unité de temps verticale, ce qui, en toute généralité, revient à représenter un nœud *ou* dans un arbre qui matérialise l'espace d'états d'un problème. L'ensemble des nœuds *ou* associés à un temps vertical donné constitue un ensemble de *perspectives temporelles*, ou encore un *feuilletage temporel*, comparable aux perspectives de TROPES (cf. § 2.3.3).

Supposons par exemple que le processus `P` modélise l'évolution des valeurs d'un attribut `A` attaché à un objet `O`. L'objet `O` est donc un objet temporel au sens du paragraphe précédent. Voici trois descriptions formelles de `O`, la première n'est pas temporelle, la seconde n'utilise pas les perspectives temporelles, qui en revanche sont employées dans la troisième :

```
(Objet O
   (A ($valeur P)))

(Objet O
   (A ($valeur (2 P11 P12 P21 P22)
               (1 P1 P2)
               (0 P))))

(Objet O
   (A ($valeur (2 (1 P11 P12 P21 P22)
                  (2 P11' P12' P21' P22'))
               (1 (1 P1 P2)
                  (2 P1' P2'))
               (0 P))))
```

Les spécifications établies pour les objets et les attributs temporels dans le paragraphe précédent doivent être étendues au cas des perspectives temporelles. Les perspectives temporelles généralisent la notion de propriété temporelle : une valeur temporelle n'est plus constituée d'une simple liste de couples *(temps valeur)*, mais d'une liste de telles listes. Plus précisément, à un temps vertical *temps-v* donné, correspond une liste de couples *(temps-v-i valeur-i)*, où les *temps-v-i* constituent les éléments de l'ensemble des temps horizontaux associé au temps vertical *temps-v*. Les temps horizontaux et verticaux sont comme précédemment rangés dans l'ordre décroissant. Pour reprendre l'exemple ci-dessus, le processus `P` possède deux points de vue temporels au temps 1, le couple `(P1 P2)` qui est associé au temps "métaphorique" `1.1`, et le couple `(P1' P2')` qui lui est associé au temps `1.2`. Les processus `P1` et `P1'`, ou `P2` et `P2'`, sont des alternatives. La méthode se généralise naturellement à un nombre quelconque

de perspectives. Un exemple de création et d'utilisation de perspectives temporelles est détaillé au chapitre 5 (cf. § 5.4.2).

2.4 Notes et histoires

Premier formalisme de représentation, et précurseurs dans bien des domaines, les réseaux sémantiques intègrent déjà les notions de partage de connaissances, d'héritage et les techniques de filtrage [Quillian, 1968]. Les graphes conceptuels de J.F. Sowa peuvent être considérés comme une formalisation des réseaux sémantiques précise et fortement orientée vers l'étude du langage naturel [Sowa, 1984]. Pour élargir et structurer le champ d'information lié aux nœuds d'un réseau sémantique, Marvin Minsky introduit la notion de frame [Minsky, 1975], représentant typique ou prototype d'une catégorie d'objets. Les premiers véritables langages de frames sont KRL [Bobrow and Winograd, 1977], FRL [Goldstein and Roberts, 1977], OWL [Martin, 1977], et UNITS [Stefik, 1979]. Ils ont servi de modèles à toute une génération de langages de représentation dont SRL [Fox *et al.*, 1986], commercialisé ensuite sous le nom de CRL, Shirka et ses satellites [Rechenmann, 1988], et KRS [Van Marcke, 1988] sont parmi les plus originaux. La très importante famille des systèmes à subsomption, qui dérivent tous de KL-ONE [Brachman and Schmolze, 1985], est présentée au chapitre suivant.

L'influence des langages de frames se retrouve actuellement dans la plupart des langages à objets dédiés à la représentation des connaissances, et a contribué à l'émergence du concept de représentation à objets. Premier de la lignée des langages hybrides, LOOPS est l'ancêtre des représentations à objets [Bobrow and Stefik, 1983], aux côtés des plus "industriels" ART [Clayton, 1984] et KEE [Fikes and Kehler, 1985]. Les français ne se sont pas si mal débrouillés puisqu'ils ont produit côté LISP, KOOL [Albert, 1988], MERING et ses différentes versions [Ferber, 1983] [Ferber, 1989], LORE [Caseau, 1987] (devenu SPOKE en France et LAURE en exil [Caseau, 1989]), LRO2 [Roche and Laurent, 1989] (devenu KEOPS), et YAFOOL [Ducournau, 1989b] (qui est resté YAFOOL), entre autres, et OBJLOG [Dugerdil, 1988] côté Prolog. Evidemment, de nombreux autres langages de frames et langages hybrides existent, mais cette conclusion ne se veut qu'une courte note historique et panoramique.

3

Le raisonnement dans une représentation à objets

Dans ce chapitre, nous nous plaçons dans le cadre des représentations à objets, où les objets sont des frames pourvus de méthodes. Dans une représentation hiérarchique classique, l'héritage est généralement le principal mécanisme d'inférence [Fox, 1979] : la valeur associée à une propriété pour une entité donnée se déduit en examinant les ascendants de l'entité dans la hiérarchie. L'héritage est surtout un mécanisme de partage de connaissances et de code. Il permet de produire de nouvelles connaissances à partir de connaissances déjà présentes, dans une certaine limite de complexité, et ne peut donc pas être l'unique mécanisme d'inférence. A l'inverse, la subsomption, relation sur laquelle repose l'arrangement des concepts dans les systèmes à subsomption [Patel-Schneider, 1991], se révèle être un mécanisme d'inférence puissant qui pallie certaines des limitations de l'héritage. Ce chapitre décrit l'association objets-subsomption, qui consiste à introduire dans un univers d'objets classique un formalisme et des outils semblables à ceux qui sont employés dans les systèmes à subsomption, pour procéder à des inférences et à un raisonnement par classification. Une telle association permet de profiter conjointement de l'héritage pour la gestion du partage de connaissances, et de la subsomption et du raisonnement par classification pour la résolution de problèmes. L'association se généralise en considérant simultanément plusieurs relations d'ordre partiel, ce qui permet d'envisager une base de connaissances comme un espace n-dimensionnel.

3.1 Quelques limitations de l'héritage

3.1.1 Représentations hiérarchiques

Il est important de distinguer graphe d'héritage et représentation hiérarchique[1]. Dans une représentation hiérarchique, les connaissances sont ordonnées grâce à une relation d'ordre partiel qui n'est pas forcément la relation d'héritage ; l'accent est mis sur l'arrangement des objets en une hiérarchie qui reflète les régularités et les relations

1. A la célèbre mise en garde de R.J. Brachman *isa isn't inheritance* dans [Brachman, 1983], nous ajoutons *isa is any partial order*.

de généralité qui existent entre les objets. En ce sens, un graphe d'héritage est un cas particulier de représentation hiérarchique, qui rend compte des relations d'héritage qui ont été déclarées pour les objets du domaine considéré.

L'utilisation habituelle de l'héritage est la programmation par spécialisation, le partage de propriétés et la réutilisation de code [Borgida *et al.*, 1984] [Snyder, 1987] [Snyder, 1991]. La gestion du mécanisme d'héritage peut reposer sur un algorithme de linéarisation qui établit l'ordre des ascendants d'un objet dans le graphe d'héritage, ordre dans lequel sont héritées les valeurs des attributs de l'objet [Ducournau and Habib, 1991]. Bien programmer nécessite alors une bonne connaissance de la hiérarchie d'héritage ainsi que celle de l'algorithme sur lequel repose la recherche d'une propriété dans cette hiérarchie.

La hiérarchie d'héritage d'un objet peut aussi être considérée comme un chemin, dont les caractéristiques dépendent des relations qui existent entre les objets appartenant au chemin, autrement dit des arcs positifs et/ou négatifs (arcs d'exception) constituant le chemin. A l'instar de la logique des défauts par exemple [Etherington, 1987], une logique particulière, munie de ses propres règles d'inférences, est définie sur l'ensemble des chemins ; elle permet de trier les chemins et de déterminer les propriétés pouvant être inférées pour un objet donné [Horty, 1991].

3.1.2 Héritage et expression de connaissances

Classiquement, la mise en place d'un objet dans une hiérarchie d'héritage est toujours à la charge du programmeur, alors qu'elle devrait plutôt être à la charge du système. En particulier, un objet est reconnu comme une instance du concept *éléphant à trois pattes* s'il a été défini comme tel, comme une instance de la classe `Elephant-a-trois-pattes` par exemple. Réciproquement, si `Jumbo` est défini comme une instance de la classe `Elephant` et qu'il perd une patte dans un accident de la route, il ne deviendra pas pour autant une instance de la classe `Elephant-a-trois-pattes`. Plus encore, il n'y a *a priori* aucune raison pour qu'un éléphant gris à trois pattes soit considéré comme un cas particulier d'éléphant à trois pattes, si rien n'est spécifié.

Le fait que les attributs d'un frame ou les variables d'instances d'une classe n'expriment que des conditions nécessaires d'appartenance d'un individu à l'ensemble dénoté par le frame ou la classe, explique en partie les limitations qui viennent d'être évoquées. Si `Jumbo` est une instance de la classe `Elephant-a-trois-pattes`, alors `Jumbo` possède les caractéristiques `C1`, ..., `Cn`, associées à la classe. L'inverse n'est pas forcément vrai : si `Jumbo` est un éléphant et que ses caractéristiques sont équivalentes à `C1`, ..., `Cn`, alors il n'existe généralement aucun mécanisme permettant de découvrir que `Jumbo` peut être une instance de la classe `Elephant-a-trois-pattes`.

3.1.3 Héritage et inférence

Supposons que soit définie une sous-classe `SC` commune aux classes `Elephant-ayant-au-moins-quatre-pattes` et `Elephant-ayant-au-plus-quatre-pattes`. Une instance de `SC` ne possèdera pas la propriété *avoir exactement quatre pattes* qui pro-

vient de la combinaison des propriétés caractéristiques des deux classes mères de SC. L'héritage ne permet pas de combiner des propriétés héritées, même dans un cas simple comme celui qui précède. Toutefois, dans le cas présent, il est possible de calculer l'intersection des domaines associés à chacune des propriétés héritées pour retrouver le résultat produit par la combinaison des propriétés.

Cependant, l'interaction entre des propriétés peut être plus complexe. Supposons que soient données les assertions suivantes [Patel-Schneider *et al.*, 1990] :

- *Tout parent est une personne.*
- *Tout parent de docteur est un parent.*
- *Tout enfant de parent de docteur est un docteur.*

La hiérarchie d'héritage associée à ces assertions se limite à un arbre où le concept *parent de docteur* a pour ascendant *parent* et où l'attribut *enfant* associé à *parent de docteur* a pour type *docteur*. Si l'héritage est le seul mécanisme d'inférence disponible, il est impossible de répondre aux deux questions suivantes :

- Si Gédéon est un parent de docteur, alors quelle profession exerce son fils Léon ?
- Si Albert est docteur, est-ce que ses parents sont reconnus comme des parents de docteur ?

La combinaison de propriétés permet aussi de détecter des incohérences : si Jumbo est un éléphant à trois pattes, alors il ne peut évidemment pas appartenir à la classe des éléphants à quatre pattes, mais en plus, il ne peut exister aucun endroit qui contienne à la fois Jumbo et rien que des éléphants à quatre pattes...

3.2 La subsomption

3.2.1 Introduction

La *subsomption*[2] est une relation définie sur un ensemble de descriptions de concepts : un concept *C1* subsume un concept *C2* si *C1* est nécessairement (ou par définition) plus général que *C2*. Ainsi, le concept *éléphant* subsume le concept *éléphant à trois pattes* car tout éléphant à trois pattes est avant tout un éléphant. La relation de subsomption organise les concepts en une hiérarchie, où les (concepts) *subsumants* sont placés à un niveau plus général ou "au-dessus" des (concepts) *subsumés*.

Supposons que soit donnée la suite d'assertions suivantes [Nebel, 1988] :

- *Un homme est un humain et un mâle.*
- *Un parent est un humain qui a au moins un enfant.*

2. Le verbe subsumer a pour racines latines *sub* et *sumere*, ce qui signifie littéralement "prendre sous".

- *Un père est un parent et un homme.*
- *Un grand-parent est un humain qui a au moins un enfant qui est un parent.*

Un concept est défini non seulement par des relations explicites qui font partie intégrante de sa description, mais aussi par des relations implicites qu'il est possible de déduire. Ainsi, le concept *grand-parent* est une spécialisation du concept *parent*, bien que ce ne soit pas explicitement précisé dans la définition de *grand-parent*. En analysant la définition de *grand-parent*, il apparaît que tous les individus qui sont susceptibles d'être décrits par *grand-parent* sont nécessairement décrits par *parent*. Une suite d'assertions comme celle qui précède contient toujours plus de connaissances que ce qui n'est exprimé explicitement. Il est donc indispensable de disposer d'outils qui découvrent les relations implicites qui existent entre les concepts décrits.

Ces connaissances explicites et implicites servent à satisfaire les requêtes classiques suivantes [Nebel, 1988] :

- Subsomption : est-ce que le concept *C1* subsume le concept *C2* ?
- Classification : étant donné un ensemble de concepts organisés en hiérarchie, quels sont les subsumants les plus spécifiques et les subsumés les plus généraux d'un concept donné ?
- Indépendance : est-il possible de déclarer ou de montrer que deux concepts sont disjoints ?
- Inconsistance : est-il possible de découvrir qu'un concept est inconsistant, au sens où un tel concept ne peut pas avoir de représentant ?
- Possession de propriétés[3] : quelles sont les propriétés détenues par un concept, les restrictions associées à ces propriétés et leurs valeurs ?

Toutes les requêtes précédentes se ramènent à des calculs déterminant des relations de subsomption : classifier un concept revient à découvrir ses subsumants les plus spécifiques et ses subsumés les plus généraux ; un concept est inconsistant s'il est subsumé par un concept déclaré à l'avance inconsistant ; deux concepts sont indépendants si leur conjonction définit un concept inconsistant ; trouver les propriétés possédées par un concept se ramène à connaître les propriétés détenues par les subsumants du concept. S'il est possible de spécifier un algorithme de complexité polynomiale pour déterminer les relations de subsomption existant entre les concepts, alors toutes les autres formes de requêtes sont aussi de complexité polynomiale.

3.2.2 Trois définitions

La relation de subsomption se définit sur un ensemble d'objets représentant des concepts. Le terme "objet" est à prendre au sens très général de conjonction de couples propriété-valeur. Voici trois manières différentes de présenter l'assertion *A subsume B* :

3. L'exemple de requête standard que permet de satisfaire le mécanisme d'héritage.

- Définition en *extension* [Levesque and Brachman, 1987] : A subsume B si l'ensemble des individus dénoté par A contient nécessairement l'ensemble des individus dénoté par B. Par exemple, l'objet qui dénote les mortels subsume l'objet dénotant les hommes.

- Définition en *intension* [Finin, 1986] [Woods, 1991] : A subsume B si tout individu décrit par B l'est aussi par A ; autrement dit, si l'ensemble des propriétés d'un individu dont la description est définie par B contient l'ensemble des propriétés qui sont spécifiées par A. Par exemple, l'ensemble des propriétés associé à l'objet qui dénote les hommes "comprend" les propriétés associées à l'objet qui dénote les mortels. Tout objet se compose donc d'une description propre définie par des propriétés *locales* et d'une description *partagée* avec ses subsumants.

- Définition *logique* [MacGregor, 1988] : A subsume B si être un individu décrit par B implique logiquement être un individu décrit par A. Ainsi, pour tout x, homme(x) $\Rightarrow$ mortel(x). Cette dernière définition peut être considérée comme une reformulation de la définition précédente.

L'inclusion ensembliste, interprétée en extension ou intension, ainsi que l'implication logique, induisent des relations d'ordre partiel, il en est donc de même de la subsomption. En particulier, la relation de subsomption est réflexive, A subsume A, et transitive : un objet B subsumé par un objet A est subsumé par tous les subsumants de A.

Par convention et par analogie avec une hiérarchie d'héritage, la hiérarchie ou graphe de la relation de subsomption possède toujours une borne supérieure, l'objet le plus général qui subsume tous les autres objets, mais pas forcément une borne inférieure[4]. Par souci d'homogénéité, les types simples comme entier, réel, chaîne de caractères, booléen, etc., sont supposés être représentés par des objets subsumés par l'objet le plus général.

Les trois définitions de la subsomption portent sur des concepts, mais elles peuvent également s'employer pour déterminer si un concept subsume un individu, en assimilant l'individu à un concept, dont l'extension ne contient qu'un seul élément, à savoir l'individu lui-même [Resnick *et al.*, 1990].

3.2.3 Exemple et complexité de la subsomption

En KL-ONE, un concept du monde réel est décrit par un objet appelé *concept*, qui est muni de rôles ou attributs et qui est comparable à un frame. Tester qu'un concept A subsume un concept B se fait en comparant les descriptions de A et de B composant par composant [Schmolze and Lipkis, 1983] [Schmolze and Israel, 1983] :

- Tous les objets qui subsument A subsument B.

- Pour tout attribut a de A, il existe un attribut b de B, qui exprime la même propriété (inclusion de l'intension de A dans celle de B) :

4. Une telle borne inférieure existe en OMEGA [Attardi *et al.*, 1986]

- Le type de la valeur de a subsume le type de la valeur de b.
- Le domaine de valeurs associé à a contient celui qui est associé à b.
- L'intervalle de cardinalité associé à a doit inclure celui qui est associé à b.

La fonction qui implante l'algorithme précédent s'appelle `SubsumeP`. Elle possède la propriété de consistance [Schmolze and Lipkis, 1983] : si `SubsumeP(A,B)` est vrai, alors l'ensemble des individus dénoté par `A` contient nécessairement l'ensemble des individus dénoté par `B`. Toutefois, la fonction ne possède pas la propriété de complétude car elle ne découvre pas forcément toutes les relations de subsomption existantes : la fonction retourne toujours un résultat, vrai ou échec, mais lorsqu'elle retourne échec, il n'est pas sûr que le concept A ne subsume pas le concept B.

Il a été prouvé que le problème consistant à déterminer qu'un objet A subsume un objet B est de complexité équivalente au problème de la satisfaction d'une formule booléenne [Levesque and Brachman, 1987]. La troisième définition de la subsomption montre que les deux problèmes sont liés : A subsume B si et seulement si A se déduit logiquement de B. Nous revenons au paragraphe 3.5.2 sur la complexité de la subsomption, qui est en rapport avec la richesse du langage de description des concepts utilisé.

3.2.4 La subsomption clausale

Une clause est une disjonction de littéraux formés de formules atomiques positives et négatives qui sont elles-mêmes composées de termes [Chang and Lee, 1973]. La subsomption est une relation qui est aussi définie sur les clauses, elle permet de déterminer si une clause en implique une autre, autrement dit, si l'information contenue dans une clause n'est pas déjà contenue dans une autre.

Il existe plusieurs définitions de la subsomption sur les clauses. Pour notre part, nous distinguerons subsomption et θ-subsomption : une clause C *subsume* une clause D si C implique logiquement D [Gottlob, 1987] ; une clause C *θ-subsume* une clause D si $\|C\| \leq \|D\|$ (le nombre de littéraux de C est inférieur à celui de D), et s'il existe une substitution θ portant sur les variables telle que $C\theta \subseteq D$ [Gottlob and Leitsch, 1985] [Stickel, 1986]. Si une clause C subsume une clause D, alors l'information contenue dans D est redondante et D peut éventuellement être éliminée. En particulier, si C θ-subsume D, alors C est plus "courte" que D. Par exemple, *P(x)* subsume et θ-subsume la disjonction $P(a) \vee P(b)$, mais $P(x) \vee P(a)$ subsume *P(a)* et $Q(x,y) \vee Q(y,x)$ subsume *Q(x,x)*, sans que la θ-subsomption soit vérifiée.

Des résultats duaux sont donc obtenus en considérant que des objets représentant des concepts sont assimilés à des conjonctions de littéraux [Hayes, 1979] : si le concept A subsume le concept B, alors l'intension de B contient celle de A et donc A est plus "court" que B comme pour les clauses, mais B implique logiquement A, à l'inverse des clauses.

3.2.5 La subsomption subsume l'héritage

Il est naturel de se demander quel rapport existe entre la relation de subsomption et celle d'héritage dans une représentation à objets. A première vue, rien ne semble distinguer les deux relations : les trois interprétations présentées précédemment rappellent les interprétations en extension, en intension et logique de la relation d'héritage [Brachman, 1983]. Toutefois, la relation de subsomption est plus générale que celle d'héritage car elle porte sur des formules booléennes qui sont des conjonctions ou des disjonctions de littéraux. Les descriptions de concepts et celles des individus recouverts par ces concepts s'apparentent à des conjonctions de littéraux, alors que, de façon duale, les disjonctions de littéraux sont des clauses [Hayes, 1979]. La subsomption ne relève donc pas du même niveau d'analyse et de description des connaissances que l'héritage. La subsomption subsume l'héritage : un concept *C1* hérite une propriété *P* s'il est subsumé par un concept possédant la propriété *P* [Patel-Schneider, 1991] ; les inférences par héritage apparaissent comme des cas particuliers d'"inférences par subsomption".

Même si un système à subsomption est implanté avec un langage à objets, il est préférable que les mécanismes d'héritage et de subsomption restent indépendants. Ainsi, dans une représentation à objets où est définie une relation de subsomption, l'héritage peut jouer partiellement le rôle de la subsomption : si un concept *C1* hérite d'un concept *C2*, alors *C2* subsume *C1*. Ces inférences sont simples et relèvent uniquement du partage de propriétés. Supposons maintenant que le concept *Père heureux* désigne un père dont toutes les filles sont mariées, que *Pierre* est un père et qu'il possède une fille nommée *Marthe*. Si *Marthe* est mariée, alors l'inférence *Pierre* est un *Père heureux* doit en découler automatiquement ; s'il apparaît que *Marthe* divorce, alors *Pierre* ne doit plus être un *Père heureux* ; pour finir, si *Pierre* est déclaré être un *Père heureux*, alors l'inférence *Marthe* est mariée doit en découler. Ces inférences ne sont pas réalisables avec le seul mécanisme d'héritage, qui ne doit donc pas être l'unique mode d'inférence dans une représentation à objets.

L'héritage est un mécanisme de partage de code et le support de la programmation par réutilisation [Snyder, 1987]. Ce point de vue n'est pas pris en compte par la relation de subsomption : les méthodes et les réflexes ne sont pas considérés comme des composants de représentation dans un système à subsomption, car il n'existe pas de sémantique précise permettant de déterminer qu'une procédure en "subsume" une autre.

3.2.6 La subsomption et les exceptions

Dans une hiérarchie d'héritage, si *A* est plus général que *B*, alors toutes les propriétés de *A* sont héritées par *B*, sauf en cas de masquage où une propriété héritable, valeur ou méthode, est redéfinie, ou encore en présence d'exceptions à l'héritage, où un ensemble de propriétés héritables est globalement ignoré. Le masquage de valeurs et les exceptions conduisent à un raisonnement non monotone [Reiter, 1987] : si une propriété affirmée à un niveau de la hiérarchie est remise en cause à un niveau inférieur, ce qui est déductible pour les objets du niveau supérieur peut ne plus l'être pour les

objets du niveau inférieur. Certaines théories sur les exceptions ont vu le jour dans le cadre d'études sur les graphes d'héritage [Ducournau and Habib, 1991] [Horty, 1991]. Dans le cadre des systèmes à subsomption, un article célèbre de R.J. Brachman fait encore autorité en la matière [Brachman, 1985] : autoriser des exceptions à la relation de subsomption conduit à définir un système de représentation où un éléphant est un singe, avec l'exception notable qu'il ne grimpe pas aux arbres...

3.2.7 La subsomption dans les représentations à objets

La relation de subsomption est définie de façon très générale au paragraphe 3.2.2. Dans ce qui suit, nous allons donner une définition formelle de la subsomption, dans le contexte des représentations à objets. Nous baptisons cette relation la subsomption *faible* ou *spécialisée*, par opposition à la subsomption générale qui porte indifféremment sur des conjonctions ou des disjonctions de termes, qui pourrait être qualifiée de subsomption "forte". Nous considérons dans la suite qu'un objet est une conjonction de propriétés attributs-valeurs (un frame muni de couples attributs-facettes par exemple).

Etant donné :

- un ensemble d'objets $\mathcal{E}$;
- un ensemble de propriétés $\mathcal{P} = \{p_i, i \in I\}$;
- une relation d'ordre partiel spécifique $\succeq_i$, dépendant de chaque p_i ;
- un ensemble $\mathcal{Q} = \{q_j, j \in J\}$ de propriétés déductibles : la valeur d'une propriété q_j dépend des valeurs des propriétés p_i et se calcule grâce à une méthode de partage de propriété spécifique notée $\mathcal{MP}$.

Nous dirons qu'un objet *O1* appartenant à $\mathcal{E}$ subsume au sens des représentations à objet (faiblement ou de façon spécialisée) un objet *O2* appartenant à $\mathcal{E}$ si :
pour tout $i \in I$, $p_i(O1) \succeq_i p_i(O2)$, où $p_i(O1)$ dénote la propriété p_i pour l'objet *O1*.

La relation d'ordre partiel résultante, notée $\succeq$, est une relation d'ordre partiel "produit", définie comme la conjonction des ordres partiels spécifiques $\succeq_i$ pour $i \in I$.

Si *O1* subsume faiblement *O2*, alors, pour tout $j \in J$, la valeur de $q_j(O2)$ se déduit par l'intermédiaire de $\mathcal{MP}$ des valeurs des propriétés p_i associées à *O1*.

La relation d'héritage est un cas particulier de subsomption faible : un objet *O1* subsume (au sens de l'héritage) un objet *O2* si *O1* est un ascendant de *O2* dans le graphe d'héritage. Plus précisément, l'ensemble d'objets $\mathcal{E}$ s'identifie au graphe d'héritage ; l'ensemble de propriétés $\mathcal{P}$, tout comme l'ensemble $\mathcal{Q}$, fait référence aux propriétés des objets, quelles qu'elles soient (les propriétés s'héritent de façon uniforme [Ducournau and Habib, 1989]) ; les relations d'ordre partiel spécifiques dépendent du type des objets (un exemple de telles relations est présenté au paragraphe 3.3.3, page 109).

La définition d'une relation de subsomption faible dans l'univers des graphes nous offre un autre exemple important, repris d'ailleurs dans le contexte des graphes moléculaires au chapitre 5. Soit $\mathcal{E}$ un ensemble de graphes ayant des nœuds et des arêtes étiquetés ; une relation d'ordre partiel $\succeq_n$ définie sur l'ensemble des nœuds $\mathcal{N}$; une relation d'ordre partiel $\succeq_a$ définie sur l'ensemble des arêtes $\mathcal{A}$. Un graphe $G1$ (= $(N1,A1)$) subsume (au sens des graphes étiquetés) un graphe $G2$ (= $(N2,A2)$) si :

- $G1$ est un sous-graphe de $G2$ (au sens standard de la théorie des graphes).
- Pour tout $n_i \in N1$, $n_i(G1) \succeq_n n_i(G2)$; pour tout $a_j \in A1$, $a_j(G1) \succeq_a a_j(G2)$.

Il n'existe pas de définition générale du partage de propriétés, qui est spécifique à l'application envisagée. Nous donnons au paragraphe 5.4 (cf. page 194) un exemple possible de tel partage.

Sauf mention explicite du contraire, subsomption signifie désormais subsomption faible.

3.3 La classification

3.3.1 Un algorithme de classification

Dans la suite, le terme "objet" continue à désigner de façon générale la description d'un concept sous la forme d'une conjonction de propriétés attributs-valeurs. Etant donné une relation de subsomption faible, et un ensemble d'objets organisés en hiérarchie par l'intermédiaire de cette relation, la classification est l'opération qui permet de placer un objet X dans la hiérarchie, en recherchant les objets A qui subsument X et les objets B qui sont subsumés par X, sans qu'aucune de ces relations de subsomption ne soit explicitement spécifiée. X est soit une nouvelle description, soit une description déjà présente dont certaines caractéristiques ont été modifiées ou doivent être précisées. Le processus de classification d'un objet X dans une représentation à objets s'appuie sur la comparaison de X avec les objets présents dans la hiérarchie et met en valeur les relations de subsomption qui existent entre X et ces objets.

Le processus de classification se décompose en trois étapes principales : recherche des subsumants les plus spécifiques ou SPS, recherche des subsumés les plus généraux ou SPG, et mise en place des relations entre l'objet à classer, ses subsumants et ses subsumés. L'algorithme qui est présenté dans la suite est inspiré de celui qui est utilisé en KL-ONE [Lipkis, 1982] ; il est indépendant de la sémantique de la relation de subsomption utilisée ; la seule contrainte imposée est que la relation employée soit une relation de subsomption faible.

L'algorithme peut évidemment être utilisé avec la relation d'héritage pour gérer un graphe d'héritage ou encore pour découvrir les relations de généralisation/spécialisation qui existent entre de nouveaux objets et les objets déjà présents dans le graphe.

```
fonction comparer (X,O*)
   si O* est marqué
     alors ne rien faire
     sinon marquer O*
           si O* ne subsume pas X
              alors retourner {}
              sinon si O* est une feuille
                       alors retourner {O*}
                       sinon SPS ← {}
                             pour chaque descendant D* de O* faire
                                   SPS ← SPS ∪ comparer (D*,X)
                             si SPS = {}
                               alors retourner {O*}
                               sinon retourner {SPS}
```

Figure 3.1. Comparaison entre l'objet courant O* et l'objet à classer X lors de la recherche des subsumants les plus spécifiques de X.

La recherche des subsumants les plus spécifiques

Dans ce qui suit, les objets appartiennent à une hiérarchie bornée supérieurement par un objet appelé la *racine* de la hiérarchie, qui subsume tous les autres objets présents. Les subsumants les plus spécifiques s'obtiennent en parcourant la hiérarchie en profondeur, en partant de la racine. La hiérarchie n'est explorée que partiellement, car les sous-hiérarchies dont les maximaux sont des objets incompatibles avec l'objet à classer sont élaguées. La figure 3.1 montre comment s'effectue la comparaison de l'objet courant O^* avec l'objet à classer X.

Si O^* ne subsume pas X, alors la sous-hiérarchie de racine O^* est élaguée. Aucun descendant de O^* ne peut subsumer X car la relation de subsomption est transitive. L'exploration se poursuit à partir du premier frère de O^* faisant encore partie de l'ensemble des objets à explorer. Si O^* subsume X, alors O^* est temporairement le subsumant le plus spécifique de la sous-hiérarchie en cours de traitement. Si O^* est une feuille de la hiérarchie, alors l'exploration est terminée pour la branche en cours de traitement et O^* est le subsumant le plus spécifique pour cette branche. Si O^* n'est pas une feuille, l'exploration se poursuit en profondeur à partir de O^*. Si aucun des subsumés de O^* ne subsume X, alors O^* est déclaré subsumant le plus spécifique de la sous-hiérarchie en cours de traitement et il est ajouté à l'ensemble dénoté par la variable locale SPS, qui contient les subsumants les plus spécifiques déjà trouvés. Le parcours continue jusqu'à ce tous les objets de la hiérarchie aient été examinés.

Dans les systèmes à subsomption, les concepts sont divisés en concepts primitifs et concepts définis. Dans certains cas, les concepts primitifs sont ignorés lors de la recherche des subsumants les plus spécifiques d'un concept à mettre en place dans la

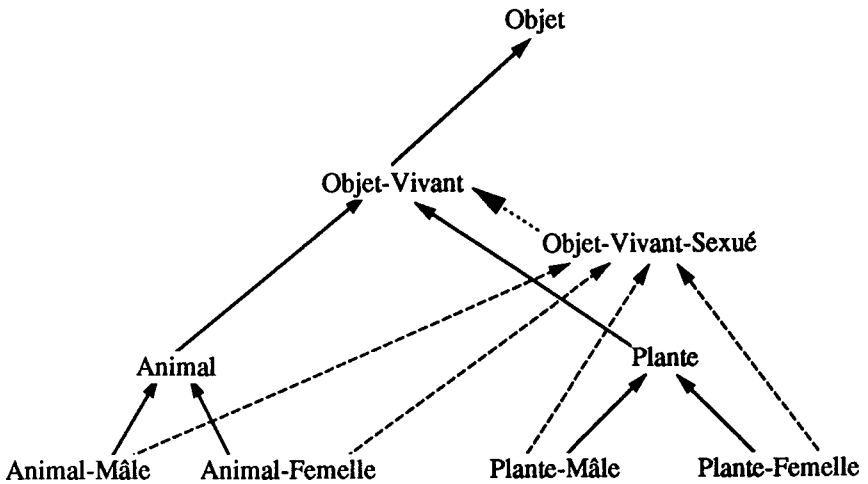


Figure 3.2. Les SPG ne sont pas forcément les descendants immédiats des SPS.

hiérarchie (cf. page 129).

La recherche des subsumés les plus généraux

La seconde partie du processus de classification concerne la recherche des subsumés les plus généraux (SPG). Puisque la relation de subsomption est transitive, l'obtention des subsumants les plus spécifiques permet de focaliser cette recherche : il suffit de n'explorer que les descendants des SPS qui possèdent les mêmes propriétés que l'objet à classer X, descendants qui sont des concepts ou des représentants de concepts.

Si $\{D\}$ désigne l'ensemble des descendants des SPS et D^* le descendant courant, l'exploration s'effectue sur $\{D\}$ de la façon suivante :

- Si X subsume D^*, alors D^* fait partie des subsumés les plus généraux.
- Si X ne subsume pas D^*, alors l'exploration se poursuit en profondeur sur les descendants de D^*, jusqu'à ce qu'un élément soit un subsumé de X ou bien que D^* n'ait plus de descendants.

Les SPG d'un concept ne sont pas forcément les descendants immédiats de ses SPS comme le montre l'exemple de la figure 3.2 [Finin, 1986]. L'objet *Objet-Vivant-Sexué* est subsumé par l'objet *Objet-Vivant*. Il ne subsume pas les descendants immédiats de *Objet-Vivant*, à savoir *Animal* et *Plante*, mais leurs descendants *Animal-Mâle*, *Animal-Femelle*, *Plante-Mâle* et *Plante-Femelle*.

Insertion d'un objet dans la hiérarchie

Lorsque les subsumants les plus spécifiques et les subsumés les plus généraux ont été découverts, de nouveaux liens exprimant les relations de subsomption existant entre

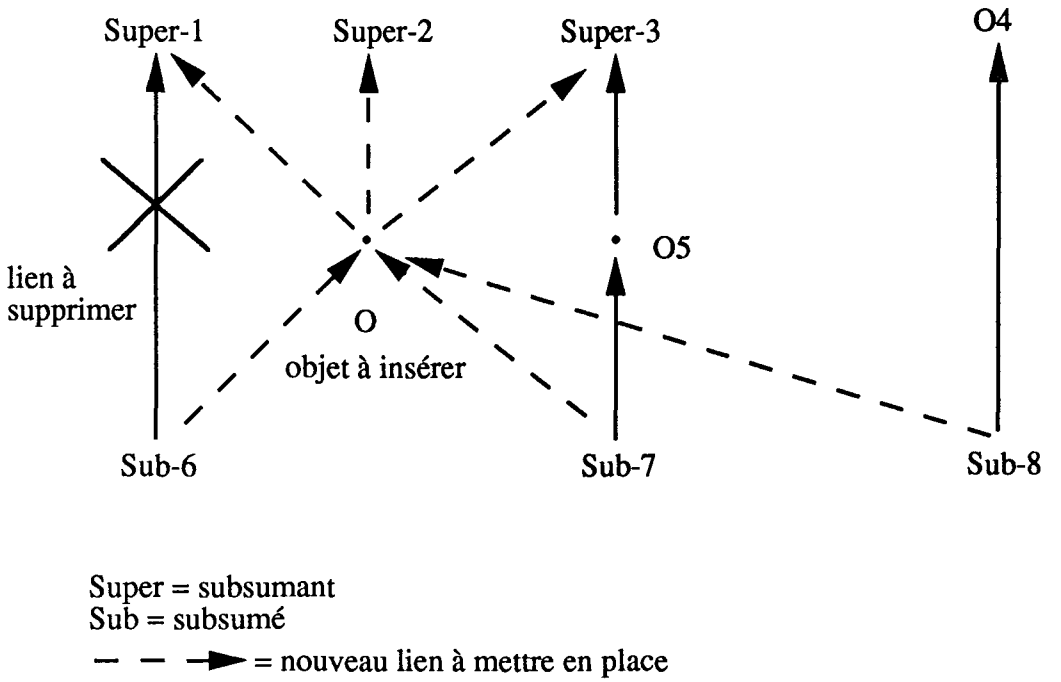


Figure 3.3. Redondances à détecter et à supprimer lors de la mise en place des liens subsumant-subsumé.

ces objets peuvent être mis en place. En particulier, de nouveaux liens d'héritage sont construits lorsque la relation de subsomption est identifiée à la relation d'héritage. La mise en place de liens entre X et ses subsumants les plus spécifiques ne pose aucun problème particulier. En revanche, l'opération d'insertion peut provoquer l'apparition de liens ou arcs de transitivité qui sont considérés comme redondants et qui doivent être supprimés (cf. Fig. 3.3). Rappelons qu'un arc de transitivité relie deux objets comparables mais non consécutifs dans le graphe de la relation considérée, ici la relation de subsomption. Un arc de transitivité est redondant, mais, à l'inverse des arcs de transitivité doublant une exception à l'héritage, il n'est pas contradictoire (cf. page 49). Supprimer cette redondance consiste simplement à supprimer l'arc de transitivité. En dehors du cas qui vient d'être évoqué, les liens déjà existants entre les objets du graphe ne doivent pas être modifiés (cf. Fig. 3.3).

Lorsque X est un objet déjà présent dans le graphe et que le processus de classification a modifié la place que X y occupait, deux cas peuvent se produire. Dans le premier cas, de nouveaux liens de subsomption sont précisés et les liens existant précédemment sont conservés. Le second cas concerne les "migrations" d'objets : les anciens liens ne sont alors pas forcément conservés, mais ce problème n'est pas traité ici [Nguyen and Rieu, 1989] [Kim, 1990].

Classification et formation incrémentielle de concepts

Le processus de classification permet d'insérer un nouvel objet dans une hiérarchie d'objets, en accord avec les informations présentes dans la hiérarchie. Par conséquent, un tel processus peut également être utilisé à une autre fin, celle de la construction incrémentielle de la hiérarchie elle-même. Une telle tâche est similaire à celle effectuée par les systèmes de formation incrémentielle de concepts [Gennari *et al.*, 1989] (cf. § 4.2.5). Dans cette optique, le processus de classification se présente donc naturellement comme un outil d'aide à l'acquisition de connaissances [MacGregor and Burstein, 1991]. Un exemple de construction incrémentielle et dynamique de hiérarchie d'objets à l'aide du processus de classification, la hiérarchie des structures réactives, est donné au chapitre 5 (cf. page 191).

3.3.2 Classification dans un graphe d'héritage

Dans ce paragraphe, la relation de subsomption est assimilée à la relation d'héritage et le processus de classification est appliqué à des objets faisant partie d'un graphe d'héritage.

Classification interactive en héritage simple

Dans [Finin, 1986], T. Finin présente une méthode de classification interactive et incrémentielle qui repose sur un parcours en profondeur d'un arbre d'héritage simple. Le processus de classification guide et aide l'utilisateur à compléter la description d'un nouvel objet X à insérer dans l'arbre d'héritage, grâce à une série de questions résultant de comparaisons faites entre X et les objets existants. Ainsi, seule une description partielle de X est nécessaire au départ de la classification et aucune connaissance préalable des objets déjà présents n'est utile.

Deux stratégies guident la classification. La première, dite *classification par profil d'attributs*, consiste à choisir un attribut dans X, puis à rechercher l'ascendant commun le plus spécifique, au sens de l'héritage, parmi tous les objets possédant cet attribut. Cet ascendant, noté A dans la suite, constitue le nœud de l'arbre d'héritage à partir duquel commence la classification. Si A subsume X au sens de l'héritage, alors A devient le subsumant le plus spécifique, A étant le seul subsumant en héritage simple. Lors de cette comparaison, tous les attributs de A sont examinés et l'utilisateur peut se voir sollicité pour approuver ou réprouver l'adjonction de certains attributs à X, qui est supposé avoir une description incomplète. Si aucun ascendant commun n'est trouvé ou aucune relation de subsomption n'est établie, un autre attribut est choisi, l'utilisateur n'étant sollicité qu'en dernier recours.

La seconde stratégie, dite *stratégie par exclusion*, s'appuie sur le fait qu'à chaque étape de la classification, l'ascendant de X dans l'arbre d'héritage, noté *SUPER*, qui est aussi le subsumant le plus spécifique de X au sens de la relation d'héritage, est unique. A chaque étape de la descente dans l'arbre d'héritage, seul un descendant de *SUPER* peut devenir le prochain ascendant de X. Si aucun descendant de *SUPER* ne

subsume X au sens de l'héritage, alors l'ascendant de X reste *SUPER* et la classification continue avec la recherche des descendants de X. Si un seul descendant de *SUPER* subsume X, alors il devient l'ascendant de X, à condition que la description de X soit compatible avec les attributs supplémentaires que pourrait détenir le descendant et que ne détiendrait pas X. L'utilisateur se voit alors sollicité pour approuver l'adjonction de tels attributs et leur ensemble de valeurs. Lorsque plusieurs descendants de *SUPER* subsument X, ils sont triés après un examen interactif de leurs attributs de façon à ce qu'il ne reste plus qu'un seul subsumant possible, pour ainsi se retrouver dans le cas précédent.

Cette approche de la classification, qui fait jouer un rôle très actif à l'utilisateur, a conduit à certaines études sur l'interactivité, qui préconisent la création de *profils d'utilisateurs* auxquels sont associés des questionnaires typiques guidant le processus de classification [Rich, 1979] [Finin and Drager, 1986].

Un exemple en Shirka

Shirka est un langage de frames, dont les frames se répartissent en objets générateurs, les *schémas de classe*, et en objets terminaux, les *schémas d'instance*, qui ne peuvent pas être instanciés[5] [Rechenmann, 1988]. Toutes les entités, schémas, attributs, facettes ou procédures, sont des schémas, et tout schéma est instance d'un schéma de classe.

La figure 3.4 montre la définition de la classe `Personne`. L'attribut `sorte-de` représente le lien d'héritage et sa valeur indique que le schéma `Personne` est une spécialisation du schéma `objet`, qui est la racine du graphe d'héritage multiple. Tout attribut possède obligatoirement un type, introduit par l'une des deux facettes de typage `$un` ou `$liste-de`, type qui est éventuellement restreint grâce aux facettes `$intervalle` et `$domaine`. Les musiciens, qui forment une catégorie particulière de personnes, sont décrits par la classe `Musicien`, spécialisation de `Personne`. Un musicien spécifique comme `Albert` est représenté par une instance qui est rattachée à sa classe par un lien `est-un` (cf. Fig. 3.5). Les valeurs des attributs définies dans `Albert`, introduites grâce au symbole `=`, doivent respecter les spécifications de la classe[6]. Une instance n'est que rarement une description complète ; elle ne contient que les valeurs connues lors de sa définition ; les autres valeurs sont retrouvées par héritage, par calcul, par filtrage ou encore par classification.

En Shirka, le filtrage permet de calculer la valeur d'un attribut en parcourant les instances d'une classe donnée et en ne retenant que celles qui satisfont certains critères. La classification permet de trouver la position d'une instance, nouvelle ou ancienne, en parcourant la hiérarchie des classes et en ne retenant que les classes les plus spécifiques compatibles avec la structure de l'instance [Pivot and Prokop, 1987]. Le graphe d'héritage est divisé en familles qui sont déterminées par les sous-graphes

5. Schéma de classe et schéma d'instance sont respectivement abrégés en classe et instance dans la suite.

6. Les facettes `$valeur`, `$defaut` et le symbole `=` différencient explicitement une valeur partagée par toutes les instances d'une classe, une valeur pouvant être redéfinie et une valeur affectée à un attribut dans une instance.

```
{Personne
   sorte-de = objet ;
   nom $un chaine ;
   prenom $liste-de chaine ;
   age $un entier
       $var-nom a
       $sib-exec {Soustraction
                   valeur1 $valeur *annee-courante* ;
                   valeur2 $var<- annee-naissance ;
                   resultat $var-> age}
   date-de-naissance $un {Date
                              annee $var-nom annee-naissance}
   enfants $liste-de Personne ;
   coaequales $liste-de Personne
              $sib-filtre {Personne
                              lui-meme $var-> coaequales ;
                              age $var<- a}}

{Musicien
   sorte-de = Personne ;
   specialite $liste-de Specialite
              $domaine Composition Arrangement Interpretation}

{Albert
   est-un = Poly-Instrumentiste ;
   nom = "Marqueur" ;
   prenom = "Albert Joseph" ;
   date-de-naissance = date-123}
```

Figure 3.4. Le schéma de classe `Personne`, une de ses spécialisations, `Musicien` et le schéma d'instance `Albert`. Un filtre est associé à l'attribut `coaequales`. Il permet de retrouver la liste des instances de la classe `Personne` ayant le même âge qu'une personne donnée.

d'héritage dont les racines sont les sous-classes directes d'`objet`. Ces racines sont appelées chefs de famille et elles définissent une partition du graphe d'héritage, au sens où un attribut est supposé avoir la même sémantique chaque fois qu'il est utilisé à l'intérieur d'une famille (à l'image des attributs associés à un concept en TROPES, voir le paragraphe 2.3.3).

L'algorithme de classification est fondé sur le parcours d'une famille en profondeur. Il s'appuie sur un processus dit de *spécialisation* qui consiste à vérifier que les caractéristiques d'une instance donnée sont compatibles avec celles de la classe vers laquelle

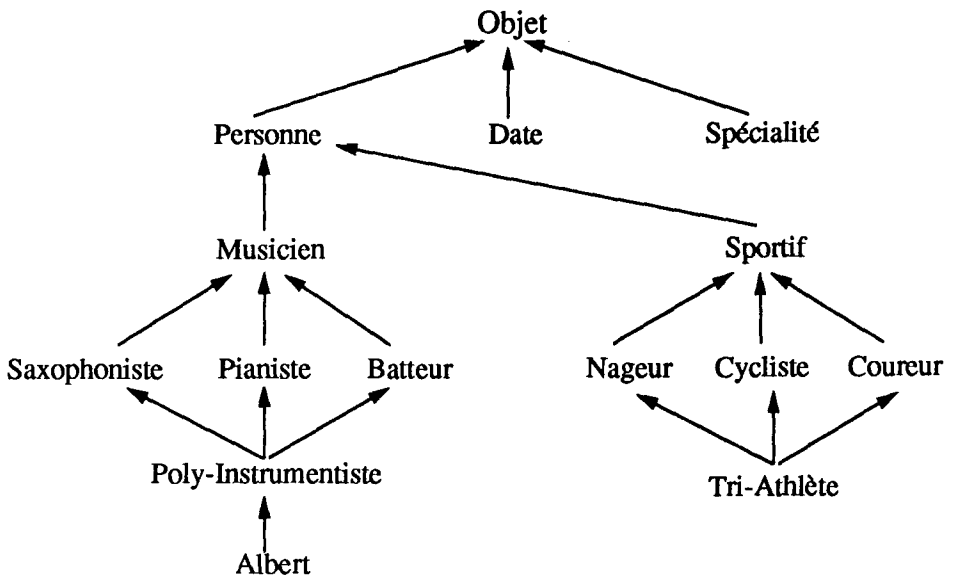


Figure 3.5. Une partie du graphe d'héritage contenant la famille `Personne`.

on souhaite spécialiser l'instance, c'est-à-dire la classe qui devrait devenir une nouvelle classe mère de l'instance. Classifier une instance revient donc à trouver l'ensemble des classes du graphe vers lesquelles l'instance peut être spécialisée. La classification s'effectue avec l'hypothèse du monde ouvert et son résultat est constitué de trois listes de classes[7] : les classes sûres dont fait partie la future classe mère de l'instance, les classes impossibles qui ne peuvent pas être des classes mères et les classes possibles, pour lesquelles manquent certaines informations pour qu'une décision soit prise. La classification est interactive et, lorsqu'un attribut n'est pas valué, le système calcule sa valeur s'il existe un réflexe si-besoin pour le faire, sinon il la demande à l'utilisateur. Une explication est associée à chaque échec (classe impossible) ou demi-échec (classe possible) du processus de spécialisation.

La spécialisation d'une instance I de classe initiale Ci vers une classe cible C s'effectue comme suit :

- Une marque *sûr* est associée à toutes les superclasses de Ci jusqu'à ce que F soit marqué, F étant le chef de famille ou bien la "superclasse commune la plus spécifique" à Ci et à C. La relation d'ordre définissant la "superclasse commune la plus spécifique" est la relation d'héritage : une classe est plus spécifique qu'une autre si la première hérite de la seconde.

7. Les termes "sûr", "possible" et "impossible" caractérisent aussi les trois catégories d'objets dans un graphe d'héritage lors d'un filtrage (cf. § 2.1.4), avec une signification légèrement différente. Pour la classification, toute superclasse d'une classe sûre est sûre. Pour le filtrage, toute sous-classe d'une classe sûre est sûre, mais la superclasse d'une classe sûre peut n'être que possible .

- La partie du graphe d'héritage composée de la sous-famille contenant C est ensuite parcourue en profondeur à partir de F. La structure de l'instance I est comparée avec celle de chaque classe de la sous-famille. Le traitement des attributs se fait conformément à une série de règles établissant les contraintes devant être satisfaites par les valeurs des attributs [Pivot and Prokop, 1987] : les types et les valeurs des attributs de I doivent vérifier les contraintes définies dans chaque classe examinée. Si I est compatible avec une classe, celle-ci est marquée avec *sûr*. Si une classe possède un attribut qui n'existe pas dans I, deux cas sont possibles : soit la valeur de cet attribut peut être calculée, soit elle est demandée à l'utilisateur. Si cette valeur ne peut être déterminée dans l'un ou l'autre cas, la classe est marquée avec *possible*, ce qui implique en particulier que les sous-classes de cette classe pourront au mieux être marquées avec *possible*. Dans tout autre cas, la classe examinée est marquée avec *impossible*, de même que ses sous-classes, qui seront ignorées ultérieurement dans le parcours.

- Le processus de spécialisation se termine lorsque la classe C est marquée avec *sûr*, *impossible* ou encore *possible*.

Examinons par exemple comment se déroule la spécialisation de l'instance `Albert` vers la classe `Tri-Athlete`, dans la famille dont le chef est l'objet `Personne` [Pivot and Prokop, 1987]. Au cours du processus de spécialisation, les classes `Poly-Instrumentiste`, `Saxophoniste`, `Pianiste`, `Batteur`, `Musicien` et `Personne` sont marquées avec *sûr*. La classe `Personne`, chef de famille, représente la superclasse commune la plus spécifique à `Poly-Instrumentiste` et à `Tri-Athlete`. L'instance `Albert` peut être spécialisée vers `Tri-Athlete` si et seulement si elle peut l'être vers `Nageur`, `Cycliste` et `Coureur`. L'appariement de l'instance `Albert` avec la classe `Sportif` ne peut réussir que si l'utilisateur répond positivement à la question *Quelle est la valeur de l'attribut pratique-un-sport pour Albert ?* puisqu'il n'existe aucun autre moyen de trouver la valeur de cet attribut (cf. Fig. 3.6). Supposons que ce soit le cas, une question similaire est alors posée pour trouver la valeur de l'attribut `specialite` associé aux classes, dans l'ordre, `Nageur`, `Cycliste` et `Coureur`. L'utilisateur a tout le loisir de demander des informations sur les domaines de valeurs acceptables lorsqu'il a des trous de mémoire. Supposons que les réponses faites aux trois questions précédentes aient été `Nage`, `Velo` et `Course`. Les classes `Nageur`, `Cycliste` et `Coureur` sont alors marquées avec *sûr* et il ne reste plus qu'à vérifier que l'instance `Albert` est compatible avec la classe `Tri-Athlete`. Ceci nécessite de connaître la valeur de l'attribut de type booléen `resistance`. Or, le réflexe si-besoin associé à cet attribut dans la classe `Tri-Athlete` retourne *vrai* si l'instance à laquelle il est appliqué est de type `Saxophoniste`. Comme c'est le cas (ouf !), l'instance `Albert` peut être spécialisée vers la classe `Tri-Athlete`. Si la réponse à l'une des questions posées lors du parcours du graphe n'est pas conforme à ce qui est attendu, la spécialisation de l'instance `Albert` vers la classe `Tri-Athlete` échoue.

La classification en TROPES (cf. § 2.3.3) repose sur des bases sensiblement identiques [Mariño *et al.*, 1990] :

- Toute superclasse d'une classe sûre est sûre.

```
{Sportif
   sorte-de = Personne ;
   pratique-un-sport $un booleen ;
   specialite $liste-de Specialite
              $domaine Nage Velo Course Perche Javelot}

{Nageur
   sorte-de = Sportif ;
   specialite $domaine Nage}

{Tri-Athlete
   sorte-de = Nageur Cycliste Coureur ;
   resistance $un booleen
              $sib-exec {Saxophoniste?
                           valeur1 $var<- est-un ;
                           resultat $var-> resistance}}
```

Figure 3.6. Détail des classes `Sportif`, `Nageur` et `Tri-Athlete`.

- Toute sous-classe d'une classe impossible est impossible.
- Toutes les classes sœurs[8] d'une classe sûre sont impossibles, car une instance ne peut dériver que d'une seule classe dans une perspective.
- Un pont a la même étiquette, sûr, possible ou impossible, quelque soit la perspective considérée. Si une classe dans un pont est marquée sûre, alors toutes les classes du pont deviennent sûres, ce qui permet de déduire rapidement des informations supplémentaires.

Un schéma d'instance ne peut exister si sa classe n'existe pas. Toutefois, contrairement à l'usage en vigueur dans un langage de classes (cf. page 27 et § 3.3.4), il est possible de classer et d'enrichir une instance de la classe la plus générale appelée `objet`, car tout schéma est un prototype, la structure d'une instance n'est pas figée et peut évoluer à tout moment, par adjonction, suppression ou même déplacement de propriétés.

3.3.3 Une approche réflexive et répartie

Dans l'étude qui suit, le processus de classification est envisagée dans le graphe d'héritage d'une représentation à objets réflexive.

8. Classes qui sont "au même niveau" dans la perspective pour la relation d'héritage.

Réduction de l'espace de classification

L'espace de filtrage d'une propriété est généralement identique au graphe d'héritage dans son entier (cf. § 2.1.4). Appelons *espace de classification* l'ensemble des objets qui doivent être explorés pendant le processus de classification, lors de la recherche des subsumants et des subsumés. Comme pour le filtrage, si aucune information n'est disponible, l'espace de classification s'identifie au graphe d'héritage. Il est possible de réduire considérablement cet espace dans une représentation à objets réflexive.

Tout attribut A est représenté par un attribut-objet de même nom A, qui fixe un attribut appelé `maximaux` ayant pour valeur l'ensemble des objets maximaux pour l'attribut (cf. page 46 et figure 2.13). L'espace de filtrage d'un attribut A est déterminé par la réunion des sous-graphes associés à ses maximaux (réunion au sens indiqué à la figure 2.5). Par extension, l'espace de classification d'un objet est défini par la réunion des espaces de filtrage des attributs de l'objet. Par exemple, si un objet O possède les attributs $A1$, $A2$ et $A3$, l'espace de classification de O est constitué par la réunion des maximaux qui fixent $A1$, $A2$ et $A3$ dans le graphe d'héritage. Ces maximaux sont mémorisés dans les attributs objets décrivant $A1$, $A2$ et $A3$. L'utilisation de ces informations réduit considérablement le volume des calculs nécessaires lors du processus de classification.

Tout objet possède naturellement une autre information réflexive qui est la liste de ses attributs, notons la `attributs`. Lorsque cette liste est accessible, il est possible de la trier selon un certain critère (comme en classification interactive, avec une stratégie par exclusion, voir page 104), ou encore de la restreindre à certains attributs importants. Dans le dernier cas, la classification de l'objet est partielle et ne reflète que le point de vue des attributs qui ont été utilisés lors de la classification.

Si l'espace de recherche des subsumants peut être réduit lors du processus de classification, il est également possible de minimiser le volume des calculs lors de la recherche des subsumés les plus généraux. L'ensemble des subsumants les plus spécifiques d'un objet à classer X ne correspond pas forcément à l'ensemble des objets maximaux pour les propriétés de X. Toutefois, si X fixe un nouvel attribut A, alors X est l'unique objet maximal pour A. Dans ce cas, la recherche des subsumés les plus généraux de X devient inutile : aucun objet ne peut dériver de X, car il devrait alors posséder l'attribut A.

Approche répartie et sémantique des facettes

Bien que l'étude qui suit concerne l'héritage, l'approche présentée est générique et peut s'employer avec toute relation d'ordre partiel déterminant une relation de subsomption faible. Les règles qui suivent peuvent être vues comme les spécifications de relations d'ordre partiel associées aux facettes (cf. § 3.2.7).

Lors du processus de classification, les objets sont comparés composant par composant. L'implantation du test de subsomption peut être répartie en un ensemble de méthodes qui sont activées par envoi de message. Une méthode générale, notée `subsumep`, est associée à la racine du graphe d'héritage. Elle est définie de la façon

suivante : un objet *O1* subsume un objet *O2* si chacun des composants de *O1* subsume le composant correspondant de *O2*. Les composants d'un objet sont ses attributs, les composants d'un attribut ses facettes. La méthode `subsumep` est redéfinie pour tous les objets pour lesquels il existe un test spécifique plus efficace. En particulier, attributs et facettes sont des objets à part entière et peuvent donc posséder leur propre méthode `subsumep`. Notons *a1* l'attribut *A* dans l'objet *O1* et *a2* le même attribut *A* dans l'objet *O2*. Le test sur les attributs est défini de façon générale par : l'attribut *a1* subsume l'attribut *a2* si les valeurs associées aux facettes de *a1* subsument celles qui lui correspondent dans *a2*.

Les facettes sont à leur tour prises en compte de la façon suivante :

- Typage (facettes `$un` et `$liste-de`) : le type de *a1* doit subsumer le type de *a2*. Si le type de *a1* est simple, entier, chaîne, booléen, etc., alors celui de *a2* doit lui être identique[9]. Si *a1* définit une relation, alors le co-domaine de *a1* doit subsumer le co-domaine de *a2* ; autrement dit, le co-domaine de *a1* doit être un ascendant du co-domaine de *a2* dans le graphe d'héritage. Les caractéristiques qui viennent d'être explicitées sont valables pour tout attribut, qu'il soit monovalué ou multivalué, puisque la facette `$liste-de`, tout comme la facette `$un`, n'admet qu'un seul argument.

- Restriction de type (facettes `$domaine` et `$intervalle`) : le domaine ou l'intervalle de valeurs associé à *a1* doit contenir celui qui est associé à *a2*.

- Cardinalité (facettes `$min` et `$max`) : l'intervalle de cardinalité associé à *a1* doit contenir l'intervalle de cardinalité associé à *a2* (l'intervalle de cardinalité d'un attribut est borné inférieurement par la valeur de la facette `$min` et borné supérieurement par la valeur de la facette `$ max`).

- Valeur par défaut (facette `$defaut`) : une valeur par défaut pouvant être masquée à n'importe quel niveau de la hiérarchie d'héritage, les valeurs par défaut de *a1* et de *a2* doivent simplement et comme il se doit, respecter les spécifications de type qui leur sont associées.

- Valeur (facette `$valeur`) : une telle valeur est considérée comme fixe. Si *a1* est monovalué, alors la valeur de *a2* doit être identique à celle de *a1*. Toutefois, si *a1* est multivalué et que la cardinalité maximale n'est pas atteinte, la valeur de *a2* peut être complétée en accord avec les spécifications associées à *a2*. Pour que soit respectée la priorité des facettes (cf. § 2.1.3 et figure 2.7), il est vérifié qu'une facette `$valeur` dans *a1* ne subsume pas une facette `$si-besoin` ou une facette `$defaut` dans *a2*.

- Si-besoin (facette `$si-besoin`) : seule facette procédurale à être prise en compte, il est vérifié, pour les raisons évoquées dans le cas ci-dessus, qu'une facette `$si-besoin` dans *a1* ne subsume pas une facette `$defaut` dans *a2*.

9. Les types simples primitifs ne possèdent pas de sous-types.

- Les facettes procédurales `$si-possible`, `$si-ajout`, `$si-enleve` et `$methode` ne sont pas prises en compte.

Les points précédents définissent une sémantique pour les facettes d'un langage de frames. Cette sémantique est proche de celle qui est adoptée dans le langage OBJLOG [Dugerdil, 1988]. Dans ce qui précède, l'attribut *a1* est soit hérité par *O2*, soit redéfini par *a2* dans *O2*. Dans les deux cas, *a1* et *a2* portent le même nom. Il n'existe aucun moyen, sans intervention extérieure, de déterminer que deux attributs *a1* et *a2* de noms différents ont la même sémantique (ou sont synonymes), même si leurs facettes vérifient les conditions précédentes. Ainsi, l'attribut *mère* associé à l'objet *Personne* et l'attribut *épouse* associé à l'objet *Homme* ont le même co-domaine, l'objet *Femme*, et la même cardinalité, mais n'ont aucune relation entre eux.

Le test de subsomption défini sur les facettes est une adaptation pour les représentations à objets du test utilisé dans les systèmes à subsomption. Toutefois, dans ces systèmes, le test s'applique uniformément à l'ensemble des concepts, alors que dans l'approche répartie, il peut être redéfini pour n'importe quel concept ou composant, à partir du moment où l'utilisateur en ressent la nécessité.

3.3.4 Classification dans les langages de classes

Décrire des concepts avec une approche classe-instance consiste à représenter à l'aide de classes les concepts les plus généraux, puis à spécialiser ces classes pour décrire des concepts qui dérivent de ces concepts généraux. Les objets particuliers manipulés dans les applications sont des instances de ces classes. Une classe encapsule des données et des procédures ; les caractéristiques d'une instance ne sont généralement pas visibles depuis l'extérieur, contrairement au modèle des prototypes sur lequel reposent les langages de frames et les représentations à objets considérées jusqu'à présent. Or, pour classer un objet, une instance d'une classe par exemple, il est nécessaire de connaître sa structure pour pouvoir la comparer à celle des objets déjà existants. L'encapsulation, qui est un des principes de base des langages de classes [Snyder, 1987], apparaît donc comme un des obstacles les plus sérieux au raisonnement par classification.

Si classer une instance revient à trouver sa classe d'appartenance, alors une telle opération n'a pas de sens dans un langage de classes, puisqu'une instance ne peut exister que si sa classe a été créée au préalable. Le processus de classification peut éventuellement consister à rechercher les classes partageant une partie de leur structure et de leur comportement avec une classe donnée. Cependant, une telle démarche soulève plusieurs problèmes qui sont difficiles, voire impossibles à résoudre. En particulier, comment reconnaître que deux variables ont la même sémantique, qu'un comportement en spécialise un autre ?

Par ailleurs, la modification dynamique du graphe d'héritage n'est pas autorisée dans les langages de classes compilés comme C++ [Stroustrup, 1989] ou Eiffel [Meyer, 1990], mais elle l'est dans les langages de classes à typage dynamique comme CLOS [Keene, 1989] ou les Flavors [Moon, 1986]. Dans ces langages, une classe peut aussi être redéfinie dynamiquement par adjonction ou suppression de variables. La mise à jour des instances d'une classe redéfinie, de ses sous-classes et de leurs instances est

automatiquement assurée par le système. Lorsque la redéfinition entraîne un changement de structure des instances, la transformation n'est effectivement réalisée qu'à la première opération d'accès effectuée sur les instances. Elle peut s'accompagner d'une intervention de l'utilisateur pour initialiser les nouvelles variables ou modifier la valeur des anciennes par exemple.

Soulignons pour terminer que le traitement d'objets évolutifs est une préoccupation relativement ancienne dans les bases de donnés [Nguyen and Rieu, 1989] [Kim, 1990], et que le modèle des prototypes a souvent été retenu pour représenter des objets évolutifs dans le contexte des classes [Borning, 1986] [Mittal *et al.*, 1986] [Stein *et al.*, 1989].

3.4 Extension de la subsomption

3.4.1 Co-subsomption = subsomption + composition

Un objet composite est formé par l'agrégation d'un certain nombre d'autres objets appelés ses composants, qui décrivent chacun une partie de l'objet composite (cf. § 2.3.2). Les composants sont liés à l'objet composite par une relation *partie de*, de relation inverse *composé de*.

La composition d'objets peut être envisagée sous un angle différent dans une représentation à objets : un objet peut être considéré comme une forme "faible" d'objet composite, dont les composants sont accessibles par l'intermédiaire de ses attributs typés. De tels attributs définissent en réalité des liens de composition, et leur valeur est un ensemble d'objets représentant les composants. Par exemple, une molécule est composée d'atomes et de liaisons ; elle est représentée par l'objet `Molecule`, qui possède deux attributs, `atomes` et `liaisons`, le premier mémorisant la liste des atomes et le second la liste des liaisons de la molécule (cf. § 5.3.3). Cependant, à l'inverse des objets composites standard, une méthode spécifique de création de représentants doit être conçue, qui est chargée de la création et de la mise en place des représentants des composants. Cette façon d'envisager la composition d'objets peut pallier certains des inconvénients liés au formalisme des objets composites présenté au paragraphe 2.3.2 : traitement des cas où le nombre de composants est inconnu à l'avance (typiquement le cas des molécules), séparation explicite de composants de même type existant en plusieurs exemplaires, expression de liens de composition "multiples" (le même atome est à la fois composant d'une molécule et d'une liaison).

Pour gérer explicitement les objets composites faibles et le partage de propriétés avec les composants, nous définissons la relation de *co-subsomption*[10] : un objet *O1* *co-subsume* un objet *O2* si *O1* est un composant de *O2*. Le sens qui est donné ici à "composant" est le premier sens fonctionnel composant-composite décrit à la figure 2.17 [Winston *et al.*, 1987] : *O1* décrit le type d'un composant de *O2*, ce qui se traduit généralement par le fait qu'un représentant de *O2* a pour composant un représentant de *O1*. Même si le sens accordé à la composition est ici le sens fonctionnel, la relation de co-subsomption peut être adaptée pour décrire n'importe quelle autre nuance de la

10. *Composition + subsomption.*

composition.

Comme la relation d'inclusion dont elle dérive, la co-subsomption est une relation d'ordre partiel. Elle est réflexive : O co-subsume O, ce qui signifie, pour nous, que le tout est aussi considéré comme une partie. Elle est anti-symétrique : si *O1* co-subsume *O2* et si *O2* co-subsume *O1*, alors *O1* et *O2* sont "égaux", aux sens où ils ont les mêmes composants, ce qui ne veut pas dire que *O1* et *O2* sont égaux structurellement, au sens où ils ont les mêmes attributs munis des mêmes valeurs. Elle est transitive : si *O1* co-subsume *O2* et si *O2* co-subsume *O3*, alors *O1* co-subsume *O3*. Comme l'ordre induit par la relation d'héritage, l'ordre induit par la relation de co-subsomption n'est pas un ordre total, tous les objets n'étant pas forcément comparables.

La relation de co-subsomption est aussi une relation de subsomption faible. En particulier, la co-subsomption organise les objets en une hiérarchie locale[11] de composition où les objets composants "dominent" l'objet composite. Pour reprendre l'exemple de l'objet `Molecule`, il est possible de définir trois méthodes qui implantent de façon répartie le test de co-subsomption. Ces méthodes sont associées respectivement aux objets `Atome`, `Liaison` et `Molecule` ; elles retournent *vrai* si l'objet auquel elles sont appliquées, atome, liaison, ou sous-structure, fait partie de la molécule étudiée. Une molécule se présente alors comme une hiérarchie locale de composition où les atomes, les liaisons et les sous-structures dominent la molécule dont ils font partie. Pour résoudre un problème de synthèse, seul le fait que certaines sous-structures remarquables co-subsument une molécule a de l'importance (cf. § 5.4) : les sous-structures co-subsumant une molécule `M` déterminent les fonctionnalités chimiques de `M`, ainsi que les réactions applicables à `M`, grâce à un mécanisme de partage de propriétés spécialement adapté à la composition moléculaire (cf. page 194). Cette forme de partage de propriétés peut être étendue au cas plus général des graphes étiquetés.

La hiérarchie locale de composition associée à un objet est construite avec l'algorithme de classification présenté au paragraphe 3.3.1, où la relation de subsomption faible est la co-subsomption. Le processus de classification joue alors son double rôle de raisonnement et d'aide à l'acquisition de connaissances. La remarque faite sur la mise en place des objets dans un graphe d'héritage prend ici un certain sens (cf. § 3.1.2) : la construction d'une hiérarchie locale de composition n'est pas statique (du ressort de l'utilisateur), mais dynamique (effectuée sous la responsabilité du système).

3.4.2 Vers des bases de connaissances n-dimensionnelles

La relation de co-subsomption associée aux structures moléculaires peut être interprétée comme la relation *sous-graphe de* et appliquée à des graphes plus généraux que les graphes moléculaires. Des méthodes particulières de partage de propriétés doivent alors être construites pour gérer le partage d'information entre subsumants et subsumés. Le processus peut se prolonger à une relation d'ordre partiel spatiale comme *à gauche de*, *à droite de*, *devant* ou encore *derrière*, à une relation d'ordre temporelle comme *avant que*, *après que*, etc..

11. Une hiérarchie locale de composition est aussi associée à chaque objet composite dans le système ORION [Kim *et al.*, 1987] [Kim *et al.*, 1989].

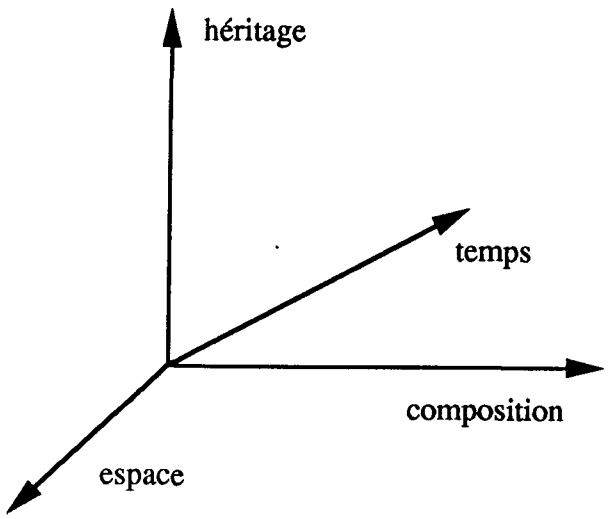


Figure 3.7. Différentes dimensions dans une base de connaissances : héritage, composition, espace, temps.

Chaque relation d'ordre partiel a sa propre sémantique et détermine une relation de subsomption faible, ce qui définit ce que nous appelons une *dimension* dans la base de connaissances. Jusqu'à présent, la dimension la plus connue et la plus étudiée (dans les représentation à objets) est la dimension de l'héritage Cette dimension est "orthogonale" à une seconde dimension, bien connue aussi, la composition. Par extension, une base de connaissances peut être vue comme un espace multi-dimensionnel, un R^n en somme, où chaque dimension décrit une relation (cf. Fig. 3.7).

Une propriété définit une dimension dès lors qu'elle exprime un ordre partiel qui peut se traduire par une relation de subsomption faible. Cette méthode de traitement des propriétés va dans le sens d'une meilleure représentation des liens existant entre concepts : traduire la finesse et la complexité des connaissances détenues par les divers liens dans un réseau de concepts, liens qui ne sont pas toujours que de simples connexions [Johnson-Laird *et al.*, 1984].

3.4.3 Subsomption, relation d'équivalence et perspectives

La relation de subsomption utilisée dans l'algorithme de classification est un ordre partiel. Cette hypothèse fondamentale détermine les caractéristiques de la vision n-dimensionnelle d'une base de connaissances et, en particulier, de l'organisation possible en hiérarchie d'objets dominants et dominés.

Supposons maintenant que la relation considérée soit une relation d'équivalence[12]. Etant donné un objet O, l'algorithme de classification ne permet plus de construire

12. Le seul axiome qui diffère est celui de l'anti-symétrie qui est remplacé par l'axiome de symétrie : si *O1* est en relation avec *O2*, alors *O2* est en relation avec *O1*.

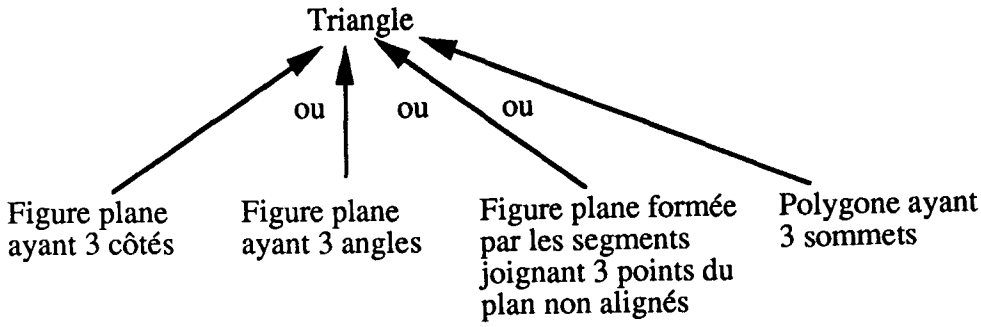


Figure 3.8. Une disjonction de descriptions associées à un concept. Si un objet O satisfait une description de la disjonction, alors il les satisfait toutes.

une hiérarchie, mais plutôt une disjonction d'objets équivalents, qui offrent des points de vue différents sur O. Par exemple, l'objet *Triangle* est représenté par les deux descriptions *figure plane à trois côtés* et *figure plane ayant trois angles*, qui expriment deux points de vue différents mais équivalents sur le concept de triangle (cf. Fig. 3.8).

Une disjonction attachée à un concept rappelle la notion de pont en TROPES (cf. § 2.3.3). Dans le même ordre d'idées, le raisonnement par classification s'appuie alors sur le principe suivant [Ferber and Volle, 1988] [Woods, 1991] : si C est un concept décrit par la disjonction $C1 \cup C2 \cup \ldots \cup Cn$, alors un objet O est un représentant de C s'il satisfait une des descriptions Ci ; réciproquement, si O satisfait une description Ci, il les satisfait toutes.

3.5 Les systèmes à subsomption

3.5.1 La technologie des systèmes à subsomption

Introduction

Il n'existe à l'heure actuelle aucun consensus sur la façon de concevoir des outils de représentation et de manipulation de connaissances. Cependant, un certain courant de recherche s'est créé autour des idées qui sont à la base de KL-ONE pour donner naissance à une famille de langages de représentation, que nous conviendrons d'appeler les *systèmes à subsomption*[13]. Ce courant de recherche actif, original et prometteur,

13. Il n'existe pas (encore) de consensus pour nommer ces langages, qui sont indifféremment qualifiés de *term subsumption languages* [Patel-Schneider *et al.*, 1990], *classification-based systems* ou *term classification systems* [MacGregor, 1991], *terminological logics* [Patel-Schneider, 1989], *term description languages* [Owsnicki-Klewe, 1989] et enfin *hybrid representation systems* [Nebel, 1990].

tente d'unifier des idées qui proviennent des études faites sur les réseaux sémantiques et sur les langages de frames [Nebel, 1990].

Les systèmes à subsomption reposent sur les principes généraux suivants [MacGregor, 1991] :

- La sémantique de la formule décrivant un concept, formule appelée aussi *terme*, est définie à l'aide d'une interprétation, comme c'est le cas pour une formule logique du premier ordre. Dans la suite, le mot "concept" désigne aussi bien le concept de la vie réelle que le terme décrivant ce concept.

- Les systèmes à subsomption dérivent des travaux de R.J. Brachman sur la représentation des connaissances [Brachman, 1977] [Brachman, 1979], qui recommandent de distinguer plusieurs niveaux dans la prise en compte des connaissances. Aussi, il existe une distinction entre le langage de description des concepts ou langage des termes (*TBox*), et le langage des *assertions* (*ABox*), qui permet d'exprimer des faits relatifs aux concepts et à leurs instances[14]. La *TBox* renferme la *connaissance terminologique* qui relève de la représentation des concepts ; la *ABox* renferme la *connaissance assertionnelle* qui relève de la représentation des faits concernant les instances de concepts.

- Les concepts sont ordonnés en une hiérarchie par une relation de subsomption ; la hiérarchie est gérée par un *classifieur* qui place les nouveaux concepts et s'occupe du maintien de la cohérence de la hiérarchie des termes. Les assertions sont traitées par le *reconnaisseur* qui est chargé, entre autres, de la gestion des instances et du maintien de la cohérence des instances interdépendantes.

Quelques systèmes à subsomption

KL-ONE [Brachman and Schmolze, 1985] est le premier système à subsomption digne de ce nom. Le langage de description des concepts est très riche, mais les calculs effectués par le classifieur ne sont pas complets [Schmolze and Lipkis, 1983] (cf. page 95). Les descendants directs de KL-ONE ont formé pendant longtemps le plus gros des troupes dans la famille des systèmes à subsomption. NIKL, pour *New Implementation of KL-ONE*, est une version épurée et améliorée de KL-ONE [Vilain, 1985] [Kaczmarek *et al.*, 1986]. Le classifieur de NIKL est beaucoup plus rapide que celui de KL-ONE, mais il n'est pas complet. Dans KRYPTON apparaît pour la première fois une *ABox*, dont le pouvoir descriptif est équivalent au calcul des prédicats du premier ordre [Brachman *et al.*, 1983] [Brachman *et al.*, 1985]. Cependant, la complexité de la *ABox* et les faiblesses du classifieur posent de nombreux problèmes. KANDOR [Patel-Schneider, 1984] et BACK [von Luck *et al.*, 1987] [Peltason *et al.*, 1989] se présentent comme des versions simplifiées de KRYPTON. Les possibilités de représentation sont plus faibles et le classifieur n'est pas complet, mais toutes les inférences sont réalisées de façon efficace. Les *ABox* de KANDOR et de BACK sont essentiellement réduites aux

14. Lorsqu'il n'y a aucune confusion possible avec les instances d'une classe dans un langage de classes, nous utilisons le mot "instance" pour désigner le représentant d'un concept.

prédicats unaires et binaires clos et deviennent, surtout la seconde, des modèles pour les systèmes à subsomption qui suivent. Des règles et contraintes sont développées pour la première fois dans LOOM [MacGregor, 1988] et MESON [Owsnicki-Klewe, 1988]. Le récent CLASSIC [Borgida *et al.*, 1989] [Brachman *et al.*, 1991] intègre la quintessence des idées qui sont à la base des systèmes de la famille KL-ONE. En particulier, le langage de description des concepts est réduit, le classifieur est complet et l'algorithme sur lequel il s'appuie est de complexité polynomiale. Citons encore le système SB-ONE [Kobsa, 1991] avec lequel est bâti le système de compréhension du langage naturel X-TRA [Allgayer *et al.*, 1989]. Bien que construit selon d'autres principes, le langage OMEGA [Attardi *et al.*, 1986] présente de nombreuses similarités avec les systèmes à subsomption [Attardi, 1991].

Notons que la vogue des systèmes à subsomption a traversé l'Atlantique pour envahir l'Allemagne et s'attaquer avec succès à l'Italie. En revanche, deux autres pays continuent à résister (pour combien de temps encore ?), la France et le Royaume Uni.

Le langage de description des concepts

Rappelons que les entités d'un domaine d'application sont représentées par des objets appelés concepts ou termes. qui sont des conjonctions de rôles exprimant des relations avec d'autres concepts. Chaque rôle peut se voir associer des restrictions qui portent sur son co-domaine et sa cardinalité, à l'instar des facettes déclaratives d'un langage de frames.

Voici par exemple la description en LOOM des concepts *un petit chien ayant trois pattes* et *un père dont toutes les filles sont mariées* [MacGregor, 1991] :

```
(defconcept PC3P
            is (and Chien Petit
                    (exactly 3 pattes)))

(defconcept Pere-heureux
            is (and Pere
                    (all fille Epouse)))
```

Les concepts sont habituellement *primitifs* ou *définis*. Les premiers dénotent de grandes catégories naturelles comme les personnes, les animaux, les plantes, etc.. Ce sont des descriptions qui expriment des conditions nécessaires mais non suffisantes : il n'est pas possible de décider qu'un individu est une instance d'un concept primitif en examinant ses seuls rôles. Les concepts primitifs servent à construire les concepts définis qui sont cette fois des descriptions complètement spécifiées, exprimant des conditions nécessaires et suffisantes d'appartenance d'un individu à l'extension du concept. Ainsi, la définition du concept `Pere-heureux` est équivalente à la formule :
$(\forall$ `x`$)$ `Pere-heureux(x)` $\Longleftrightarrow$ `Pere(x)` et $(\forall$ `y`$)$ `fille(x,y)` $\Rightarrow$ `Epouse(y)`.
Cette formule donne la sémantique de la description du concept `Pere-heureux`.

La subsomption

La définition ensembliste de la subsomption, présentée à la page 94, est adoptée dans la plupart des systèmes à subsomption. Le classifieur détermine les relations de subsomption qui existent à la création des concepts ou encore lors des modifications de concepts adjacents[15].

La relation de subsomption induit un ordre partiel sur l'ensemble des concepts qui est organisé en une hiérarchie comparable à un graphe d'héritage multiple. S'ils ne sont pas redéfinis, les rôles d'un concept subsumant sont partagés avec les concepts subsumés. Le processus de classification permet de mettre en place une nouvelle description dans la hiérarchie grâce à une recherche des subsumants les plus spécifiques et des subsumés les plus généraux de la nouvelle description.

A chaque concept est associé son extension, ou ensemble des individus recouverts par le concept. Toutefois, deux concepts différents n'ont pas forcément des extensions différentes : les concepts *rond carré* et *nombre premier plus petit que 1* sont différents, mais ont la même extension, à savoir l'ensemble vide [Woods, 1991].

Les calculs de subsomption sont généralement consistants mais non complets. Ainsi, toute relation de subsomption détectée est correcte du point de vue de la sémantique de la représentation, mais toutes les relations de subsomption existantes ne sont pas forcément détectées. Ces problèmes sont abordés de façon formelle au paragraphe suivant (cf. § 3.5.2).

Le langage des assertions

Une *assertion* exprime un fait ou une contrainte qui porte sur des concepts. Par exemple, `PC3P(Albert)` est une assertion qui indique que l'individu `Albert` est un petit chien ayant trois pattes. Les assertions prennent la forme de prédicats unaires, comme `PC3P(Albert)`, ou de prédicats binaires, comme `fille(Pierre,Marthe)`, mais sont rarement de complexité supérieure. En réalité, il n'existe aucun consensus au sujet des fonctionnalités que doit recouvrir le langage des assertions, si ce n'est la possibilité d'attacher des contraintes aux concepts [Nebel, 1990].

Les implications de LOOM ou les règles de CLASSIC peuvent être considérées comme des cas particuliers et sophistiqués d'assertions. Elles expriment des conditions nécessaires (mais non suffisantes) : *C1 implique C2* signifie qu'être un individu appartenant à l'extension du concept *C1* implique être un individu appartenant à l'extension du concept *C2*. Une implication ne doit pas être confondue avec une relation de subsomption : une relation de subsomption est d'ordre analytique et reste vraie quelque soit l'interprétation choisie, alors qu'une implication n'est pas forcément vraie dans toutes les interprétations. Ainsi, `(implies Pere-heureux Tranquille)` est une implication qui est vraie pour un individu `I` temporairement, tant que toutes les filles (connues) de `I` sont mariées, état de fait qui cesse dès qu'une des filles de `I` divorce ou que `I` ne devienne à nouveau père d'une fille.

L'ensemble des concepts dont dépend un individu est quelquefois appelé le *type*

15. Deux concepts sont adjacents si la valeur d'un des rôles du premier dépend du deuxième.

de l'individu [MacGregor, 1988]. Un programme spécialisé, appelé *reconnaisseur* ou *ajusteur*[16], a pour but de maintenir la cohérence des types des individus présents dans la base à un instant donné et de mettre à jour les rôles de concepts interdépendants lorsque l'un d'entre eux est modifié. Le reconnaisseur gère des assertions et joue, vis-à-vis des individus, un rôle comparable à celui que le classifieur joue vis-à-vis des concepts.

3.5.2 Une définition formelle de la subsomption

Terminologie et fonction d'extension

Dans ce qui suit, nous donnons la définition formelle d'une *terminologie* et de la subsomption dans une terminologie. Les descriptions de concepts sont formées à partir de concepts primitifs et d'opérateurs de définition de rôles. Voici une grammaire minimale permettant d'engendrer un langage de descriptions de concepts appelé $\mathcal{FL}$ [Brachman and Levesque, 1984] [Nebel, 1988] :

```
<concept> ::= <concept-primitif> |
              (and <concept>+) |
              (all <role> <concept>) |
              (some <role>)

<role> ::= <role-primitif> |
           (restr <role> <concept>)
```

Un concept et ses rôles sont définis à partir d'autres éléments de même nature. Les opérateurs employés sont `and`, qui indique que le concept dérive d'une conjonction de concepts, `all`, qui précise le co-domaine d'un rôle (le type de la valeur du rôle), `some` qui introduit un rôle et `restr` qui associe une restriction à un rôle sous la forme d'une description de concept. Voici les termes qui décrivent les concepts `Homme` et `Grand-parent`, en admettant que `Humain` et `Male` soient des concepts primitifs, et que `enfant` soit un rôle primitif :

```
Homme = (and Humain Male)
Grand-Parent = (and Humain
                    (some (restr enfant
                                 (and Humain (some enfant)))))
```

La définition de `Grand-parent` peut aussi être construite à partir de la définition de `Parent` de la façon suivante :

```
Parent = (and Humain
              (some enfant))

Grand-Parent = (and Parent
                    (restr enfant (and Parent)))
```

16. *Recognizer*.

L'opérateur `all` est comparable aux facettes `$un` ou `$liste-de` d'un langage de frames, `and` est comparable au lien `sorte-de`.

Considérons maintenant un ensemble d'objets D et une fonction $\mathcal{E}$, définie sur l'ensemble des concepts et à valeur dans l'ensemble des parties de D d'une part, et sur l'ensemble des rôles et à valeur dans l'ensemble des parties du produit cartésien D X D d'autre part. $\mathcal{E}$ est une fonction *d'extension* si et seulement si elle possède les propriétés suivantes :

- $\mathcal{E}(C)$ est inclus dans D pour tout concept C.
- $\mathcal{E}(R)$ est inclus dans D X D pour tout rôle R.
- $\mathcal{E}[(and\ C1\ ...\ Cn)] = \{x \in D \mid x \in \mathcal{E}[C1] \wedge ... \wedge x \in \mathcal{E}[Cn]\}$:
  un objet x appartient à l'extension du concept *(and C1 ... Cn)* si et seulement si x appartient à l'intersection des extensions de chacun des concepts *C1*, ..., *Cn*.
- $\mathcal{E}[(all\ R\ C)] = \{x \in D \mid (\forall y) : (x\ y) \in \mathcal{E}[R] \Rightarrow y \in \mathcal{E}[C]\}$:
  l'ensemble des valeurs du rôle R est composé d'individus qui sont des représentants de C.
- $\mathcal{E}[(some\ R)] = \{x \in D \mid \exists\ y : (x\ y) \in \mathcal{E}[R]\}$:
  il existe un rôle R associé au concept considéré.
- $\mathcal{E}[(restr\ R\ C)] = \{(x\ y) \in D\ X\ D \mid (x\ y) \in \mathcal{E}[R] \wedge y \in \mathcal{E}[C]\}$:
  la description du rôle R peut être spécialisée grâce à des contraintes additionnelles décrites par le concept C.

Nous sommes maintenant en mesure de donner une définition formelle de la subsomption [Nebel, 1988] : un concept *C1* subsume un concept *C2* si et seulement si, pour tout ensemble D, domaine de l'extension, et toute fonction d'extension $\mathcal{E}$ définie sur $D : \forall\ t\ :\ t \in \mathcal{E}[C2] \Rightarrow t \in \mathcal{E}[C1]$.

Le langage $\mathcal{FL}$ est "trop riche" pour que les calculs de subsomption soient complets. En revanche, avec le langage $\mathcal{FL}-$, obtenu en supprimant l'opérateur `restr` dans $\mathcal{FL}$, les calculs de subsomption sont complets et de complexité polynomiale bornée par $O(n^2)$, où n est la somme des longueurs des deux descriptions comparées [Brachman and Levesque, 1984].

Le langage $\mathcal{FL}-$ fournit des possibilités de description relativement pauvres. Sans pour autant modifier la complexité des calculs de subsomption, il est possible d'enrichir le pouvoir expressif de $\mathcal{FL}-$, en introduisant deux opérateurs, notés `atleast` et `atmost`, qui généralisent l'opérateur `some`. Ces deux opérateurs permettent de préciser les nombres minimal et maximal de valeurs admises pour un rôle ; ils s'utilisent dans des formules de type `(atleast <entier> <role>)` et `(atmost <entier> <role>)`. C'est d'ailleurs de `atleast` et de `atmost` que dérivent les facettes `$min` et `$max` des langages de frames. Ils sont définis formellement de la façon suivante :

- $\mathcal{E}[(atleast\ Min\ R)] = \{x \in D \mid ||\{y \in D \mid (x\ y) \in \mathcal{E}[R]\}|| \geq Min\}$:
  le nombre minimal de valeurs pour le rôle R est Min ($||\ S\ ||$ dénote la cardinalité de l'ensemble S).

- $\mathcal{E}[(atmost\ Max\ R)] = \{x \in D \mid ||\{y \in D \mid (x\ y) \in \mathcal{E}[R]\}|| \leq Max\}$: le nombre maximal de valeurs pour le rôle R est Max.

Grâce à l'introduction des opérateurs `atleast` et `atmost`, il est possible de construire un terme inconsistant, en l'occurrence `(and (atmost 1 R) (atleast 2 R))`. L'extension de ce terme, noté *T-vide*, est vide : une instance de *T-vide* doit être munie simultanément d'au moins deux valeurs pour le rôle `R` et d'au plus une valeur unique pour `R`. Le terme *T-vide* sert à tester l'inconsistance d'autres termes en vérifiant qu'ils sont subsumés par *T-vide* : si *T-vide* subsume un terme *T?*, alors l'extension de *T?*, qui est contenue dans celle de *T-vide*, est forcément vide.

L'introduction séparée ou simultanée des opérateurs `atleast` et `atmost` ne modifie pas la complexité des calculs de subsomption pour le langage de termes obtenus. Une alternative consiste à enrichir le langage de description de rôles en introduisant l'opérateur `androle` qui permet de définir un rôle relativement à une conjonction de rôles, dans des formules de type `(androle <role> <role>)` (pour simplifier, `androle` est supposé ne prendre que deux arguments). Ainsi, l'opérateur `androle` permet de créer des rôles en spécialisant d'autres rôles, éventuellement primitifs. Il est comparable à l'opérateur `and` pour les concepts. Il est formellement défini par :

$$\mathcal{E}[(androle\ R\ P)] = \{(x\ y) \in D\ X\ D \mid (x\ y) \in \mathcal{E}[R] \wedge (x\ y) \in \mathcal{E}[P]\}.$$

Les rôles sont alors traités comme des concepts et, en particulier, font partie de la hiérarchie des concepts.

Si l'adjonction séparée du couple `atleast-atmost` ou de l'opérateur `androle` ne modifie pas la complexité de $\mathcal{FL}-$, leur introduction simultanée rend les calculs de subsomption incomplets. Malgré ce problème de complétude, la grammaire minimale, appelée aussi *terminologie*[17], sur laquelle est fondée la majorité des systèmes à subsomption, est donnée par :

```
<concept> ::= <concept-primitif> |
              (and <concept>+) |
              (all <role> <concept>) |
              (atleast <entier> <role>) |
              (atmost <entier> <role>)

<role> ::= <role-primitif> |
           (androle <role> <role>)
```

Le langage dérivant de cette grammaire est noté dans la suite $\mathcal{FL}+$.

Complexité de la subsomption dans une terminologie

Il existe plusieurs analyses de la complexité de la subsomption dans des terminologies [Levesque and Brachman, 1987] [Nebel, 1988] [Schmidt-Schauss, 1989]. Nous

17. Cette présentation des terminologies est une simplification (voir [Nebel, 1990] pour une définition précise).

donnons ci-dessous de brefs détails de l'analyse faite pour le langage $\mathcal{FL}+$. Considérons la définition T suivante :

```
(and (atleast 2 R)
     (atleast 2 (androle R Rprim1))
     (atleast 1 (androle R Rprim2))
     (all (androle R Rprim1) (atleast 4 P))
     (all (androle R Rprim2) (atmost 3 P)))
```

Le terme T peut s'interpréter comme le terme décrivant tous les individus qui ont :

- au moins 2 valeurs distinctes pour le rôle `R` ;
- au moins 2 valeurs distinctes pour `R` qui satisfont la définition de **`Rprim1`** et qui ont au moins 4 valeurs différentes pour `P` ;
- au moins une valeur pour `R` qui satisfait la définition de **`Rprim2`** et qui a au plus 3 valeurs différentes pour `P`.

Bien que l'expression `(atleast 2 R)` indique que la cardinalité du rôle `R` est au moins 2, au moins trois valeurs différentes sont nécessaires pour satisfaire les spécifications données ci-dessus : deux valeurs pour le sous-rôle `(androle R Rprim1)` et une pour le sous-rôle `(androle R Rprim2)`. De plus, ces valeurs sont forcément différentes, car elles doivent obéir à des restrictions disjointes[18] : les valeurs du premier sous-rôle vérifient `(atleast 4 P)` et celles du second sous-rôle `(atmost 3 P)`. Un algorithme de subsomption complet doit être capable de détecter une telle relation de subsomption.

Par ailleurs, H.J. Levesque et R.J. Brachman ont montré que détecter des relations de subsomption pour des termes écrits avec le langage $\mathcal{FL}+$ est un problème de complexité équivalente au problème NP-complet qui consiste à établir qu'un ensemble de formules booléennes en forme normale conjonctive est contradictoire [Brachman and Levesque, 1984] [Levesque and Brachman, 1987]. Plus précisément, la subsomption est un problème dont la résolution se réduit polynomialement à un tel problème NP-complet.

Pour sa part, B. Nebel a montré que la subsomption dans $\mathcal{FL}+$ est de complexité équivalente au problème NP-complet suivant [Nebel, 1988] : étant donné une collection C de sous-ensembles d'un ensemble fini S, existe-il une partition de S en deux sous-ensembles *S1* et *S2* telle qu'aucun sous-ensemble de C ne soit entièrement contenu dans *S1* ou *S2* ? Dans la démonstration, le problème de partition est transformé en un test de subsomption entre le terme `(atleast 3 R)` et un terme `S` qui décrit conceptuellement la partition de l'ensemble S.

Dans les deux cas, la subsomption se réduit polynomialement à un problème NP-complet, type de problème que l'on a de bonnes raisons de croire de complexité exponentielle [Garey and Johnson, 1979]. Toutefois, certains chercheurs considèrent que les algorithmes de subsomption généralement utilisés sont suffisamment efficaces dans

18. Le concept `(and (atleast 4 P) (atmost 3 P))` est inconsistant.

la plupart des cas courants, que les contre-exemples invoqués ne sont pas réalistes, et donc, pour finir, que le problème de la complexité de la subsomption est mineur [Attardi, 1991] [MacGregor, 1991].

3.5.3 CLASSIC

Le langage de description des concepts

CLASSIC [Borgida *et al.*, 1989] [Brachman *et al.*, 1991] est le dernier né de la famille KL-ONE. Il a ainsi bénéficié de l'expérience accumulée au cours de la conception des autres descendants de KL-ONE. CLASSIC dérive plus directement de KANDOR [Patel-Schneider, 1984] et, comme son prédécesseur, il possède un langage minimal de description de concepts, de type $\mathcal{FL}+$, et un classifieur efficace. En outre, en utilisant l'unique langage de description des concepts, il est possible de spécifier des requêtes et des règles qui s'activent en chaînage avant. Pour faciliter la représentation de connaissances incomplètes, la représentation des concepts s'appuie sur l'hypothèse du monde ouvert. La qualité des services d'interrogation et de manipulation des concepts est équivalente à celle des services de représentation. En particulier, il est possible d'appréhender d'une façon plus satisfaisante un problème classique en interrogation de bases de données/connaissances : la réponse à une requête ne doit pas être seulement une extension (un ensemble d'individus), mais doit pouvoir être, le cas échéant, de nature descriptive. Par exemple, le système doit être en mesure de fournir des hypothèses descriptives, plus précisément, de donner la description ou l'ébauche d'une description des individus qui satisfont la requête, même si aucun individu du type voulu n'est déjà présent dans la base de données/connaissances.

Une expression en CLASSIC est une composition d'expressions plus simples. La description d'un concept primitif est une des expressions les plus simples. Un tel concept se définit à l'aide du constructeur `PRIMITIVE` et décrit des catégories très générales d'individus. A l'exception de `THING`, qui est le concept le plus général, tout concept dérive d'un concept qui est plus général. En particulier, les termes primitifs qui sont des objets du langage hôte, LISP ou C, dérivent du concept général `HOST`. Le constructeur `DISJOINT-PRIMITIVE` permet, quant à lui, de définir des concepts primitifs disjoints, pour représenter par exemple des concepts indépendants (à l'image des concepts définissant des perspectives, cf. 2.1.2 et 2.3.3). Comme dans tout système à subsomption, la position d'un concept non primitif dans la hiérarchie est induite par la définition du concept et déterminée automatiquement par le classifieur.

Dans la figure 3.9, la première expression définit le concept primitif `Auto`. Classiquement, les concepts primitifs n'expriment que des conditions nécessaires : si `de-dion-bouton-1` est un représentant de `Auto-de-sport`, alors `de-dion-bouton-1` dérive des concepts `Auto` et `Article-de-luxe`, la réciproque n'étant pas forcément vraie ; un représentant des concepts `Auto` et `Article-de-luxe` n'est pas obligatoirement un représentant du concept `Auto-de-sport`. Un concept peut être décrit comme un type énuméré, par l'intermédiaire du constructeur `ONE-OF` qui introduit une liste d'individus qui définissent le type en extension. C'est le cas par exemple pour le concept `Constructeur-francais`. Les restrictions sur les rôles sont également classiques, `ALL` pour

```
(PRIMITIVE THING Auto)
(PRIMITIVE (AND Auto Article-de-luxe Auto-de-sport))
(PRIMITIVE (ONE-OF Peugeot Renault Citroen)
           Constructeur-francais)
(ALL vehicule-conduit Auto)
(AT-LEAST 3 roue)
(AT-MOST 4 roue)
(SAME-AS (conducteur) (assure-principal))

(AND Etudiant
     (ALL vehicule-conduit
          (AND Auto-de-sport
               (ALL constructeur Usine-italienne)))
     (AT-LEAST 1 vehicule-conduit)
     (AT-MOST 2 vehicule-conduit))
```

Figure 3.9. Description en CLASSIC d'automobiles et d'étudiants qui conduisent ces automobiles (d'après [Borgida *et al.*, 1989]).

exprimer des contraintes sur le co-domaine du rôle, `AT-LEAST` et `AT-MOST` pour signifier des contraintes sur la cardinalité du rôle et, moins classique, `SAME-AS`, pour spécifier qu'un rôle a la même valeur qu'un autre rôle. Des opérateurs comme `OR` ou `NOT` n'ont pas été pris en compte : le langage de description des concepts est moins riche, mais en contrepartie, les calculs de relations de subsomption sont beaucoup moins complexes, donc beaucoup plus rapides. Signalons qu'il est également possible de spécifier des éléments de représentation procéduraux sous la forme de fonctions de type `TEST`, qui sont associées à un concept et qui jouent un rôle comparable à celui des réflexes si-possible.

La sémantique d'un concept est donnée par sa dénotation, qui est une fonction définie sur l'ensemble des états de la base et à valeur dans l'ensemble des objets qui satisfont une description conceptuelle dans un état donné. Deux concepts sont équivalents si et seulement si leurs dénotations sont identiques : ils décrivent alors les mêmes individus.

Interactions avec une base de connaissances

Les trois principales interactions avec une base de connaissances sont la définition d'un concept, la mise à jour d'un concept et la spécification d'une requête. Ces trois opérations peuvent quelquefois être exécutées simultanément. Les opérateurs `define-concept` et `define-role` définissent un concept et un rôle respectivement, l'opérateur `create-ind` crée un individu : l'individu `Rocky` est de type `THING` tant que

```
define-concept [Gosse-de-riche
                (AND Etudiant
                       (ALL vehicule-conduit Auto-de-sport)
                       (AT-LEAST 2 vehicule-conduit))]

define-concept [Etudiant
                (AND Personne
                       (AT-LEAST 1 Inscription))]

 (a) create-ind [Rocky]
 (b) assert-ind [Rocky (FILLS vehicule-conduit volvo-17)]
 (c) assert-ind [Rocky (CLOSE vehicule-conduit)]

 (d) assert-ind [Rocky Personne]

 (e) assert-ind [Rocky (AT-LEAST 1 vehicule-conduit)]
 (f) assert-ind [Rocky (ALL vehicule-conduit Auto-de-sport)]
 (g) assert-ind [Rocky (ALL vehicule-conduit
                              (ALL constructeur (ONE-OF Ferrari)))]

 (h) assert-ind [Rocky (FILLS inscription Nancy-1)]

 (i) assert-ind [Rocky (SAME-AS (passion) (vehicule-conduit))]
 (j) assert-ind [Rocky (AT-MOST 1 vehicule-conduit)]
```

Figure 3.10. La définition des concepts `Gosse-de-riche`, `Etudiant` et celle de l'individu `Rocky` (d'après [Borgida *et al.*, 1989]).

rien d'autre n'est précisé (cf. Fig. 3.10).

L'affinement de la description d'un individu s'effectue de façon incrémentielle grâce à l'opérateur `assert-ind`. L'expression `(d)` fait dériver `Rocky` du concept `Personne` *a posteriori*. L'opérateur `FILLS` donne une valeur à un rôle et `CLOSE` déclare que le rôle est fermé (il ne peut plus recevoir de valeur). L'expression `(c)` ferme le rôle `vehicule-conduit` associé à `Rocky`. Toutefois, la prise en compte simultanée des expressions `(e)` et `(j)` est un autre moyen de déclarer que le rôle `vehicule-conduit` est fermé, sans employer explicitement l'opérateur `CLOSE`.

Les objets associés à un concept peuvent être décrits sans être jamais nommés : en supposant que les informations données par `(a)` et `(b)` soient caduques[19], les expressions `(e)`, `(f)` et `(g)` décrivent l'ensemble des caractéristiques de la voiture de `Rocky`,

19. En réalité, par souci de simplicité, aucun opérateur de suppression de concepts, d'individus ou de rôles, n'a été prévu.

sans que l'objet associé à cette voiture ne soit nommé explicitement. Lorsqu'une nouvelle information est apportée à la description d'un individu, il est vérifié que la description résultante est en accord avec les contraintes exprimées dans les concepts dont il dérive.

Les expressions `(d)` et `(h)` permettent de déduire que l'individu `Rocky` est de type `Etudiant`, et ainsi de répondre à des requêtes qui pourraient découler de son appartenance à l'extension du concept `Etudiant`.

Définition de règles

Une règle exprime une connaissance dynamique du type : si un individu est un représentant du concept *C1*, alors il est aussi un représentant du concept *C2*. Une règle est attachée à un concept et elle fournit une information additionnelle qui apparaît comme une conséquence de l'appartenance d'un individu à l'extension d'un concept. Une telle information est bien additionnelle et elle ne doit pas être possédée par l'individu pour dériver du concept considéré. Ainsi, l'assertion :

```
assert-rule [Etudiant (ALL mange Restauration-de-qualite-mediocre)]}
```

est une règle qui indique que tout ce que mange un étudiant est de qualité médiocre. Autrement dit, tout représentant du concept `Etudiant` est aussi un représentant de la catégorie des individus dont la restauration est de qualité médiocre. Cette règle est déclenchée chaque fois qu'une nouvelle instance du concept `Etudiant` est classifiée dans la hiérarchie des concepts.

Les règles de CLASSIC sont comparables aux implications de LOOM. Comme pour une implication, définir une règle comme la précédente n'est pas équivalent à ajouter l'expression `(ALL mange Restauration-de-qualite-mediocre)` au concept `Etudiant`, ce qui reviendrait à ne reconnaître un individu comme une instance du concept `Etudiant` que s'il possède un rôle `mange` muni de la valeur `Restauration-de-qualite-mediocre`, ce qui n'est (évidemment ?) pas une condition nécessaire pour être représentant du concept `Etudiant`.

Une règle établit les conséquence de l'appartenance d'un individu à la catégorie dénotée par un terme et s'apparente plutôt à un réflexe si-ajout. Lorsqu'un individu est reconnu comme étant le représentant d'un concept, toutes les règles attachées au concept sont déclenchées en chaînage avant, ce qui produit éventuellement une cascade de déclenchements : l'appartenance d'un individu à une catégorie déclenche un règle qui stipule l'appartenance de l'individu à une nouvelle catégorie à laquelle est attachée une autre règle, et ainsi de suite.

Les requêtes

Une requête est une demande d'information qui concerne un concept C ou bien ses rôles. Une des requêtes principales consiste à rechercher l'ensemble des concepts dont dépend C selon un certain "point de vue". Il est possible de se demander par exemple quels sont les concepts qui sont les subsumants les plus spécifiques de C, ou

bien, les subsumés les plus généraux de C. L'opérateur `concept-subsumes[C1 C2]` retourne vrai si `C1` subsume `C2`, ou encore, si et seulement si, dans tout état de la base, tout individu qui satisfait la description de `C2` satisfait aussi celle de `C1`. Par extension, un concept `C1` est équivalent au concept `C2` si `concept-subsumes[C1 C2]` et `concept-subsumes[C2 C1]` sont simultanément vrais.

Les requêtes concernant les rôles sont de même type que les requêtes utilisées dans une base de données relationnelle [Brodie *et al.*, 1984]. En particulier, modulo l'hypothèse du monde ouvert[20], une base de connaissances CLASSIC peut être assimilée à une base de données relationnelle si un concept est assimilé à une relation unaire et un rôle à une relation binaire. Cependant, contrairement à l'usage en vigueur dans l'univers des bases de données, il n'existe pas de langage de requête séparé de type SQL. Toute expression du langage CLASSIC peut être interprétée comme une requête. Par exemple, l'expression `E = (AT-LEAST 2 vehicule-conduit)` recouvre tous les individus qui sont décrits par un concept dont la définition inclut `E`. Ces mêmes individus satisfont la requête `E`.

Une requête peut aussi être "ouverte" :

```
?: Personne

(AND Etudiant
     (ALL vehicule-conduit
          ?: (ALL constructeur (ONE-OF Ferrari)))

(AND Etudiant (ALL mange ?: THING))
```

La première requête retourne tous les individus de type `Personne`, la seconde, tous les individus de type `Etudiant` conduisant un véhicule dont le constructeur est `Ferrari`, la troisième, tout ce qu'est susceptible de manger un individu de type `Etudiant`.

La plupart des réponses à une requête nécessite le calcul de relations de subsomption [Resnick *et al.*, 1990] : la requête est d'abord considérée comme un concept, normalisée, puis, par classification, elle est insérée dans la hiérarchie des concepts ; les individus satisfaisant la requête sont des instances des subsumants les plus spécifiques de la requête. De cette façon, seuls des individus satisfaisant la *forme* de la requête sont testés.

3.5.4 LOOM

Comme dans les autres systèmes à subsomption, les expressions conceptuelles en LOOM recouvrent les définitions de concepts et de rôles ; les assertions expriment des faits vérifiés par les instances des concepts. Les opérateurs `primitive` et `defconcept` jouent le même rôle que leurs homonymes en CLASSIC. L'opérateur `defrelation` sert à définir un rôle qui peut être annoté par les opérateurs `all`, `at-least`, `at-most` et enfin `exactly` pour spécifier le nombre exact de valeurs du rôle (cf. Fig. 3.11). Des

20. L'hypothèse du monde clos est généralement en vigueur dans une base de données relationnelle.

```
(defconcept Personne primitive)
(defconcept Male (and Personne primitive))
(defconcept Femelle (and Personne primitive))
(defconcept Epoux-se (and Personne primitive))

(defrelation enfant primitive
             (implies (domain Personne) (range Personne))
             closed-world)

(defrelation fille
             (and enfant (range Femelle)))

(defconcept Pere
            (and Male (at-least 1 enfant)))

(defconcept Pere-heureux
            (and Pere (all fille Epoux-se)))

(A1) : (tell (Epoux-se Pierre))
(A2) : (tell (enfant Pierre Marthe))
(A3) : (tell (Male Pierre))
(A4) : (tell (Femelle Marthe))
(A5) : (tell (Epoux-se Marthe))
(A6) : (forget (Epoux-se Marthe))
(A7) : (tell (Pere-heureux Pierre))
```

Figure 3.11. Quelques concepts définissant, en LOOM, les éléments d'une famille et les liens qui existent entre eux (d'après [MacGregor, 1988]).

contraintes additionnelles s'expriment par l'intermédiaire des opérateurs `domain`, pour décrire le domaine d'un rôle, `range`, pour décrire le co-domaine d'un rôle et `inverse` pour spécifier un "rôle inverse", équivalent d'une relation inverse. Ces opérateurs ressemblent aux facettes d'un langage de frames dans la forme et la fonction.

L'opérateur `implies` sert à définir des implications qui sont comparables aux règles de CLASSIC. L'annotation `closed-world` indique que l'hypothèse du monde clos doit être prise en compte localement, pour un rôle. En complément, les connecteurs `and`, `or` et `not` servent à construire n'importe quelle expression booléenne composée. Les nombreux rôles qui viennent d'être présentés montrent que LOOM est beaucoup plus riche que CLASSIC de ce point de vue : en CLASSIC, il n'existe pas d'opérateurs de disjonction ou de négation et, en outre, la suppression d'informations n'est pas autorisée.

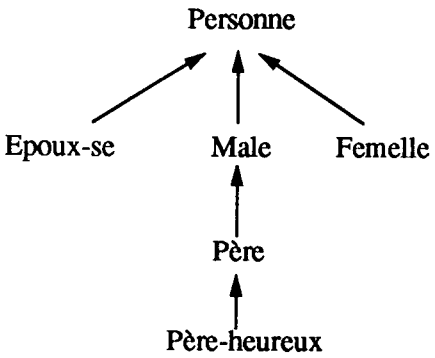


Figure 3.12. Visualisation de la hiérarchie des concepts définis en LOOM à la figure 3.11.

Classification et ajustement de type

Lors du placement d'un concept C dans la hiérarchie, les propriétés sur lesquelles s'appuie le classifieur de LOOM sont classiques [MacGregor, 1988] :

- Les concepts primitifs ne sont pas testés.
- Un concept D n'est pas testé si un de ses descendants subsume C.
- Un concept D n'est testé que si au moins un de ses ascendants subsume C.

Le *type* d'une expression conceptuelle E est l'ensemble des concepts qui subsument E. L'ensemble des subsumants les plus spécifiques de E constitue la "valeur externe" du type ou type réduit de E. Par exemple, le type de l'expression `(and Epoux-se Personne Femelle)` est le triplet (`Epoux-se`[21] `Personne Femelle`) et le type réduit est le couple (`Epoux-se Femelle`) (cf. Fig. 3.12). Le type d'une instance I est l'ensemble des concepts desquels I dérive. Plus précisément, I appartient à l'intersection des extensions des concepts faisant partie de son type. Trouver le type d'une instance I revient donc à rechercher tous les concepts les plus spécifiques dont I est susceptible de dériver.

L'évolution temporelle d'une base de connaissances implique une mise à jour ou *ajustement* constant du type des individus présents dans la base. Cette tâche, comparable au maintien de la cohérence d'une base de connaissances, est prise en charge par le reconnaisseur (ou ajusteur). Trois sortes d'opérations nécessitent le réajustement du type d'un individu :

- Mise en place avec l'opérateur `tell`, ou suppression avec l'opérateur `forget`, d'une assertion portant sur I (le dernier opérateur est une spécialité de LOOM).
- Modification de la valeur d'un rôle attaché à I.

21. `Epoux-se` : il faut bien contenter tout le monde !

- Modification du type d'une instance adjacente à I (instance en relation avec I par l'intermédiaire d'un rôle).

Le reconnaisseur joue pour les assertions un rôle symétrique à celui que joue le classifieur pour les termes. Le classifieur exploite la hiérarchie de haut en bas, du général au spécifique, alors que le reconnaisseur travaille de bas en haut, du spécifique au général. Plus précisément, la stratégie qu'emploie le reconnaisseur pour traiter les assertions s'articule généralement autour de trois étapes :

- Propagation ("en chaînage avant") : chaque fois qu'une nouvelle instance est introduite, ou qu'une instance est modifiée, les modifications qui en résultent sont propagées aux concepts adjacents à l'instance modifiée.

- Abstraction : une abstraction A, qui décrit l'instance I dont il faut ajuster le type, est créée. L'abstraction A est construite directement à partir de la description de I, comme dans NIKL ou BACK, ou de façon incrémentielle comme dans LOOM. Dans ce dernier cas, il y a un échange continu d'informations entre A et I, jusqu'à ce que le nouveau type ait été calculé.

- Classification : l'abstraction A est classifiée dans la hiérarchie des concepts. L'ensemble des subsumants de A constituent le type de I. Si A est une abstraction complète, alors la classification de A s'effectue avec les informations disponibles sur A, sinon, la classification de A est incrémentielle et se fait parallèlement à l'enrichissement de A ; la classification progresse grâce à à l'activation de requêtes chargées de glaner des suppléments d'information en vue d'affiner la mise en place de A dans la hiérarchie des concepts.

En LOOM, la construction de l'abstraction A se fait à partir de la description des concepts qui subsument I au moment où le reconnaisseur commence l'ajustement du type de I. Ces subsumants sont mémorisés dans une liste qui est notée `TYPE` (le type de I) et qui est attachée à A. Deux autres listes, notées respectivement `HITS` et `MISSES`, mémorisent les questions qui reçoivent respectivement une réponse positive et négative lors des ajustements précédents de I. A chaque traitement d'instance adjacente à I, les listes `HITS` et `MISSES` attachées à I sont consultées : si les réponses aux questions contenues dans `HITS` restent vraies et celles contenues dans `MISSES` restent fausses, alors rien ne se passe, sinon le type de I doit être ajusté.

Explicitons le fonctionnement du reconnaisseur en examinant les 7 assertions données à la figure 3.11. L'assertion `A1 : (tell (Epoux-se Pierre))` provoque la construction d'une abstraction décrivant `Pierre` :

```
(defconcept Pierre (and Epoux-se primitive))
TYPE : (Epoux-se)
HITS : ()
MISSES : ()
```

Le type de `Pierre` est mis en place dans `TYPE`, mais aucune question n'est posée au sujet de ce type. L'assertion `A2 : (tell (enfant Pierre Marthe))` provoque la mise

en place du type de Marthe, en l'occurrence Personne, grâce à l'implication associée au rôle enfant. Par ailleurs, aucun réajustement de type n'est effectué. L'assertion A3 : (tell (Male Pierre)) provoque la modification de l'abstraction décrivant Pierre en :

```
(defconcept Pierre (and Male Epoux-se primitive))
TYPE : (Pere Epoux-se)
HITS : ((at-least 1 enfant))
MISSES : ((all fille Epoux-se))
```

Le classifieur inspecte l'instance Pierre pour savoir si la réponse à la question (at-least 1 enfant) est vraie, puisque cette propriété est associée à Pere, subsumé immédiat de Male. La réponse étant positive, le type de Pierre est ajusté et la valeur de HITS attachée à Pierre est mise à jour. Ensuite, le classifieur essaie de savoir si Pierre peut être une instance de Pere-heureux, subsumé immédiat de Pere. Il pose donc la question (all fille Epoux-se) qui reçoit une réponse négative.

L'assertion A4 : (tell (Femelle Marthe)), provoque la mise à jour du type de Marthe qui devient Femelle. Les instances adjacentes à Marthe sont "prévenues" de ce changement de type. Par suite, Marthe devient la fille de Pierre, mais aucune réponse n'est donnée à la question (all fille Epoux-se), le type de Pierre n'est donc pas modifié.

L'assertion A5 : (tell (Epoux-se Marthe)) provoque un ajustement du type de Marthe en (Femelle Epoux-se). La réponse à la question (all fille Epoux-se) est positive, ce qui, en cascade, implique un ajustement du type de Pierre :

```
(defconcept Pierre (and Male Epoux-se primitive
                        (all fille Epoux-se)))
TYPE : (Pere-heureux Epoux-se)
HITS : ((at-least 1 enfant) (all fille Epoux-se))
MISSES : ()
```

L'assertion A6 : (forget (Epoux-se Marthe)), qui est une suppression de fait, provoque un retour des choses à l'état de l'étape A4. La modification du type de Marthe remet en cause une des réponses positives contenue dans la liste HITS associée à Pierre.

L'assertion A7 : (tell (Pere-heureux Pierre)), donne un exemple de déduction "en chaînage avant". Le type de Pierre est modifié, ce qui provoque également la modification du type des instances qui sont adjacentes à Pierre, en l'occurrence le type de Marthe retourne à l'état qui était le sien en A5. Toutefois, puisque Pere-heureux a été associé directement à Pierre et que Pere-heureux implique Pere, le classifieur ne pose aucune question sur les rôles associés à Pierre.

3.6 L'apport des systèmes à subsomption

Comme celle des langages de frames, la technologie de représentation des systèmes à subsomption repose sur le rejet d'une approche de la représentation des connaissan-

ces qui soit purement logique [Minsky, 1975]. En ce sens, et par la forme des entités qui décrivent des concepts dans l'une et l'autre famille, les systèmes à subsomption sont très proches des langages de frames et des représentations à objets. Cependant, un langage de frames classique tire l'essentiel de son pouvoir expressif et déductif du mécanisme d'héritage, qui permet le partage de propriétés en propageant les caractéristiques d'un frame à l'autre. Dans un système à subsomption, toutes les contraintes qui s'appliquent à un concept sont propagées à ses subsumés, comme si ces derniers "héritaient" ces contraintes. Le partage de propriétés ne constitue qu'une facette de la classification, les autres recouvrant essentiellement la mise en place des objets dans une hiérarchie et le maintien de la cohérence de la hiérarchie.

Un des mérites de la technologie des systèmes à subsomption est d'avoir mis l'accent sur l'existence de différents niveaux de connaissances, qui doivent se refléter dans une représentation et dans le langage utilisé pour construire cette représentation. La *TBox* recouvre la définition des descriptions conceptuelles. Le classifieur est le gestionnaire de la *TBox* : insertion des nouveaux concepts et maintien de la cohérence de la hiérarchie des concepts. La *ABox* recouvre l'expression des assertions, généralement sous la forme de termes unaires ou binaires clos. Le reconnaisseur est le gestionnaire de la *ABox*. Il est chargé du traitement des assertions, de la mise à jour du type des individus considérés dans les assertions et du maintien de la cohérence des individus interdépendants. Comme dans certaines représentations à objets hybrides, il n'est pas rare qu'un niveau particulier de langage serve à exprimer des règles ou des implications, mais les avis restent partagés en ce qui concerne la suppression de faits.

De nombreuses études ont prouvé que seuls des langages de description de concepts "pauvres", de type $\mathcal{FL}-$, ont un classifieur consistant et complet, d'où le fameux dilemme : faut-il avoir un classifieur consistant et complet avec un langage de description de concepts limité – ce qui est petit est toujours plus joli – ou bien un langage de description riche et un classifieur théoriquement incomplet ? L'expérience prouve que les utilisateurs préfèrent disposer d'un langage riche, aux possibilités nombreuses. Les impératifs ne sont pas les mêmes pour tous : le fait que le classifieur est forcément incomplet à partir d'un certain nombre de connecteurs est un argument théorique qui intéresse les concepteurs de systèmes à subsomption, mais qui n'intéresse pas les utilisateurs désirant construire un système destiné à devenir un produit industriel. Il vaut alors peut-être mieux que certaines défaillances soient dues à l'incomplétude du classifieur, plutôt que de travailler avec un langage de descriptions de concepts trop pauvre[22].

Par ailleurs, l'opération qui en réalité a le plus d'importance est celle qui est liée à l'ajustement du type des individus. Cette opération est consistante et complète si le système travaille avec des connaissances *caches*[23] [Levesque, 1986b] [Etherington *et al.*, 1989]. Les connaissances caches déterminent un nouveau niveau dans une base de connaissances, qui peut être alors vue comme un empilement de niveaux de repré-

22. C'est souvent à ce moment là que l'utilisateur décide de construire son propre système, dont les capacités dépassent rarement celles du système à subsomption qu'il utilisait au départ.

23. Ce terme est une traduction possible pour *vivid knowledge*, qui désigne un ensemble de termes clos en rapport avec l'univers représenté et situé dans une mémoire "à court terme".

sentations, allant du plus simple au plus complexe. La richesse de la description des concepts d'un niveau est en rapport avec la richesse du langage de description associé à ce niveau [MacGregor, 1991]. Le comportement du classifieur dépend du niveau des concepts manipulés : un concept de première classe se décrit avec les connecteurs de $\mathcal{FL}-$ (niveau de description où le classifieur est complet) et n'est en relation qu'avec des concepts de première classe ; les autres concepts sont de seconde classe. Un utilisateur connaît alors la qualité des inférences produites, qui est directement en rapport avec la classe des concepts manipulés. Les opérations de classification et de réajustement effectuées sur les concepts de première classe sont complètes, par définition de cette classe. Les concepts de seconde classe sont exploitables sans problème de complétude dans toute requête ne nécessitant pas de classification. Cette approche de la représentation de concepts peut être qualifiée de "conviviale", puisque son but est de satisfaire le maximum d'utilisateurs, en toute connaissance de cause.

4
La catégorisation

L'interprétation de grands ensembles de données est une activité scientifique courante, en particulier dans les sciences expérimentales : classification du règne animal en zoologie, classification du règne végétal en botanique, classification périodique des éléments en chimie... Une telle interprétation, appelée *catégorisation*, consiste à décrire et à expliquer des données en les distribuant en un nombre restreint de classes homogènes regroupant des objets ayant des caractéristiques et un comportement similaires. Deux approches sont classiquement employées pour catégoriser un ensemble d'objets : l'approche numérique qui comprend l'analyse de données, l'approche symbolique qui s'appuie sur des techniques d'intelligence artificielle et qui comprend, entre autres, la catégorisation conceptuelle et divers courants de l'apprentissage symbolique. La seconde approche doit beaucoup à la première, les liens existant entre les deux ayant d'ailleurs tendance à se resserrer, comme le montre l'émergence actuelle d'une approche mixte symbolique-numérique.

Ce chapitre illustre les liens qui existent entre la catégorisation, le raisonnement par classification et les techniques d'apprentissage symbolique. En particulier, il est d'abord fait un parallèle entre le raisonnement par classification et la formation de hiérarchies de concepts, puis entre le raisonnement par classification et le raisonnement par cas. Ce dernier, qui repose sur l'analogie et, d'une certaine façon, sur le principe de remémoration, intègre une importante phase d'apprentissage.

4.1 L'approche numérique : l'analyse de données

Les méthodes d'analyse de données permettent l'étude globale d'un ensemble de données décrites par un grand nombre de caractères essentiellement numériques. Dans la suite, nous nous intéressons aux méthodes employées en *classification automatique*, qui consistent à analyser et à catégoriser des données avec ou sans point de vue hiérarchique. Les méthodes non hiérarchiques, appelées aussi méthodes de partitionnement, produisent directement une partition en un nombre fixé de classes. Les méthodes hiérarchiques produisent des partitions en classes de plus en plus spécialisées, à l'image des graphes d'héritage. Ces méthodes de classification automatique permettent de construire des classes d'équivalence d'objets suivant les valeurs de leurs attributs. Elles ont de nombreuses applications, comme la segmentation de nuages de points selon certains

critères, la reconnaissance de formes [Duda and Hart, 1973], etc..

4.1.1 La catégorisation non hiérarchique

La catégorisation non hiérarchique consiste à regrouper n individus en k classes, le nombre de classes étant généralement connu à l'avance [Lerman, 1970] [Bouroche and Saporta, 1987]. Les individus d'une classe doivent se ressembler le plus possible et les classes doivent être séparées le plus possible. Pour ce faire, il est nécessaire de disposer d'une *distance* qui discrimine les individus afin de les partitionner en classes, ainsi que d'un critère global mesurant la qualité de la partition obtenue. La distance inter-individus se définit en fonction d'attributs jugés discriminants. Un attribut est d'autant plus discriminant qu'il induit une catégorisation plus fine ; la séparation entre classes est plus importante ou bien le nombre de classes produites est plus élevé. Lorsqu'un individu est décrit par un vecteur contenant les valeurs de ses attributs, il peut être vu comme un point dans un espace multi-dimensionnel, chaque dimension correspondant à un attribut. La position des points dans l'espace détermine la distance qui les sépare. Cette pratique relève de la géométrie euclidienne et la catégorisation se décrit comme la recherche d'une partition d'un nuage de n points en k sous-nuages.

La moyenne des carrés des distances des points au centre de gravité du nuage, appelée *inertie*, est généralement utilisée pour mesurer la dispersion des points. Une classe est d'autant plus homogène que son inertie est faible. La somme des inerties d'un ensemble de classes, l'inertie étant calculée par rapport au centre de gravité de chaque classe, s'appelle inertie *intraclasse*. Un ensemble de classes est d'autant plus homogène que son inertie intraclasse est faible. L'inertie des centres de gravité de chaque classe, calculée par rapport au centre de gravité du nuage, est appelée inertie *interclasse*. Plus cette inertie est élevée, plus les classes sont séparées. Par ailleurs, la somme des inerties intraclasse et interclasse est constante : elle correspond à l'inertie *totale* du nuage de n points, qui est calculée comme la moyenne des carrés des distances de chaque point au centre de gravité du nuage. Ainsi, minimiser l'inertie intraclasse revient à maximiser l'inertie interclasse. Les inerties intraclasse et interclasse servent de critères globaux pour évaluer la qualité d'une partition.

La plupart des techniques de catégorisation non hiérarchiques procèdent par améliorations successives d'une partition initiale. Par exemple, la technique de *regroupement autour de centres mobiles* consiste à partager un ensemble d'individus en k classes, d'abord en fonction de la distance des individus à k centres choisis au départ, puis en fonction des distances aux centres de gravité des classes obtenues. Le processus est itéré, les centres étant modifiés jusqu'à ce que la catégorisation soit stable, c'est-à-dire qu'un optimum local au sens d'un critère choisi soit obtenu. La méthode des *nuées dynamiques* de E. Diday en est une généralisation [Diday *et al.*, 1982]. Une classe n'est alors plus définie par un seul point, mais par un noyau composé d'un nombre fixé d'individus jugés représentatifs de la classe. Les noyaux jouent le rôle de centres de gravité et servent en outre à interpréter les classes obtenues (cf. Fig. 4.1).

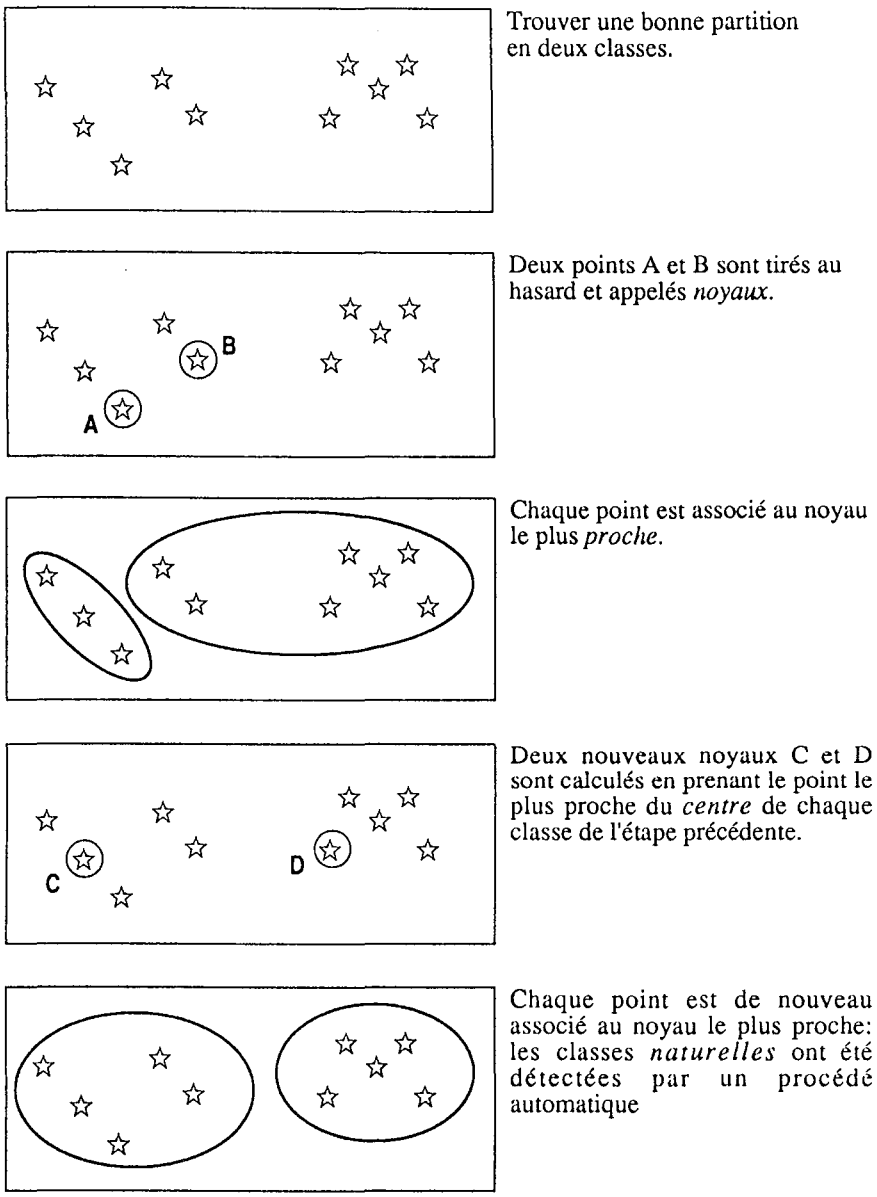


Figure 4.1. Un exemple de recherche de classes reposant sur la technique des nuées dynamiques.

4.1.2 La catégorisation hiérarchique

La catégorisation hiérarchique permet de regrouper un ensemble d'individus en classes, classes qui sont à leur tour organisées en une hiérarchie [Benzecri, 1973]. Le nombre de classes devant être obtenues n'est pas forcément donné à l'avance. Dans ce contexte, les deux techniques classiques de discrimination d'un ensemble d'individus sont la *division* et l'*agrégation*. La division est descendante : l'ensemble des objets initiaux est envisagé comme une classe unique qu'il faut partitionner en classes de plus en plus spécifiques, qui sont partiellement ordonnées en une hiérarchie [Rao, 1971].

L'agrégation est une technique ascendante, et opère par fusions successives de l'ensemble des objets initiaux, qui sont considérés au départ comme autant de classes. La hiérarchie est généralement une arborescence construite de façon incrémentielle : une partition en k classes est obtenue en regroupant en une seule classe deux classes faisant partie de la partition en $k + 1$ classes. Un des problèmes principaux consiste à définir le critère de regroupement de deux classes. Lorsque les individus sont vus comme des points dans un espace euclidien, il est possible de fusionner les deux classes pour lesquelles la perte d'inertie interclasse est la plus faible. D'autres types de distance peuvent être utilisés, ayant chacun avantages et inconvénients. Parmi les plus courants, citons la distance du saut minimal, qui mesure la plus petite des distances entre les éléments pris deux à deux, la distance du diamètre, qui mesure la plus grande de ces distances, et la distance de la moyenne, qui mesure la moyenne éventuellement pondérée des distances entre les éléments pris deux à deux.

Les méthodes d'analyse de données produisent d'excellents résultats dans les contextes numériques, sans avoir recours à des connaissances externes sur le domaine étudié. Elles sont cependant limitées dès lors qu'il s'agit de traiter des individus décrits par des critères qualitatifs, ou encore de personnaliser le traitement des différents attributs discriminants. Les catégories recherchées ne sont jamais considérées comme des entités pour elles-mêmes et, la plupart du temps, aucune explication ni description globale des classes obtenues n'est élaborée, l'interprétation des résultats étant laissée à la charge de l'utilisateur.

4.2 L'approche symbolique

4.2.1 Un algorithme de catégorisation conceptuelle

L'approche numérique est relativement éloignée de la façon de raisonner d'un être humain, qui, comme nous l'avons vu au chapitre 1, se sert plutôt de prototypes pour repérer des catégories : les objets sont alors partitionnés en fonction des attributs qu'ils partagent, et non pas en fonction des valeurs des attributs qu'ils possèdent. L'approche symbolique, ou *catégorisation conceptuelle*[1], consiste à regrouper des objets, non seulement parce qu'ils sont proches au sens d'une certaine distance, mais aussi parce qu'ils matérialisent l'extension d'un certain concept lorsqu'ils sont considérés en groupe. La catégorisation dépend généralement d'un objectif, qui peut être atteint en s'appuyant sur des connaissances relatives au domaine étudié. Les objets sont discriminés en fonction des propriétés qu'ils partagent mais aussi en fonction des connaissances disponibles. Par exemple, un système de reconnaissance de caractères partitionne un ensemble de points selon leur distribution topologique, mais aussi selon leur appartenance à un groupement de points susceptible de représenter une partie d'une lettre connue.

R.S. Michalski et R.E. Stepp ont établi une théorie et présenté un algorithme de catégorisation conceptuelle, dont le but est de trouver la catégorisation qui obtient

1. *Conceptual clustering*.

le meilleur score au sens d'un critère global de qualité [Michalski and Stepp, 1983] [Michalski and Stepp, 1984] [Stepp and Michalski, 1986]. Un tel critère est plus difficile à définir que dans un contexte numérique, car il dépend cette fois de paramètres qualitatifs. Il peut être établi à partir d'informations comme :

- le nombre d'attributs communs partagés par les individus d'une classe, qui est à rapprocher de l'inertie intraclasse ;
- le nombre d'attributs discriminants, qui est à rapprocher de l'inertie interclasse ;
- le nombre d'attributs discriminants à eux seuls, car une catégorisation est d'autant meilleure qu'il faut moins d'informations pour classer un individu ;
- la simplicité des catégories obtenues, car les catégories les plus significatives sont supposées être les plus simples.

La recherche d'attributs discriminants est donc de première importance. Par exemple, la catégorisation des espèces d'oiseaux induite par un attribut comme *aquatique* est plus sélective et donc plus fine que la catégorisation induite par l'attribut *couleur*, car le premier implique plus de propriétés que le second, entre autres oiseau *nageur*, *à pieds palmés*, *mangeur de poisson*.

L'algorithme de catégorisation conceptuelle permet de traiter un ensemble initial de n objets, dont k sont considérés comme les objets de référence ou les prototypes de k catégories. Les descriptions des prototypes sont utilisées pour procéder à une première discrimination des objets en k classes disjointes. Les objets les plus représentatifs de chaque classe sont recherchés pour servir de nouveaux prototypes au cours de l'itération suivante. Le processus s'arrête dès que les itérations convergent vers une solution stable, ou dès que la catégorisation n'évolue plus au sens d'un critère global de qualité choisi à l'avance. Le nombre optimal de classes est rarement connu à l'avance. Il est quelquefois possible de le déterminer en limitant, par exemple, le nombre de sous-classes d'une classe donnée, puis de faire varier ce nombre de sous-classes, et de choisir la catégorisation qui obtient le meilleur score au sens du critère global de qualité.

L'algorithme a été testé, entre autres, sur le célèbre exemple des petits trains (cf. Fig. 4.2). Les catégorisations obtenues sont proches de celles qui sont réalisées par des êtres humains. Elles ont permis d'étudier l'influence de deux facteurs importants sur la catégorisation, qui sont la donnée d'un objectif et la prise en compte de connaissances externes. Ainsi, même si l'algorithme de catégorisation conceptuelle ressemble beaucoup à la technique des centres mobiles ou à celles des nuées dynamiques, il fournit cependant une bonne interprétation qualitative des résultats obtenus. Une telle interprétation peut évoluer selon le contexte choisi et peut difficilement être obtenue avec les seules techniques d'analyse de données classiques.

Il faut toutefois noter que les résultats présentés par R.S. Michalski et R.E. Stepp, ainsi que la nouveauté de leur méthode, sont contestés dans un article de M.B. Dale [Dale, 1985], qui fournit des références à des méthodes numériques semblables (et même meilleures selon lui) mises au point dans les années soixante. Parallèlement, une application directe de l'algorithme de catégorisation conceptuelle à des données

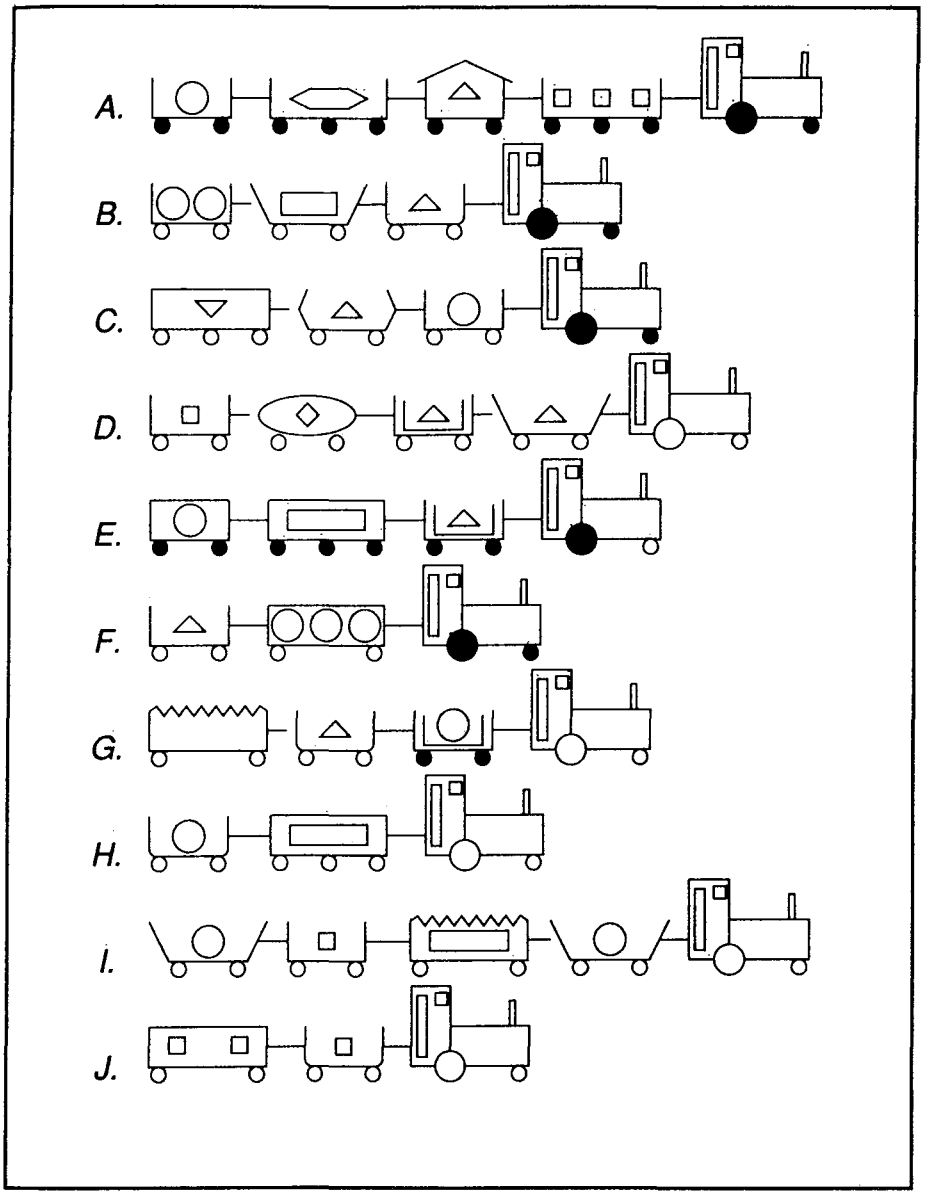


Figure 4.2. La catégorisation de trains qui transportent des marchandises (d'après [Stepp and Michalski, 1986]).

complexes s'est révélée irréaliste, les temps de calculs nécessaires à la reconnaissance des catégories étant beaucoup trop long [Nugues, 1989].

4.2.2 Une note sur l'approche symbolique-numérique

La problématique de l'analyse de données, celles de la catégorisation conceptuelle et de l'apprentissage symbolique sont semblables [Gascuel and Guénoche, 1990] : expli-

quer un ensemble de données apparemment disparates en construisant une catégorisation dont la qualité est mesurée à l'aide de certains critères. Toutefois, dans la première approche, sont recherchées des méthodes efficaces de traitement d'informations numériques, alors que dans la seconde, des informations symboliques plus riches sont manipulées en relation avec des connaissances relatives au domaine étudié. Cependant, les méthodes utilisées en catégorisation conceptuelle et dans les méthodes d'apprentissage qui s'y rattachent, montrent que les deux approches peuvent se compléter avantageusement pour produire une catégorisation ayant un fort pouvoir explicatif et prédictif. La combinaison des approches a tendance à prendre de plus en plus d'importance, comme le montrent l'introduction d'objets symboliques dans un contexte numérique [Diday, 1989] et les techniques numériques utilisées en catégorisation conceptuelle. Le système PLAGE est un élément représentatif des systèmes élaborés dans un tel contexte [Gascuel, 1987]. Le but de ce système est de produire des descripteurs servant à discriminer des données, la pertinence des descripteurs étant mesurée à l'aide de critères statistiques.

Les approches symbolique et numérique ne peuvent que s'enrichir mutuellement, la première en élargissant le champ d'applications de la seconde, le recours à des critères numériques pouvant augmenter l'efficacité des méthodes de la première.

4.2.3 La formation incrémentielle de hiérarchies de concepts

La catégorisation conceptuelle permet de produire une partition, éventuellement hiérarchique, grâce à une série d'opérations de discriminations effectuées sur un ensemble d'objets. Elle peut être interprétée comme une opération de reconnaissance, qui revient à déterminer la catégorie d'un objet à partir d'informations partielles, ou encore comme un apprentissage à partir d'observations, qui est proche de la façon de raisonner de l'être humain [Reed, 1972] : un individu observe un ensemble d'entités, objets ou événements, à partir duquel il essaie d'induire des concepts qui décrivent ces entités ; la tâche d'apprentissage réside alors dans la découverte et la formation des concepts, ainsi que dans la mise en place de leur organisation globale.

Plus précisément, étant donné un ensemble d'objets hétérogènes considérés comme des instances de concepts, il s'agit de construire une catégorisation hiérarchique qui regroupe ces objets. Les catégories retenues représentent l'intension des concepts découverts. Dans la suite du chapitre, nous appellerons *catégorisation hiérarchique incrémentielle* le processus de formation incrémentielle de hiérarchies de concepts. Dans la méthode de catégorisation conceptuelle présentée au paragraphe précédent, l'ensemble des objets à traiter est immédiatement disponible. En catégorisation hiérarchique incrémentielle, les objets sont au contraire traités les uns après les autres, au fur et à mesure de leur apparition. La catégorisation s'appuie alors sur un processus de classification qui est suivi d'une restructuration de la hiérarchie des concepts en cours de construction. Ce traitement successif des objets distingue nettement la catégorisation hiérarchique incrémentielle des approches numériques et conceptuelles classiques.

La catégorisation hiérarchique incrémentielle s'appuie sur les principes suivants [Gennari *et al.*, 1989] :

- Représentation des concepts : les concepts découverts appartiennent à une hiérarchie d'objets qui est un arbre ou un graphe. Ces objets, éléments de la hiérarchie, décrivent l'intension d'un concept ; ils sont ordonnés par une relation d'ordre partiel comparable à la relation d'héritage.

- Classification des instances : une instance à classer est d'abord comparée à l'élément le plus général de la hiérarchie, puis aux éléments plus spécialisés, tout comme dans la recherche de relations de subsomption (cf. § 3.3.1). Toutefois, le parcours de la hiérarchie est généralement effectué en profondeur ordonnée, sous le contrôle d'une fonction d'évaluation qui estime la distance qui sépare la nouvelle instance des concepts déjà présents dans la hiérarchie.

- Apprentissage non supervisé : la classification d'une instance est réalisée sans prise d'avis extérieur. Le nombre de catégories à obtenir et l'ensemble des objets à traiter n'est pas connu à l'avance ; la hiérarchie évolue à l'apparition de chaque nouvelle instance à classer.

- Apprentissage incrémentiel : une seule instance est traitée à la fois. La classification d'une instance guide la tâche de catégorisation ou formation de la hiérarchie des concepts, qui consiste à modifier la hiérarchie obtenue, par création, fusion ou suppression de concepts, en considérant les informations détenues par l'instance. Il est donc nécessaire de disposer d'opérateurs qui procèdent à de telles opérations de chirurgie sur la hiérarchie et sur les concepts.

Deux autres techniques sont principalement utilisées pour acquérir automatiquement des concepts. En apprentissage *empirique* ou apprentissage par *détection de similarités* (SBL[2]), les nouveaux concepts sont créés en déterminant les caractéristiques communes à une série d'exemples [Michalski, 1984] [Kodratoff, 1986]. En apprentissage par *explications* (EBL[3]), un nouveau concept est créé sur la base d'une généralisation d'exemples, qui doit être justifiée par des explications [Ellman, 1989]. Ces deux techniques se démarquent de la formation incrémentielle de hiérarchies de concepts en plusieurs points, notamment : le résultat de l'apprentissage n'est pas une hiérarchie de concepts mais plutôt la description d'un ou quelquefois de plusieurs concepts ; l'apprentissage est supervisé à l'image d'un professeur donnant un ensemble d'exemples d'un nouveau concept à ses élèves, pour qu'ils en tirent une généralisation.

4.2.4 Un système de catégorisation hiérarchique incrémentielle

La description de UNIMEM[4] [Lebowitz, 1987], un représentant typique de la famille des systèmes de catégorisation hiérarchique incrémentielle, va illustrer les idées abordées précédemment. Le but du système est de construire une hiérarchie de concepts semblable à une hiérarchie d'héritage, en observant les régularités que possèdent des

2. *Similarity-Based Learning.*
3. *Explanation-Based Learning.*
4. *UNIversal MEMory.*

objets ou instances qui lui sont présentés séquentiellement. Chaque concept est défini par un ensemble de couples attribut-valeur appelés caractéristiques, une valeur pouvant être aussi bien symbolique que numérique. La fréquence d'apparition d'une valeur dans les instances traitées par le système est mesurée par un score associé à la valeur et appelé score de prédiction. Un autre score, qui mesure la fréquence d'apparition de l'attribut, est associé directement à l'attribut. Ce dernier score sert à déterminer l'importance de l'attribut dans la description du concept.

Les algorithmes de parcours et de modification de la hiérarchie des concepts

Une nouvelle instance est classée après une recherche en profondeur dans la hiérarchie, pendant laquelle le système recherche les concepts les plus spécifiques "subsumant" la nouvelle instance (une instance peut dépendre de plusieurs concepts). La recherche des subsumants est contrôlée par une fonction d'évaluation qui tient compte du degré de "proximité" de l'instance avec un concept, ce degré de proximité étant un paramètre du système. L'appariement d'une instance et d'un concept n'est donc pas forcément complet.

Lorsque le subsumant (ou les subsumants) le plus spécifique a été trouvé, le système compare la nouvelle instance avec les instances du subsumant. Lorsqu'aucune instance ne partage suffisamment de caractéristiques avec la nouvelle instance, cette dernière est simplement rattachée au concept subsumant et les scores associés à chaque caractéristique du concept sont mis à jour. S'il existe une instance qui partage suffisamment de caractéristiques avec la nouvelle instance, un nouveau concept, spécialisation du concept subsumant, est créé sur la base de ces caractéristiques communes. La hiérarchie des concepts est donc modifiée dynamiquement. La nouvelle instance ainsi que celles qui s'apparient avec elle sont rattachées au nouveau concept. Les scores associés à chaque caractéristique du nouveau concept sont mis à jour en conséquence.

Lors de l'actualisation des scores, les concepts les plus généraux sont examinés avant les concepts les plus spécifiques. Lorsque le score associé à une caractéristique dépasse un seuil fixé par l'utilisateur, la caractéristique est considérée comme définitivement acquise par le concept : son score est fixé et ne peut plus être modifié. Parallèlement, si un score descend en dessous d'un autre seuil fixé par l'utilisateur, la caractéristique est supprimée. La généralité d'un concept peut ainsi augmenter, mais jusqu'à une certaine limite : un concept jugé trop général est supprimé puisque son pouvoir descriptif et discriminant est devenu trop faible. Les spécialisations du concept supprimé et leurs instances sont réajustées sur les ascendants du concept supprimé.

Quatre opérations principales permettent donc de modifier la hiérarchie des concepts pendant la classification et sont à la base du processus de formation de concepts :

- Mise en place d'une nouvelle instance.
- Création d'un nouveau concept sur la base de caractéristiques communes à deux instances.
- Suppression d'un concept.

- Adjonction définitive ou suppression d'une caractéristique selon la valeur du score qui lui est associé.

Le système UNIMEM partage de nombreuses caractéristiques avec le système CYRUS [Kolodner, 1983]. Il a eu plusieurs descendants, dont COBWEB [Fisher, 1987], ADECLU [Decaestecker, 1989] et CLASSIT[5] [Gennari *et al.*, 1989]. Les systèmes COBWEB et ADECLU incorporent des méthodes probabilistes pour gérer les scores associés aux caractéristiques des concepts.

Un exemple de construction incrémentielle d'une hiérarchie

Les figures 4.3 et 4.4 montrent un scénario possible de construction d'une hiérarchie de concepts après l'analyse de six instances. Au départ, le système crée un concept, notons-le *C1*, qui correspond aux caractéristiques de la première instance qu'il analyse. Pour simplifier, les couples attribut-valeur sont réduits à une valeur. Après analyse de la seconde instance, la caractéristique *un-noyau* qui figurait dans la description du concept *C1* disparaît car son score devient nul, la seconde instance ayant trois noyaux. Les scores associés aux caractéristiques *sombre* et *deux-cils* sont augmentés de 1. La troisième instance ne peut être prise en compte par *C1* car ses caractéristiques sont trop éloignées de celles de *C1*. Un nouveau concept, noté *C2*, est alors créé pour décrire cette instance. La quatrième instance s'apparie complètement avec une des instances de *C1*. Par suite, un nouveau concept, noté *C3*, est créé, et sa description contient la caractéristique *trois-noyaux*, qui ne fait pas partie des caractéristiques de *C1*. Les scores de *C1* et du nouveau concept *C3* sont mis à jour. La cinquième instance est rattachée à *C2*, mais elle partage la caractéristique *deux-cils* avec *C1*. Par suite, le score associé à *deux-cils* est tout de même augmenté de 1, ce qui montre que l'ensemble des extensions des concepts créés ne forme pas forcément une partition disjointe de l'ensemble des instances. La prise en compte de la sixième instance se fait exactement comme celle de la quatrième instance pour donner naissance à un nouveau concept, spécialisation de *C2*. Signalons pour terminer que la hiérarchie obtenue est dépendante de l'ordre de présentation des instances [Lebowitz, 1988].

4.2.5 Classification et catégorisation hiérarchique incrémentielle

Le raisonnement par classification et la catégorisation hiérarchique incrémentielle présentent certaines analogies. L'algorithme sur lequel repose le premier a été détaillé au paragraphe 3.3.1. Il contrôle notamment l'insertion d'un nouvel objet dans une hiérarchie d'objets déjà constituée, insertion qui peut être vue comme un enrichissement de la hiérarchie sur laquelle est opérée la classification. Le raisonnement par classification a donc, dans une certaine mesure, une fonction d'acquisition de connaissances [MacGregor and Burstein, 1991].

5. L'article décrivant CLASSIT présente une synthèse des caractéristiques des principaux systèmes de formation incrémentielle de concepts et inclut l'étude de l'un des plus anciens d'entre eux, nommé EPAM [Feigenbaum and Simon, 1984].

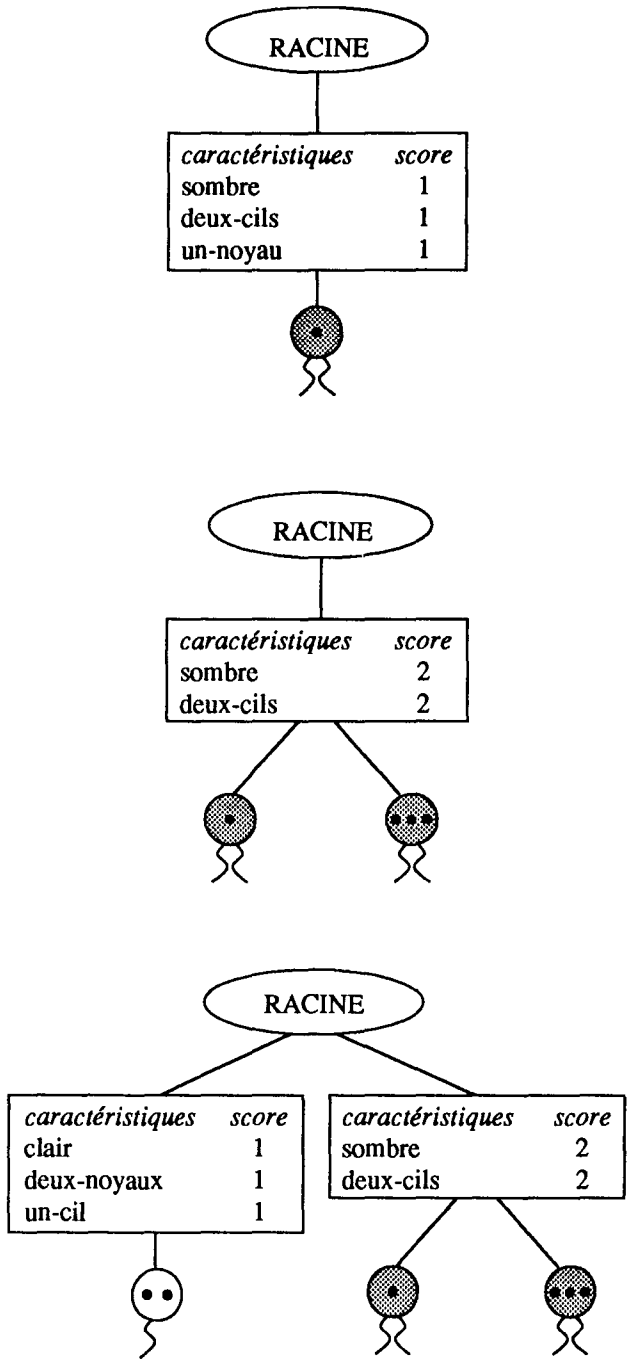


Figure 4.3. Formation incrémentielle de concepts (1).
Un scénario possible de formation incrémentielle d'une hiérarchie de concepts. Les instances représentent des cellules et se distinguent par leur couleur, le nombre de noyaux qu'elles renferment, ainsi que par le nombre de cils qui leur sont attachés.

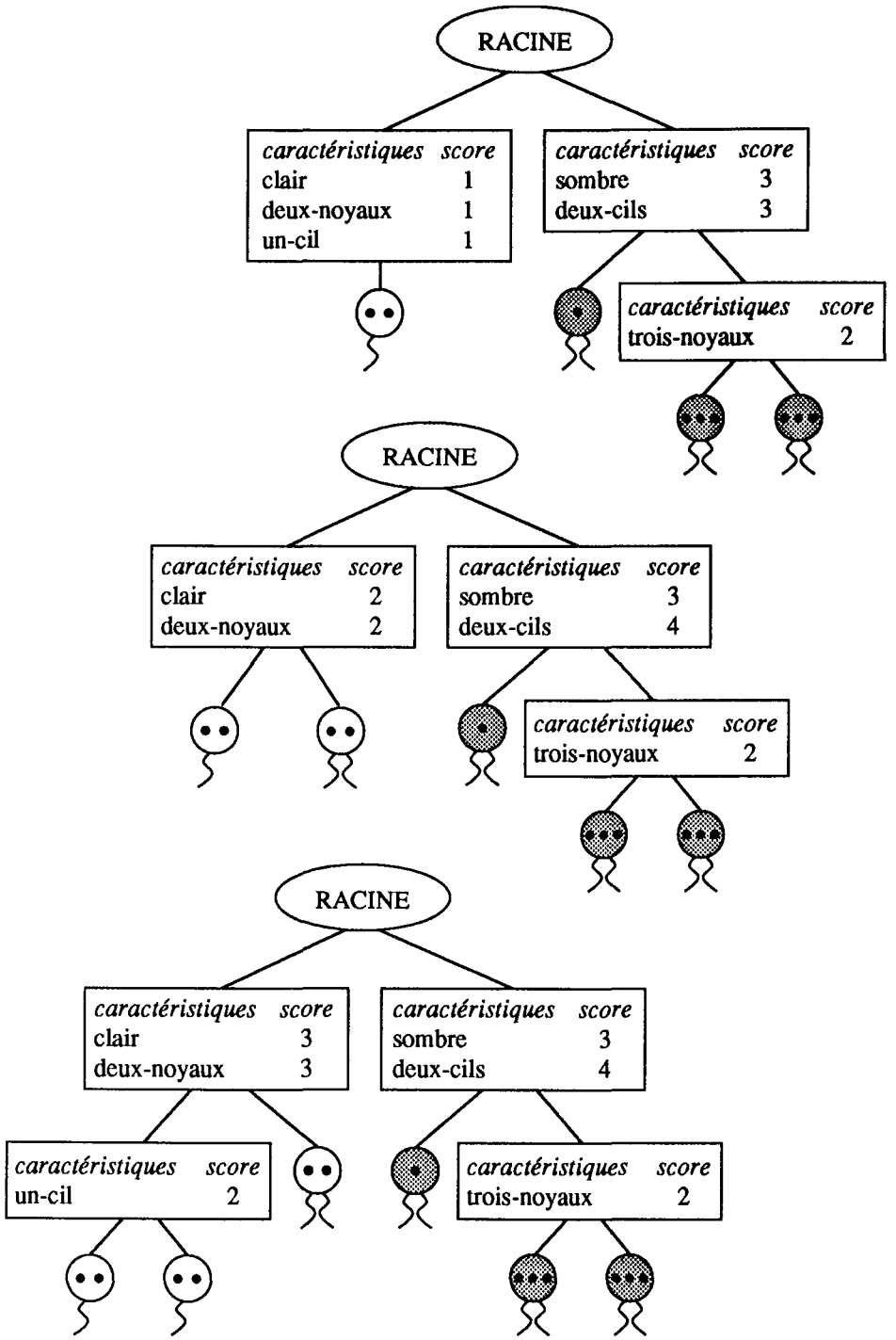


Figure 4.4. Formation incrémentielle de concepts (2).

```
fonction CHI (O RACINE)
Comparer O avec chaque descendant de RACINE
cas si un descendant unique D de RACINE est un modèle possible pour O
      alors mettre à jour la description de D
            si D est une feuille alors Placer O
                                  sinon appeler CHI (O D)
   si plusieurs descendants de RACINE sont des modèles possibles pour O
      alors Fusionner les descendants en un unique concept F
            mettre à jour la description de F
   si aucun descendant de RACINE n'est un modèle possible pour O
      alors choisir le descendant P le plus ''proche'' de O
            Comparer O avec les descendants de P
            Si il existe des descendants de P compatibles avec O
               alors Scinder P en deux sous-concepts P1 et P2
                     (P1 décrit O et les descendants de P compatibles)
                     (P2 décrit les descendants de P non compatibles)
                     mettre à jour les descriptions de P1 et P2
               sinon Créer un nouveau concept décrivant O
   autre cas NIL
```

Figure 4.5. Un algorithme de catégorisation hiérarchique incrémentielle (d'après [Hadzikadic and Yun, 1989]). L'objet courant à classer `O` est supposé compatible avec l'objet `RACINE`, qui est la racine de la sous-hiérarchie en cours d'exploration.

Parallèlement, les méthodes de catégorisation hiérarchique incrémentielle font partie intégrante des méthodes d'apprentissage symbolique, et, de façon complémentaire, des méthodes d'acquisition de connaissances [Gennari *et al.*, 1989]. Comme pour tout processus de catégorisation conceptuelle ou numérique, la fonction de la catégorisation hiérarchique incrémentielle est de "résumer" pour l'expliquer un ensemble de données brutes *a priori* disparates en une hiérarchie de concepts auxquels se rattachent ces données. Un algorithme simplifié de catégorisation hiérarchique incrémentielle est présenté à la figure 4.5. Les quatre opérations fondamentales dans l'algorithme sont :

- *Placer* un objet : cette opération consiste à rattacher à un concept C de la hiérarchie l'objet courant, noté O dans la suite, lorsque le degré d'appariement entre C et O dépasse un seuil donné $S1$. Dans ce cas, C est dit un modèle pour O, et O fait partie de l'extension du concept C. La mise à jour de la description d'un concept consiste à prendre en compte dans cette description les éléments d'information détenus par l'objet courant O. Plus l'objet C est situé "bas" dans la hiérarchie, plus le placement de O est optimal.

- *Fusionner* la description de deux concepts (ou plus) : cette opération consiste

à remplacer deux concepts *C1* et *C2* auxquels *O* peut se rattacher par un seul nouveau concept noté *F*, qui généralise *C1* et *C2*. La description de *F* factorise les propriétés communes à *C1* et *C2*, qui deviennent des spécialisations de *F* une fois que leurs descriptions ont été mises à jour, tout comme *O* d'ailleurs.

- *Scinder* ou *Diviser* un concept en deux sous-concepts : cette opération s'applique lorsque le degré d'appariement entre *O* et les concepts en cours d'exploration n'est pas assez élevé, mais reste acceptable. Ce degré d'appariement est inférieur à *S1*, mais supérieur à un seuil *S2*, et les concepts vérifiant cette propriété sont dits compatibles avec *O*. Notons alors *P* le concept compatible dont le degré d'appariement avec *O* est maximal. Les descendants de *P* sont comparés avec *O*. S'il existe des descendants de *P* dont le degré d'appariement dépasse une seuil donné *S3*, alors *P* est scindé en deux nouveaux concepts *P1* et *P2*, *P1* décrivant *O* et les descendants de *P* "suffisamment proches" de *O*, *P2* décrivant les autres descendants de *P*. Dans certains systèmes de catégorisation hiérarchique incrémentielle, lorsque le degré d'appariement entre *O* et les concepts en cours d'exploration n'est pas assez élevé, un nouveau concept est créé systématiquement.

- *Créer* un concept : cette opération consiste à créer de toutes pièces un nouveau concept qui puisse décrire *O*. C'est le cas lorsque le degré d'appariement entre *O* et n'importe quel concept est inférieur au seuil *S2*. L'objet *O* est alors unique en son genre, mais d'autres objets à venir seront peut-être de même type que *O*.

En catégorisation incrémentielle, les appariements ne sont généralement pas exacts, mais partiels ou encore approchés. Le degré d'appariement entre un concept et l'objet courant est mesuré et puis comparé à des seuils de tolérance qui indique l'opération à effectuer.

Une rapide comparaison entre l'algorithme présenté ci-dessus et l'algorithme de classification présenté au paragraphe 3.3.1 nous permet de constater que :

- L'organisation de l'espace des descriptions sur laquelle opère le raisonnement par classification, et la catégorisation hiérarchique incrémentielle est dans les deux cas une hiérarchie conceptuelle.

- Les concepts appartenant à ces hiérarchies ont une structure également semblable, généralement de type frame-attributs-facettes [Aguirre, 1989].

- Un parcours de l'espace des descriptions est effectué dans les deux cas. Le degré d'appariement entre les concepts et l'objet courant fait office de fonction d'évaluation lors du parcours. Toutefois, l'appariement est généralement exact en ce qui concerne le raisonnement par classification, alors qu'il est partiel en catégorisation hiérarchique incrémentielle.

- Deux opérations fondamentales se retrouvent dans les deux algorithmes, *Placer* et *Créer*, tandis que les opérations *Fusionner* et *Scinder* sont uniquement utilisées en catégorisation hiérarchique incrémentielle.

Ainsi, le processus de classification travaille sur une hiérarchie où tous les objets sont connus à l'avance, un seul objet devant être inséré dans la hiérarchie. L'opération de création consiste à donner la description du nouveau concept ; l'opération de placement consiste à trouver l'endroit approprié de la hiérarchie devant accueillir le nouveau concept. En ce sens, le parcours de la hiérarchie des concepts effectué par le processus de classification est descendant, des concepts génériques vers les concepts spécifiques. L'insertion dans la hiérarchie a plus un rôle de résolution de problèmes que d'enrichissement de la hiérarchie, et par conséquent d'apprentissage et d'acquisition de concepts : le but de l'insertion est de retrouver des informations détenues par des subsumants pour faire avancer la résolution d'un problème.

En catégorisation hiérarchique incrémentielle, le parcours de la hiérarchie en cours de construction n'est pas systématiquement descendant, car la hiérarchie est justement modifiée lors de ce parcours. Les deux opérations fondamentales de modification de la hiérarchie sont alors la fusion et la division de concepts. En ce sens, le processus de catégorisation hiérarchique incrémentielle intègre d'une certaine façon l'idée de retour arrière, en remettant en cause la forme de la hiérarchie. Il a donc beaucoup plus que le raisonnement par classification un rôle en apprentissage et en acquisition de concepts. Intervenant en amont de son homologue, le processus de raisonnement par classification, le processus de catégorisation hiérarchique incrémentielle le complète : il permet, grâce aux fonctionnalités qui lui sont intégrées, de construire une hiérarchie sur laquelle raisonne le processus de classification.

4.2.6 La conception d'une hiérarchie d'objets

Décrire les objets appartenant à l'univers étudié, les placer dans des catégories, puis organiser globalement ces catégories, sont les opérations préliminaires principales dans la conception d'une hiérarchie d'objets, dans le cadre des représentations à objets. Ces opérations sont délicates à mettre en œuvre, car il n'existe pas de méthodologie universelle de conception de représentations hiérarchiques [Griffith, 1982] [Booch, 1990] [Ferber, 1990] [Rumbaugh *et al.*, 1991]. La conception dépend de nombreux facteurs, du but poursuivi, du point de vue et de l'expérience du concepteur relativement au domaine modélisé, etc.. Brièvement, voici quelques critères qui donnent une idée des étapes principales de la marche à suivre :

- Identification des concepts du domaine susceptibles d'être représentés par des objets, description de l'état et du comportement des individus recouverts par ces concepts ; le comportement fait référence aux opérations que les individus doivent être en mesure d'effectuer.

- Identification des relations qui existent entre les différents concepts, en regroupant ceux qui partagent une partie de leurs propriétés et de leur comportement ; organisation des concepts en hiérarchie d'héritage.

- Modélisation du traitement à effectuer pour résoudre un problème donné sur le domaine étudié, mise en place des méthodes générales de résolution de problèmes standard sur le domaine étudié.

La hiérarchie ainsi construite décrit des connaissances propres à un certain domaine, à l'intérieur duquel se posent des problèmes qu'il faut résoudre. Le processus de résolution de problèmes est en règle générale réparti en un ensemble de traitements locaux dont la responsabilité incombe à certains individus prenant en charge les opérations à exécuter.

La construction d'une hiérarchie est un processus *incrémental* : la hiérarchie évolue et s'améliore en fonction des résultats obtenus, jusqu'à ce qu'une certaine stabilité soit atteinte et que les résultats soient jugés satisfaisants.

4.3 Catégorisation, classification et analogie

C'est le petit matin, il fait encore un peu sombre. A une certaine distance, une forme allongée se déplace parmi les herbes et se rapproche. Se devinent une tête, quatre pattes qui supportent un corps puissant de couleur brunâtre, et une grande queue. Il s'agit d'un animal, tous les indices le prouvent, mais lequel ? Il ne peut pas s'agir d'un chat, l'animal est trop gros, ni même d'un chat sauvage, d'ailleurs que ferait-il ici ? Il ne s'agit pas non plus d'un chien, un chien n'a pas cette allure. Le supposé animal est maintenant très proche : contre toute attente, il apparaît qu'il s'agit d'un lion et qu'il n'a pas l'air vraiment amical, mieux vaut s'en protéger. La rencontre pourrait mal tourner ; elle me rappelle trop d'anciennes rencontres faites dans la rue avec des chiens agressifs. Tout cela est très étonnant. La prochaine fois, je saurai qu'il est possible de faire ce genre de rencontre inattendue dans un tel endroit.

Ce début d'histoire illustre la problématique du raisonnement par cas. L'analyse d'une nouvelle situation ou la résolution d'un nouveau problème ne s'envisagent pas *ex nihilo*, mais plutôt à partir d'expériences mémorisées ayant un certain rapport avec la situation ou le problème courants [Riesbeck and Schank, 1989] [Slade, 1991] [Kolodner, 1991]. La compréhension du déroulement d'événements présents peut s'expliquer à partir de situations ou cas "analogues" vécus antérieurement. Le cas explicatif est adapté à la situation présente pour en devenir un modèle : les caractéristiques qui sont d'actualité sont conservées, celles qui ne le sont pas, ou qui sont susceptibles d'engendrer des erreurs d'analyse sont écartées. Le cas adapté devient un nouveau cas qui est mémorisé afin d'être réutilisé ultérieurement dans des circonstances analogues. La phase d'explication d'une nouvelle situation est comparable à une phase de catégorisation, tandis que la phase d'enrichissement des connaissances qui en résulte relève de l'apprentissage.

Le raisonnement par cas s'envisage selon deux points de vue principaux, comme une tentative de modélisation du comportement cognitif humain, et comme un des principes de base dans la construction de systèmes à bases de connaissances. Du point de vue psychologique, il suppose que l'activité cognitive humaine est plutôt en accord avec des théories comme celles des modèles mentaux [Johnson-Laird, 1983] [Read and Cesa, 1991], et que ce type de théorie permet d'expliquer au mieux cette activité[6]. Du

6. Une comparaison avec la présentation de la catégorisation faite au paragraphe 1.2.1 montre le parallélisme existant entre la catégorisation, le principe de remémoration, et le raisonnement par cas.

point de vue de l'intelligence artificielle, le raisonnement par cas apporte des éléments de solution aux questions concernant l'adaptation et l'évolution des système à bases de connaissances travaillant avec des informations incomplètes ou bruitées. Il peut être considéré comme un principe général de raisonnement qui relève de l'analogie et qui s'appuie sur l'exploitation d'expériences acquises antérieurement. Il peut donc servir à résoudre des problèmes en adaptant une ancienne solution à un problème nouveau, ou encore en se laissant guider par des démonstrations présentant une certaine similitude avec la résolution présente ; il peut aussi servir à interpréter une situation nouvelle en termes de situations vécues et assimilées. Il fait donc appel à des techniques développées en intelligence artificielle qui traitent de l'organisation de la mémoire, de l'apprentissage, de la planification, et enfin de la résolution de problèmes.

Le raisonnement par cas s'appuie sur un modèle de la mémoire qui favorise la représentation, la recherche et la manipulation de cas, qui est de plus dynamique et évolutif. Les études réalisées pour le raisonnement par cas qui concernent l'organisation de la mémoire proviennent en droite ligne des recherches menées sur la remémoration, les scripts et les frames (cf. page 25 et paragraphe 1.2.2). Toutefois, les théories de l'apprentissage "adaptatif" à partir d'échecs ou d'erreurs ont également très fortement contribué à l'essor du raisonnement par cas [Schank, 1982] [Hall, 1986] [Hammond, 1990]. D'une certaine façon, l'apprentissage à partir d'échecs s'oppose à la tendance classique de la généralisation, ou apprentissage à partir d'exemples, qui pourrait être qualifié d'apprentissage à partir de ce qui "marche". Le premier consiste plutôt à apprendre à partir de ce qui "ne marche pas", apprendre en s'adaptant car ce qui était prévu ne s'est pas concrétisé. Classiquement, un échec ou une erreur surviennent lorsque les connaissances disponibles ne permettent pas d'expliquer la situation courante, soit parce qu'il n'existe aucune connaissance sur le sujet, soit parce que les connaissances ayant un rapport avec le sujet sont en désaccord ou même en contradiction avec celui-ci. La prise en compte de la situation courante implique une révision des connaissances disponibles. Cette mise à jour revient à ajouter aux connaissances un cas décrivant la situation courante, accompagné d'une explication sur sa bonne et mauvaise réutilisation future[7]. L'explication provient d'une analyse de l'échec consistant à établir les causes et les conséquences de l'échec, en déterminant l'écart existant entre le cas mémorisé le plus "proche" et la situation courante, et finalement en modifiant les paramètres appropriés pour réduire l'écart et ajuster le cas le plus "proche" à la situation courante. L'association explications-connaissances joue un rôle primordial dans le raisonnement et la manipulation de connaissances.

La figure 4.6 montre le schéma de fonctionnement d'un système employant le raisonnement par cas. Le cycle de raisonnement consiste à localiser dans la mémoire des cas une expérience passée qui s'apparie partiellement avec la situation en cours d'analyse, à évaluer les différences existant entre le cas trouvé et la situation, et à corriger le cas en conséquence pour en faire un nouveau cas, qui est alors mémorisé pour une utilisation analogue future. Voici le détail des six étapes principales du cycle :

7. Ce qui suggère que mémoriser des connaissances dans une base revient à stocker des connaissances brutes comme dans une base de données, en les accompagnant toutefois d'un mode d'emploi.

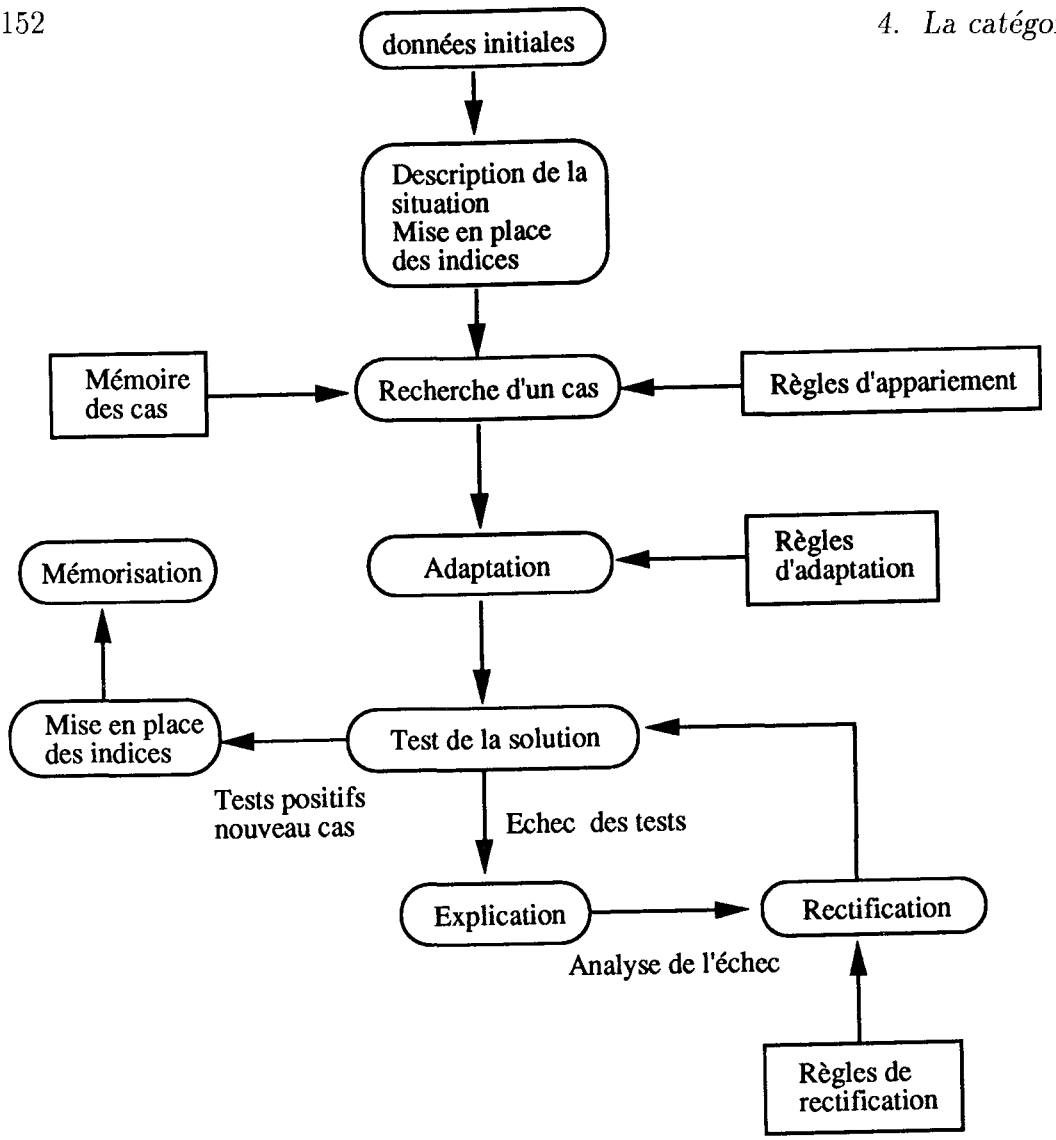


Figure 4.6. Le mécanisme du raisonnement par cas (d'après [Riesbeck and Schank, 1989]).

- Description de l'entité courante : cette entité peut recouvrir un individu, un événement, une situation. Les caractéristiques de l'entité courante servent de descripteurs et d'indices ou d'index de recherche. Les indices sont utilisés pour résumer aussi fidèlement que possible l'entité courante. Des règles d'indexation aident à identifier les caractéristiques qui sont *a priori* de bons indices, qui facilitent la recherche d'un cas dans la mémoire. Les indices ont le même usage que ceux d'une enquête policière. Parallèlement, la recherche de bons indices reste un problème très difficile.

- Recherche d'un cas dans la mémoire des cas, appelée aussi mémoire des expériences : les indices sont utilisés pour rechercher un cas qui s'apparie, en général partiellement, avec l'entité courante. Un cas est une description dont le contenu est variable, et peut comprendre par exemple la donnée d'un problème, d'une

solution, d'un mode d'emploi de la solution, ainsi que les contre-indications associées. Le cas retrouvé donne une idée de la connaissance disponible sur l'entité en cours d'étude. Lorsque plusieurs cas sont potentiellement capables d'expliquer l'entité courante, des règles de sélection de cas peuvent émettre des suggestions et aider à sélectionner le cas le plus "proche" de la situation actuelle.

- Modification-adaptation : le cas sélectionné, appelé "solution" par la suite, est modifié pour s'adapter à l'entité courante ; il est en effet assez rare que la solution s'apparie complètement à l'entité courante. Des règles d'adaptation peuvent indiquer les facteurs à modifier en priorité pour que le degré d'appariement entre la solution et l'entité courante soit le plus élevé possible. Ces règles sont généralement très dépendantes du domaine sur lequel le système travaille.

- Tests : durant cette phase, des tests mesurent la prédisposition de la solution à expliquer l'entité courante. Autrement dit, l'entité en cours d'analyse doit être comprise grâce aux explications fournies par la solution.

- Mémorisation : si la solution explique "suffisamment bien" l'entité courante, alors la solution est mémorisée ; la solution est différente du cas dont elle provient puisqu'elle a été adaptée à la prise en compte de l'entité courante. Un ensemble de descripteurs qui résument la solution mémorisée est associé à cette dernière, afin de la caractériser dans la mémoire des cas, pour utilisation ultérieure. Cette étape d'apprentissage a pour résultat un enrichissement de la connaissance du système.

- Echec, explication de l'échec, rectification : si la solution ne contribue pas à expliquer l'entité courante de façon satisfaisante, il est nécessaire de trouver la cause de l'échec et de rectifier la solution en conséquence. Les causes de l'écart existant entre la solution et l'entité courante sont mémorisées et associées à la solution corrigée, pour éviter ultérieurement de refaire les mêmes erreurs dans un cas analogue. Des règles de rectification peuvent éventuellement indiquer quels sont les paramètres à modifier en priorité en cas d'échec. Ces règles sont aussi dépendantes du domaine de travail du système, comme les règles d'adaptation. La solution rectifiée est finalement renvoyée en phase de test pour validation.

Le raisonnement par cas s'appuie largement sur l'hypothèse que la mémoire est une mémoire de cas, dynamique et évolutive, contenant pour l'essentiel des expériences passées facilement accessibles. Ces expériences sont d'autant plus faciles à retrouver et à utiliser qu'elles sont richement imagées et indexées[8]. Le raisonnement par cas est guidé par l'expérience : l'adaptation à un nouveau cas, sa prise en compte et sa mémorisation sont provoquées par un échec à une explication prévue. Ce mode d'apprentissage se démarque de l'apprentissage par généralisation, tout en permettant comme ce dernier d'engendrer de nouvelles connaissances en les extrapolant à partir d'anciennes connaissances. S'il existe un consensus sur la modélisation globale du raisonnement

8. De fil en aiguille : plus une expérience est en rapport avec un nombre élevé de situations, plus elle est facile d'accès.

par cas, de nombreux et importants problèmes de détail restent encore à résoudre, portant notamment sur la représentation et le contenu d'un cas, sur l'organisation de la mémoire des cas, sur la gestion de l'adaptation des cas et celle de l'évolution de la mémoire des cas. Des revues des principaux systèmes de raisonnement par cas se trouvent dans [Slade, 1991] et dans [Riesbeck and Schank, 1989]. Parmi les systèmes les plus importants, citons CYRUS [Kolodner, 1983] et MEDIATOR [Kolodner *et al.*, 1985] [Kolodner, 1991], CHEF [Hammond, 1990] et PROTOS [Porter *et al.*, 1990].

Nous allons maintenant analyser les rapports existant entre le raisonnement par cas et le raisonnement par classification. Leur fonctionnement de base est identique : ils sont guidés par l'expérience, comme le suggère le principe de remémoration. Toutefois, à l'image des scripts, une expérience dans une mémoire de cas a le plus souvent un rôle d'"épisode temporel" à adapter à une situation courante, que n'ont pas forcément les objets d'une hiérarchie sur lequel opère le raisonnement par classification. La problématique du raisonnement par cas recouvre donc un univers plus large que celle du raisonnement par classification : un parallèle peut être fait entre le raisonnement par classification et les deux premières phases du raisonnement par cas, à savoir la description de la situation courante et la recherche d'un cas. Les phases suivantes d'apprentissage et d'acquisition de connaissances ne font pas partie du cycle standard du raisonnement par classification. Cependant, il n'est pas fait mention dans le raisonnement par classification d'évolution dynamique de la mémoire par adaptation de nouveaux cas à de nouvelles situations. Seul un réarrangement dynamique de la mémoire est concevable : il correspond à la construction dynamique d'une hiérarchie d'objets qui soit orthogonale aux hiérarchies déjà existantes et qui reflète la dimension définie par une nouvelle relation de subsomption.

Un mariage des deux modes de raisonnement peut être envisagé. Il revient à organiser la mémoire des cas sous la forme de hiérarchies croisées d'objets, gérées par le raisonnement par classification. Les cas sont représentés par des objets, et l'indexation des cas se fait par l'intermédiaire des attributs associés aux objets, ou encore par des relations de subsomption lorsqu'il existe un index ayant une sémantique de relation d'ordre partiel. Retrouver les cas qui s'apparient partiellement avec une situation en cours d'analyse consiste à retrouver les subsumants et les subsumés de la situation dans une des hiérarchies de cas. Des techniques de filtrage approximatif peuvent alors être employées pour favoriser l'appariement partiel [Vignard, 1985] [Granger, 1988]. Les subsumants d'une situation en cours d'analyse fournissent des informations de nature générale sur la situation, tandis que l'étude de l'écart entre la situation courante et les subsumés peut fournir la description du nouveau cas adapté.

4.4 Une note finale sur la catégorisation naturelle

Les informations sensorielles qui parviennent à un être humain lui permettent de percevoir les entités qui l'entourent, puis de les reconnaître, et enfin de réagir en conséquence, en s'appuyant sur son expérience dans le meilleur des cas, sinon en improvisant. Le comportement final de l'être humain fait suite à une série d'inférences qui dérive de la reconnaissance de l'entité. Cette reconnaissance peut être vue comme

une classification de l'entité dans une catégorie donnée. La classification complète donc la perception, et autorise la prédiction de caractéristiques non observables à partir de celles qui ont été observées. Un dernier facteur, l'environnement, a aussi toute son importance. La réaction d'un individu individu face à un lion diffère selon qu'il se trouve dans la savane ou dans un zoo. Là encore, l'expérience acquise sur le milieu environnant guide les décisions prises : si la barrière du parc des lions dans le zoo est cassée, il faut (vite) improviser un nouveau comportement.

Pour qu'un individu puisse prédire un comportement, il doit rechercher la catégorie de l'entité qu'il observe et tenir compte de l'univers dans lequel il se trouve. Ceci suppose entre autre que les connaissances sont organisées en catégories "naturelles", hiérarchisées ou non. La précision des inférences tirées d'une classification dépend de l'abstraction des catégories sur lesquelles le raisonnement par classification s'applique. Les catégories dites naturelles définissent les entités en fonction de leurs caractéristiques et de leurs interactions avec le milieu dans lequel ils sont plongés [Mervis and Rosch, 1981] [Bobick, 1987]. Idéalement, découvrir l'appartenance d'une entité à une telle catégorie doit provoquer des inférences qui sont en rapport avec le but de l'observateur. La prédiction de caractéristiques non observables n'est possible que si ces dernières dépendent de celles qui sont observables, ce qui suppose une certaine structuration de l'univers, qui dépend à la fois de l'expérience de l'observateur, de son point de vue sur l'univers et de son but.

Les techniques de catégorisation abordées dans le chapitre, l'analyse de données, la catégorisation conceptuelle, la catégorisation hiérarchique incrémentielle, les techniques d'apprentissage, divisent et ordonnent les entités étudiées, mais souvent sous un seul point de vue. Or, il y a autant de catégorisations, et donc de classifications possibles, qu'il y a d'interprétations des entités classées, de perspectives sur les entités [Minsky, 1988]. Par exemple, puisque le lion est le roi de la jungle, il est à ce titre l'équivalent d'un souverain, puisqu'il est aussi un tueur d'antilopes, il est à ce titre l'équivalent d'un chasseur, puisqu'il est très photogénique, il est à ce titre l'équivalent d'un modèle... Savoir construire et gérer des catégorisations reflétant ainsi plusieurs points de vue est une tâche primordiale dans le construction de systèmes à bases de connaissances. L'utilisation du raisonnement par classification laisse entrevoir une certaine possibilité d'exprimer ces perspectives, en considérant une base de connaissances comme un espace n-dimensionnel (cf. § 3.4.2). Une association tripartite, catégorisation hiérarchique incrémentielle pour construire les hiérarchies d'objets, raisonnement par classification pour opérer sur ces hiérarchies selon plusieurs points de vue (exprimés par une relation d'ordre partiel) et raisonnement par cas pour faire évoluer la base de connaissances, semble être une voie qui mériterait d'être explorée.

5

Un système d'aide à la conception de plans de synthèse

Ce chapitre décrit la partie expérimentale de la thèse et illustre de nombreux points théoriques de programmation abordés dans les chapitres précédents. Il présente tout d'abord la problématique de la synthèse organique, qui consiste à concevoir des corps chimiques complexes à partir de corps plus simples. Un rapide tour d'horizon des systèmes d'aide à la synthèse existants, systèmes qui sont essentiellement procéduraux, montre qu'une des préoccupations principales reste la représentation des réactions. La démarche présentée dans la suite est originale car elle inverse cette tendance classique, en donnant la priorité à la représentation des données plutôt qu'à celle des traitements. Ainsi, ce sont les environnements favorables et défavorables à l'application d'une réaction qui sont avant tout décrits, plutôt que la réaction accompagnée de ses différents cas d'application. L'ensemble des informations chimiques relatives à la description d'une synthèse se retrouve alors structuré en différentes hiérarchies croisées d'objets. Le raisonnement par classification est alors utilisé pour résoudre des problèmes de synthèse, plus précisément pour développer le plan de synthèse d'une molécule cible. En regard des études effectuées précédemment sur la conception artificielle de plans de synthèse, la démarche est novatrice et a servi de support à l'implantation d'un système appelé YCHEM.

5.1 La synthèse organique

5.1.1 La problématique de la synthèse organique

La synthèse de molécules organiques est une opération qui consiste à préparer une molécule, dite molécule *cible*, à partir de réactifs de départ, qui sont des molécules disponibles commercialement ou accessibles au moyen de synthèses connues. Une molécule est une composition d'atomes et de liaisons pouvant être considérée selon le point de vue de sa structure, ou *squelette* et celui de sa *fonctionnalité* (cf. Fig. 5.1). Le squelette est constitué par l'enchaînement des atomes de carbone que contient la molécule. Les *groupes fonctionnels* s'articulent autour des liaisons d'ordre multiple et des liaisons comportant au moins un hétéroatome (un hétéroatome est un atome qui

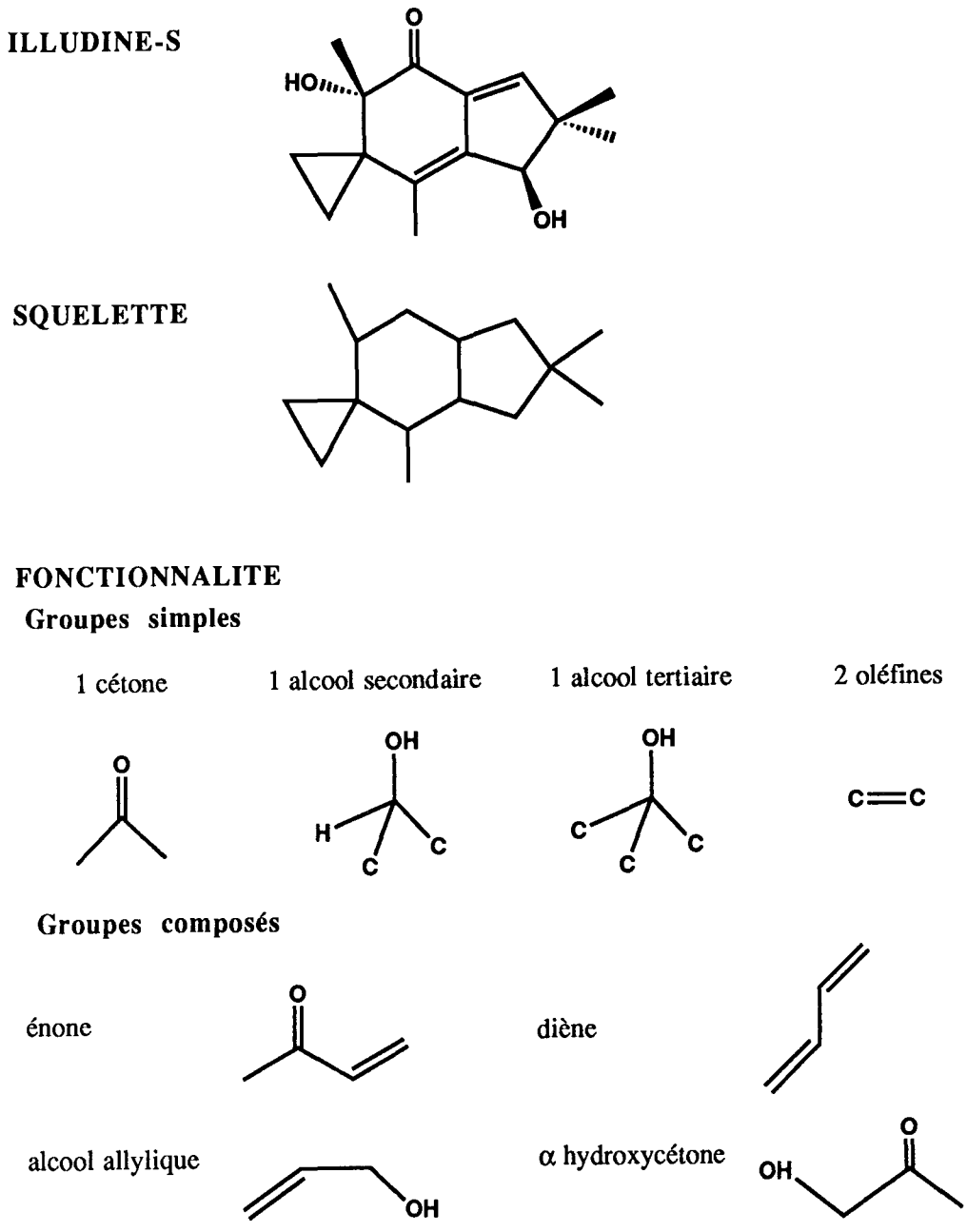


Figure 5.1. Squelette et groupes fonctionnels d'une molécule organique.

n'est ni un carbone ni un hydrogène). Ils déterminent la fonctionnalité ou le comportement chimique de la molécule. Le problème de base de la synthèse organique s'énonce alors de la façon suivante : *construire le squelette d'une molécule cible en s'aidant des groupes fonctionnels qu'elle renferme.*

Pratiquement, construire une molécule cible nécessite l'élaboration puis l'expérimentation d'un *plan de synthèse*. Un tel plan est défini par un ou plusieurs *chemins de synthèse*, chacun étant composé d'une suite de réactions, qui mènent des réactifs de départ à la molécule cible. Pour élaborer un plan de synthèse, le chimiste commence par dégager grossièrement un ou plusieurs chemins de synthèse, puis vérifie que les applications des réactions contenues dans les chemins exhibés sont valides. Chaque étape d'un chemin correspond à la mise en place d'un *objectif* à atteindre au cours de la synthèse. Un objectif matérialise une partie du squelette de la cible à construire et peut être conditionné par des aménagements de la fonctionnalité de la cible. La planification consiste donc à déterminer et à ordonner une série d'objectifs à atteindre. Parallèlement, la simulation de réactions sert à évaluer la faisabilité des étapes synthétiques pour atteindre les objectifs fixés.

La réussite d'une synthèse est intimement liée à la qualité du plan de synthèse, dont la conception requiert de l'imagination et des connaissances étendues portant sur les produits de départ disponibles, sur les stratégies de synthèse, sur les chemins de synthèse répertoriés, sur l'intérêt et les limites d'un certain nombre de réactions génériques, sachant qu'une réaction générique peut avoir quelques milliers de cas particuliers. L'élaboration d'un tel plan relève aussi de l'expérience personnelle et d'un certain savoir-faire dont il est difficile de dégager des règles générales.

5.1.2 Une première formalisation

Le but d'un problème de synthèse est de concevoir une nouvelle molécule, ou encore de déterminer un nouveau plan pour obtenir une molécule connue. L'espace d'états de ce problème de conception est décrit par un état final, la molécule cible, un état initial qui regroupe un ensemble de réactifs de départ, et un ensemble d'opérateurs, les réactions. Un état, que nous appelons *réacteur*, correspond à une distribution des liaisons entre atomes qui détermine les réactifs en présence. Les réactions permettent de passer d'un état à l'autre, donc d'une distribution des liaisons à la distribution suivante. Un chemin de synthèse est constitué d'une suite d'étapes qui correspondent chacune à l'application d'un opérateur à un état. Résoudre un problème de synthèse consiste donc à trouver au moins un chemin de synthèse qui mène de réactifs de départ à la molécule cible, les réactifs de départ étant des produits connus répertoriés à l'avance.

Les réactions peuvent s'envisager selon différents niveaux conceptuels (cf. Fig. 5.2) :

- Le niveau spécifique décrit une réaction particulière s'appliquant sur des molécules spécifiques. Il s'accompagne de la description de conditions réactionnelles précises.

- Le niveau générique fait intervenir des familles de molécules, ainsi que des condi-

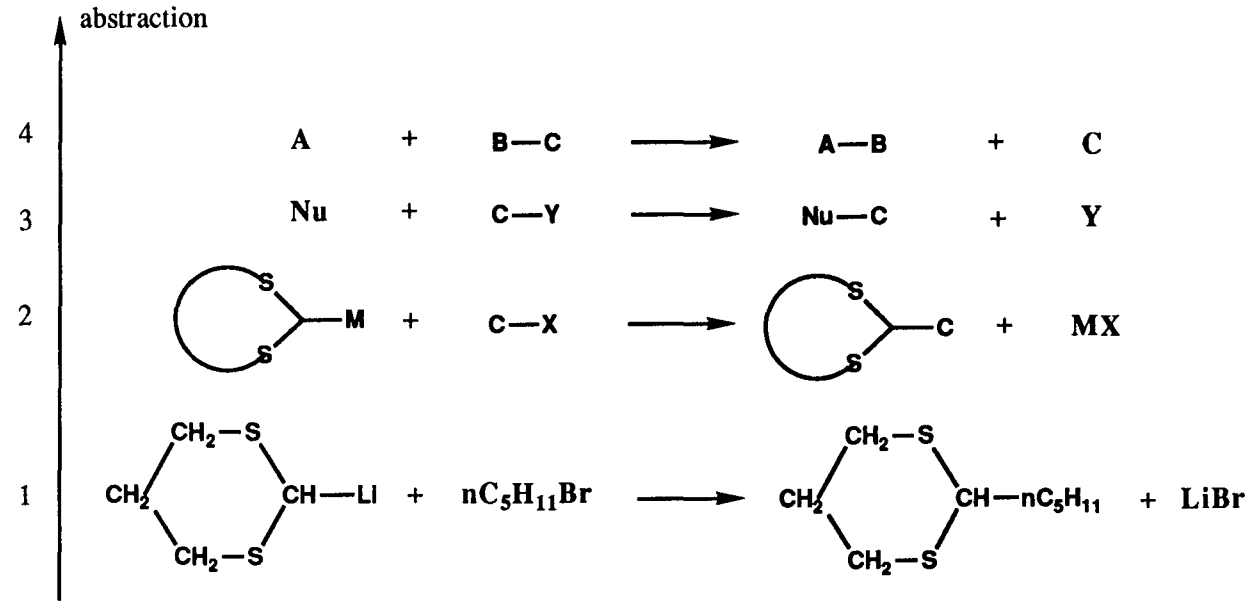


Figure 5.2. Une réaction envisagée selon quatre niveaux conceptuels.

tions réactionnelles génériques.

- La réaction élémentaire `Nu + C/Y ⟶ Nu/C + Y` fait intervenir les propriétés électroniques des réactifs en présence[1].
- Le niveau formel `A + B/C ⟶ A/B + C` donne le bilan de la redistribution des liaisons entre atomes (de type quelconque).

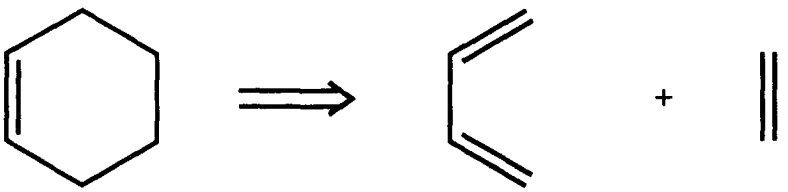
Une réaction modifie le squelette ou la fonctionnalité d'une cible. Lorsque cette modification a un intérêt synthétique majeur, elle sert à préparer une famille de produits par exemple, la réaction est appelée une *méthode de synthèse*. Ainsi, la méthode de Wittig :

$$\texttt{C//O + C/X} \longrightarrow \texttt{C//C}$$

permet de construire la double liaison `C//C` à partir de deux réactifs contenant respectivement les fragments `C//O` et `C/X`[2]. En réalité, la méthode de Wittig comprend elle-même plusieurs étapes intermédiaires. Le schéma réactionnel précédent en donne

1. Le symbole `Nu` désigne un atome nucléophile, qui est un atome ayant tendance à donner des électrons. Pour être compatible avec le code décrivant les objets chimiques, les liaisons sont notées avec le symbole / pour les liaisons simples, // pour les liaisons doubles et /// pour les liaisons triples.

2. Le symbole `X` désigne un halogène en général, qui est soit un fluor `F`, un chlore `Cl`, un brome `Br` ou un iode `I`. Il faut toutefois noter que la méthode de Wittig n'est généralement pas réalisée à partir de fluorures.



Sous-structure présente dans la cible

Sous-structures qui se trouvent dans les précurseurs

Figure 5.3. La transformation de Diels-Alder.

une vision macroscopique qui est la vision standard considérée dans la suite. Ce schéma réactionnel est l'expression *synthétique* de la méthode de Wittig. Il existe aussi une expression *rétrosynthétique* de la méthode de Wittig :

$$\texttt{C//C} \Longrightarrow \texttt{C//O} + \texttt{C/X}$$

Par convention [Corey *et al.*, 1985], l'expression rétrosynthétique d'une méthode de synthèse est appelée *transformation* et se note avec le symbole $\Longrightarrow$. La transformation de Wittig s'interprète alors de la façon suivante : pour construire le fragment `C//C` dans une molécule cible, il faut disposer de deux réactifs de départ, l'un renfermant un fragment de type `C/X` et l'autre un fragment de type `C//O`. La reconnaissance de fragments comme `C//C` dans une molécule cible est donc de première importance, car elle détermine l'ensemble des transformations applicables à la cible. Ainsi, la transformation de Diels-Alder n'est applicable à une molécule cible que si cette dernière renferme un cycle à six atomes bien particulier (cf. Fig. 5.3).

Parallèlement à la double interprétation d'une méthode de synthèse, la résolution d'un problème de synthèse est dirigée par les données ou par les buts. Dans le premier cas, le mode de résolution est *synthétique* ; dans le second, il est *rétrosynthétique*. Le mode rétrosynthétique relève de la résolution de problèmes par réduction et s'applique le plus souvent dans un tel contexte, car la molécule cible, qui correspond à l'état final, est généralement le seul état bien connu.

5.1.3 Le mode rétrosynthétique

Le mode rétrosynthétique a été formalisé par E.J. Corey [Corey and Wipke, 1969]. Il consiste à appliquer une ou plusieurs transformations à la molécule cible pour obtenir un certain nombre de précurseurs. Plus précisément, le mode rétrosynthétique

comporte une étape de reconnaissance de *rétrons* dans la cible. Les rétrons sont des sous-structures moléculaires susceptibles d'être obtenues par des méthodes de synthèse connues. L'étape de reconnaissance est suivie d'une étape de génération de précurseurs, qui sont obtenus après application aux rétrons des transformations correspondant aux méthodes de synthèse retenues à l'étape précédente. Le processus est ensuite réitéré sur les précurseurs, jusqu'à la génération de précurseurs connus, qui deviennent des produits de départ potentiels. L'arbre et-ou obtenu, ayant pour racine la cible et pour feuilles les produits de départ, est appelé *arbre de synthèse*.

Génération d'objectifs et de sous-objectifs

Comme dans tout problème de conception complexe, il n'est pas envisagable de trouver la solution d'un problème de synthèse en engendrant et en explorant systématiquement un espace d'états complet, qui serait obtenu en appliquant toutes les transformations possibles à une molécule cible, puis à ses précurseurs, et ainsi de suite. Une telle attitude serait irréaliste, car il existe des dizaines de milliers de molécules disponibles qui sont autant de produits de départ possibles, comme il existe des dizaines de milliers de réactions connues qui sont toutes des opérateurs potentiels. Un problème de synthèse doit donc être résolu sous le contrôle de *stratégies de synthèse* [Corey *et al.*, 1985] [Trombini, 1987a; Trombini, 1987b; Trombini, 1987c]. Ces stratégies consistent généralement à repérer les parties structurelles de la cible à construire en priorité, constructions qui deviennent autant d'*objectifs structurels*, puis à ordonner chronologiquement ces constructions. Les stratégies de synthèse permettent ainsi de développer l'arbre de synthèse d'une molécule cible de la façon la moins combinatoire possible.

Lorsqu'un objectif structurel ne peut pas être atteint, parce que la transformation adéquate ne peut pas s'appliquer directement, la cible doit être aménagée par la mise en œuvre d'une liste ordonnée de *sous-objectifs*, qui correspondent généralement à des ajustements de fonctionnalité, échange, introduction ou suppression de groupes fonctionnels sur un site réactionnel. Ainsi, la première étape du chemin de synthèse de la figure 5.4 est un ajustement de fonctionnalité qui conditionne l'application de la transformation de Diels-Alder.

Les stratégies associées au mode rétrosynthétique

Il est possible de classer les objectifs en objectifs structurels lorsque l'intérêt de la stratégie porte d'abord sur la structure de la molécule cible, en objectifs mécaniques lorsque l'intérêt de la stratégie porte sur les transformations. Voici quelques stratégies pouvant être associées au mode rétrosynthétique :

- La stratégie orientée par la topologie consiste à identifier une ou plusieurs liaisons dont la déconnexion va conduire à une simplification importante de la structure de la cible.
- La stratégie orientée par la stéréochimie consiste à privilégier la conservation ou bien la suppression des stéréocentres présents dans la molécule cible.

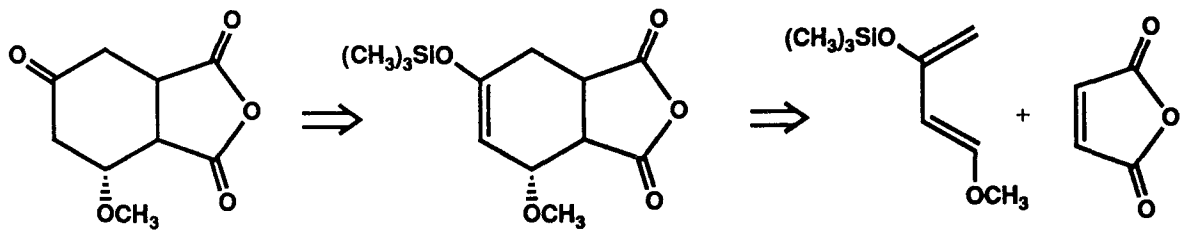


Figure 5.4. L'application de la transformation de Diels-Alder est conditionnée par l'obtention d'un sous-objectif, qui correspond à un ajustement de fonctionnalité.

- La stratégie orientée par les groupes fonctionnels s'appuie sur l'idée que les sites réactionnels se situent dans l'environnement des groupes fonctionnels. Elle consiste à examiner les groupes fonctionnels présents dans la cible et à repérer les liaisons pouvant être créées grâce à la présence de ces groupes.

- La stratégie orientée par les transformations consiste à privilégier l'utilisation d'une transformation signifiante dans le contexte étudié. L'utilisation de la transformation de Diels-Alder montrée à la figure 5.4 donne un bon exemple de cette stratégie.

- La stratégie orientée par la symétrie consiste à déterminer les parties symétriques et, le cas échéant, à privilégier l'application de transformations qui fragmentent la cible en parties symétriques (le même précurseur est alors obtenu plusieurs fois).

Une autre stratégie importante est la stratégie orientée par les produits de départ, qui s'articule autour de deux étapes principales :

- La molécule cible est comparée à une liste de produits de départ potentiels facilement accessibles. La comparaison tient compte de la structure et de la fonctionnalité. Les produits de départ retenus sont ceux qui sont les plus "proches" de la cible, pour un critère donné.

- Un chemin de synthèse menant des produits de départ retenus à la molécule cible est déterminé synthétiquement ou rétrosynthétiquement.

La recherche de produits de départ appropriés est un problème excessivement compliqué. Dans [Wipke and Rogers, 1984], les auteurs proposent un algorithme pour simplifier une molécule cible et rechercher dans une base de molécules celles qui ont un squelette structurellement proche de celui de la cible pouvant servir de produits de départ.

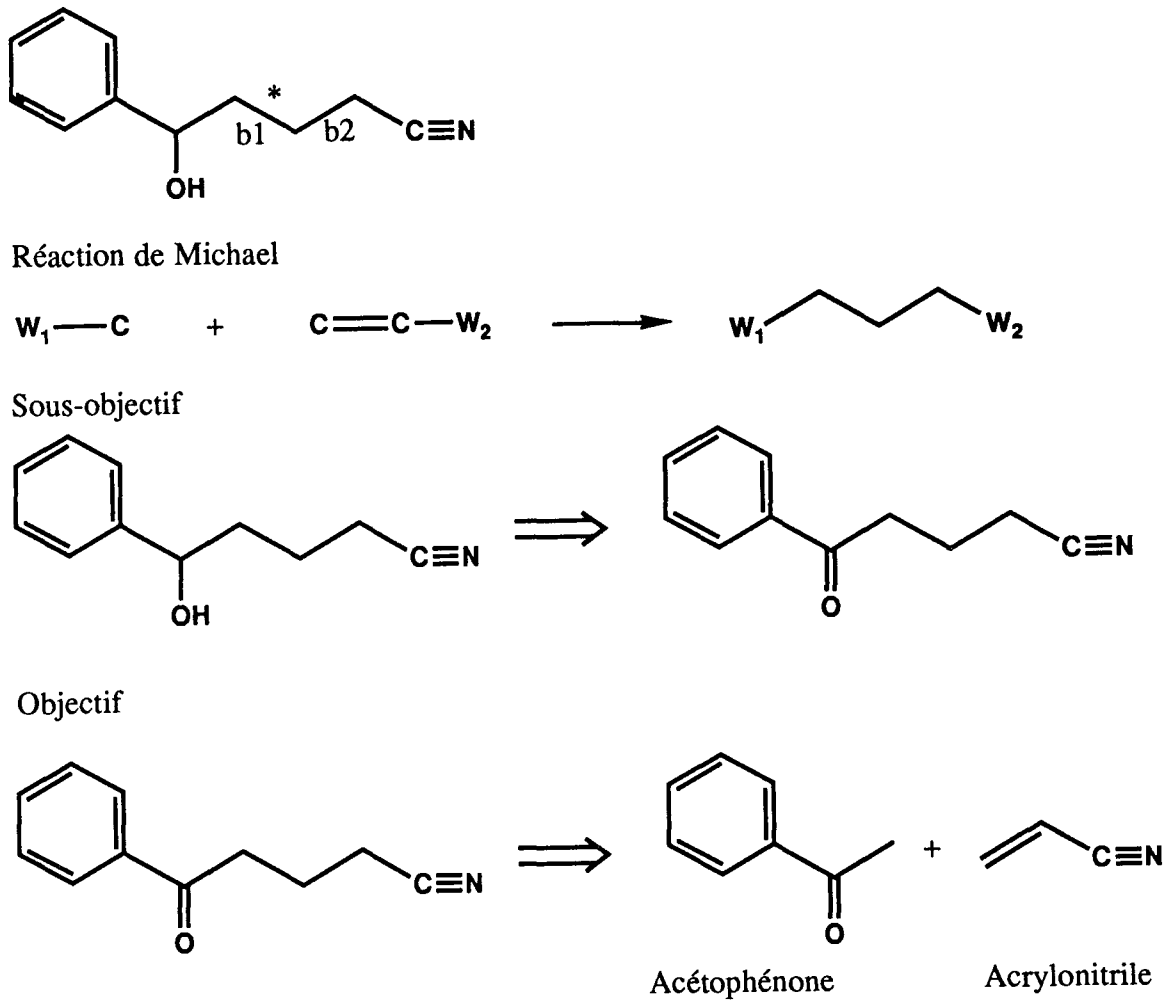


Figure 5.5. La construction de la liaison marquée par * est un objectif. Cet objectif est atteint grâce à la transformation de Michael, qui n'est applicable qu'après modification de la fonctionnalité de la cible.

L'appariement entre la cible et un produit de départ potentiel n'est jamais complet. Les différences qui apparaissent peuvent être exploitées pour mettre au point le plan de synthèse de la cible. Ainsi, le groupe de A.P. Johnson, qui travaille sur le système de synthèse LHASA et sur la base de données chimiques ORAC, a une démarche intéressante et rigoureuse [Johnson, 1985] : étant donné une molécule cible, il faut d'abord trouver un produit de départ structurellement proche de la cible, ensuite définir une série d'objectifs à atteindre en comparant la cible et le produit de départ choisi, puis construire le plan de synthèse de la cible en utilisant un mode rétrosynthétique. Considérons par exemple la molécule cible décrite à la figure 5.5. Cette cible possède une chaîne carbonée latérale qui présente une certaine parenté avec l'acrylonitrile (C//C/C///N). Toutefois, la chaîne carbonée est plus longue et elle est saturée (toutes les liaisons carbone-carbone sont simples), ce qui met en valeur deux objectifs : casser

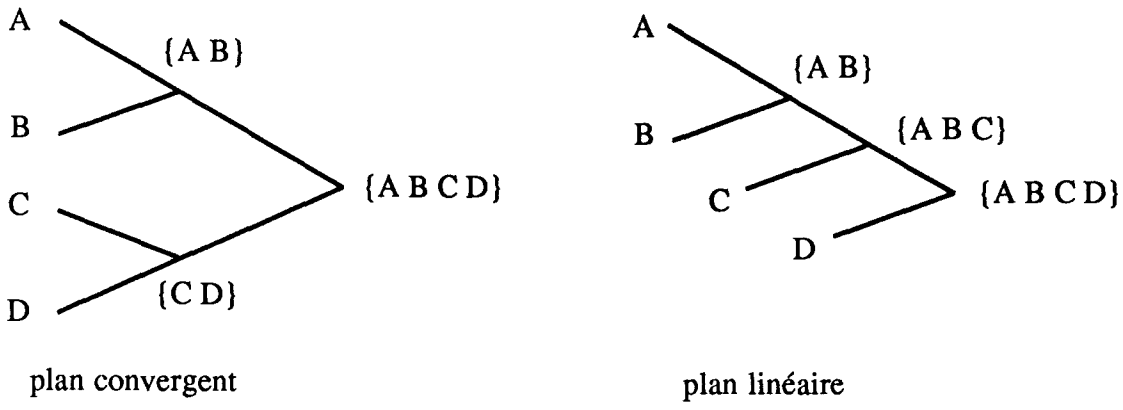


Figure 5.6. Plans de synthèse linéaire et convergent.

la liaison notée *b1* et rendre double la liaison notée *b2*. Il faut alors trouver une méthode de synthèse qui permettent d'atteindre l'un des objectifs, ou mieux, les deux. La transformation de Michael est potentiellement applicable, moyennant un ajustement de fonctionnalité dans la cible. Une façon simple d'obtenir la fonctionnalité désirée est d'échanger l'alcool C/OH avec une cétone C//O. Ainsi, le chemin de synthèse permettant de passer de la cible aux deux réactifs de départ compte deux étapes, la première matérialisant un sous-objectif, la seconde un objectif.

Une autre approche intéressante et importante sur le point de vue économique est celle de J.B. Hendrickson. Cette approche consiste à fragmenter en deux parties (et deux fois de suite au plus) le squelette de la molécule cible de toutes les façons possibles, sous les conditions suivantes [Hendrickson, 1990] : chaque fragment contient au moins trois atomes de carbone et il s'apparie avec au moins un produit de départ répertorié dans un catalogue. Une fois tous les fragments possibles obtenus, la formation d'une liaison qui réunit deux fragments constitue un objectif à atteindre. Le critère retenu pour ordonner les objectifs est celui de la *convergence maximale* du plan résultant. La convergence d'un plan de synthèse se mesure au nombre d'étapes du plan qui peuvent être effectuées en parallèle, donc indépendamment les unes des autres. Plus un plan de synthèse est convergent, plus la synthèse est courte, et en général, meilleur est le rendement, et moins la synthèse est chère (cf. Fig. 5.6).

Une note sur le mode synthétique

Le mode *synthétique* consiste à appliquer une ou plusieurs méthodes de synthèse à un ensemble de réactifs de départ vérifiant certaines caractéristiques [Jauffret *et al.*, 1986]. En réalité, il n'existe pas encore de formalisation de stratégies de développement de chemins de synthèse qui soit associée à ce mode de résolution. L'intérêt majeur du mode synthétique est de s'assurer de la pertinence du choix d'une méthode de synthèse pour résoudre un problème : le produit souhaité est majoritaire ou la méthode choisie n'est pas en compétition avec d'autres. En ce sens, le mode synthétique complète le

mode rétrosynthétique en contrôlant la validité et la qualité d'un chemin de synthèse.

5.2 La conception artificielle de plans de synthèse

5.2.1 L'approche classique

Ce sont sur les idées qui viennent d'être présentées que sont bâtis les systèmes d'aide à la conception de plans de synthèse, appelés aussi systèmes de synthèse assistée par ordinateur, SAO en abrégé. Ces systèmes ont pour but d'aider les chimistes à établir des plans de synthèse. Actuellement, une des fonctions essentielles de ces systèmes est de transmettre des connaissances et un savoir-faire, puisés chez un ou plusieurs experts en synthèse organique, à des non spécialistes, étudiants, chimistes et ingénieurs de l'industrie chimique venant d'autres domaines, etc.. Le système "idéal" d'aide à la conception de plans de synthèse comporte trois modules principaux :

- Un module graphique gère les entrées-sorties, entrée des connaissances en général, entrée des données relatives à un problème particulier, saisie graphique des produits de départ et visualisation graphique des résultats obtenus.
- Un module de planification perçoit les caractéristiques structurelles de la molécule cible et élabore en conséquence un plan de construction préliminaire correspondant à une série d'objectifs à atteindre.
- Un module de simulation de méthodes de synthèse ou de transformations s'occupe de percevoir dans le détail les caractéristiques fonctionnelles de la cible. Il produit en conséquence une liste de sous-objectifs visant à adapter la cible afin que les transformations choisies pour atteindre les objectifs fixés puissent être appliquées.

La réalisation complète d'un tel système est loin d'être achevée et demande des moyens relativement importants en hommes et en machines.

5.2.2 Notes sur quelques systèmes de SAO

Les systèmes procéduraux

La majorité des programmes de SAO fonctionnant actuellement sont *procéduraux*. Brièvement[3], ces systèmes sont tous bâtis sur la même architecture, qui comprend une base de transformations et un programme chargé d'appliquer ces transformations à des réactifs (cf. Fig. 5.7). Les transformations décrivent des réactions et leurs conditions

3. La thèse de Claude Laurenço, qui contient une revue et une analyse critique des principaux systèmes procéduraux est la référence française en la matière [Laurenço, 1985]. Le document de présentation de l'exposition *Computer Session*, tenue lors de la 8ième *International IUPAC Conference on Organic Synthesis*, à Helsinki du 23 au 27 juillet 1990, procure une bonne actualisation des principaux systèmes de synthèse assistée par ordinateur.

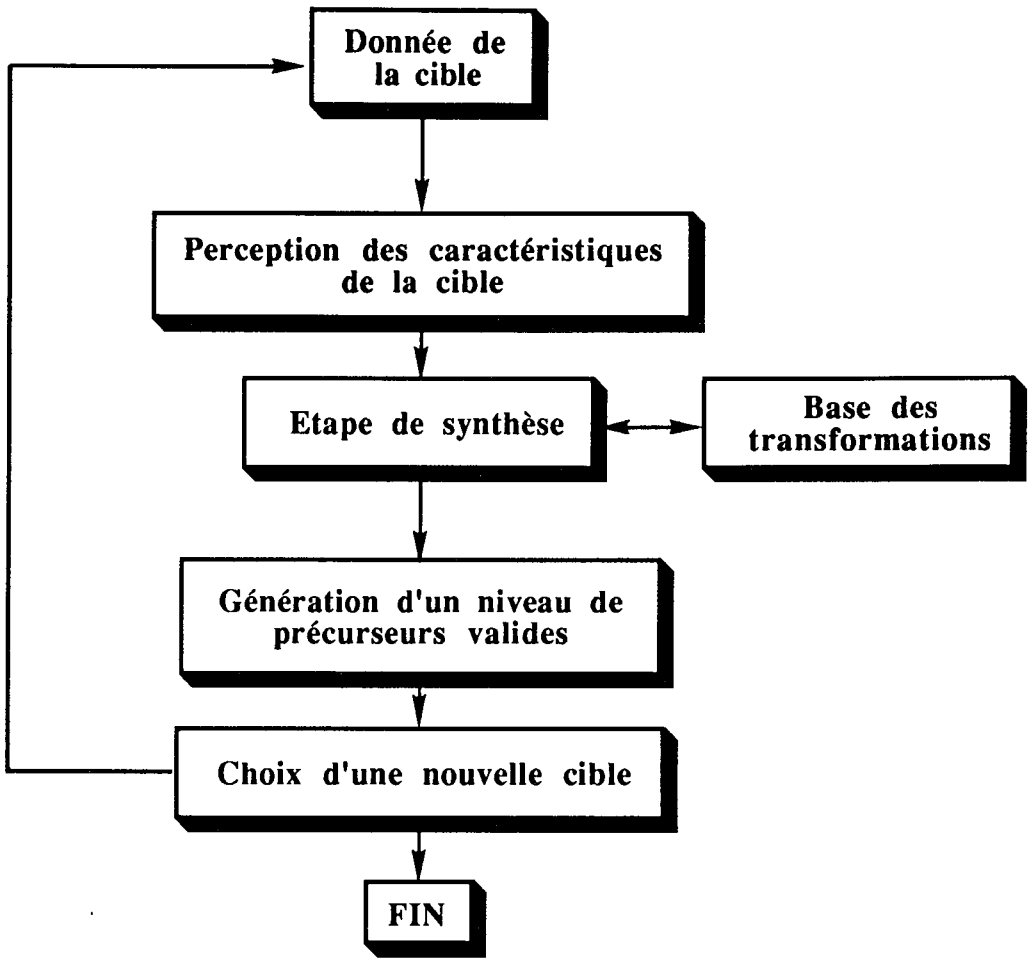


Figure 5.7. Schéma de fonctionnement standard d'un système de SAO procédural employant un mode rétrosynthétique.

d'applicabilité. Elles sont représentées par des procédures écrites dans des langages spécialisés. Les systèmes procéduraux sont généralement interactifs et l'utilisateur se voit sollicité pour choisir de nouvelles cibles ou encore pour élaguer l'arbre de synthèse. Ils permettent de gérer de façon efficace différentes bases de données chimiques, mais souffrent des limitations inhérentes aux représentations procédurales [Winograd, 1975] : représentation des molécules sous la forme de tables de connexion difficiles à manipuler (cf. Fig. 5.8), représentations des transformations peu lisibles et peu évolutives, etc.. Ces derniers points expliquent la relative pauvreté des différentes bases de données associées à ces systèmes.

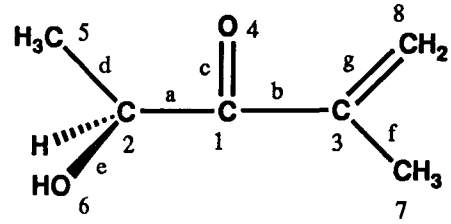


Table de connexion des atomes

Atome No.	Type	Stéréocentre	Nombre atomes	Atomes voisins	Nombre liaisons	Liaisons voisines
1	1 (C)	0	3	234	4	abc
2	1	1	3	165	3	aed
3	1	0	3	178	4	bfg
4	4 (O)	0	1	1	2	c
5	1	0	1	2	1	d
6	4	0	1	2	1	e
7	1	0	1	3	1	f
8	1	0	1	3	2	g

Table de connexion des liaisons

Liaison No.	Type	Stéréochimie	Atome1	Atome2
a	1	0	1	2
b	1	0	1	3
c	2	0	1	4
d	1	0	2	5
e	1	1	2	6
f	1	0	3	7
g	1	0	3	8

Figure 5.8. Une molécule et la table de connexion qui lui est associée (d'après [Barone and Chanon, 1986]).

Les systèmes fondés sur la logique

Toute une génération de systèmes à base de connaissances particulièrement bien adaptés au diagnostic s'appuient sur le calcul des prédicats ou sur le formalisme des règles de production. Théoriquement, un expert construit ou aide à construire une base de connaissances, où sont déclarés les faits et les règles propres au domaine d'application, sous la forme de formules logiques ou de règles de production, sans se soucier de leur manipulation. Cette tâche est laissée à un moteur d'inférences,

chargé de la manipulation des connaissances et du raisonnement [Laurière, 1982]. A l'instar de DENDRAL [Buchanan and Feigenbaum, 1978], des systèmes spécialisés dans l'interprétation de spectres de masse ou de spectres RMN ont été conçus selon ce modèle. En revanche, peu de systèmes de SAO ont été bâtis en utilisant cette approche, qui se prête mal à la construction d'un système destiné à résoudre un problème complexe comme celui de la synthèse organique : règles et formules logiques servent à représenter naturellement et facilement des relations existant entre des objets, mais s'avèrent plutôt inadaptées lorsqu'il s'agit de représenter les objets eux-mêmes. Ainsi, seuls quelques prototypes de systèmes d'analyse de sites synthétiques utilisent un formalisme "logique" [Wipke and Dolata, 1986] [Napoli *et al.*, 1986] [Pfertzel-Ramphft, 1989], les capacités de ces systèmes étant loin d'égaler celles des systèmes procéduraux.

Quelques limitations des systèmes existants

Les faiblesses des systèmes de SAO existant actuellement sont principalement liées aux techniques de représentation des connaissances et de raisonnement utilisées :

- Les molécules sont représentées par des tables de connexion. Les enquêtes évaluant la potentialité synthétique d'un fragment dans une molécule sont traduites dans des langages spécialisés. Ces différents types de représentation sont contraignants pour le chimiste et relativement infidèles, car ils ne permettent d'exprimer que partiellement le savoir-faire de l'expert. Les représentations obtenues sont le plus souvent redondantes, imprécises, difficiles à améliorer et à faire évoluer.

- Les connaissances ne sont décrites qu'à un niveau superficiel, comme dans la plupart des systèmes experts de première génération. En particulier, les réactions ne sont prises en compte qu'à un seul niveau de représentation, en général trop spécifique. Or, des connaissances décrites sur plusieurs niveaux de généralité sont plus fidèles et utilisables dans des contextes différents. La description de l'univers est ainsi plus riche et la manipulation des objets représentés plus précise. Tout ou partie des niveaux conceptuels présentés à la figure 5.2 est utilisable lors de l'analyse et de la manipulation d'une molécule cible. Ainsi, le niveau générique sert à choisir une ou plusieurs méthodes de synthèse et le niveau spécifique sert à valider ces choix.

- Le volume de connaissances disponibles est généralement assez faible, sauf dans certains cas très particuliers [Gelernter *et al.*, 1984]. Cette limitation n'est cependant pas réservée aux seuls systèmes de SAO.

- La formalisation des stratégies reste encore très insuffisante.

Plus récemment, dans le domaine de la chimie organique et de la synthèse en particulier, quelques systèmes à base de connaissances ont été réalisés en utilisant des représentations à objets. En dehors du système qui va être présenté de façon détaillée dans la suite, citons le système d'élaboration de chemins de synthèse OASIS [Sun, 1989], qui a été développé au CRIN/INRIA LORRAINE, le système d'interprétation de spectres

SICLOB [Marie *et al.*, 1989] et le système de description de mécanismes réactionnels et d'apprentissage MAROCO [Marie, 1991].

5.2.3 Une nouvelle approche de la SAO

Un système de synthèse assistée par ordinateur a pour but de simuler le raisonnement du chimiste durant l'étape de planification, donc de délivrer un plan de synthèse, qui doit être composé d'un ensemble ordonné d'objectifs correspondant à autant d'applications de réactions à des ensembles de réactifs intermédiaires. Il est possible de distinguer trois niveaux d'abstraction dans la conception d'un plan de synthèse :

- Au niveau politique sont définis des objectifs premiers comme la synthèse totale ou partielle de la molécule cible.
- Au niveau stratégique sont planifiés des objectifs structurels qui correspondent à la construction des différentes parties de la structure de la cible.
- Au niveau tactique sont choisies les différentes méthodes de synthèse (ou transformations) permettant d'atteindre les objectifs structurels.

Les systèmes actuellement opérationnels ont pour la plupart une bonne connaissance tactique, les choix politiques et stratégiques étant laissés à l'utilisateur. En outre, la représentation procédurale dont usent la plupart des systèmes actuellement opérationnels rend leur évolutivité difficile, mais surtout n'offre pas la possibilité d'utiliser facilement et naturellement les différents niveaux d'abstraction pouvant traduire les natures fort diverses des connaissances du domaine.

Dans le modèle que nous avons conçu, une synthèse est vue comme un système temporel qui évolue d'un état initial vers un état final en passant par une suite d'états intermédiaires [Laurenço *et al.*, 1990]. Un état est décrit par un réacteur se composant d'un ensemble d'objets primaires qui sont des atomes et des liaisons. La connectivité des liaisons détermine les objets secondaires qui sont les molécules entrant en jeu dans la synthèse. Un état intermédiaire de la synthèse au temps $t+1$ correspond à une redistribution des liaisons due à l'action des transformations de synthèse applicables au réacteur existant au temps t. Les états sont ordonnés dans le temps et matérialisent des objectifs devant être atteints au cours de la synthèse. Un objectif peut à son tour se décomposer en plusieurs sous-objectifs. En choisissant un mode rétrosynthétique, l'état initial est constitué d'une molécule cible et l'état final d'un ensemble de réactifs de départ connus et disponibles. Les actions associées à un objectif ont un caractère constructif, comme la modification du squelette de la cible par exemple, tandis que celles qui sont associées à un sous-objectif correspondent à des ajustements de fonctionnalités.

La réalisation d'un système à base de connaissances modélisant une telle approche de la synthèse nécessite le choix d'un formalisme de représentation. L'expérience acquise lors de l'étude et de l'élaboration d'autres systèmes dans le domaine nous a amené

à utiliser une représentation à objets implantée en YAFOOL et le raisonnement par classification [Napoli, 1990]. Puisqu'il fallait lui donner un nom, le système résultant a été baptisé YCHEM, "Y" pour Y3 bien sûr [Ducournau, 1989b]...

Dans notre approche, il est important que l'expert en synthèse ait une idée de la modélisation et de la programmation. En effet, contrairement à l'approche "ingénieur cogniticien-expert", où le premier individu monopolise le pouvoir relatif à la machine, il est indispensable que le chimiste ait une connaissance et une certaine conscience des outils employés, de leurs avantages et de leurs limites, surtout en ce qui concerne les modes de représentation des connaissances et de raisonnement choisis. Ainsi, lors du transfert d'expertise, les traditionnels problèmes d'incompréhension entre informaticien au sens large et chimiste se voient en tout cas fortement diminués, s'ils ne disparaissent pas complètement : le premier ne peut plus promettre au second ce que le second ne peut plus espérer avoir. Notre approche est distribuée et honnête, au sens où elle considère que l'expert est un réel collaborateur, qui a donc un rôle à jouer dans la modélisation et, éventuellement, dans la programmation lors de l'élaboration du système.

5.3 YCHEM : une modélisation des objets de la chimie organique

L'objet `Objet-chimique` est la racine de la hiérarchie d'héritage ou univers qui contient les modèles de base utilisés en chimie organique, les atomes et les différents types atomiques, les liaisons et les liaisons fonctionnelles, les molécules et les structures moléculaires particulières. Une molécule est composée d'objets primaires qui sont des atomes et des liaisons ; une liaison est composée de deux atomes extrémités et possède un ordre. Plusieurs molécules existant simultanément et devant réagir sont insérées dans réacteur. L'application d'une transformation aux réactifs présents dans un réacteur provoque une redistribution des liaisons à l'intérieur du réacteur. Au niveau de l'implantation, les modèles d'objets chimiques sont représentés par des frames dotés d'un ensemble d'attributs qui caractérisent l'état du modèle, et d'un ensemble de méthodes, au sens programmation par objets cette fois, qui décrivent les différentes opérations applicables au modèle. Ces frames sont appelés indistinctement objets, frames, ou, s'il n'y a aucune ambiguïté, classes. Les représentants de ces frames sont appelés instances. Le terme "méthode" employé seul désigne une méthode associée à un objet, par opposition à "méthode de synthèse".

5.3.1 Les atomes

L'atome générique

Un atome est représenté par un objet qui est une spécialisation directe de `Objet-chimique`. L'atome générique possède deux sortes d'attributs, les premiers décrivent les propriétés de l'atome en tant qu'entité autonome, les seconds décrivent son envi-

```
(defmodele Atome
  (sorte-de
    ($valeur Objet-chimique))
  (valence
    ($un entier)
    ($intervalle [1 6]))
  (numero
    ($un entier)
    ($intervalle [1 103]))
  (charge
    ($un entier)
    ($domaine [-2 -1 0 1 2])
    ($defaut 0))
  (masse
    ($un reel))
  (incidentes
    ($liste-de Liaison)
    ($lien-inverse extremites)
    ($defaut nil))
  (molecule
    ($un Molecule)
    ($lien-inverse atomes))
  (configuration
    ($un symbole)
    ($domaine [R S])
    ($defaut nil))
  (degre
    ($methode +degre))
  (voisins
    ($methode +voisins))
  (creation
    ($methode creer-atome))
  (subsumep
    ($methode a-subsumep))
  (matchp
    ($methode a-matchp)))
```

Figure 5.9. L'objet décrivant l'atome générique.

ronnement (cf. Fig. 5.9). Ainsi, `valence`, `numero` pour le numéro atomique, `charge`, et `masse`, font partie de la première catégorie. Les attributs `incidentes`, pour la liste des liaisons dans lesquelles est engagé l'atome, `molecule`, pour le nom de la molé-

```
methode +degre (frame)
  s ← 0 ;
  pour liaison ∈ (lire frame incidentes) faire
    s ← s + (lire liaison ordre) ;
  retourner((lire frame valence) + (lire frame charge) - s)
```

Figure 5.10. Définition formelle de la méthode `degre`.
L'expression `(lire frame attribut)` retourne la valeur de l'attribut `attribut` de l'objet `frame`.

cule qui contient l'atome, `configuration` pour la configuration spatiale des atomes tétracoordinés, sont des attributs qui appartiennent à la seconde catégorie. Les attributs `incidentes` et `molecule` établissent des relations entre le frame `Atome` et les frames `Liaison` et `Molecule`, de relations inverses `extremites` et `atomes` respectivement (cf. 2.1.1). Chaque fois qu'une liaison est créée ou détruite, la valeur de l'attribut `incidentes` des atomes extrémités est automatiquement mise à jour. De même, chaque fois qu'une nouvelle molécule est créée, chacun de ses atomes voit la valeur de l'attribut `molecule` mise à jour.

L'objet `Atome` est pourvu de cinq méthodes. La méthode `degre` calcule l'état d'hydrogénation d'un atome en fonction de sa charge, de sa valence et de la somme des ordres des liaisons incidentes. Pour une majorité d'atomes, ce calcul consiste à sommer la valence et la charge de l'atome et à soustraire la somme des ordres des liaisons incidentes, la charge pouvant être éventuellement négative (cf. Fig. 5.10). Cette formule de calcul est redéfinie en particulier pour l'hydrogène et le carbone. Par analogie avec la théorie des graphes, où le degré d'un sommet dans un graphe non orienté compte le nombre d'arêtes incidentes au sommet, la méthode `degre` est utilisée pour vérifier que les modifications apportées à l'environnement structurel d'un atome dans une molécule sont correctes. Plus le "degré d'un atome" est proche de la valeur de la valence de l'atome, plus l'atome peut accepter de nouveaux voisins. Inversement, si le degré est nul, l'atome n'accepte plus de nouveau voisin.

La méthode `creation` est l'équivalent d'une méthode d'instanciation dans un langage de classe. Elle sert à créer des représentants de l'objet `Atome` et masque la méthode standard de création de représentants associée à `Objet-ideal`, racine du graphe d'héritage.

Pour un atome donné, la méthode `voisins` calcule la liste des atomes voisins en fonction des liaisons incidentes à l'atome. Cette liste de voisins est calculée chaque fois que c'est nécessaire par une méthode. Un tel calcul pourrait aussi être effectué par un réflexe si-besoin. Toutefois, dans notre cas, une valeur calculée par un réflexe si-besoin est systématiquement mise en place dans la facette `$valeur` associée à l'attribut. Il est alors nécessaire de prévoir des réflexes *a posteriori* qui gèrent l'évolution de la valeur

de l'attribut et de celle des attributs qui en dépendent.

Les méthodes **subsumep** et **matchp** sont des opérateurs utilisés pour résoudre des problèmes de synthèse. La première possède un rôle particulièrement important dans le raisonnement par classification. Elle vérifie que l'environnement d'un atome **A1** est plus général que celui d'un atome **A2** : **A1** et **A2** doivent avoir le même type atomique et l'environnement de **A1** doit être moins contraint que celui de **A2**, autrement dit le degré du premier doit être supérieur au degré du second (cf. § 3.2.7). La méthode **matchp** teste si un atome **A1** s'apparie avec un atome **A2** en vérifiant cette fois que **A1** et **A2** ont non seulement le même type atomique mais aussi le même degré.

Les informations figurant dans l'objet **Atome** sont d'ordre macroscopique, car elles ne tiennent pas compte d'éléments comme les couches électroniques externes par exemple, qui ont une certaine importance lorsque les mécanismes réactionnels entrent en jeu [Marie, 1991].

Le carbone et quelques types atomiques particuliers

L'objet **Atome** possède de nombreuses spécialisations, dont l'hydrogène (**H**), le carbone (**C**), l'oxygène (**O**), le phosphore (**P**), etc., (cf. Fig. 5.11). Les spécialisations de **Atome** sont des modèles générateurs des instances particulières qui sont effectivement manipulées dans une application.

La méthode **degre** est redéfinie pour l'hydrogène et le carbone. En réalité, il n'existe pas de calcul général du degré d'un atome et les formules présentées ici ne traitent que les cas les plus courants en chimie organique. Par exemple, le gaz rare xénon, qui a normalement une valence égale à 0, a une valence égale à 4 dans le tétrafluorure de xénon XeF4 ; l'iode, qui a une valence normalement égale à 1, a pour valence 7 dans le périodate de sodium NaIO4. Le soufre, le phosphore, le bore sont d'autres cas spéciaux. Par défaut, la valence la plus haute possible est attribuée à ces types atomiques. Il est très délicat de vouloir établir une formule générale du calcul du degré d'un atome sans connaître *a priori* la valence de l'atome étudié, donc sans avoir un aperçu global de la molécule étudiée. Les formules utilisées sont des approximations valides pour tous les cas courants. En l'occurrence, pour l'hydrogène, la valeur délivrée par **degre** s'obtient en soustrayant la valeur absolue de la **charge** à la **valence**, les atomes **H+** et **H-** ne pouvant être liés à un autre atome. Pour le carbone, il faut soustraire à la valence de l'atome la valeur absolue de la charge et la somme des ordres des liaisons incidentes. Les redéfinitions du calcul du degré établies pour l'hydrogène et le carbone sont valables pour les atomes faisant respectivement partie de la même colonne que l'hydrogène et le carbone dans la classification périodique des éléments.

La table de Mendeleev

La classification périodique des éléments est une organisation des 103 atomes répertoriés actuellement par numéro atomique croissant (cf. Fig. 5.12). Elle est représentée par un tableau à deux entrées, appelé aussi table de Mendeleev. Le statut d'un atome, son état et son comportement, dépendent d'informations "internes" et "externes" : les

```
(Hydrogene
  (sorte-de ($valeur Atome))
  (valence ($valeur 1))
  (numero ($valeur 1))
  (masse ($valeur 1.0079))
  (degre ($methode +degre-H)))

(Carbone
  (sorte-de ($valeur Atome))
  (valence ($valeur 4))
  (numero ($valeur 6))
  (masse ($valeur 12.011))
  (degre ($methode +degre-C)))

(Oxygene
  (sorte-de ($valeur Atome))
  (valence ($valeur 2))
  (numero ($valeur 8))
  (masse ($valeur 15.99994)))

(Phosphore
  (sorte-de ($valeur Atome))
  (valence ($domaine [3 5])
           ($defaut 5))
  (numero ($valeur 15))
  (masse ($valeur 30.9738)))
```

Figure 5.11. Les objets `Hydrogene`, `Carbone` et `Oxygene` sont parmi les atomes les plus utilisés en chimie organique. L'objet `Phosphore` décrit un atome ayant plusieurs valences possibles.

premières sont de trois types, intrinsèques ou directement associées à l'atome, liées à la ligne et liées à la colonne où se trouve situé l'atome dans le tableau. Les secondes proviennent de l'environnement moléculaire dans lequel se trouve l'atome.

La classification périodique des éléments n'est pas, à l'image des classifications qui ont été envisagées jusqu'ici, hiérarchique : tous les atomes ont le même statut et sont considérés sur le même plan. Cependant, s'il n'existe pas d'ordre naturel, il existe des équivalences naturelles (cf. § 3.4.3) et donc différents points de vue pour envisager les informations attachées à un atome. Voici par exemple une liste de points de vue possibles :

Tableau périodique des éléments

Numéro atomique → 6
C ← Symbole
Carbone ← Nom
Masse atomique → 12.01

I A												III A	IV A	V A	VI A	VII A	VIII A
1 H Hydrogène 1.008	II A							VIII B			I B	II B					2 He Hélium 4.00
3 Li Lithium 6.94	4 Be Béryllium 9.01												5 B Bore 10.81	6 C Carbone 12.01	7 N Azote 14.01	8 O Oxygène 16.00	9 F Fluor 19.00	10 Ne Néon 20.18
11 Na Sodium 22.99	12 Mg Magnésium 24.30	III B	IV B	V B	VI B	VII B							13 Al Aluminium 26.98	14 Si Silicium 28.09	15 P Phosphore 30.97	16 S Soufre 32.06	17 Cl Chlore 35.45	18 Ar Argon 39.95
19 K Potassium 39.10	20 Ca Calcium 40.08	21 Sc Scandium 44.96	22 Ti Titane 47.90	23 V Vanadium 50.94	24 Cr Chrome 52.00	25 Mn Manganèse 54.94	26 Fe Fer 55.85	27 Co Cobalt 58.93	28 Ni Nickel 58.70	29 Cu Cuivre 63.55	30 Zn Zinc 65.38		31 Ga Gallium 69.72	32 Ge Germanium 72.59	33 As Arsenic 74.92	34 Se Sélénium 78.96	35 Br Brome 79.90	36 Kr Krypton 83.80
37 Rb Rubidium 85.47	38 Sr Strontium 87.62	39 Y Yttrium 88.91	40 Zr Zirconium 91.22	41 Nb Niobium 92.91	42 Mo Molybdène 95.94	43 Tc Technétium 98.91	44 Ru Ruthénium 101.07	45 Rh Rhodium 102.91	46 Pd Palladium 106.4	47 Ag Argent 107.87	48 Cd Cadmium 112.41		49 In Indium 114.82	50 Sn Etain 118.69	51 Sb Antimoine 121.75	52 Te Tellure 127.60	53 I Iode 126.90	54 Xe Xénon 131.30
55 Cs Césium 132.91	56 Ba Baryum 137.33	57* La Lanthane 138.91	72 Hf Hafnium 178.49	73 Ta Tantale 180.95	74 W Tungstène 183.85	75 Re Rhénium 186.21	76 Os Osmium 190.2	77 Ir Iridium 192.22	78 Pt Platine 195.09	79 Au Or 196.97	80 Hg Mercure 200.59		81 Tl Thallium 204.37	82 Pb Plomb 207.2	83 Bi Bismuth 208.98	84 Po Polonium (209)	85 At Astate (210)	86 Rn Radon (222)
87 Fr Francium (223)	88 Ra Radium 226.03	89† Ac Actinium 227.03	104 § (261)	105 § (262)	106 § (263)	107 § (262)	108 § (265)	109 § (267)										

Lanthanides	58 Ce Cérium 140.12	59 Pr Praséodyme 140.91	60 Nd Néodyme 144.24	61 Pm Prométhium (145)	62 Sm Samarium 150.4	63 Eu Europium 151.96	64 Gd Gadolinium 157.25	65 Tb Terbium 158.93	66 Dy Dysprosium 162.50	67 Ho Holmium 164.93	68 Er Erbium 167.26	69 Tm Thulium 168.93	70 Yb Ytterbium 173.04	71 Lu Lutétium 174.97
Actinides	90 Th Thorium 232.04	91 Pa Protactinium 231.04	92 U Uranium 238.03	93 Np Neptunium 237.05	94 Pu Plutonium (244)	95 Am Américium (243)	96 Cm Curium (247)	97 Bk Berkélium (249)	98 Cf Californium (251)	99 Es Einsteinium (254)	100 Fm Fermium (257)	101 Md Mendélévium (258)	102 No Nobélium (259)	103 Lr Lawrencium (260)

§ I.U.P.A.C. n'a pas adopté de noms ou de symboles officiels pour ces éléments.
Les nombres décimaux sont représentés ici avec un point décimal et non avec une virgule.

Figure 5.12. La classification périodique des éléments, qui peut être utile pour lire ce chapitre.

- Point de vue interne intrinsèque : il contient les caractéristiques minimales que sont le numéro atomique et la masse.
- Point de vue interne "ligne" : les atomes qui font partie d'une même ligne ont une structure électronique semblable, qui comporte le même nombre de couches électroniques, mais qui diffère pour la dernière couche.
- Point de vue interne "colonne" : les atomes qui font partie d'une même colonne ont un comportement électronique semblable. Ils ont la même couche électronique externe, qui détermine leur valence, valence par défaut pour les atomes en ayant plusieurs, ainsi que leur électronégativité. Les colonnes sont habituellement répertoriées par un chiffre romain, qui dénote le nombre d'électrons sur la dernière couche. L'électronégativité mesure la tendance d'un atome à accepter des électrons et, sur une même ligne, elle croît de gauche à droite.
- Point de vue externe "structurel" : il décrit l'environnement de l'atome, la molécule à laquelle il appartient, la configuration de l'atome, ses liaisons incidentes et ses atomes adjacents.

Ces différents points de vue servent à représenter un atome d'une façon homogène, conforme à la vision du chimiste. Les points de vue n'interfèrent pas, mais communiquent, les informations détenues par l'un étant disponibles pour l'autre lorsque c'est nécessaire.

Les superatomes

Le comportement réactionnel *électroattracteur* (attracteur d'électrons) ou *électrodonneur* (donneur d'électrons) d'une sous-structure moléculaire s'explique en partie par la proximité de groupements structurels particuliers, suivant que ces groupements accumulent respectivement des atomes pauvres ou riches en électrons[4]. Il existe des familles de groupements électroattracteurs et électrodonneurs qui possèdent chacune des propriétés chimiques spécifiques. L'effet électroattracteur ou électrodonneur s'exerce à partir d'un atome particulier, appelé *atome central*, qui est à rapprocher de l'atome *origine* d'un groupement, introduit par E.J. Corey [Corey and Wipke, 1969].

Un *W-groupe* est un modèle de groupe fonctionnel dont les spécialisations ont un atome origine électroattracteur. La cétone et les aldéhydes (caractérisés par `C//O`), les esters (`O//C/O/C`), les amines (`C/N` et les nitriles `C///N`) en sont des exemples. Les W-groupes sont considérés comme des *superatomes* de symbole `W`. Ils ont un comportement chimique analogue mais une structure très différente, l'atome central pouvant être de type `C`, `P`, `N`, `S`, etc.. Pour un atome, le fait d'être électroattracteur est contextuel, puisqu'il dépend de l'environnement de l'atome, mais, pour un groupe, la propriété d'être électroattracteur est intrinsèque : le fait "une cétone est un groupe électroattracteur" n'est jamais remis en cause.

Les hétéroatomes et les halogènes sont deux autres familles d'atomes dont la représentation pose des problèmes analogues à celle des W-groupes. Les hétéroatomes se définissent en exprimant une exception, les halogènes en explicitant les membres de la famille : un hétéroatome, noté `A*`, est un atome de n'importe quel type, excepté carbone et hydrogène ; un halogène, noté `X`, est un fluor, ou un chlore, ou un brome, ou un iode. Il est très commode, comme pour les W-groupes, de décrire les hétéroatomes et les halogènes par des superatomes. Mais l'assimilation d'une famille à un élément pose certains problèmes pratiques, comme celui de fixer la masse du superatome, son numéro atomique ou encore sa valence.

La représentation des superatomes se fait de deux façons qui coexistent, en extension ou en intension (cf. § 2.1.2). Dans les trois cas précédents, il est facile de construire un prédicat qui sélectionne des W-groupes, des hétéroatomes ou des halogènes. En revanche, la représentation en intension est plus difficile à mettre en forme. La figure 5.13 montre la définition des superatomes `W`, `A*` et `X`. Le superatome `W` a par défaut des caractéristiques semblables à celles de l'atome de carbone, car le carbone est le plus souvent l'atome central d'un W-groupe. La méthode `subsumep` associée à `W` recherche les sous-structures qui sont susceptibles d'être des W-groupes, en vérifiant qu'une des liaisons incidentes au superatome est de même type qu'une des liaisons caractéristiques

4. Ces explications restent superficielles et la réalité s'avère plus complexe, car le type des électrons peut aussi entrer en jeu.

```
(defmodele W
  (sorte-de
    ($valeur Atome))
  (valence
    ($defaut 4))
  (numero
    ($defaut 103))
  (subsumep
    ($methode w-subsumep)))

(defmodele A*
  (sorte-de
    ($valeur Atome))
  (valence
    ($defaut 6))
  (numero
    ($defaut 103))
  (subsumep
    ($methode a-subsumep)))

(defmodele X
  (sorte-de
    ($valeur Atome))
  (valence
    ($valeur 1))
  (numero
    ($domaine [9 17 35 53])
    ($defaut 53))
  (subsumep
    ($methode x-subsumep)))
```

Figure 5.13. Les superatomes `W` pour les W-groupes, `A*` pour les hétéroatomes et `X` pour les halogènes.

des W-groupes, à savoir (`C//O C/N S//O P//O C//N C///N`).

Le superatome `A*` possède des attributs ayant des valeurs par défaut arbitrairement grandes, car il est censé représenter n'importe quel type atomique. Le superatome `X` est construit selon le même principe que `A*`. La valence est égale à 1 pour tous les halogènes, mais la valeur par défaut du numéro atomique est celui de l'iode, le plus grand des numéros atomiques des halogènes[5]. La méthode `subsumep` associée à `A*`

5. C'est la raison pour laquelle les superatomes sont en tête de la liste des atomes d'une structure

retourne vrai si le type atomique de l'atome auquel elle est appliquée est différent de `C` ou de `H`, à la manière du prédicat qui sélectionne les hétéroatomes dans une liste d'atomes. La méthode `subsumep` associée à `X` retourne vrai si le type atomique de l'atome auquel elle est appliquée est `F`, `Cl`, `Br` ou `I`.

Les problèmes de représentation qui viennent d'être évoqués sont liés à la représentation d'une famille d'objets par l'un d'entre eux, les objets étant regroupés à cause d'une certaine ressemblance, mais n'ayant pas forcément tous les mêmes caractéristiques [Winograd, 1978] [Borgida, 1988].

5.3.2 Les liaisons

La liaison générique

L'objet `Liaison` est caractérisé par les attributs `ordre`, qui indique l'ordre ou type de la liaison, `extremites` qui répertorie les deux atomes engagés dans la liaison, et `molecule`, qui donne le nom de la molécule à laquelle appartient la liaison (Fig. 5.14). Il existe cinq types de liaisons, les liaisons covalentes simple, double, triple, aromatique et la liaison ionique. Une liaison ionique unit un cation à un anion comme dans le cristal de chlorure de sodium `Na+Cl-` et a un ordre égal à 0. Deux atomes sont liés par une liaison covalente s'ils ont des orbitales atomiques ne contenant qu'un seul électron et que ces orbitales se combinent de sorte que les deux électrons appartiennent aussi bien à l'une qu'à l'autre. La liaison est simple ou d'ordre 1 si chaque atome ne met en jeu qu'une seule orbitale, double s'il y a deux orbitales et au plus triple s'il y en a trois. Le cas des liaisons aromatiques est plus compliqué. En première approximation, une liaison faisant partie d'un cycle aromatique est considérée comme une liaison aromatique[6]. Un cycle est aromatique s'il vérifie la loi de Hückel : le nombre d'électrons *pi* doit être égal à $4n+2$, où n est un entier naturel [Métivier, 1987]. Il y a deux électrons *pi* pour une liaison double. Ainsi, pour un benzène, qui peut être considéré comme un cycle à 6 liaisons, dont 3 sont doubles, le nombre d'électrons *pi* est 6, donc égal à $4n+2$, pour n valant 1. En réalité, l'ordre d'une liaison aromatique est fixé à 1.5 car les liaisons doubles sont délocalisées dans un cycle aromatique, chaque liaison du cycle étant équivalente ($6*1.5 = 3*2 + 3*1$).

Avant la création d'une nouvelle liaison, le réflexe `liaison?`, associé à l'attribut `extremites`, teste la validité de la formation de cette nouvelle liaison en vérifiant que le degré de chaque atome engagé dans la liaison est supérieur ou égal à l'ordre de la liaison à construire (cf. Fig. 5.15). Le réflexe si-ajout `+extremites` associé à l'attribut `extremites` sert à ordonner les deux atomes engagés dans la liaison, en mettant en tête l'atome de plus haut numéro atomique et de plus haut degré.

Une relation d'ordre plus élaborée, appelée *signature*, tient également compte de l'environnement structurel de l'atome [Fischer and Napoli, 1991]. Elle est définie par la formule :

---

moléculaire

6. Il s'agit bien d'une approximation, car il est possible de rencontrer des liaisons non aromatiques dans un cycle aromatique [Métivier, 1987].

```
(defmodele Liaison
  (sorte-de
    ($valeur Objet-chimique))
  (ordre
    ($un Nombre)
    ($domaine [0 1 1.5 2 3])
    ($defaut 0))
  (extremites
    ($liste-de Atome)
    ($lien-inverse incidentes)
    ($si-possible liaison?)
    ($min 2)
    ($max 2)
    ($si-ajout +extremites)
    ($defaut nil))
  (molecule
    ($un Molecule)
    ($lien-inverse liaisons))
  (creation
    ($methode creer-liaison))
  (subsumep
    ($methode b-subsumep))
  (matchp
    ($methode b-matchp)))
```

Figure 5.14. L'objet `Liaison` décrit la liaison générique.

*sgn(A) = 100 * no-at(A) + 10 * insaturation(A) + valence(A) - nb-H(A)*
où *no-at(A)* indique le numéro atomique de l'atome A, *nb-H(A)* indique le nombre d'atomes d'hydrogène liés à A. L'insaturation d'un atome A vaut 0 si l'atome est saturé au sens où toutes ses liaisons incidentes sont simples ; elle vaut 1 si l'atome fait partie d'un cycle aromatique ; elle vaut 2^*N s'il existe N liaisons doubles incidentes ; elle vaut 3^*N s'il existe N liaison triples incidentes.

Ajouter un atome à une structure moléculaire passe obligatoirement par la création d'une nouvelle liaison. Il ne faut pas que l'environnement d'un atome soit modifié sans qu'une nouvelle liaison soit créée ou supprimée. Il est donc naturel que les tâches liées à la modification de l'environnement d'un atome soit de la responsabilité des liaisons et non de celle des atomes. Ainsi, la mise en place ou la suppression des atomes extrémités d'une liaison provoque la mise à jour des attributs `incidentes` des atomes extrémités de la liaison, car l'attribut `extremites` définit une relation avec le frame `Atome`, de relation inverse `incidentes`. Comme pour `Atome`, l'attribut `molecule`

```
fonction liaison? (frame atome-1 atome-2)
  ordre ← (lire frame ordre) ;
  si (degre atome-1) ≥ ordre
  et (degre atome-2) ≥ ordre
     alors retourner(vrai)
     sinon retourner(faux) fsi

fonction +extremites (frame atome-1 atome-2)
  cas si (lire atome-1 numero) > (lire atome-2 numero)
         alors (ecrire frame extremites (atome-1 atome-2))
       si (lire atome-1 numero) = (lire atome-2 numero)
       et (degre atome-1) ≥ (degre atome-2)
         alors (ecrire frame extremites (atome-1 atome-2))
       autre cas (ecrire frame extremites (atome-2 atome-1))
```

Figure 5.15. Définitions formelles du réflexe si-possible `liaison?` et du réflexe si-ajout `+extremites`.

définit une relation avec le frame `Molecule`, de relation inverse `liaisons`.

Puisqu'une liaison est constituée de deux atomes et d'un ordre, la valeur de l'attribut `incidentes` du frame `Atome` permet de connaître la liste des atomes qui sont liés à un atome à un instant donné, calcul effectué par la méthode `voisins`. Si `voisins` était un attribut, il serait adjacent à l'attribut `incidentes`, car la connaissance de la valeur de l'un entraîne la connaissance de la valeur de l'autre (il s'agit ici d'adjacence au sens des attributs, voir page 53). Toutefois, il est plus simple de considérer `voisins` comme une méthode, et non comme un attribut[7] : en premier lieu, il ne serait pas possible de définir une relation entre l'attribut `voisins` et l'attribut `extremites`, parallèle à celle qui existe entre `incidentes` et `extremites`, car le co-domaine de `voisins` serait `Atome` et non `Liaison` ; en second lieu, puisque les valeurs des attributs `voisins` et `incidentes` évoluent en parallèle, il serait nécessaire de mettre en place un mécanisme *ad hoc* qui gère l'évolution de ces valeurs, mécanisme calqué sur celui des relations et qui s'appuie sur des réflexes *a posteriori* [Napoli *et al.*, 1991].

Comme le frame `Atome`, le frame `Liaison` est pourvu de méthodes de sélecteurs `creation`, `matchp` et `subsumep` qui effectuent les mêmes tâches que les méthodes homonymes associées au frame `Atome`. La première crée une nouvelle liaison avec pour données un couple d'atomes et un ordre. Les méthodes `subsumep` et `matchp` modélisent les définitions suivantes : une liaison *L1* subsume une liaison *L2* si *L1* et *L2* ont le même ordre et si les atomes extrémités de *L1* subsument les atomes extrémités

7. La redondance d'information qui serait entraînée par la coexistence des deux attributs serait compensée par une meilleure lisibilité des atomes manipulés dans une application.

correspondants de *L2*, sachant que les atomes extrémités sont rangés dans un ordre bien établi (cf. § 3.2.7) ; une liaison *L1* s'apparie avec une liaison *L2* si *L1* et *L2* ont même ordre et si les atomes extrémités de *L1* s'apparient avec les atomes extrémités correspondants de *L2*. Par exemple, la liaison générique `C/X`, où `X` est le superatome halogène, subsume les liaisons `C/F`, `C/Cl`, `C/Br` et `C/I`, sans que l'inverse soit vrai.

Les liaisons de type spécial

Le frame `Liaison` ne possède pas de spécialisations représentant les différents types de liaisons, simples, doubles, triples et aromatiques. Cependant, certaines spécialisations génériques comme les doubles liaisons carbone-carbone, carbone-oxygène, etc., qui possèdent des propriétés spécifiques, sont représentées par des groupes fonctionnels simples (cf. § 5.3.4). Par ailleurs, toute liaison contenue dans une molécule est appelée à évoluer dans le temps. Avec un tel modèle de liaison, la modification de l'environnement d'une liaison se traduit par la seule modification des valeurs des attributs `ordre` et `extremites` attachés à la liaison modifiée. Cette opération de mise à jour de valeurs d'attributs est beaucoup moins coûteuse qu'une migration d'objets dans le graphe d'héritage, qu'il faudrait envisager dès qu'une liaison verrait son ordre ou ses extrémités modifiés [Nguyen and Rieu, 1989].

5.3.3 Les molécules

La structure moléculaire

La molécule générique est représentée par l'objet `Molecule`, dont les attributs `atomes` et `liaisons` ont respectivement pour valeur la liste des atomes et des liaisons de la molécule, chaque atome étant un représentant du frame `Atome`, et chaque liaison un représentant du frame `Liaison` (Fig. 5.16).

Ainsi, une molécule en tant que composition d'atomes et de liaisons est un exemple d'objet composite "faible" (cf. § 3.4.1). La méthode `creation` sert à construire un représentant de `Molecule`, en fonction d'une liste de connexion qui est obtenue à partir du dessin de la molécule. Le détail de la création d'un tel représentant va nous permettre d'expliciter le fonctionnement des différents réflexes attachés aux objets `Atome` et `Liaison`. La création comporte trois phases principales :

- Un représentant "vide" de `Molecule` est tout d'abord créé, ses attributs `atomes` et `liaisons` n'ayant aucune valeur.
- Les objets représentant les atomes de la molécule sont ensuite créés, suivant la donnée du type et de la charge des atomes. Ces objets sont rangés dans la liste constituant la valeur de l'attribut `atomes`. Le mécanisme des relations met à jour la valeur de l'attribut `molecule` de chaque représentant de l'objet `Atome`. Le réflexe si-ajout `+atomes` trie ensuite la liste des atomes en fonction du numéro atomique des atomes et place les atomes de plus haut numéro atomique en tête de liste. Cette information est notamment utilisée lors des appariements de structures moléculaires.

```
(defmodele Molecule
  (sorte-de
    ($valeur Objet-chimique))
  (atomes
    ($liste-de Atome)
    ($lien-inverse molecule)
    ($si-ajout +atomes)
    ($defaut nil))
  (liaisons
    ($liste-de Liaison)
    ($lien-inverse molecule)
    ($si-ajout +liaisons)
    ($defaut nil))
  (creation
    ($methode creer-molecule))
  (subsumep
    ($methode m-subsumep))
  (matchp
    ($methode m-matchp)))
  (+liaison
    ($methode +liaison)))
  (-liaison
    ($methode -liaison)))
  (-atome
    ($methode -atome)))
```

Figure 5.16. L'objet `Molecule` décrit la structure moléculaire générique.

- Les objets représentant les liaisons sont finalement créés, avec pour données l'ordre et les extrémités de chaque liaison. L'initialisation de la valeur de l'attribut `molecule` pour chaque représentant de l'objet `Liaison` se fait comme pour les atomes. La mise en place des atomes extrémités d'une liaison est précédée de l'activation du réflexe `liaison?` qui vérifie que la liaison peut effectivement être construite. Cette activation provoque en cascade l'envoi d'un message de sélecteur `degre` à chaque atome extrémité. Une fois la mise en place des extrémités autorisée, le mécanisme des relations met à jour la valeur de l'attribut `incidentes` de chaque atome extrémité et le réflexe si-ajout `+extremites` ordonne les atomes extrémités dans chaque liaison. Pour finir, un tri des liaisons de la molécule qui vient d'être construite est effectué par le réflexe si-ajout `+liaisons` attaché à l'attribut `liaisons` de l'objet `Molecule`. La relation d'ordre utilisée pour le tri compare deux liaisons en considérant leur *poids*, qui est calculé en fonction du numéro atomique des atomes extrémités et de l'ordre de la liaison. Une liaison est

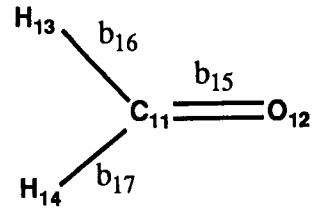


Figure 5.17. La molécule de formol.

d'autant plus "lourde" qu'elle possède un ordre élevé et des atomes extrémités de numéro atomique élevé[8].

Voici par exemple la représentation simplifiée de la molécule de formaldéhyde, plus connue sous le nom de formol (cf. Fig. 5.17), et de deux de ses composants, le carbone `c11` et la liaison `b15`[9] :

```
(formol
   (est-un ($valeur Molecule))
   (atomes ($valeur (o12 c11 h13 h14)))
   (liaisons ($valeur (b15 b16 b17))))

(c11
   (est-un ($valeur C))
   (incidentes ($valeur (b15 b16 b17)))
   (molecule ($valeur formol)))

(b15
   (est-un ($valeur Liaison))
   (ordre ($valeur 2))
   (extremites ($valeur o12 c11))
   (molecule ($valeur formol)))
```

La subsomption et l'appariement de structures moléculaires

Les méthodes `subsumep` et `matchp` permettent de comparer deux structures moléculaires entre elles. Elles implantent respectivement la *subsomption de structures* (ou co-subsomption, voir les paragraphes 3.2.7 et 3.4.1) et l'*appariement de structures* :

- Une molécule *M1* *co-subsume*, ou plus simplement *subsume* s'il n'y a aucune ambiguïté possible, une molécule *M2*, si *M1* est une sous-structure de *M2*, au

8. Le poids d'une liaison peut aussi être défini comme la somme des signatures de chaque atome extrémité.

9. Dans le code, le lien `est-un` lie une instance à son modèle.

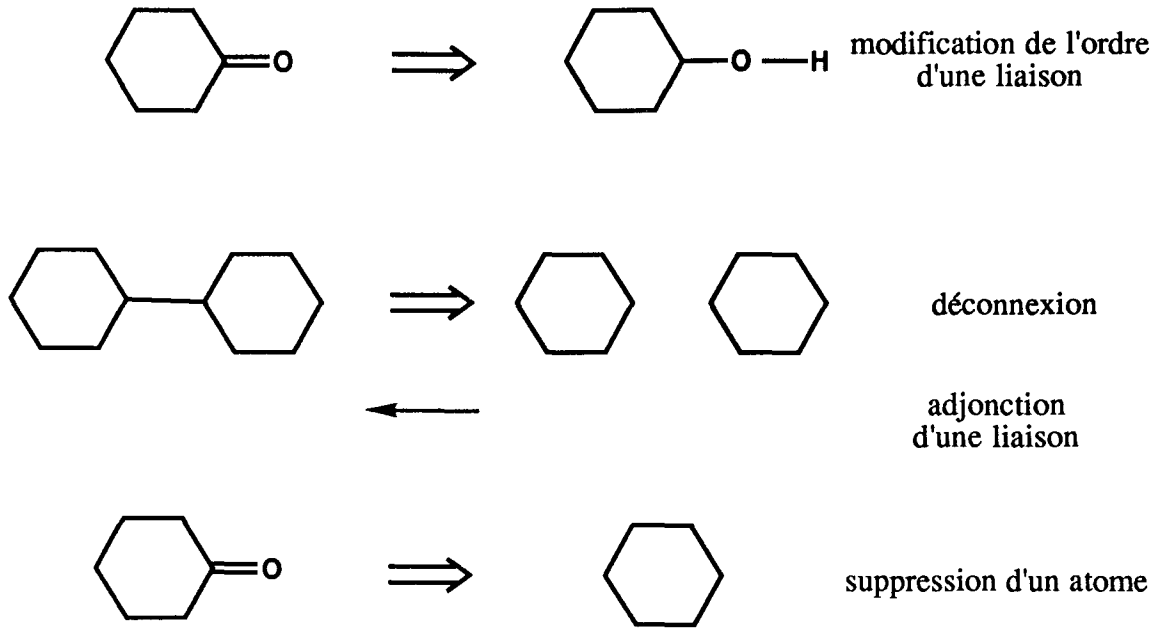


Figure 5.18. Les principales opérations de chirurgie qui sont effectuées sur une structure moléculaire.

sens où le graphe associé à *M1* est un sous-graphe[10] du graphe associé à *M2*, et si les atomes et les liaisons de *M1* subsument respectivement les atomes et les liaisons de *M2*. La molécule *M1* est quelquefois appelée le *dominant* et *M2* le *dominé*.

- Une molécule *M1* *s'apparie* avec une molécule *M2* si le graphe associé à *M1* est isomorphe au graphe associé à *M2*.

Les opérations de chirurgie moléculaire

Les opérations effectuées sur une structure moléculaire sont essentiellement de trois types, l'adjonction, la suppression et la modification de l'ordre d'une liaison (cf. Fig. 5.18). Parallèlement, l'adjonction et la suppression d'un atome sont deux opérations qui dérivent des deux premières. Seule la suppression d'un atome est définie par convenance. La suppression d'une liaison est effectuée par la méthode `-liaison` et consiste à supprimer les extrémités `a1` et `a2` de la liaison, ce qui provoque la mise à jour des attributs `incidentes` de `a1` et `a2` par l'intermédiaire du mécanisme des relations. L'attribut `molecule` associé à chaque extrémité est mis à jour si les atomes `a1` et `a2` sont associés à une nouvelle structure moléculaire.

10. La relation est à prendre au sens de la théorie des graphes : l'ensemble des sommets caractérisant *M1* est un sous-ensemble de l'ensemble des sommets caractérisant *M2* ; la relation définissant la connexion des atomes dans *M1* est la restriction à *M1* de la relation définissant la connexion des atomes dans *M2*.

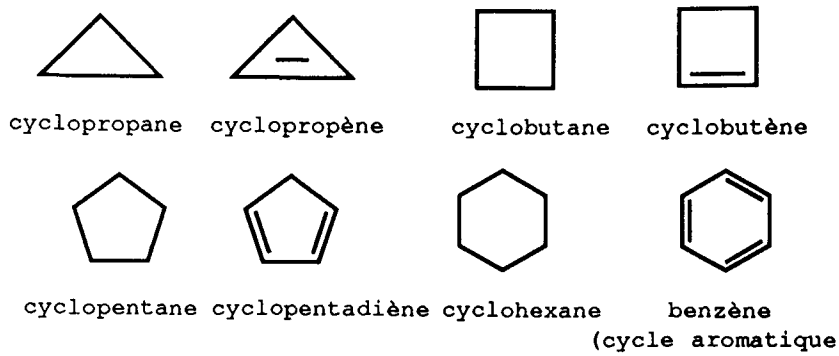


Figure 5.19. Quelques cycles fondamentaux.

Rappelons qu'une synthèse est modélisée par une suite de réacteurs qui sont constitués d'un ensemble d'atomes et de liaisons évoluant au cours du temps. Pour conserver la trace des opérations effectuées sur les réacteurs, la structure représentant la molécule de départ dans une opération de chirurgie n'est pas modifiée, mais une nouvelle structure est créée, pour rendre compte des modifications introduites dans le réacteur. Dans le cas où le graphe associé à une structure moléculaire de départ n'est plus connexe, il y a création d'autant de nouvelles molécules que de composantes connexes existantes. Cette façon de faire est conservée en cas d'utilisation d'objets temporels (cf. § 5.4.2).

L'adjonction d'une liaison à une structure moléculaire `M` est effectuée par la méthode `+liaison`. Elle consiste à lier un atome `a2` à un atome `a1` déjà présent dans la structure. L'adjonction se déroule en deux phases : vérifications préliminaires incluant un test sur la présence de `a1` dans `M`, puis création d'une nouvelle liaison entre `a1` et `a2`, en fonction de la donnée de `a2` et de l'ordre de la liaison à construire. Les mises à jour qui résultent de la mise en place de la nouvelle liaison sont identiques à celles qui sont effectuées lors de la création d'une molécule. La modification de l'ordre d'une liaison déjà existante est un cas particulier de l'opération d'adjonction.

Si l'adjonction d'un atome est une opération qui ne peut être réalisée sans qu'il y ait adjonction de liaison, la suppression d'un atome est une opération implantée de façon indépendante, qui s'avère pratique dans certains cas, bien qu'elle puisse s'exprimer comme la suppression de l'ensemble des liaisons incidentes à l'atome.

Les structures cycliques

Les cycles et polycycles sont des structures moléculaires particulières, spécialisations de `Molecule`. La figure 5.19 recense les cycles de base qui servent à construire les systèmes polycyliques. La méthode `subsumep` est redéfinie pour les structures cycliques. Elle permet de retrouver les cycles qui dominent une molécule et n'est activée que lorsque l'entier défini par la formule $1+ \mid liaisons \mid - \mid atomes \mid$ est strictement

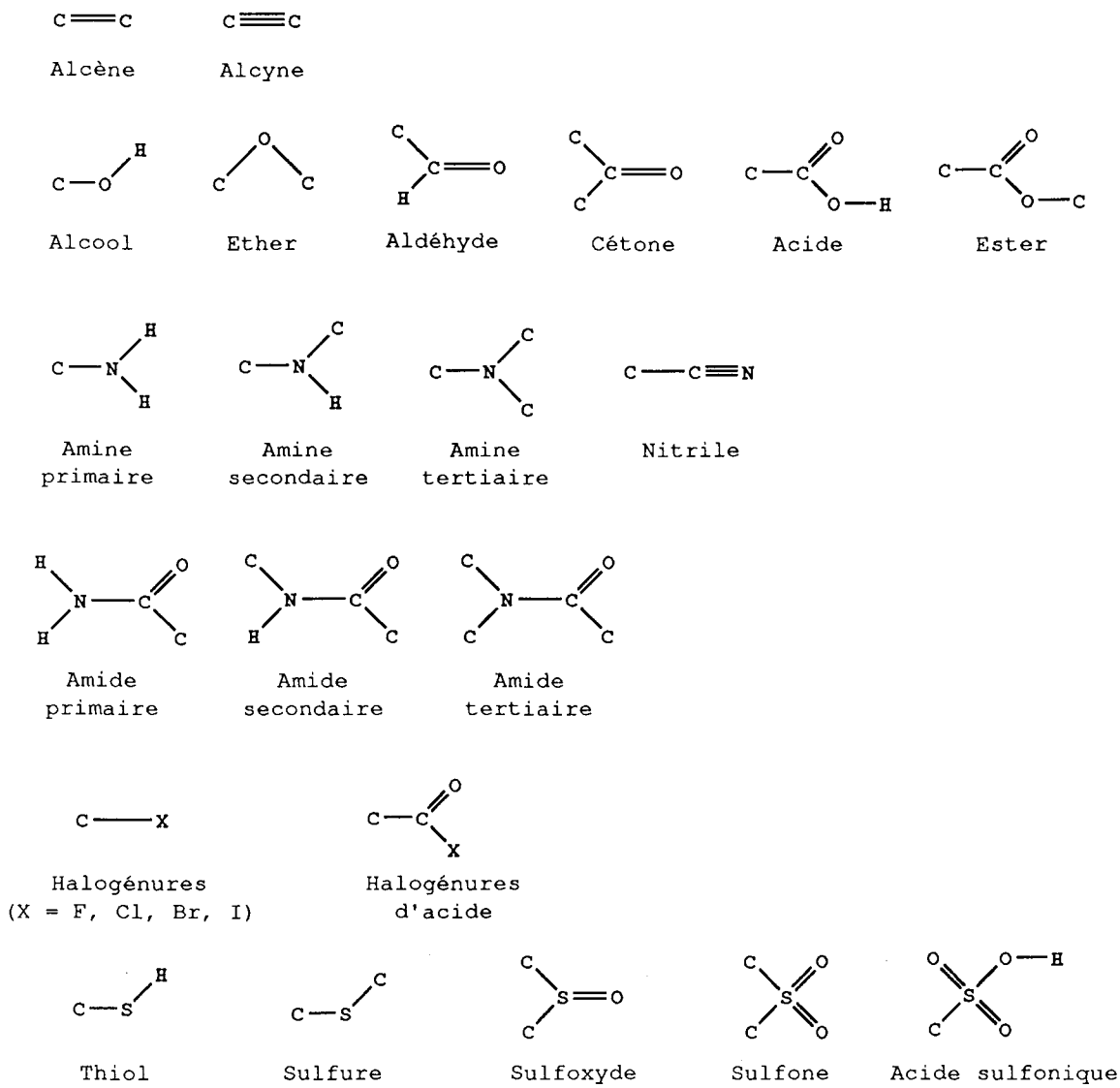


Figure 5.20. Quelques groupes fonctionnels.

positif[11], où | *atomes* | et | *liaisons* | désignent respectivement le nombre d'atomes et de liaisons de la structure moléculaire.

5.3.4 Les groupes fonctionnels

Un groupe fonctionnel est une sous-structure moléculaire décrivant une propriété caractéristique d'une famille de molécules. Les fonctions alcool `C/O/H` ou cétone `C//O` en sont des exemples (cf. Fig. 5.20). Comme une molécule, un groupe fonctionnel est une composition d'atomes et de liaisons. Il a, de ce fait, une représentation très

11. La formule est mise en défaut pour les molécules dont le graphe est non planaire, car elle dérive de la formule d'Euler, qui est définie pour les graphes planaires : | *faces* | + | *sommets* | − | *arcs* |= 2.

proche de celle d'une molécule. Toutefois, pour traiter plus efficacement les problèmes d'appariement et de recherche de sous-structures, les groupes fonctionnels dérivent de deux objets génériques, `Groupe1` et `Groupe2`. Le premier décrit les groupes simples qui ne sont constitués que d'une seule liaison, alors que le second décrit les groupes plus complexes qui comportent plus d'une liaison. La séparation en groupes primaires et groupes secondaires est uniquement un artifice de calcul qui ne recouvre aucune réalité chimique.

Les groupes primaires ou simples

Les groupes primaires comprennent les liaisons génériques carbone-carbone d'ordre multiple comme `C//C`, `C///C`, les liaisons génériques carbone-hétéroatome courantes comme `C/O`, `C//O`, `C/X` où `X` est un halogène, etc., ou encore les liaisons génériques hétéroatome-hétéroatome comme `O/O`. Un groupe fonctionnel simple dérive de `Groupe1`, spécialisation de `Liaison`. Voici par exemple la représentation associée au groupe fonctionnel cétone `C//O` :

```
(defmodele C//O
   (est-un ($valeur Groupe1))
   (ordre ($valeur 2))
   (type-extremites ($valeur (C O))))
```

La méthode `subsumep` est redéfinie pour les groupes simples. Un groupe simple *g1* domine une instance *b1* de `Liaison`, si *b1* a le même ordre et des extrémités de même type atomique que *g1* ; *g1* domine une molécule *M* s'il existe une liaison dans *M* qui soit subsumée par *g1*. Par exemple, le groupe cétone `C//O` subsume ou domine la molécule de formol présentée plus haut (cf. Fig. 5.17). Il existe aussi une représentation arborescente des groupes fonctionnels simples, qui est conforme à la notation linéaire des chemins de synthèse présentée dans [Laurenço *et al.*, 1990] (voir aussi le paragraphe 5.5.2). Cette représentation sert à construire un nom local unique utilisé pour effectuer des reconnaissances locales de groupes fonctionnels [Fischer and Napoli, 1991].

Les groupes secondaires ou composés

Les groupes secondaires représentent des sous-structures complexes comportant plus d'une liaison, qui sont construites à partir des groupes primaires. Ils sont représentés par l'objet `Groupe2`, spécialisation de `Molecule`. Un groupe secondaire est indifféremment une chaîne d'atomes comme le groupe alcool `C/O/H`, ou une structure plus complexe comme les carbonates ou les esters.

La description des objets chimiques statiques les plus importants est terminée. La figure 5.21 montre le graphe d'héritage où sont classés ces objets.

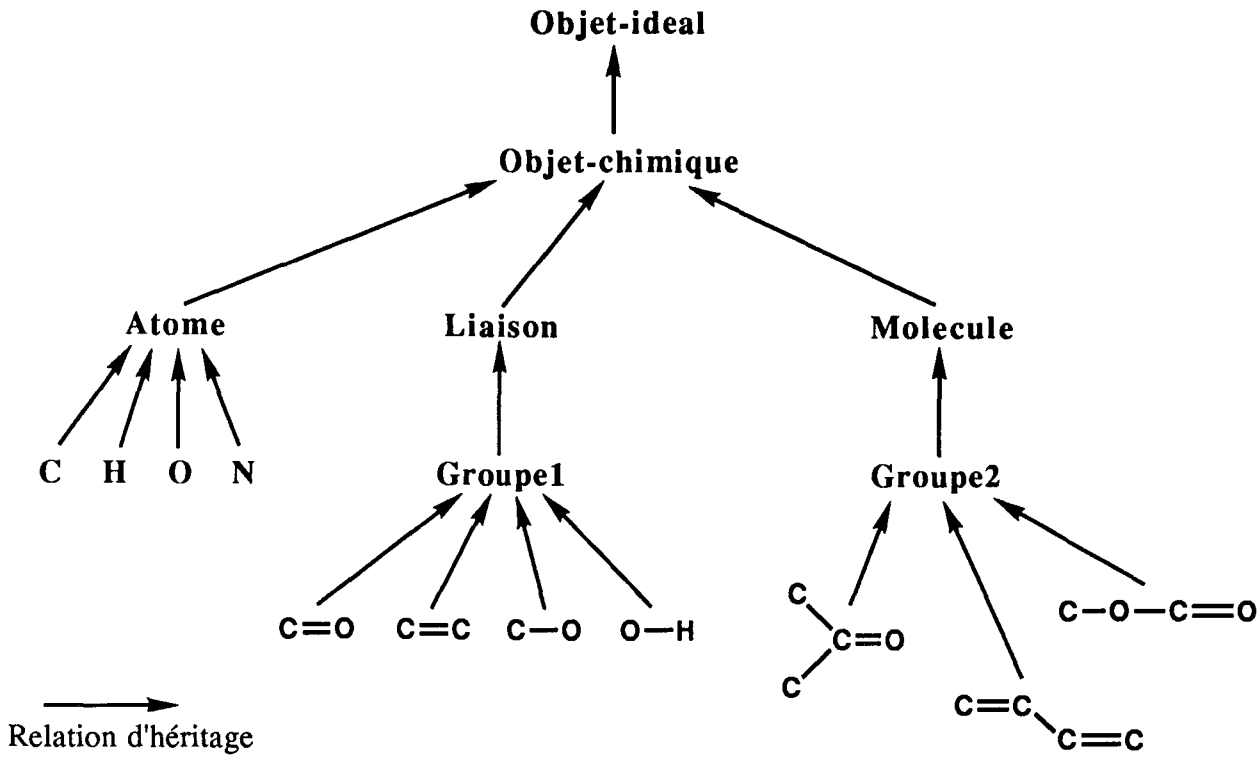


Figure 5.21. L'univers `Objet-chimique`.

5.3.5 Les transformations

Une réaction est un processus par lequel une ou deux molécules appelées *réactifs* se transforment en une ou deux molécules différentes appelées *produits*. Une réaction requiert de la chaleur (réaction thermique) ou de la lumière (réaction photochimique). Un certain nombre d'autres *conditions réactionnelles* doivent également être remplies. L'application d'une réaction provoque la rupture, la formation ou la modification de l'ordre d'une liaison. Les réactions se divisent en réactions de *construction* et réactions d'*échange* de groupes fonctionnels. Les premières servent à bâtir une partie d'un squelette moléculaire, comme la méthode de Wittig qui construit la double liaison carbone-carbone par exemple. Les secondes permettent de passer d'un groupe fonctionnel à l'autre comme la réaction faisant passer d'un alcool `C/O/H` à une cétone `C//O` (déshydrogénation ou oxydation des alcools) ou d'un nitrile `C///N` à une amine `C/NH2` (réduction par hydrogénation catalytique). Dans les deux cas, une réaction se résume à une composition de fonctions élémentaires de chirurgie, adjonction et suppression de liaisons.

La conception de plans de synthèse présentée dans la suite s'appuie sur un mode de résolution de problèmes de synthèse rétrosynthétique. Ce sont donc des transformations qui sont prises en compte, autrement dit l'expression rétrosynthétique des réac-

```
methode wittig (cible)
Reconnaissance dans la cible des sites réactionnels C//C potentiels
  sites-reactionnels ← (chercher C//C cible)
  Choix d'un site réactionnel C//C
  c//c ← (choix sites-reactionnels)
  Mémorisation des extrémités du site C//C retenu
  c1 ← (premier (lire c//c extremites))
  c2 ← (second (lire c//c extremites))
  Mise en place du réacteur
  reacteur ← (concatener (lire cible atomes) (lire cible liaisons))
  Suppression de la liaison C//C retenue
  reacteur ← (-liaison reacteur c//c)
  Construction de la nouvelle liaison C//O
  reacteur ← (+liaison reacteur c1 (creation O) 2)
  Construction de la nouvelle liaison C/X
  reacteur ← (+liaison reacteur c2 (creation X) 1)
  Construction des deux nouvelles molécules produites
  produits ← (creation (composantes-connexes reacteur))
```

Figure 5.22. Une description formelle de la méthode qui est associée à l'objet C//C et qui implante la transformation de Wittig : C//C $\Longrightarrow$ C/X + C//O.

tions. Une transformation est représentée par une méthode attachée à l'objet décrivant la sous-structure produite par la transformation. Par exemple, la transformation de Wittig C//C $\Longrightarrow$ C/X + C//O est associée à l'objet décrivant le groupe fonctionnel C//C (une description formelle de la transformation est donnée à la figure 5.22).

5.4 Synthèse et raisonnement par classification

5.4.1 Le mode rétrosynthétique réactualisé

Dans la suite, résoudre un problème de synthèse consiste à élaborer un plan de synthèse en employant un mode rétrosynthétique. La préparation de la molécule cible reste théorique. Le plan de synthèse de la molécule cible est défini en fonction des groupes fonctionnels qu'elle renferme. Une étape du plan correspond à la classification des fragments moléculaires produits à l'étape précédente, dans une hiérarchie particulière, appelée *hiérarchie des structures réactives*. Un plan de synthèse apparaît donc comme une succession de classifications qui se termine lorsque les produits obtenus sont des précurseurs connus et répertoriés.

Raisonnement rétrosynthétique dirigé par les structures

Dans l'approche procédurale standard, la représentation d'une transformation comprend une série d'enquêtes servant à déterminer l'applicabilité de la transformation [Laurenço, 1985]. Ce type d'approche oblige le chimiste à décrire sa connaissance sous la forme de conditionnelles qui ne lui permettent pas toujours d'exprimer pleinement son savoir, ni de représenter un site réactionnel ou une transformation selon plusieurs points de vue. Notre approche est nouvelle et repose sur un principe totalement différent. L'applicabilité d'une transformation à une molécule cible est déterminée après classification de la molécule dans la hiérarchie des structures réactives. Lors de cette classification, les groupes fonctionnels présents dans une molécule cible sont retrouvés de façon ordonnée, du plus simple au plus complexe[12] : les groupes primaires sont recherchés en premier lieu ; les résultats obtenus lors de ce premier filtrage, qui est très rapide vu la simplicité des structures recherchées, orientent la recherche des groupes secondaires présents dans la cible. La classification de la cible est terminée lorsque les groupes secondaires les plus complexes ont été déterminés. Les groupes présents orientent la fragmentation de la cible en précurseurs ayant une structure moins complexe que celle de la cible.

Dans cette approche, la tâche du chimiste consiste à préciser les différents sites réactionnels potentiels et non potentiels pour une transformation, sachant qu'un site réactionnel peut intervenir dans plusieurs transformations. Par ailleurs, certaines des études réalisées pour mettre en forme des transformations procédurales peuvent être facilement adaptées à ce type de description.

La hiérarchie des structures réactives

La subsomption de structures moléculaires dérive de la relation de co-subsomption vue au paragraphe 3.4. Elle ordonne les groupes fonctionnels répertoriés par niveaux de généralité. Le graphe de cette relation est baptisé *hiérarchie des structures réactives* et il est construit grâce à l'algorithme de classification présenté au paragraphe 3.3.1. Les groupes fonctionnels primaires y dominent les groupes fonctionnels secondaires : par exemple, le groupe primaire `C/O` est un ascendant des groupes secondaires `C/O/H` et `C/O/O/C`, car `C/O` (co-)subsume `C/O/H` et `C/O/O/C` (cf. Fig. 5.23).

La relation de subsomption sur les structures moléculaires s'appuie sur la relation de composition et n'a donc pas la même sémantique que la relation d'héritage : une structure moléculaire *M2* qui a pour composant une structure *M1* peut partager certaines propriétés avec *M1*, mais le partage de propriétés n'est pas systématique comme dans l'héritage. Par conséquent, la subsomption de structures moléculaires et son graphe, la hiérarchie des structures réactives, déterminent une nouvelle dimension de la base de connaissances "orthogonale" à la dimension de l'héritage.

Lorsqu'un groupe fonctionnel est une *structure réactive*, il possède trois attributs nommés `schema+`, `schema-` et `schema` (cf. Fig. 5.24). Pour une structure réactive don-

12. Un certain rapprochement peut être fait avec les techniques de filtrage structuré [Bylander *et al.*, 1989].

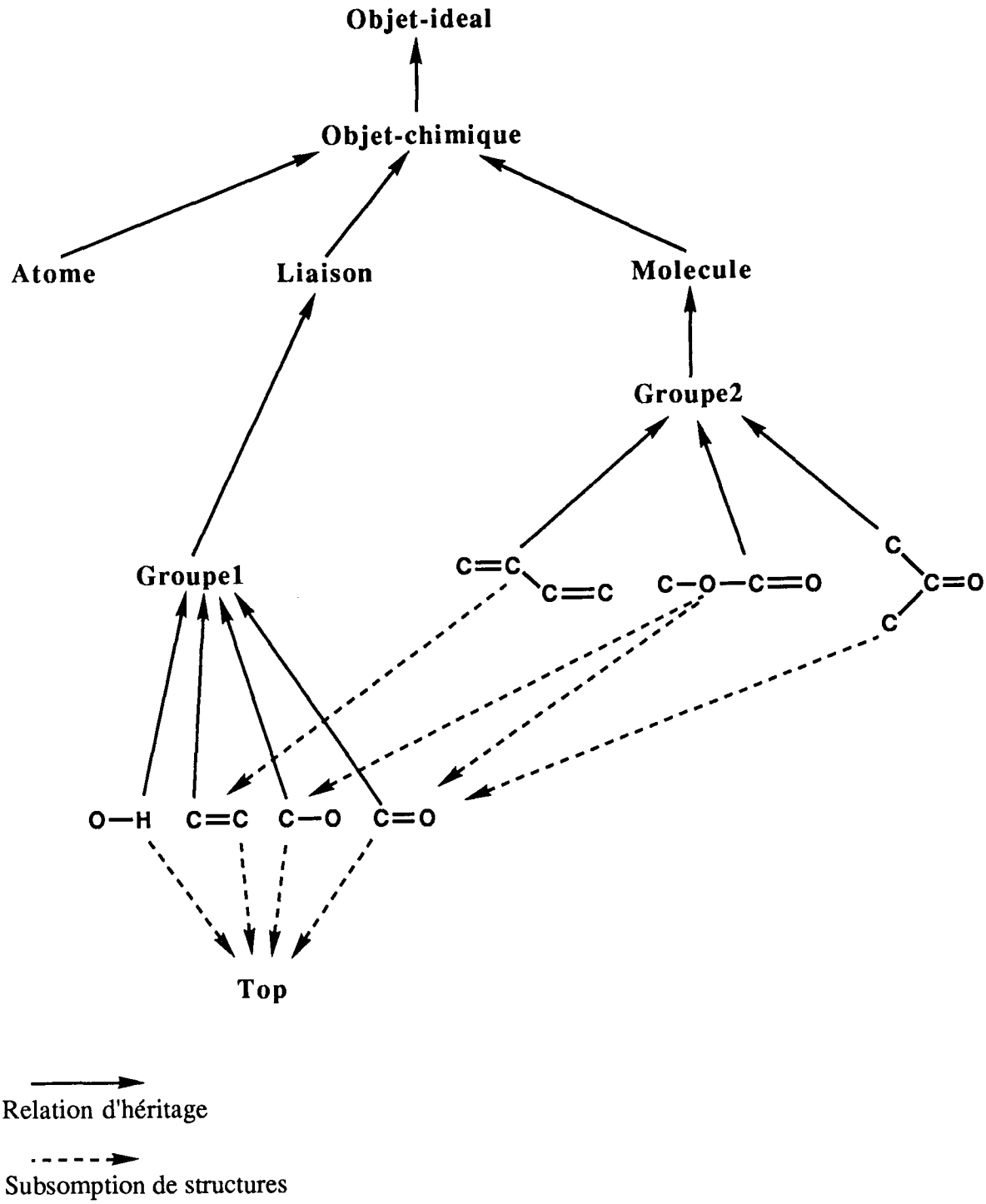


Figure 5.23. L'univers des objets chimiques est de dimension 2 : la hiérarchie des structures réactives est "orthogonale" au graphe d'héritage.

```
(defmodele Objet-chimique
  (sorte-de
    ($valeur Objet-ideal))
  (subsumants
    ($liste-de Groupe)
    ($defaut nil))
  (subsumes
    ($liste-de Groupe)
    ($defaut nil))
  (schema+
    ($liste-de Transformation)
    ($defaut nil)
    ($si-possible (schema+?)))
  (schema-
    ($liste-de Transformation)
    ($defaut nil)
    ($si-possible (schema-?)))
  (schema
    ($liste-de Transformation)
    ($defaut nil)
    ($si-besoin +schema)))
```

Figure 5.24. La partie de l'objet générique `Objet-chimique` dédiée à la gestion des structures réactives. La valeur de l'attribut `schema` est calculée par un réflexe si-besoin.

née SR, l'attribut `schema+` a pour valeur la liste des transformations qui construisent SR, mais qui ne construisent pas les subsumants de SR dans la hiérarchie des structures réactives. De façon duale, l'attribut `schema-` a pour valeur la liste des transformations qui construisent les subsumants de SR, mais qui ne construisent pas SR.

Par exemple, l'attribut `schema+` attaché au groupe fonctionnel `C//C` a pour valeur `wittig`, ce qui exprime que la transformation de Wittig permet de construire le groupe `C//C` (cf. Fig. 5.25). L'attribut `schema-` attaché à la structure `O//P/C//C` vaut aussi `wittig`, ce qui signifie cette fois que le groupe `C//C` ne peut plus être construit par la transformation de Wittig si l'environnement de `C//C` est de type `O//P/C//C`. L'attribut `schema-` joue un rôle équivalent au masquage, car il remet en cause le fait qu'une transformation puisse construire une fonctionnalité dans un environnement donné.

Les contraintes associées aux attributs `schema+` et `schema-` sont implantées par des réflexes si-possible et s'énoncent comme suit : étant donné une structure réactive SR, une transformation T appartient à la valeur de `schema+` si et seulement si elle n'est pas "héritable" pour SR ; T appartient à la valeur de `schema-` si et seulement si elle est "héritable" pour SR, où "héritable" a un sens qui est précisé dans le paragraphe

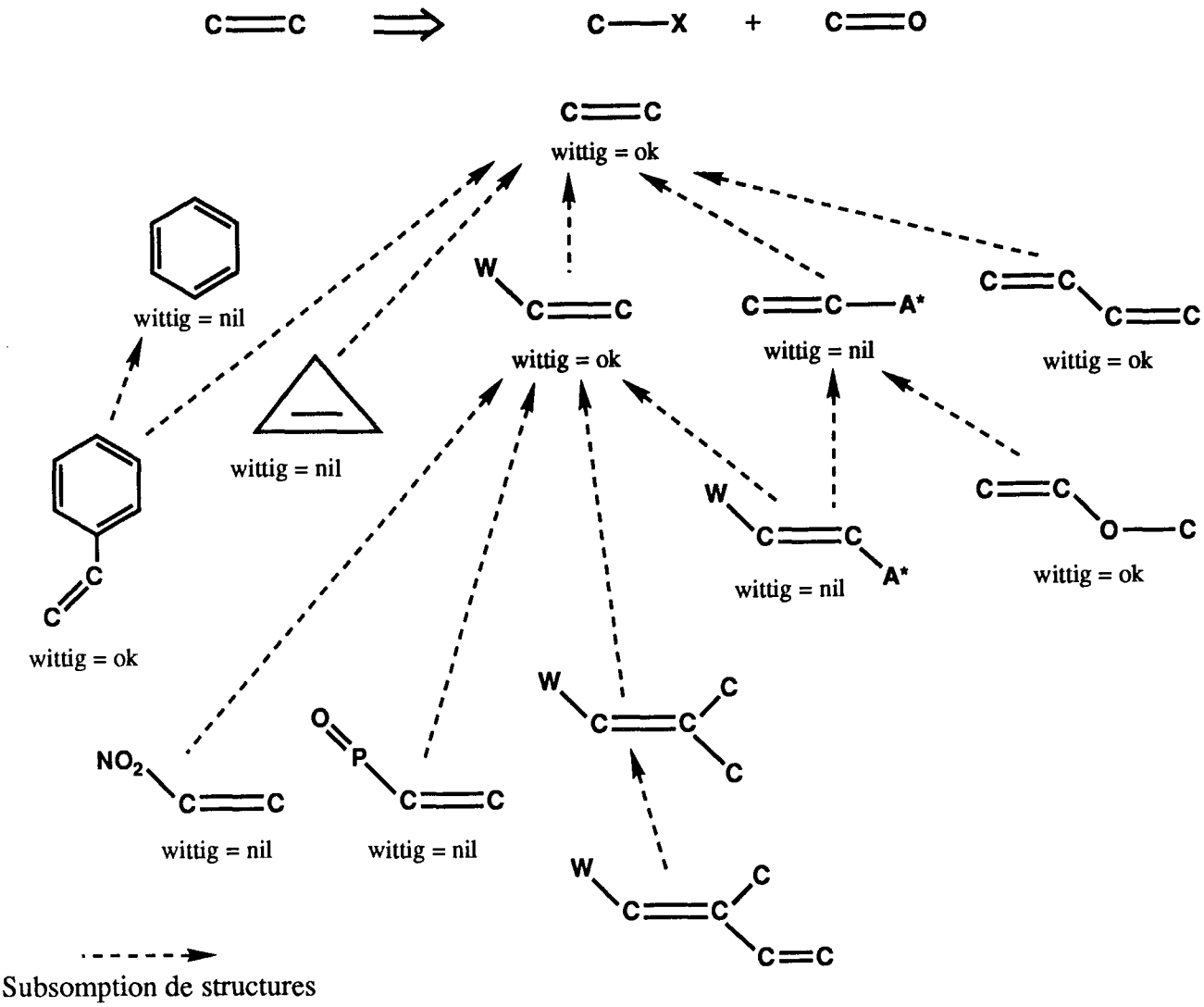


Figure 5.25. Les sites réactionnels valides et non valides associés à la transformation de Wittig.

suivant.

Le partage de propriétés dans la hiérarchie des structures réactives

Etant donné une structure réactive SR, une transformation T est dite *positivement héritable* dans la hiérarchie des structures réactives si elle appartient à la valeur de l'attribut `schema+` d'un subsumant de SR dans la hiérarchie.

De façon duale, une transformation T est dite *négativement héritable* dans la hiérarchie des structures réactives si elle appartient à la valeur de l'attribut `schema-` d'un subsumant de SR dans la hiérarchie.

La combinaison de l'héritage positif et négatif, autrement dit le partage des valeurs

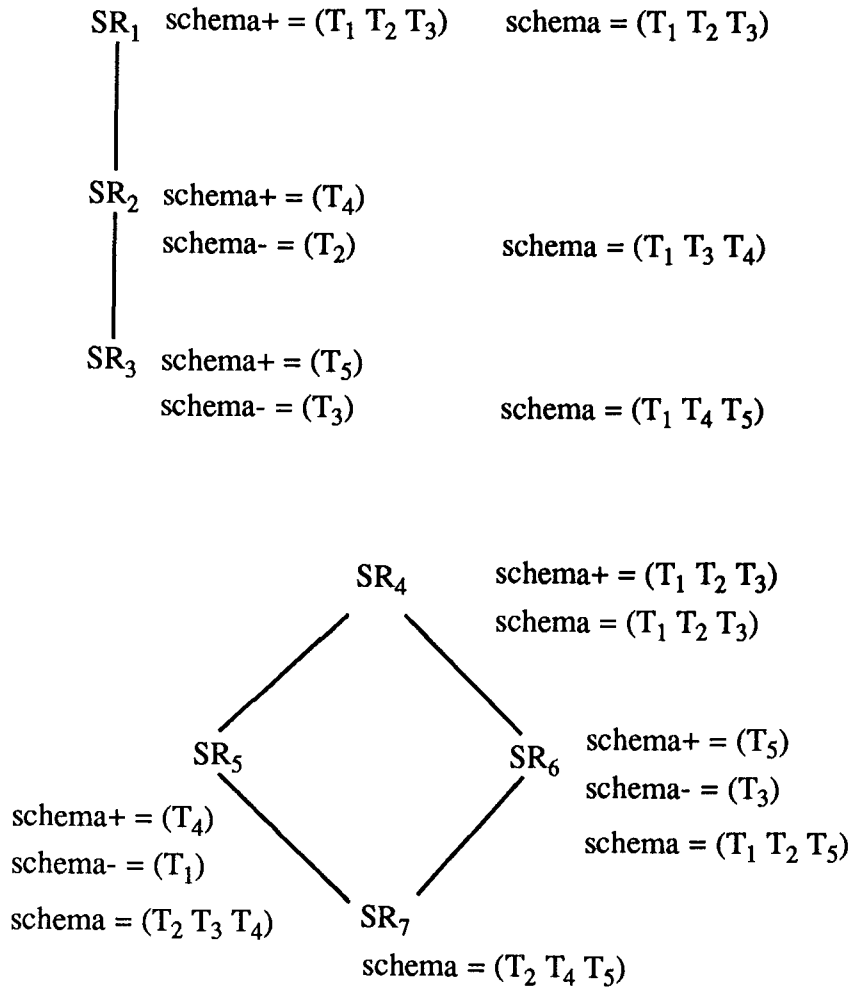


Figure 5.26. Exemples de calcul de la valeur de l'attribut `schema`.

des attributs `schema+` et `schema-`, se fait par l'intermédiaire de l'attribut `schema`, dont la valeur est calculée par la formule[13] :

```
schema(SR) = schema+(SR) ∪ [schema(s) - schema-(SR)], s ∈ SUPERS
           = ∪ schema+(s) - ∪ schema-(s), s ∈ SR ∪ SUPERS
```

Le terme `schema(SR)` dénote la valeur de l'attribut `schema` pour la structure `SR` ; `SUPERS` dénote l'ensemble des subsumants de `SR` dans la hiérarchie des structures réactives ; le symbole ∪ dénote la réunion ensembliste (cf. Fig. 5.26). Une valeur est dite *héritable* pour une structure réactive `SR` si elle appartient par calcul à la valeur de l'attribut `schema` associé à `SR`. Dans le présent contexte, la valeur de cet attribut s'interprète comme la liste des transformations qui construisent effectivement la structure réactive `SR`.

13. Cette formule est un cas particulier de calcul de cumul de valeurs de propriétés multivaluées présenté dans [Ducournau, 1992].

La formule du calcul de la valeur de l'attribut `schema` tient compte des éventuels conflits, analogues aux conflits d'héritage, qui pourraient survenir. En particulier, une transformation `T` n'est héritable pour une structure `SR` que si `T` n'est pas simultanément positivement et négativement héritable pour `SR`. Il existe une certaine analogie avec le traitement des contradictions pures dans les problèmes d'exception à l'héritage (cf. Fig. 2.6) : la priorité est donnée à l'héritage si une propriété P est héritée lorsqu'il existe un chemin d'héritabilité positif (un chemin sans arc d'exception) qui permet d'en hériter, sinon la priorité est donnée à l'exception et la propriété P n'est pas héritée [Ducournau and Habib, 1991]. Dans le cas présent, la priorité est donnée à l'exception, plus précisément à l'héritage négatif : une transformation n'est héritable que si elle n'est pas négativement héritable, ce qui est chimiquement cohérent.

Ainsi, considérons `SR1` et `SR2` les deux subsumants d'une structure `SR` dans la hiérarchie des structures réactives, `T` une transformation qui est dans `schema+` pour `SR1` et dans `schema-` pour `SR2`. Dans ce cas, le fragment `SR1` $\cap$ `SR2` situé dans `SR` est dans un contexte défavorable et `T` ne peut pas lui être appliqué.

Le développement de l'arbre de synthèse d'une molécule cible donnée commence par la recherche des différents groupes fonctionnels simples que la cible renferme. Chaque groupe devient un site réactionnel potentiel. La cible est ensuite classifiée dans la hiérarchie des structures réactives à partir de chacun des groupes simples retenus. A l'issue de la classification, les différentes transformations construisant des fragments de la cible sont calculées et appliquées le cas échéant.

Lorsqu'une cible renferme plusieurs occurrences d'un même groupe simple, sont alors effectuées autant de classifications dans la hiérarchie des structures réactives qu'il existe de groupes simples. Par exemple, la réunion `C//C`-cyclo-propène contient deux groupes de type `C//C`. L'un, celui qui appartient au cyclo-propène, est dans un contexte défavorable vis-à-vis de la transformation de Wittig. En revanche, l'autre, qui est extérieur au cyclo-propène, est dans un contexte favorable et la transformation peut lui être appliquée.

Le traitement d'un exemple simple

La molécule cible `M` donnée à la figure 5.27 a pour subsumants les structures réactives suivantes :

- L'ester `O//C/O/C` subsume `O1//C2/O3/C4`.
- L'éther d'énol `C//C/O/C` subsume `C5//C6/O7/C8`.
- La structure réactive `W/C//C/A*` subsume `C2/C5//C6/O7`, où `W` subsume le W-groupe ester `C4/O3/C2//O1`, qui est lié au carbone `C5`, et l'hétéroatome générique `A*` subsume `O7`, qui est lié au carbone `C6`.
- Le diène `C//C/C//C` subsume `C5//C6/C9//C10`.

Dans le cas général, la transformation de Wittig s'applique à la structure `C//C`, à l'éther d'énol `C//C/O/C`, au diène `C//C/C//C`, mais elle ne s'applique pas à la sous-structure `W/C//C/A*` (cf. Fig. 5.25). Or, il existe deux groupes de type `C//C` dans

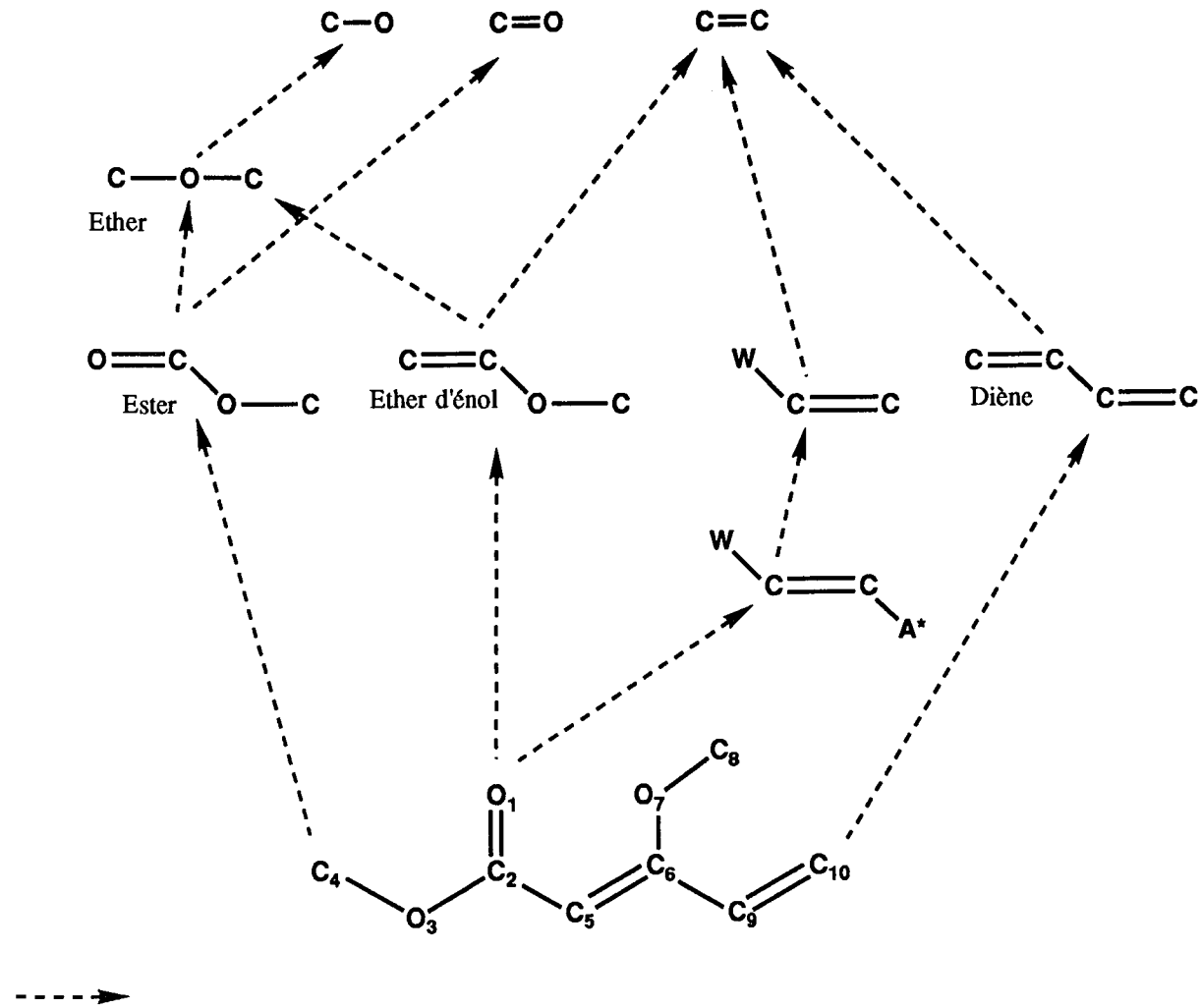


Figure 5.27. La molécule cible M et ses subsumants dans la hiérarchie des structures réactives.

la cible M, quatre en tenant compte des symétries carbone-carbone à l'intérieur d'un groupe C//C (C5//C6 et C6//C5, C9//C10 et C10//C9). Lorsque la cible est classifiée dans la hiérarchie des structures réactives à partir du groupe primaire C5//C6, les subsumants les plus spécifiques sont C//C/O/C, C//C/C//C et W/C//C/A*. Le conflit qui existe sur l'héritabilité de la transformation de Wittig est résolu au profit de l'exception car C5//C6 est dans le contexte défavorable W/C//C/A*. Par suite, la transformation de Wittig n'est pas applicable à C5//C6. En revanche, en classifiant la cible M à partir du groupe primaire C9//C10, l'unique subsumant est le diène C5//C6/C9//C10 qui est un contexte favorable pour l'application de la transformation de Wittig. La transformation est alors appliquée, ce qui donne naissance aux précurseurs P1 et P2 (cf. Fig. 5.28).

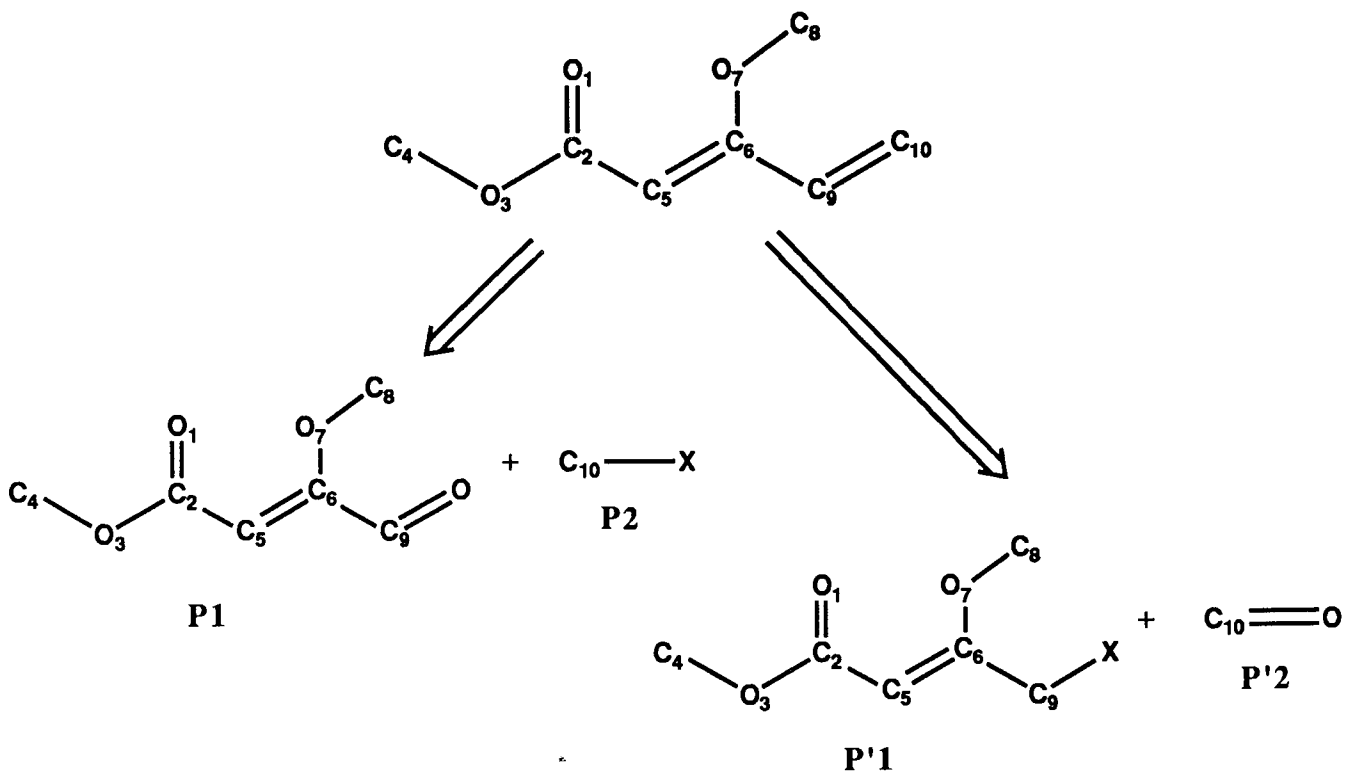


Figure 5.28. Le début du développement d'un arbre de synthèse. Les produits P1 et P2 sont obtenus parallèlement aux produits P1' et P2'.

Le double rôle de la classification

Cette nouvelle approche du mode rétrosynthétique s'appuie sur un raisonnement par classification qui a pour cadre la hiérarchie des structures réactives. Savoir construire une structure moléculaire M quelconque consiste à insérer M dans cette hiérarchie, les subsumants de M détenant les transformations construisant M. La classification de la molécule cible M est réalisée en s'appuyant sur l'algorithme de classification vu au paragraphe 3.3.1, où la relation de comparaison choisie est la subsomption de structures moléculaires. Ce mode de résolution de problèmes de synthèse fournit un bon exemple du double rôle du processus de classification, construction de la hiérarchie des structures réactives et classification d'une cible dans cette même hiérarchie. De plus, contrairement à la classification heuristique (cf. § 1.2.4), le raisonnement par classification est constructif, puisqu'il y a construction de la hiérarchie des structures réactives – cette hiérarchie correspond à l'univers de résolution du problème – et construction d'une solution particulière en classifiant la cible dans l'univers de résolution.

5.4.2 Le développement d'un arbre de synthèse

Le développement de l'arbre de synthèse d'une molécule cible M donnée apparaît comme une "instanciation" du cycle de base du raisonnement par classification (cf. § 1.2.3), qui comporte les étapes suivantes :

- Description de la molécule cible M : le choix de la molécule cible peut s'appuyer sur certains critères politiques ou encore sur des résultats fournis par le système (traitement d'un précurseur par exemple).
- Recherche des différentes structures réactives primaires dans M, puis classification de M dans la hiérarchie des structures réactives, selon les points de vue donnés par les structures primaires.
- Calcul des transformations applicables pour les points de vue développés à l'étape précédente (calcul de la valeur de l'attribut `schema` pour les différentes structures réactives retenues), application des transformations valides et enfin génération des précurseurs.

Lors de la dernière étape, un niveau supplémentaire de l'arbre de synthèse est construit. Il correspond à la génération des précurseurs résultant de l'application des transformations valides aux réactifs du niveau précédent. Le développement de l'arbre de synthèse se poursuit tant que les précurseurs engendrés à une étape ne font pas partie de la base des produits de départ répertoriés ; ces précurseurs sont alors traités comme de nouvelles cibles et le cycle de classification leur est appliqué. L'arrêt du processus intervient lorsque les feuilles de l'arbre de synthèse correspondent à des précurseurs connus et directement accessibles.

Structures moléculaires temporelles

Lors de la conception d'un plan de synthèse, tous les objets chimiques manipulés, atomes, liaisons et molécules, sont des objets temporels (cf. § 2.3.4). Les groupes fonctionnels font exception : ils conservent une représentation atemporelle car leurs propriétés chimiques sont considérées comme immuables dans le temps. La représentation des objets temporels est très souple et elle autorise un partage maximal des objets présents à un instant donné. En particulier, l'historique des opérations de chirurgie moléculaire effectuées sur les différents réacteurs (adjonction ou suppression de liaison), ainsi que la trace de l'évolution de chaque objet sont conservés, pour éventuellement servir à la description globale du plan de synthèse. Par suite, si un assemblage moléculaire perd son caractère connexe après une opération de chirurgie, il se crée, pour un réacteur donné, autant de molécules qu'il existe de composantes connexes dans le réacteur.

Voici un extrait de la représentation temporelle de la cible `M` au temps `0`, les atomes et les liaisons ont été numérotés comme ceux de la figure 5.27 :

```
FRAME M
```

```
 est-un   $value = (Molecule-t)
 atomes   $value = ((0 o3 o7 o1 c6 c5 c8 c4 c9 c2 c10))
 liaisons $value = ((0 b11 b16 b13 b17 b12 b15 b19 b18 b14))
```

Après application de la transformation de Wittig à la double liaison C9//C10, la représentation temporelle au temps 1 des atomes de carbones c9 et c10, qui sont les extrémités de la liaison b19, devient :

```
FRAME c9
 est-un    $value = (Atome-t C)
 incidentes $value = ((1 b21 b18) (0 b19 b18))
 molecule  $value = ((1 . P1) (0 . M))

FRAME c10
 est-un    $value = (Atome-t C)
 incidentes $value = ((1 b23) (0 b19))
 molecule  $value = ((1 . P2) (0 . M))
```

les produits obtenus sont les molécules P1 et P2 :

```
FRAME P1
 est-un   $value = (Molecule-t)
 atomes   $value = ((1 o7 o1 o20 o3 c2 c8 c5 c6 c4 c9))
 liaisons $value = ((1 b11 b12 b17 b16 b13 b21 b15 b18 b14))

FRAME P2
 est-un   $value = (Molecule-t)
 atomes   $value = ((1 x22 c19))
 liaisons $value = ((1 b23))
```

Les carbones c9 et c10 sont liés au temps 0 par la liaison b19. Au temps 1, c9 est lié à l'oxygène o20 par la liaison b21 et c10 est lié à l'halogène x22 par la liaison b23 :

```
FRAME b19
 est-un     $value = (Liaison-t)
 ordre      $value = ((1 . 0) (0 . 2))
 extremites $value = ((1) (0 c9 c10))
 molecule   $value = ((1) (0 . M))

FRAME b21
 est-un     $value = (Liaison-t)
 ordre      $value = ((1 . 2))
 extremites $value = ((1 o20 c9))
 molecule   $value = ((1 . P1))

FRAME b23
```

```
est-un     $value = (Liaison-t)
ordre      $value = ((1 . 1))
extremites $value = ((1 x22 c10))
molecule   $value = ((1 . P2))
```

En tenant compte de la symétrie du groupe C9//C10, le résultat n'est pas un arbre de synthèse, mais plutôt un feuilletage de perspectives temporelles, dont une des perspectives est le couple (P1 P2), et l'autre, le couple (P1' P2').

```
FRAME P1'
 est-un   $value = (Molecule-t)
 atomes   $value = ((1 x24 o7 o1 o3 c2 c8 c5 c6 c4 c9))
 liaisons $value = ((1 b25 b11 b12 b17 b16 b13 b15 b18 b14))

FRAME P2'
 est-un   $value = (Molecule-t)
 atomes   $value = ((1 o26 c19))
 liaisons $value = ((1 b27))
```

Les perspectives temporelles apparaissent alors dans la représentation des carbones c9 et c10 :

```
FRAME c9
 est-un    $value = (Atome-t C)
 incidentes $value = ((1 (b21 b18) (b25 b18)) (0 b19 b18))
 molecule  $value = ((1 . (P1 P1')) (0 . M))

FRAME c10
 est-un    $value = (Atome-t C)
 incidentes $value = ((1 (b23) (b27)) (0 b19))
 molecule  $value = ((1 . (P2 P2')) (0 . M))
```

Le problème des stratégies

Une stratégie de synthèse permet de sélectionner les objectifs à atteindre pour élaborer au mieux un plan de synthèse. Le problème de la formalisation des stratégies de synthèse n'a pas encore été véritablement abordé. L'étude et la mise en œuvre de stratégies sont difficiles, car elles demandent beaucoup d'expérience et de savoir-faire sur les synthèses [Winter, 1984] [Winter, 1985]. Actuellement, seul le niveau tactique du système est bien formalisé : une transformation est applicable à une sous-structure si l'environnement dans lequel se trouve la sous-structure est favorable. Aucun niveau stratégique n'est encore implanté. La mise en forme des stratégies nécessite d'effectuer des classifications de transformations et de chemins de synthèse, ce qui constitue une des "suites logiques" de cette étude.

5.5 La représentation de réactions et de chemins de synthèse

5.5.1 Un réseau de molécules

Le gestionnaire de graphes imaginé par R.A. Levinson permet de construire des systèmes où les connaissances sont décrites sous forme de graphes étiquetés non orientés [Levinson, 1984] [Levinson, 1985]. L'organisation des objets se démarque de celle qui est habituellement employée dans les représentations à objets ou dans les systèmes à subsomption. Tous les objets sont décrits comme des sous-graphes d'un graphe unique, appelé *graphe universel*. Ils sont ordonnés de façon hiérarchique, des plus "petits" qui correspondent aux graphes les plus généraux, appelés aussi graphes primitifs, aux plus "grands", qui sont les plus complexes et les plus spécifiques.

Les capacités de ce gestionnaire de graphes ont été mises à profit pour construire un système original de synthèse assistée par ordinateur [Wilcox and Levinson, 1986]. La base de connaissances du système contient des graphes qui représentent à l'aide d'un formalisme unique aussi bien des molécules que des réactions. L'organisation des objets dans la base des graphes a été réalisée de façon à optimiser les temps de réponse à quatre questions fondamentales pour l'aide à la conception de plans de synthèse et, plus généralement, pour la recherche d'information :

- Appariement exact : le graphe G fait-il partie de la base de connaissances ?
- Recherche de supergraphes : quels sont les graphes de la base qui contiennent G ?
- Recherche de sous-graphes : quels sont les graphes de la base ayant G comme supergraphe ?
- Recherche d'appariements partiels : quels sont les graphes de la base ayant les plus grands sous-graphes communs avec G ?

Le graphe universel

Un graphe contenu dans la base de connaissances est représenté par un sous-ensemble de sommets du graphe universel. La relation d'ordre partiel *sous-graphe-de* permet d'ordonner les graphes : un graphe *G1* est un *prédécesseur* du graphe *G2* si et seulement si *G1* est un sous-graphe de *G2*, au sens de la théorie des graphes (voir page 184). Le graphe *G2* est alors un *successeur* de *G1*.

A chaque graphe présent dans la base de connaissances est associé un pointeur vers ses prédécesseurs et vers ses successeurs immédiats. Tous les graphes pointent donc vers le graphe universel, éventuellement par transitivité. Initialement, la base n'est constituée que de graphes primitifs représentant des faits connus et intangibles. Les graphes primitifs sont des structures très générales, véritables briques de base servant à construire des graphes plus complexes. Dans un contexte chimique, les atomes, les

groupes fonctionnels simples et la liaison carbone-carbone C/C sont des exemples de graphes primitifs.

Le graphe universel est construit de façon incrémentielle grâce à un mécanisme de classification qui permet d'insérer chaque nouveau graphe en respectant l'ordre partiel défini sur les éléments de la base de connaissances. A l'inverse d'une hiérarchie d'héritage, les objets situés dans les niveaux supérieurs de la hiérarchie sont les objets les plus spécifiques et les plus complets. L'objet le plus spécifique est le graphe universel, qui se trouve au "sommet" de cette hiérarchie et qui contient tous les autres graphes. Les graphes primitifs, quant à eux, sont les objets les plus généraux et ils sont situés au niveau le plus bas de la hiérarchie (cf. Fig. 5.29).

La relation d'ordre partiel *sous-graphe-de* découpe la base de telle façon que seule une petite partie des graphes est explorée lors d'une interrogation, ce qui minimise le nombre d'opérations de comparaison nécessaires pour répondre à une question. L'algorithme de recherche utilisé repose sur un parcours ascendant de la hiérarchie et se divise en deux parties. La première partie consiste à rechercher l'ensemble *IP(G)* des prédécesseurs immédiats d'un graphe G, qui sont comparables aux "subsumants les plus spécifiques" de G pour la relation *sous-graphe-de* (cf. Fig. 5.30). Dans la première phase d'appariement, seuls les graphes primitifs sont comparés à G car aucun d'eux ne possède de prédécesseurs, en particulier *IP(p)* est vide pour tout graphe primitif p. Cette opération est accomplie d'autant plus rapidement que les graphes primitifs sont simples. La comparaison se poursuit de façon ascendante avec les successeurs des graphes trouvés dans la première phase et ainsi de suite. Lorsque le processus s'arrête, $P = IP(G)$. Cet algorithme est d'un type similaire à celui de l'algorithme de classification vu au paragraphe 3.3.1, si ce n'est que le parcours de la hiérarchie des graphes se fait ici en largeur.

Détaillons le calcul de *IP(G)* pour le graphe $G = a$–b–d. La première série de comparaisons se termine avec les graphes primitifs a, b et d marqués avec vrai et placés dans P, alors que les graphes primitifs c, e et f sont marqués avec faux. Lors de la deuxième série de comparaisons, seuls les graphes a–b et b–d sont examinés. Les autres graphes de même niveau sont ignorés puisque leurs prédécesseurs ne sont pas tous marqués avec vrai. A l'issue de cette seconde étape, $P = \{a\text{–}b, b\text{–}d\}$. Lors de la troisième série de comparaisons, il apparaît que G n'est pas un prédécesseur des graphes *G1*, *G2* et *G3*, qui sont les successeurs des éléments de P. La recherche des prédécesseurs de G se termine avec $P = \{a\text{–}b, b\text{–}d\}$.

La seconde partie de l'algorithme consiste à déterminer l'ensemble noté *IS(G)* des successeurs immédiats de G. Ces derniers sont les graphes les plus généraux parmi les graphes qui sont plus spécifiques que G, ils sont comparables aux "subsumés les plus généraux" de G pour la relation *sous-graphe-de*. L'ensemble *IS(G)* contient des graphes qui sont des successeurs communs à tous les éléments de *IP(G)*. Plus précisément, chaque successeur S d'un élément P de *IP(G)* est marqué avec un symbole *M(P)* qui dépend du prédécesseur P. Ce marquage est fait systématiquement pour tous les éléments de *IP(G)* sauf un. Un traitement spécial est effectué pour ce "dernier" élément, noté Q dans la suite. Tous les successeurs de Q qui sont déjà marqués avec *tous* les symboles *M(Pi)* des éléments *Pi* de *IP(G)*, Q excepté, sont des successeurs

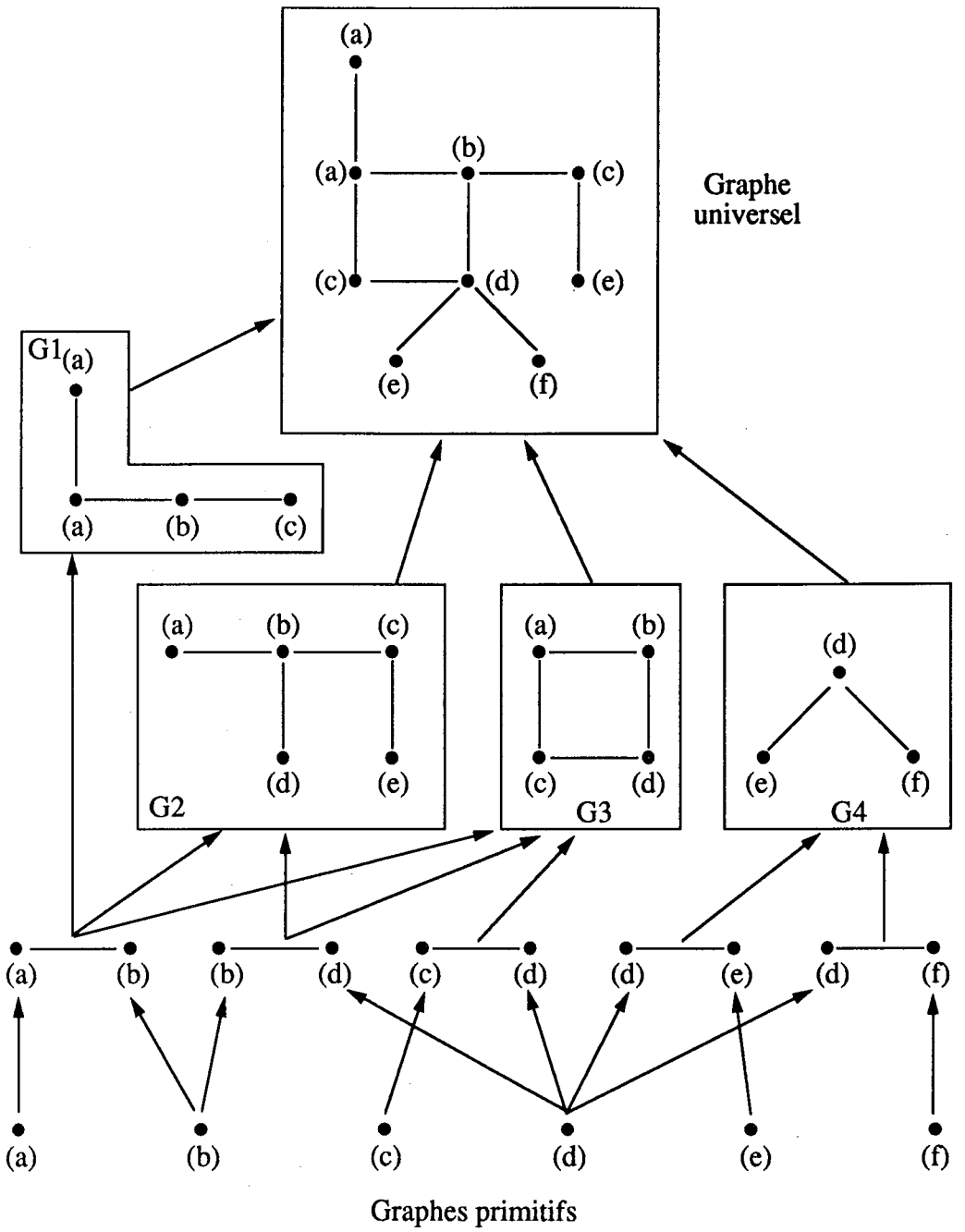


Figure 5.29. Le graphe universel est l'objet le plus spécifique et contient tous les graphes. Les graphes primitifs sont les objets les plus généraux et ils pointent vers des graphes plus spécifiques. Les étiquettes associées à chaque nœud correspondent au type du nœud et réfèrent à des graphes primitifs.

```
Tant qu'il existe un graphe g non marqué dans la base B tel que
chaque élément de IP(g) est marqué vrai ou IP(g) = nil
faire
   si g est un prédécesseur de G
      alors marquer g vrai
            P = [P - IP(g)] ∪ g
      sinon marquer g faux
```

Figure 5.30. Algorithme de recherche de l'ensemble *IP(G)* des prédécesseurs immédiats d'un graphe *G*.

potentiels de *G*. Un test vérifiant que *G* est bien un prédécesseur de ces successeurs potentiels est effectué. Si le test est positif, le successeur potentiel devient un véritable successeur : il est alors mémorisé dans *IS(G)* et ses propres successeurs sont marqués avec faux afin qu'ils ne soient plus pris en compte ultérieurement, puisque seuls les successeurs les plus spécifiques doivent être obtenus. Une fois que tous les éléments de *IP(G)* ont été examinés, le processus est reconduit en prenant cette fois comme éléments de départ les successeurs des éléments de *IP(G)* qui ne sont pas déjà dans *IS(G)* et qui ne sont pas marqués avec faux. Le processus se termine lorsque l'ensemble des successeurs à considérer est vide.

Détaillons le calcul de *IS(G)* pour le graphe $G = a\text{–}b\text{–}d$ (cf. Fig. 5.29). L'ensemble *IP(G)* a pour valeur $\{a\text{–}b, b\text{–}d\}$. Les successeurs immédiats de $a\text{–}b$ sont les graphes *G1*, *G2* et *G3*, qui sont marqués avec le symbole *M(a–b)*. Les successeurs immédiats de $b\text{–}d$ sont les graphes *G2* et *G3*, qui sont alors marqués avec le symbole *M(b–d)*. Comme ils sont déjà marqués avec *M(a–b)*, le système vérifie que *G* est un prédécesseur de *G2* et *G3*. Puisque c'est effectivement le cas, les successeurs de *G2* et *G3*, ici uniquement le graphe universel, sont marqués avec faux. Tous les successeurs de *G1*, *G2* et *G3* étant marqués, le processus se termine avec $IS(G) = \{G2, G3\}$.

Les réponses aux quatre questions mentionnées au début du paragraphe sont obtenues de la façon suivante :

- Appariement exact : si $IP(G) = IS(G)$, alors *G* est l'unique élément de ces ensembles et *G* fait partie de la base de connaissances.
- Les sous-graphes de *G* correspondent aux graphes marqués avec vrai lors du calcul de *IP(G)*.
- Les supergraphes sont obtenus en faisant l'union de tous les successeurs de chaque élément de *IS(G)*.
- Les appariements partiels sont obtenus en considérant les successeurs des éléments de *IP(G)* qui ne sont pas des supergraphes de *G*.

Le graphe universel est construit de façon à être le plus petit possible. La plupart de ses nœuds sont partagés par de nombreux graphes. Ce partage de caractéristiques et la représentation ensembliste des graphes permettent de ramener certaines opérations de manipulation des graphes à des opérations ensemblistes élémentaires. Par exemple, si Gs est un supergraphe G, il est possible de réduire G à l'ensemble de ses sommets dans Gs et ainsi de déterminer une occurrence de G dans le graphe universel. Les supergraphes de G sont tous les graphes dont l'ensemble de sommets contient l'ensemble des sommets de G, aucune comparaison sommet par sommet n'étant alors nécessaire. De même, les graphes ayant un sous-graphe commun avec G sont ceux dont l'ensemble de sommets a une intersection non vide avec l'ensemble des sommets de G.

La représentation de réactions

En suivant une approche classique, une réaction est représentée par deux graphes, l'un représentant les produits de départ et l'autre les produits obtenus [Laurenço, 1985]. Il faut donc au moins *deux* structures pour décrire l'évolution temporelle d'un ensemble de produits. Dans le système de synthèse assistée par ordinateur mis au point par C.S. Wilcox et R.A. Levinson, une réaction est représentée par un graphe *unique*, contenant des liaisons temporellement *évolutives*, qui décrivent les liaisons qui ont été modifiées au cours de la réaction [Wilcox and Levinson, 1986]. Les molécules et les réactions sont alors représentées dans un formalisme unique. La figure 5.31montre les différentes façons dont est appréhendée la réaction de Diels-Alder, avec en particulier un graphe à liaisons évolutives et un graphe condensé de la réaction.

Chaque fois qu'un nouveau graphe décrivant une molécule ou une réaction est introduit dans la base de connaissances, ses prédécesseurs et ses successeurs immédiats sont recherchés. Des informations additionnelles sont aussi associées à une réaction : un schéma réactionnel général et un schéma spécifique, ainsi que le sous-graphe commun maximal qu'elle partage avec des réactions qui lui sont "proches".

Ce système peut être utilisé comme un gestionnaire de bases de données de molécules et de réactions, mais aussi comme un système de synthèse assistée par ordinateur qui fonctionne selon le mode rétrosynthétique. Chercher la solution d'un problème de synthèse revient à retrouver les précurseurs d'une molécule cible. Pour cela, le système classifie la cible dans le graphe universel et recherche les réactions, qui sont représentées sous la même forme que la cible, qui partagent le plus de liaisons évolutives avec la cible. Les réactions retenues sont alors appliquées dans le sens rétrosynthétique et des précurseurs sont engendrés. Le cycle peut alors se poursuivre selon les besoins de l'utilisateur.

Les possibilités de recherche de plus grands sous-graphes communs sont, quant à elles, employées pour découvrir des réactions génériques, en généralisant les descriptions de plusieurs réactions spécifiques. La validité d'une généralisation est mesurée grâce à une estimation de la réactivité des sites associés à la réaction, accompagnée d'une comparaison avec des réactions déjà répertoriées. Lorsque la validité d'une réaction dépasse un certain seuil, la réaction est ajoutée à la base pour être utilisée ultérieurement. le cas échéant.

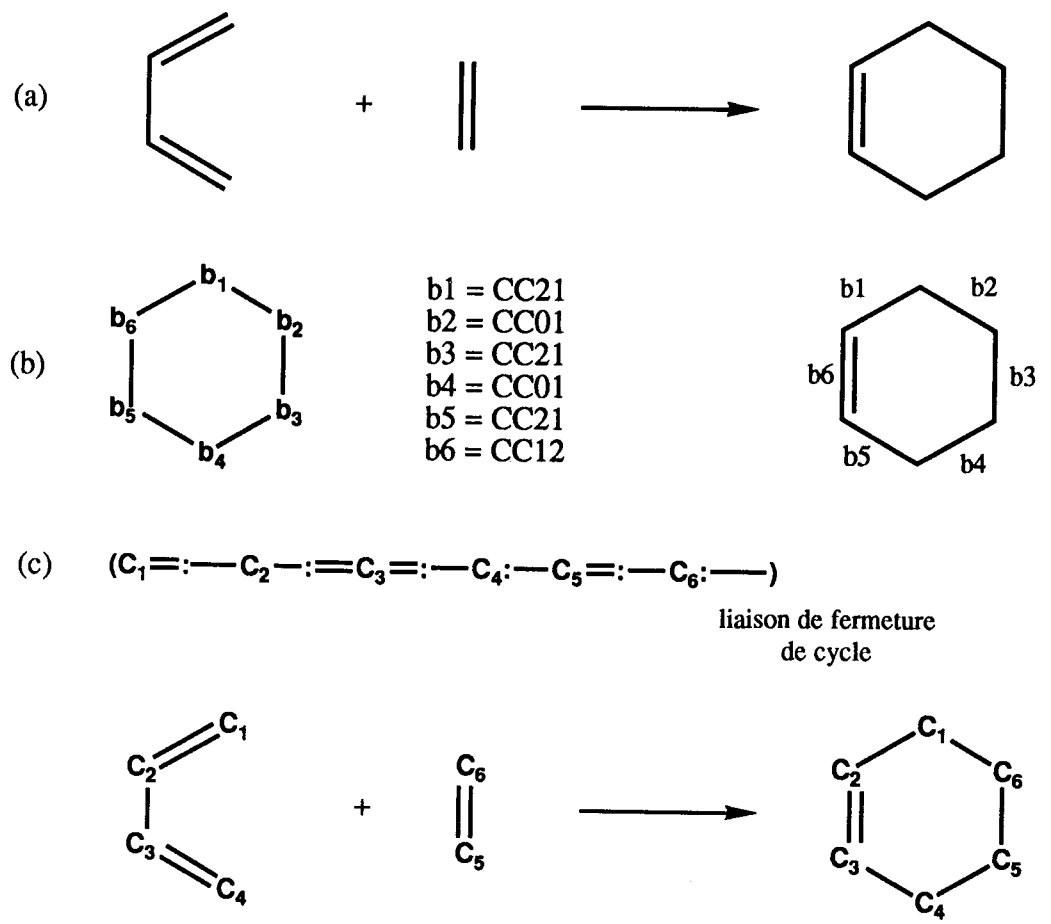


Figure 5.31. Différentes représentations de la réaction de Diels-Alder : (a) est la forme synthétique descriptive classique, (b) est un graphe condensé de la réaction, dont les sommets correspondent à des liaisons pour lesquelles sont donnés le type atomique des extrémités et l'ordre de la liaison avant et après la réaction, enfin (c) est une notation linéaire, surtout employée pour saisir les réactions, dans laquelle le passage d'une liaison carbone-carbone de l'ordre 1 à l'ordre 2 se note *C1-:=C2*.

5.5.2 Une représentation des synthèses en YAFOOL

Un modèle de représentation

Le paragraphe précédent a décrit une façon non classique de représenter avec un formalisme unique des molécules et les réactions qui agissent sur ces molécules. L'étude qui suit est en quelque sorte une généralisation de ce qui précède : les objets représentés sont non seulement des molécules et les réactions qui agissent sur ces molécules, mais en plus les chemins de synthèse où s'inscrivent des séquences de telles réactions. Cette étude a été effectuée en collaboration avec Michel Py au LIRMM de Montpellier, sous la direction de Claude Laurenço [Laurenço *et al.*, 1990]. Bien que l'implantation qui

```
(defmodele Synthese
  (est-un ($valeur Objet-Chimique))
  (nom-canonique ($liste-de Symbole))
  (cible ($un Molecule))
  (produits-depart ($liste-de Molecule))
  (chemin-de-synthese ($liste-de Symbole))
  (bibliographie ($liste-de Reference)))
```

Figure 5.32. L'objet `Synthese`.

est présentée dans la suite ait été réalisée en YAFOOL, la syntaxe de définition des objets est identique à celle qui a été utilisée jusqu'ici.

Le modèle sur lequel s'appuie la description des chemins de synthèse est une extension du *principe d'isomérisme* : deux molécules sont des isomères constitutionnels lorsqu'elles ont la même formule brute mais une distribution différente des liaisons entre leurs atomes. Lorsque ce principe est étendu aux ensembles de molécules, une réaction est alors considérée comme l'isomérisation d'un ensemble de réactifs en un ensemble de produits, un chemin de synthèse comme une suite d'isomérisations faisant évoluer dans le temps un ensemble de réactifs vers une cible et des sous-produits. Une synthèse est alors vue comme un objet structuré complexe, composé d'atomes et de liaisons. A un instant donné, la connexité de l'ensemble des liaisons détermine les molécules en présence, dont certaines sont des produits de la réaction précédente et des réactifs de la réaction à venir.

La figure 5.32 montre la description de l'objet `Synthese` : une synthèse est caractérisée par un nom canonique, une molécule cible, des produits de départ, un chemin de synthèse et des références bibliographiques. Les valeurs des attributs sont calculées à partir d'une formule linéaire. La figure 5.33 montre la formule linéaire décrivant la partie constitutionnelle de la synthèse convergente donnée à la figure 5.34.

Dans cette liste, le premier élément correspond au carbone pentalié noté `C1` et chaque symbole atomique correspond à un atome. Les liaisons sont désignées par les caractères `/`, `//`, `///` pour les liaisons simples, doubles et triples, `%` pour les liaisons aromatiques et `?` pour les liaisons d'ordre 0 ou neutres. Une liaison dynamique entre un carbone et un oxygène qui est d'ordre 1 au temps 1, puis qui évolue à l'ordre 2 au temps 2 et qui devient neutre au temps 3, est notée `(C (/1//2?3 O))`. Dans la formule, un entier suivant un symbole atomique indique une fermeture de cycle et correspond au rang de l'atome sur lequel se referme le cycle. Par exemple :
`(% C 1 (% C (% C (% C (% C (% C (% C 1))))))))` décrit un cycle aromatique, où le carbone `C` de rang 1 est l'atome de fermeture de cycle.

Cette notation linéaire est complexe, mais elle n'est pas manipulée directement. Elle sert de base à la création d'un réseau d'objets temporels qui représente la syn-

```
(C 1
  (//3? O) ; b1
  (/2? Cl) ; b2
  (?3// C ; b3
    (///2/ C 5 ; b4
      (?2// O ; b5
        (//1/2? C 7 ; b6
        (?1/ C 5) ; b7
          (/C (/C (/C (/C (/C ; b8-b9-b10-b11-b12
        (/2//3/ C 7) ; b13
          (?3/ C ; b14
            (/C 1) ; b15
            (//2/ C ; b16
              (?2/ C 18 ; b17
                (% C (% C (/ O (/ C)) ; b18-b19-b20-b21
                  (% C (% C (% C (% C 18) ; b22-b23-b24
  (?2/ C 1))))))))))))))))))) ; b25
```

Figure 5.33. Une formule linéaire décrivant la partie constitutionnelle d'une synthèse.

thèse particulière associée à la formule. Les objets temporels décrivent les composants élémentaires qui entrent en jeu dans une synthèse, à savoir les atomes, les liaisons et les molécules.

Les frames temporels

Le modèle d'objets temporels qui est présenté dans ce paragraphe est celui de YAFOOL. Un objet évoluant dans le temps est représenté par deux frames, l'un atemporel qui décrit la partie immuable de l'objet, l'autre temporel qui décrit l'état de l'objet à un instant donné [Ducournau, 1989b]. Pour un objet temporel donné, il existe un frame atemporel qui définit un ensemble de propriétés constantes ou non concernées par le temps et un frame temporel qui définit les propriétés variables et la façon dont elle varient. Un frame temporel est toujours associé au temps où il est créé. Contrairement au modèle d'objets temporels présenté au paragraphe 2.3.4, ce ne sont pas les propriétés du frame qui sont temporelles, mais le frame en entier.

La figure 5.35 montre la définition des deux frames temporels `Atome-t` et `Liaison-t`. Un frame temporel est caractérisé par la présence de l'attribut prédéfini `atemporel` qui a pour valeur le frame atemporel qui lui est associé, par exemple `Atome` pour `Atome-t`. Réciproquement, un frame atemporel possède un attribut prédéfini `temporel` qui indique le frame temporel associé.

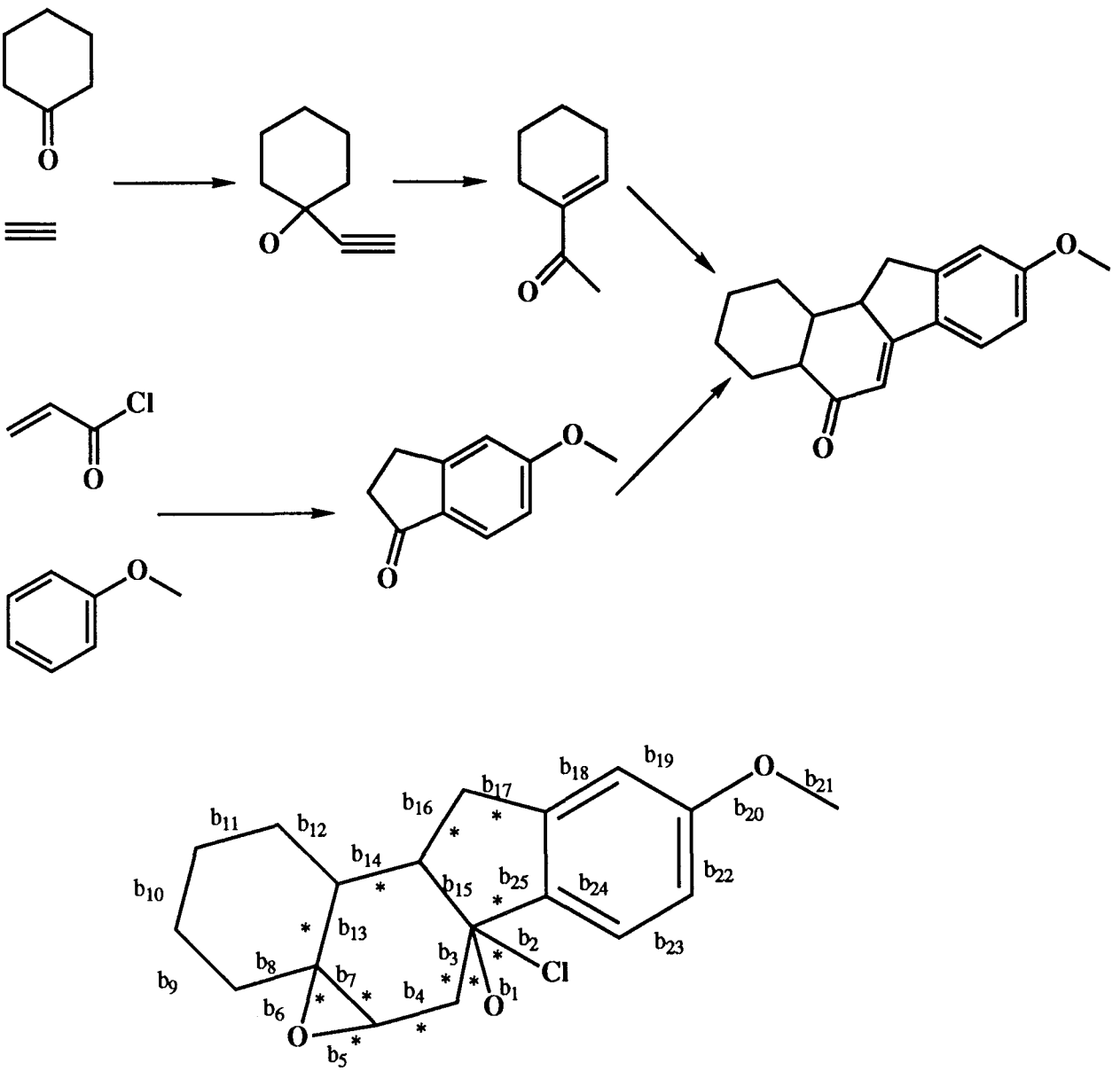


Figure 5.34. Une synthèse convergente et sa description condensée, d'après [Laurenço *et al.*, 1990]. Dans la description condensée, les sommets du graphe sans étiquette correspondent aux atomes de carbone ; les hétéroatomes sont explicitement nommés et les atomes d'hydrogène sont implicites. Les liaisons statiques sont notées de façon classique, alors que les liaisons *dynamiques*, qui décrivent les liaisons modifiées au cours de la synthèse, sont marquées d'une étoile.

```
(defmodele Atome-t
  (est-un ($valeur Atome))
  (atemporel ($valeur Atome))
  (charge ($si-besoin get-previous-value))
  (valence ($si-besoin get-previous-value))
  (configuration   ($domaine (R S P M neutre))
                   ($si-besoin get-previous-value))
  (degre ($methode +degre-t))
  (voisins ($methode +voisins)))

(defmodele Liaison-t
  (est-un ($valeur Liaison))
  (atemporel ($valeur Liaison))
  (configuration  ($domaine [E Z P M neutre])
                  ($si-besoin get-previous-value))
  (ordre ($si-besoin get-previous-value)))

(defun get-previous-value ()
  (:: slot* (instances (atemporel frame*) (- (time*) 1))))
```

Figure 5.35. La définition des frames temporels `Atome-t` et `Liaison-t`. La fonction `get-previous-value` retourne la valeur de l'attribut au dernier temps précédent. La fonction `time*` joue le même rôle que la variable globale `temps*` qui a pour valeur le temps courant (cf. § 2.3.4).

Les frames `Atome-t` et `Liaison-t` sont respectivement des spécialisations des frames `Atome` et `Liaison`, dont ils héritent les caractéristiques. La description des frames `Atome` et `Liaison` est similaire à celle des objets de même nom qui a été donnée précédemment (cf. Figures 5.9 et 5.14). Un frame temporel hérite les propriétés du frame atemporel qui lui est associé sans que la réciproque soit vraie ; un représentant d'un frame temporel hérite donc d'abord des caractéristiques du frame temporel dont il dérive, puis de celle du frame atemporel associé. Les deux frames généraux `Objet-atemporel` et `Objet-temporel` décrivent respectivement le comportement des frames atemporels et temporels. Ils figurent automatiquement dans la hiérarchie d'héritage des frames concernés, en fonction de la possession d'un attribut `temporel` pour les frames non temporels et `atemporel` pour les frames temporels. Le comportement des frames temporels est principalement défini par des méthodes de création et de suppression de représentants. Par exemple, la méthode `creation-t` associée à `Objet-atemporel` prend en argument un temps et s'applique à un frame atemporel pour créer un représentant temporel au temps donné en argument. Le nom d'un représentant temporel est formé du nom du frame atemporel dont il est issu, suf-

fixé par l'unité de temps. L'origine des temps est donnée par `t = -1`. Pour chaque objet temporel `objet`, un objet `objet.-1` est créé automatiquement par le système. L'évolution d'un objet dans le temps est décrit par un réseau de représentants ou instances temporels, chaque instance représentant l'objet au temps correspondant. Ainsi, l'évolution d'un objet `objet` entre les temps `0` et `2` est modélisée par le réseau d'instances (`objet.-1`, `objet.0`, `objet.1`, `objet.2`). Ces instances dérivent du même objet temporel, mais elles ne sont pas liées entre elles, car une instance temporelle ne peut pas engendrer une autre instance temporelle. Une telle représentation implique la création d'un très grand nombre d'instances temporelles. Toutefois, au cours de l'évolution d'un processus, il n'est pas nécessaire d'associer des instances temporelles aux objets qui sont stables pendant un certain laps de temps. Seules les instances temporelles correspondant à un temps où au moins une propriété a été modifiée sont alors créées.

Les accès en lecture ou en écriture à des propriétés temporelles se font grâce aux applicateurs qui sont redéfinis pour tenir compte du temps. Par défaut, si elle n'est pas fixée par le frame temporel, la valeur d'une propriété temporelle est héritée du frame atemporel. Pour cette raison, ainsi que pour simuler la continuité d'une synthèse à l'aide d'une série d'instances temporelles, des réflexes si-besoin ont été systématiquement associés aux attributs évolutifs des frames `Atome-t` et `Liaison-t`. Ces réflexes si-besoin activent la fonction `get-previous-value`, qui retournent la valeur de l'attribut au dernier temps où il a été modifié. La fonction `get-previous-value` utilise la méthode `instances` qui prend en argument un frame et un temps et qui retourne l'instance temporelle associée à ce temps si elle existe, sinon l'instance immédiatement antérieure.

Sur deux façons de représenter des objets temporels

La représentation des objets temporels en YAFOOL pose trois problèmes principaux, qui sont d'ailleurs en relation : la multiplication des instances temporelles, le partage de propriétés entre objets temporels et atemporels et la modélisation d'un processus continu. La création d'une instance temporelle à chaque pas de temps permet de manipuler des objets indépendants qui représentent des points de vue sur l'objet considéré à un temps donné. En contrepartie, l'ensemble des instances temporelles ne forment pas, comme on pourrait s'y attendre, une "chaîne" temporelle, à la façon d'une chaîne de Markov, où l'objet au temps `n` dépend de l'objet au temps `n-1`. Il s'ensuit que le partage de propriétés se fait toujours entre instance temporelle et frames temporel-atemporel, quelque soit le temps considéré, ce qui nécessite le recours à l'utilisation anti-naturelle de réflexes si-besoin. Ces défauts, qu'il fallait écarter, sont à l'origine de la nouvelle spécification des objets et propriétés temporels présentée au paragraphe 2.3.4.

5.6 Epilogue

Nous avons présenté dans ce chapitre une représentation d'objets graphiques évolutifs, dans le contexte de la synthèse de molécules organiques. La méthodologie mise au point pour décrire les structures moléculaires s'adapte à la représentation de structures graphiques quelconques. Les caractéristiques statiques des entités chimiques, atomes, liaisons, molécules, etc., sont décrites par des objets qui sont insérés dans une base bi-dimensionnelle formée de deux hiérarchies croisées, une dimension correspondant à la relation d'héritage et l'autre à la relation de composition. Les opérations élémentaires employées pour transformer les structures moléculaires sont représentées par des méthodes attachées aux objets génériques décrivant ces structures. L'évolution temporelle d'une structure moléculaire est prise en compte par le mécanisme des objets temporels. L'ensemble des deux hiérarchies croisées est assimilable à un réseau de contraintes sur lequel opère le raisonnement par classification. Il sert à construire et à gérer la hiérarchie associée à chaque dimension et, de façon plus originale, de méthode de resolution de problèmes. Cette méthode de résolution se généralise à tout univers qui se décrit sous la forme d'un réseau de contraintes, où la mise en place d'un objet dans le réseau apporte des informations sur la marche à suivre pour manipuler l'objet au présent et éventuellement dans le futur, conformément au principe de remémoration. Des méthodes de résolution de même type ont été appliquées dans le contexte du jeu d'échecs [Levinson, 1985] et dans celui de de la reconnaissance de formes [Granger, 1988]. Typiquement, le raisonnement par classification se révèle bien adapté à la résolution de "problèmes physiques" où les formes étudiées sont soumises à un ensemble de contraintes, éventuellement temporelles. Dans cet ordre d'idées, le mode de raisonnement qualitatif adapté à des modèles physiques composites présenté dans [Falkenhainer and Forbus, 1991] présente certaines analogies avec le raisonnement par classification.

Dans ce chapitre a également été introduit un mode original de partage de propriétés assurant l'"héritabilité" des transformations associées à une structure moléculaire le long de la dimension hiérarchique liée à la composition de structures. Ce mode de partage de propriétés se transpose à la gestion d'exceptions à l'héritage des attributs d'un objet et, plus généralement, à la gestion d'exceptions à l'héritabilité de propriétés le long d'une dimension hiérarchique définie par une relation de subsomption. Pour réaliser complètement une telle extension, il faut fixer la sémantique du partage de propriétés désiré et implanter les méthodes de gestion correspondantes.

Dans ce qui précède, nous avons privilégié et implanté le mode rétrosynthétique, qui est assimilable à un mode de résolution de problèmes de synthèse dirigé par les buts. Le mode synthétique apparaît comme la démarche complémentaire, dirigée par les données : la classification d'une molécule fournit alors un ensemble de méthodes de synthèse applicables dans le sens synthétique. Cette démarche est d'un grand intérêt, car elle permet de vérifier la pertinence des méthodes de synthèse employées : par exemple, le produit attendu est-il majoritaire, les méthodes de synthèse sont-elles peu ou pas du tout en compétition avec d'autres méthodes de synthèse ? La cohabitation des deux modes est une des suites envisagées de ce travail, tout comme l'étude et

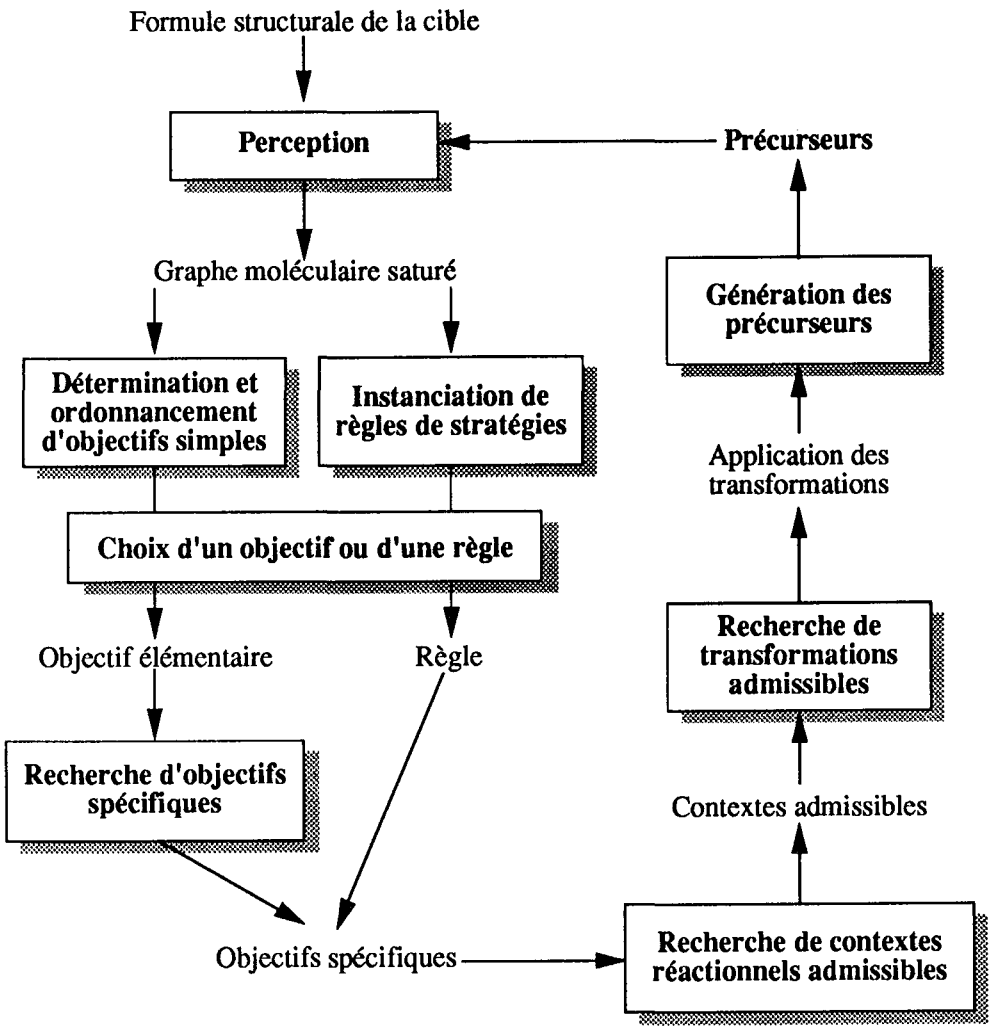


Figure 5.36. Schéma de fonctionnement du système RESYN.

l'implantation de stratégies de planification d'objectifs. Un arbre de synthèse décrit une suite d'étapes ordonnées dans le temps, chaque étape correspondant à un objectif à atteindre au cours de la synthèse. Les objectifs se divisent eux-mêmes en sous-objectifs à la façon classique des méta-plans et plans d'un système comme MOLGEN [Stefik, 1981a; Stefik, 1981b]. Pour l'avenir, nous envisageons d'orienter la recherche sur la planification d'objectifs vers des modes de raisonnement distribués comme celui de l'éco-résolution par exemple [Bura *et al.*, 1991] ; les molécules deviennent des agents régis par des procédures d'attraction-répulsion qui correspondent aux réactions permettant de passer ou de ne pas passer d'une famille de molécules à une autre.

Les idées présentées dans le chapitre 5 ont servi de base à l'implantation du système YCHEM. Cette implantation n'est pas encore complète, puisque la base de connaissances associée au système est limitée du point de vue des transformations. Les mêmes idées ont aussi servi de base à l'implantation du système RESYN, effectuée sous la di-

rection de C. Laurenço et de J. Quinqueton [Laurenço *et al.*, 1992] (cf. Fig. 5.36). La connaissance détenue par le système RESYN a été recueillie par C. Laurenço auprès de E. Toromanoff[14]. La démarche dans ce recueil d'expertise a été originale. Une première étape, transfert d'expertise chimique "pur", s'est déroulée principalement entre les deux chimistes, l'expert, E. Toromanoff, et le "relayeur", C. Laurenço. Lors de la seconde étape, le relayeur a trié et digéré les informations obtenues, puis après reflexion, a proposé une première mise en forme[15] de l'expertise recueillie. La troisième étape a mis en présence le relayeur et les informaticiens chargés de la programmation du système RESYN. La mise en forme de la connaissance incluse dans RESYN n'aurait sans doute jamais été possible sans le maillon intermédiaire qu'est le relayeur. Ce qui fait l'originalité de RESYN provient de l'originalité des connaissances de l'expert qui ne sont pas (seulement !) livresques. Un tel savoir ne pouvait être assimilé que par un chimiste dominant lui-même suffisamment bien la synthèse organique pour pouvoir trier et structurer les connaissances de l'expert. Dans le futur, le système YCHEM devrait être couplé au système RESYN ; il sera alors spécialisé dans l'étude de la fonctionnalité des molécules cible et sa tâche principale consistera à ajuster les fonctionnalités des cibles pour atteindre les objectifs fixés.

14. E. Toromanoff est un chimiste unanimement reconnu dans le milieu industriel comme un des meilleurs experts français de la synthèse organique. Il a quitté la société Roussel-Uclaf dans laquelle il travaillait pour prendre sa retraite. Cette société, qui voulait "conserver" une partie du savoir de E. Toromanoff, a été l'initiateur du projet RESYN.

15. La mise en forme de l'expertise s'est faite en collaboration avec E. Gaussens de la société FRAMENTEC.

6
Bilan et perspectives

Cette thèse aborde les notions de catégorisation et de raisonnement par classification dans le contexte de la représentation de connaissances. La catégorisation et le raisonnement par classification y jouent des rôles complémentaires : la tâche de la première consiste à organiser des connaissances en hiérarchies sur lesquelles opère le second. La catégorisation est d'abord présentée dans le premier chapitre, où nous mettons l'accent sur le côté psycho-linguistique du phénomène. Nous définissons ensuite le principe de remémoration sur lequel s'appuie le raisonnement par classification, principalement dans le contexte des représentations à objets.

Le deuxième chapitre spécifie les caractéristiques statiques d'une représentation à objets. Il explicite la structure attribut-facette ainsi que le comportement des objets à organisation hiérarchique que nous employons pour décrire des concepts du monde réel. Une représentation à objets est alors vue comme un environnement de programmation par objets pour la représentation de concepts. Les caractéristiques classiques d'un tel environnement sont semblables à celles des systèmes de représentation hybrides ; leur description est accompagnée d'études originales portant sur la sémantique des facettes, la programmation dirigée par les accès, les objets composites, et enfin, les objets temporels.

La programmation par objets pour la représentation de connaissances a des impératifs différents de ceux de la programmation par objets issue du courant du génie logiciel. La première se rapproche de la gestion de base de données évolutives, ainsi que des modes de programmation pratiqués dans les systèmes à subsomption. Le filtrage et, plus généralement, le raisonnement par classification, sont des opérations fondamentales attachées aux représentations à objets. Une étude détaillée des mécanismes propres au raisonnement par classification est donnée au chapitre 3, qui constitue le cœur de cette thèse. Cette étude concerne aussi bien l'utilisation du raisonnement par classification en gestion de hiérarchies d'objets qu'en résolution de problèmes. Contrairement à l'usage, nous faisons intervenir la subsomption comme support principal de l'ordre partiel régissant une hiérarchie d'objets dans le contexte des systèmes à héritage classiques. En particulier, la subsomption généralise l'héritage et elle permet de procéder à des inférences complexes, de l'ordre de celles qui sont effectuées en calcul des prédicats. Nous montrons à cette occasion différentes façons de conduire le raisonnement par classification dans le graphe d'héritage d'une représentation à objets, lorsque la relation de subsomption est identifiée à la relation d'héritage. Plus généralement, et

c'est là une des originalités de la démarche, nous envisageons une base de connaissances comme un espace n-dimensionnel : une relation de subsomption spécifique détermine la métrique réglementant les rapports entre les objets le long d'une dimension. Chaque dimension correspond à une perspective de la base de connaissances. Une représentation à objets apparaît alors comme un environnement de programmation intégrant dans l'univers des systèmes à héritage des outils de raisonnement empruntés pour une bonne part aux systèmes à subsomption. Cette intégration se traduit par un couplage du filtrage et du raisonnement par classification avec les modes de programmation procéduraux que sont la programmation dirigée par les accès et l'envoi de message.

Dans le chapitre 4, nous mettons d'abord en évidence les connexions qui existent entre l'analyse de données, la catégorisation conceptuelle et certaines théories de l'apprentissage automatique symbolique, comme la catégorisation hiérarchique incrémentielle. Ces trois disciplines ont le même but, trier et classer des données hétérogènes pour les expliquer. Comme le principe de remémoration y fait implicitement référence, le raisonnement par classification relève dans une certaine mesure du raisonnement par analogie, à l'instar du raisonnement par cas. En particulier, la relation de subsomption peut servir de mode d'indexation de cas : expliquer une nouvelle situation revient alors à la classer dans une mémoire de cas organisée en hiérarchie.

Le chapitre 5 constitue la partie expérimentale de la thèse. Il illustre tous les points théoriques originaux abordés dans les chapitres précédents. La synthèse organique s'est avérée un terrain d'expérimentation privilégié pour les techniques de programmation par objets pour la représentation de connaissances. Cela est dû en grande partie à la nature graphique du langage couramment employé par les chimistes. Ce langage a un très fort pouvoir d'expression : une formule chimique développée véhicule une masse considérable d'informations de toutes natures. La formule développée d'une molécule peut se voir comme un graphe étiqueté, structure bien étudiée en informatique. Parallèlement, effectuer une synthèse revient à transformer une molécule en une autre molécule. Il faut donc décrire les transformations agissant sur les structures chimiques et les conditions d'application attachées à ces transformations. Le problème de la modélisation du plan de synthèse d'une molécule organique se pose alors dans les termes suivants : trouver un environnement de programmation où les graphes sont représentés, manipulés et modifiés au cours du temps de la façon la plus naturelle possible. Or, jusqu'à présent, l'accent dans le domaine a surtout été mis sur la représentation procédurale des transformations. Notre démarche est originale, car elle inverse la façon classique de procéder, rappelant dans une certaine mesure le dualisme programmation procédurale avec priorité aux traitements et programmation par objets avec priorité aux données. Nous avons choisi de représenter en premier lieu, et sous la forme d'objets, les structures moléculaires et les opérations qui leur sont associées. Plutôt que de décrire une transformation et ses différents cas d'application, ce sont les environnements favorables à l'application de la transformation qui sont décrits dans notre modèle. Les actions associées à une transformation se réduisent à une suite d'opérations chirurgicales élémentaires. L'ensemble des informations chimiques relatives à la description d'un plan de synthèse est structuré en différentes hiérarchies croisées d'objets. Dans ce contexte, le raisonnement par classification montre alors toute sa mesure en tant que méthode de résolution de problèmes : classifier une molécule dans une hiérarchie

d'environnements revient à franchir une étape dans la résolution d'un problème de synthèse.

Les fonctionnalités de YCHEM décrites dans le chapitre 5 sont implantées en YAFOOL. La base de connaissances est actuellement limitée du point de vue des réactions et doit encore être enrichie. Dans le futur, le système va être spécialisé dans l'étude des échanges de fonctionnalités qui doivent être apportés à une molécule cible pour atteindre un objectif dans une synthèse. Cette évolution du système devrait faire de YCHEM l'assistant du système RESYN. De nombreux problèmes restent encore à résoudre, en particulier le couplage du système avec des bases de données de molécules et de réactions, ainsi que la mise en forme et l'utilisation d'un raisonnement par analogie pour exploiter cette masse de données supplémentaire. Les résultats enregistrés jusqu'à présent sont encourageants, en ce qui concerne la validation des idées qui sont à la base du système YCHEM, son architecture, sa conception, son évolutivité et sa maintenance. Dans cette voie, les systèmes RESYN et YCHEM font figure de précurseurs, et d'autres systèmes de conception de plans de synthèse devraient les suivre.

Bibliographie

[Agha, 1986] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Massachusetts, 1986.

[Aguirre, 1989] J.L. Aguirre. *Construction automatique de taxonomies à partir d'exemples dans un modèle de connaissances par objets.* Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 1989.

[Aikins, 1983] J.S. Aikins. Prototypical Knowledge for Expert Systems. *Artificial Intelligence*, 20(2):163–210, 1983.

[Aït-Kaci and Podelski, 1991] H. Aït-Kaci and A. Podelski. Towards a Meaning of LIFE. In W. Bibel and Ph. Jorrand, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming (Passau, Germany)*, Lecture Notes in Computer Science (528), pages 255–274. Springer-Verlag, Berlin, 1991.

[Albert, 1988] P. Albert. KOOL: Merging Object Frames and Rules. In J. Demongeot, T. Hervé, V. Rialle, and C. Roche, editors, *Artificial Intelligence and Cognitive Sciences*, pages 15–21. Manchester University Press, New York, 1988.

[Allen, 1983] E. Allen. YAPS: A Production Rule System Meets Objects. In *Proceedings of AAAI'83, Washington, D.C.*, pages 5–7, 1983.

[Allen, 1984] J.F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(2):123–154, 1984.

[Allgayer *et al.*, 1989] J. Allgayer, K. Harbusch, A. Kobsa, C. Reddig, N. Reithinger, and D. Schmauks. XTRA: a natural-language access system to expert systems. *International Journal of Man-Machine Studies*, 31(2):161–195, 1989.

[Arnauld and Nicolle, 1662] A. Arnauld and P. Nicolle. *La logique ou l'art de penser.* Flammarion (1970), Paris, 1662.

[Attardi *et al.*, 1986] G. Attardi, A. Corradini, S. Diomedi, and M. Simi. Taxonomic Reasoning. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI'86), Brighton, England*, pages 236–245, 1986.

[Attardi, 1991] G. Attardi. An Analysis of Taxonomic Reasoning. In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 29–49. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Audureau *et al.*, 1987] E. Audureau, L. Fariñas Del Cerro, and P. Enjalbert. Théorie de la programmation et logique temporelle (1ère partie : Validation d'algorithmes séquentiels. *Techniques et Sciences Informatiques*, 6(6):527–540, 1987.

[Bachelard, 1973] G. Bachelard. *Essai sur la connaissance approchée.* Librairie philosophique J. Vrin, Paris, 1973.

[Barone and Chanon, 1986] R. Barone and M. Chanon. Computer-Aided Organic Synthesis (CAOS). In G. Vernin and M. Chanon, editors, *Computer Aids to Chemistry*, pages 19–102. Ellis Horwood, Chichester, West Sussex, 1986.

[Barr *et al.*, 1981] A. Barr, E.A. Feigenbaum, and P.R. Cohen, editors. *The Handbook of Artificial Intelligence.* William Kaufmann, Inc., Los Altos, California, 1981.

[Bdbc, 1987] Actes des Journées d'Etude AFCET *Des Bases de Données aux Bases de Connaissances.* AFCET/édiTESTS, 1987.

[Beech, 1988] D. Beech. Intensional Concepts in an Object Database Model. In *Proceedings of the 3rd OOPSLA, San Diego, California, ACM SIGPLAN Notices 23(11)*, pages 164–175, 1988.

[Benzecri, 1973] J.-P. Benzecri. *L'Analyse des Données (1. La Taxinomie).* Dunod, Paris, 1973.

[Bestougeff and Ligozat, 1989] H. Bestougeff and G. Ligozat. *Outils logiques pour le traitement du temps.* Masson, Paris, 1989.

[Blake and Cook, 1987] E. Blake and S. Cook. On Including Part Hierarchies in Object-Oriented Languages, with an Implementation in Smalltalk. In *Proceedings of 1st European Conference on Object-Oriented Programming (ECOOP'87), Paris, Lecture Notes in Computer Science 276*, pages 45–54, 1987.

[Bobick, 1987] A.F. Bobick. Natural Object Categorization. Technical Report AI-TR-1001, AI Lab, MIT, Cambridge, Massachusetts, 1987.

[Bobrow and Stefik, 1983] D.G. Bobrow and M. Stefik. The LOOPS Manual: A Data and Object Oriented Programming System for Interlisp. Knowledge-Based VLSI Design Group Memo KB-VLSI-81-13, Xerox PARC, Palo Alto, California, 1983.

[Bobrow and Winograd, 1977] D.G. Bobrow and T. Winograd. An Overview of KRL, a Knowledge Representation Language. *Cognitive Science*, 1(1):3–46, 1977.

[Bobrow *et al.*, 1986] D.G. Bobrow, K. Kahn, G. Kiczales, L. Masinter, M. Stefik, and F. Zdybel. Commonloops: Merging LISP and Object-Oriented Programming. In *Proceedings of the 1st OOPSLA, Portland, Oregon, ACM SIGPLAN Notices 21(11)*, pages 17–29, 1986.

[Bond and Gasser, 1988] A.H. Bond and L. Gasser, editors. *Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[Booch, 1990] G. Booch. *Object Oriented Design with Applications.* Benjamin/Cummings, Reading, Massachusetts, 1990.

[Borgida *et al.*, 1984] A. Borgida, J. Mylopoulos, and H.K.T. Wong. Generalization/Specification as a Basis for Software Specification. In M.L. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modelling*, pages 87–117. Springer-Verlag, Berlin, 1984.

[Borgida *et al.*, 1989] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the ACM/SIGMOD International Conference on the Management of Data, Portland, Oregon, SIGMOD RECORD, 18(2)*, pages 59–67, 1989.

[Borgida, 1988] A. Borgida. Class Hierarchies in Information Systems: Sets, Types, or Prototypes? In M.P. Atkinson, P. Buneman, and R. Morrison, editors, *Data Types and Persistence*, pages 137–154. Springer-Verlag, Berlin, 1988.

[Borning *et al.*, 1987] A.H. Borning, R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf. Constraints Hierarchies. In *Proceedings of the 2nd OOPSLA, Orlando, Florida, ACM SIGPLAN Notices 22(12)*, pages 48–60, 1987.

[Borning, 1986] A.H. Borning. Classes Versus Prototypes in Object-Oriented Languages. In *ACM-IEEE Fall Joint Computer Conference (FJCC'86)*, pages 36–39, 1986.

[Bouroche and Saporta, 1987] J-M. Bouroche and G. Saporta. *L'analyse des donnés.* Presses Universitaires de France, (Que-sais-je 1854), Paris, 1987.

[Brachman and Levesque, 1984] R.J. Brachman and H.J. Levesque. The Tractability of Subsumption in Frame-Based Description Language. In *Proceedings of AAAI'84, Austin, Texas*, pages 34–37, 1984.

[Brachman and Schmolze, 1985] R.J. Brachman and J.G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al.*, 1983] R.J. Brachman, R.E. Fikes, and H.J. Levesque. Krypton: A Functional Approach to Knowledge Representation. COMPUTER, 16(10):67–73, 1983.

[Brachman *et al.*, 1985] R.J. Brachman, V.P. Gilbert, and H.J. Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON. In *Proceedings of the 9th IJCAI, Los Angeles, California*, pages 532–539, 1985.

[Brachman *et al.*, 1991] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Resnick, and A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE Language. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Brachman, 1977] R.J. Brachman. What's in a Concept: Structural Foundations for Semantic Networks. *International Journal of Man-Machine Studies*, 9:127–152, 1977.

[Brachman, 1979] R.J. Brachman. On the Epistemological Status of Semantic Networks. In N.V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3–50. Academic Press, New York, 1979.

[Brachman, 1983] R.J. Brachman. What IS-A is and isn't: An Analysis of Taxonomic Links in Semantic Networks. COMPUTER, 16(10):30–37, 1983.

[Brachman, 1985] R.J. Brachman. "I Lied about the Trees" or, Defaults and Definitions in Knowledge Representation. *The AI Magazine*, 6(3):80–93, 1985.

[Brachman, 1988] R.J. Brachman. The Basics of Knowledge Representation and Reasoning. *AT&T Technical Journal*, 67(1):7–24, 1988.

[Brachman, 1990] R.J. Brachman. The Future of Knowledge Representation. In *Proceedings of AAAI'90, Boston, Massachusetts*, pages 1082–1092, 1990.

[Briot and Cointe, 1987] J.P. Briot and P. Cointe. A Uniform Model for Object-Oriented Languages Using The Class Abstraction. In *Proceedings of the 10th IJCAI, Milano, Italy*, pages 40–43, 1987.

[Briot and Cointe, 1989] J.P. Briot and P. Cointe. Programming with Explicit Metaclasses in Smalltalk-80. In *Proceedings of the 4th OOPSLA, New Orleans, Louisiana, ACM SIGPLAN Notices (24)10*, pages 419–431, 1989.

[Brodie *et al.*, 1984] M.L. Brodie, J. Mylopoulos, and J.W Schmidt, editors. *On Conceptual Modelling.* Springer-Verlag, Berlin, 1984.

[Buchanan and Feigenbaum, 1978] B.G. Buchanan and E.A. Feigenbaum. Dendral and Meta-Dendral: Their Applications Dimensions. *Artificial Intelligence*, 11:5–24, 1978.

[Bura *et al.*, 1991] S. Bura, A. Drogoul, J. Ferber, and E. Jacopin. Eco-résolution : un modèle de résolution de problèmes par interactions. In *Actes du 8ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA'91), Lyon-Villeurbanne*, pages 1299–1308, 1991.

[Bylander and Mittal, 1986] T. Bylander and S. Mittal. CSRL: a Language for Classificatory Problem Solving and Uncertainty Handling. *The AI Magazine*, 7(3):66–77, 1986.

[Bylander *et al.*, 1989] T. Bylander, T.R. Johnson, and A. Goel. Structured Matching: A Task-Specific Technique for Making Decisions. In *IEEE International Workshop on Tools For Artificial Intelligence (IWTAI'89), Fairfax, Virginia*, pages 138–145, 1989.

[Carnap, 1958] R. Carnap. *Introduction to Symbolic Logic and its Applications.* Dover Publications, Inc., New York, 1958.

[Carré and Geib, 1990] B. Carré and J-M. Geib. The Point of View Notion for Multiple Inheritance. In *Proceedings of OOPSLA-ECOOP'90, Ottawa, Canada, ACM SIGPLAN Notices (25)10*, pages 312–321, 1990.

[Carré, 1989] B. Carré. *Méthodologie orientée objet pour la représentation des connaissances. Concepts de points de vue, de représentation multiple et évolutive d'objet.* Thèse de l'Université des Sciences et Techniques de Lille Flandres Artois, 1989.

[Caseau, 1987] Y. Caseau. *Etude et réalisation d'un langage objet : LORE.* Thèse de Doctorat d'Etat, Université de Paris-Sud, 1987.

[Caseau, 1989] Y. Caseau. *The Laure System: Documentation. Version 1.05.* Database Research Group Bellcore, Bell Communications Research Center, Morristown, New Jersey, 1989.

[Cauzinille-Marmèche *et al.*, 1990] E. Cauzinille-Marmèche, D. Dubois, and J. Mathieu. Catégories et processus de catégorisation. In G. Netchine-Grynberg, editor, *Développement et fonctionnement cognitifs chez l'enfant (Des modèles généraux aux modèles locaux)*, pages 93–119. Presses Universitaires de France, Paris, 1990.

[Chabre, 1988] B. Chabre. Le filtrage en YAFOOL. Rapport technique, SEMA.METRA, Montrouge, 1988.

[Chambers *et al.*, 1991] C. Chambers, D. Ungar, B-W. Chang, and U. Hölzle. Parents are Shared Parts of Objects: Inheritance and Encapsulation in SELF. *LISP and Symbolic Computation*, 4(3):207–222, 1991.

[Chandrasekaran and Goel, 1988] B. Chandrasekaran and A. Goel. From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task. IEEE *Transactions on Systems, Man and Cybernetics*, 18(3):415–424, 1988.

[Chang and Lee, 1973] C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Chein and Cogis, 1989] M. Chein and O. Cogis. Un modèle pour la programmation orientée réflexe. In *Actes du 7ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA '89), Paris*, pages 1623–1630, 1989.

[Cherniak, 1984] C. Cherniak. Prototypicality and Deductive Reasoning. *Journal of Verbal Learning and Verbal Behavior*, 23:625–642, 1984.

[Chouraqui *et al.*, 1985] E. Chouraqui, H. Farreny, D. Kayser, and H. Prade. Modélisation du raisonnement et de la connaissance. *Techniques et Sciences Informatiques*, 4(4):391–399, 1985.

[Clancey, 1985] W.J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[Clayton, 1984] B.D. Clayton. *ART Programming Primer.* Inference Corporation, Los Angeles, California, 1984.

[Cohen and Murphy, 1984] B. Cohen and G.L. Murphy. Models of Concepts. *Cognitive Science*, 8(1):27–58, 1984.

[Cointe, 1988] P. Cointe. A Tutorial Introduction to Metaclass Architecture as provided by Class Oriented Languages. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), Tokyo*, pages 592–608, 1988.

[Corey and Wipke, 1969] E.J. Corey and W.T. Wipke. Computer–Assisted Analysis of Complex Syntheses. *Science*, 166:178–192, 1969.

[Corey *et al.*, 1985] E.J. Corey, A.K. Kong, and S.D. Rubenstein. Computer–Assisted Analysis of in Organic Synthesis. *Science*, 228:408–4418, 1985.

[Cuvillier, 1954] A. Cuvillier. *Cours de philosophie.* Librairie Armand Colin, Paris, 1954.

[Dale, 1985] M.B. Dale. On the Comparison of Conceptual Clustering and Numerical Taxonomy. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 7(2):241–244, 1985.

[Davis, 1987] H.E. Davis. VIEWS: Multiple Perspectives and Structured Objects in a Knowledge Representation Language. Bachelor and master of science, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1987.

[Dean and McDermott, 1987] T.L. Dean and D.V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32(1):1–55, 1987.

[Decaestecker, 1989] C. Decaestecker. Incremental Concept Formation via a Suitability Criterion. In E. Diday, editor, *Proceedings of the Conference on Data Analysis, Learning Symbolic and Numeric Knowledge, Juan-Les-Pins, France*, pages 435–442. Nova Science Publisher Inc., New-York, 1989.

[Delpech *et al.*, 1987] P-M. Delpech, A. Potet, and C. Sayettat. Démonstration automatique et logiques temporelles. *Techniques et Sciences Informatiques*, 6(6):541–557, 1987.

[Desclés and Kanellos, 1991] J.P. Desclés and I. Kanellos. La notion de typicalité : une approche formelle. In D. Dubois, editor, *Sémantique et cognition - Catégories, prototypes, typicalité*, pages 223–244. Editions du CNRS, Paris, 1991.

[Desclés, 1986] J.P. Desclés. Implication entre concepts : la notion de typicalité. *Travaux de Linguistique et de Littérature, Centre de Philologie et de Littératures Romanes de l'Université de Strasbourg*, XXIV(1):179–202, 1986.

[Diday *et al.*, 1982] E. Diday, J. Lemaire, J. Pouget, and F. Testu. *Eléments d'analyse des données.* Dunod, Paris, 1982.

[Diday, 1989] E. Diday. Introduction à l'approche symbolique en analyse des données. *RAIRO Recherche Opérationnelle*, 23(2):193–236, 1989.

[Dorst, 1971] J. Dorst. *La vie des oiseaux.* Bordas, Paris, 1971.

[Dubois, 1986] D. Dubois. La compréhension de phrases : représentations sémantiques et processus. Thèse de Doctorat d'Etat, Université de Paris 8, 1986.

[Ducournau and Habib, 1989] R. Ducournau and M. Habib. La multiplicité de l'héritage dans les langages à objets. *Techniques et Sciences Informatiques*, 8(1):41–62, 1989.

[Ducournau and Habib, 1991] R. Ducournau and M. Habib. Masking and Conflicts, or To Inherit is Not To Own! In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 223–244. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Ducournau, 1989a] R. Ducournau. *Y3. Interfaces graphiques.* SEMA GROUP, Montrouge, 1989.

[Ducournau, 1989b] R. Ducournau. *Y3. Langage à objets. Version 3.22.* SEMA GROUP, Montrouge, 1989.

[Ducournau, 1992] R. Ducournau. *Quelques considérations sur l'héritage.* Rapport interne SEMA GROUP, Montrouge, 1992.

[Duda and Hart, 1973] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons Ltd, New York, 1973.

[Dugerdil, 1987a] P. Dugerdil. Les mécanismes d'héritage d'OBJLOG : vertical et sélectif multiple avec point de vue. In *Actes du 6ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA'87), Antibes*, pages 259–273, 1987.

[Dugerdil, 1987b] P. Dugerdil. Représentation orientée objet et systèmes experts : une introduction. Note interne 176, Groupe Représentation et Traitement des Connaissances, Marseille, 1987.

[Dugerdil, 1988] P. Dugerdil. *Contribution à l'étude de la représentation des connaissances fondée sur les objets. Le langage OBJLOG.* Thèse de l'Université d'Aix-Marseille II, 1988.

[Dugerdil, 1991] P. Dugerdil. Inheritance Mechanisms in the OBJLOG Language: Multiple Selective and Multiple Vertical with Points of View. In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 245–256. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Ellman, 1989] T. Ellman. Explanation-Based Learning: A Survey and Perspectives. ACM *Computing Surveys*, 21(2):163–221, 1989.

[Etherington *et al.*, 1989] D.W. Etherington, A. Borgida, R.J. Brachman, and H. Kautz. Vivid Knowledge and Tractable Reasoning: Preliminary Report. In *Proceedings of the 11th IJCAI, Detroit, Michigan*, pages 1146–1152, 1989.

[Etherington, 1987] D.W. Etherington. Formalizing Nonmonotonic Reasoning Systems. *Artificial Intelligence*, 31(1):41–85, 1987.

[Euzénat, 1990] J. Euzénat. Un système de maintenance de la vérité à propagation de contextes. Thèse de l'Université Joseph Fourier, Grenoble, 1990.

[Falkenhainer and Forbus, 1991] B. Falkenhainer and K.D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[Feigenbaum and Simon, 1984] E.A. Feigenbaum and H. Simon. EPAM-like Models of Recognition and Learning. *Cognitive Science*, 8:305–336, 1984.

[Ferber and Briot, 1988] J. Ferber and J.P. Briot. Design of a Concurrent Language for Distributed Artificial Intelligence. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), Tokyo*, pages 755–762, 1988.

[Ferber and Ghallab, 1988] J. Ferber and M. Ghallab. Problématique des univers multi-agents intelligents. In *Actes des Journées Nationales du PRC IA, Toulouse*, pages 295–320, 1988.

[Ferber and Volle, 1988] J. Ferber and P. Volle. Using Coreference in Object Oriented Representations. In *Proceedings of the 8th* ECAI, *Munich, West Germany*, pages 238–240, 1988.

[Ferber, 1983] J. Ferber. *MERING : un langage d'acteurs pour la représentation et la manipulation des connaissances.* Thèse de Docteur Ingénieur, Université de Paris 6, 1983.

[Ferber, 1986] J. Ferber. Systèmes experts et approches orientées objets. In *Actes des 6èmes Journeés Internationales sur les Systèmes Experts et leurs Applications, Avignon*, pages 525–542, 1986.

[Ferber, 1989] J. Ferber. Objets et agents : une étude des structures de représentation et de communications en Intelligence Artificielle. Thèse de Doctorat d'Etat, Université de Paris 6, 1989.

[Ferber, 1990] J. Ferber. *Conception et programmation par objets.* Hermès, Paris, 1990.

[Fikes and Kehler, 1985] R. Fikes and T. Kehler. The Role of Frame-Based Representation in Reasoning. *Communications of the* ACM, 28(9):904–920, 1985.

[Filman, 1988] R.E. Filman. Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environment. *Communications of the* ACM, 31(4):382–401, 1988.

[Finin and Drager, 1986] T. Finin and D. Drager. GUMS: A General User Modeling System. In *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence*, Montreal, Canada, 1986.

[Finin, 1986] T.W. Finin. Interactive Classification: A Technique for Acquiring and Maintaining Knowledge Bases. *Proceedings of the* IEEE, 74(10):1414–1421, 1986.

[Fischer and Napoli, 1991] D. Fischer and A. Napoli. Perception des groupes fonctionnels d'une molécule à l'aide d'un nom linéaire local. In *Actes de la 3ème Journée Informatique Avancée et Chimie Organique, Marseille*, 1991.

[Fisher, 1987] D.H. Fisher. Knowledge Acquisition Via Conceptual Clustering. *Machine Learning*, 2:139–172, 1987.

[Fox *et al.*, 1986] M.S. Fox, J.M. Wright, and D. Adam. Experiences with SRL: An Analysis of a Frame-based Knowledge Representation. In L. Kerschberg, editor, *Proceedings of the 1st International Workshop on Expert Database Systems, Charleston, South Carolina*, pages 161–172. Benjamin/Cummings Publishing, Menlo Park, California, 1986.

[Fox, 1979] M.S. Fox. On Inheritance in Knowledge Representation. In *Proceedings of the 6th IJCAI, Tokyo*, pages 282–284, 1979.

[Franke, 1990] D.W. Franke. Imbedding Rule Inferencing in Applications. IEEE *Expert*, pages 8–14, December 1990.

[Frege, 1893] G. Frege. *Grundsetze der Aritmetik, begriffsschriftlich abgeleitet.* Jena, 1893.

[Frege, 1971] G. Frege. *Ecrits logiques et philosophiques.* Seuil, Paris, 1971.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[Garlatti, 1990] S. Garlatti. Exceptions à l'héritage et logiques non monotones. Thèse de l'Université de Rennes, 1990.

[Gascuel and Guénoche, 1990] O. Gascuel and A. Guénoche. Aspects de l'interface entre symbolique et numérique. In *Actes des Journées Nationales du PRC IA, Paris*, pages 91–112. Hermès, 1990.

[Gascuel, 1987] O. Gascuel. PLAGE : Un outil pour construire des systèmes d'apprentissage. In *Actes du 6ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA'87), Antibes*, pages 863–877, 1987.

[Gelernter *et al.*, 1984] H. Gelernter, G.A. Miller, D.L. Larsen, and D.J. Berndt. Realization of a Large Expert Problem-Solving System SYNCHEM2: A Case Study. In *Proceedings of the 1st Conference on Artificial Intelligence Applications, Denver, Colorado*, pages 92–106, 1984.

[Gennari *et al.*, 1989] J.H. Gennari, P. Langley, and D. Fischer. Models of Incremental Concept Formation. *Artificial Intelligence*, 40:11–61, 1989.

[Goldberg and Robson, 1983] A. Goldberg and D. Robson. *Smalltalk-80, the Language and its Implementation.* Addison Wesley, Reading, Massachusetts, 1983.

[Goldstein and Bobrow, 1980] I.P. Goldstein and D.G. Bobrow. Descriptions for a Programming Environment. In *Proceedings of AAAI'80, Stanford University, California*, pages 187–189, 1980.

[Goldstein and Roberts, 1977] I.P. Goldstein and R.B. Roberts. NUDGE, a Knowledge-Based Scheduling Program. In *Proceedings of the 5th IJCAI, Cambridge, Massachusetts*, pages 257–263, 1977.

[Gomez and Segami, 1989] F. Gomez and C. Segami. The recognition and classification of concepts in understanding scientific texts. *Journal of Experimental & Theoretical Artificial Intelligence*, 1(1):51–77, 1989.

[Gottlob and Leitsch, 1985] G. Gottlob and A. Leitsch. On the Efficiency of Subsumption Algorithms. *Journal of the* ACM, 32(2):280–295, 1985.

[Gottlob, 1987] G. Gottlob. Subsumption and Implication. *Information Processing Letters*, 24:109–111, 1987.

[Granger, 1988] C. Granger. An Application of Possibility Theory to Object Recognition. *Fuzzy Sets and Systems*, 28(4):351–362, 1988.

[Griffith, 1982] R.L. Griffith. Three Principles of Representation for Semantic Networks. ACM *Transactions on Database Systems*, 7(3):417–442, 1982.

[Guesguen *et al.*, 1987] H.W. Guesguen, U. Junker, and A. Voss. Constraints in a Hybrid Knowledge Representation System. In *Proceedings of the 10th IJCAI, Milano, Italy*, pages 30–33, 1987.

[Guesguen, 1989] H.W. Guesguen. A Universal Programming Language. In *Proceedings of the 11th IJCAI, Detroit, Michigan*, pages 60–65, 1989.

[Haase, 1986] K. Haase. ARLO: Another Representation Language Offer. Technical Report 901, AI Lab, MIT, Cambridge, Massachusetts, 1986.

[Hadzikadic and Yun, 1989] M. Hadzikadic and D.Y.Y. Yun. Concept Formation by Incremental Conceptual Clustering. In *Proceedings of the 11th IJCAI, Detroit, Michigan*, pages 831–836, 1989.

[Hall, 1986] R.J. Hall. Learning by Failing to Explain. Technical Report 906, AI Lab, MIT, Cambridge, Massachusetts, 1986.

[Hammond, 1990] K.J. Hammond. Explaining and Repairing Plans That fail. *Artificial Intelligence*, 45(1-2):173-228, 1990.

[Haton *et al.*, 1991] J.-P. Haton, N. Bouzid, F. Charpillet, M.-C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, and A. Napoli. *Le raisonnement en intelligence artificielle.* InterEditions, Paris, 1991.

[Hayes, 1979] P.J. Hayes. The Logic of Frames. In D. Metzing, editor, *Frame Conception and Text Understanding*, pages 46-61. de Gruyter, Berlin, 1979.

[Hendrickson, 1990] J.B. Hendrickson. Organic synthesis in the age of computers. *Angewandte Chemie - International Edition in English*, 29:1286-1295, 1990.

[Hewitt and Jong, 1982] C.E. Hewitt and P. de Jong. Open Systems. AI Memo 691, AI Lab, MIT, Cambridge, Massachusetts, 1982.

[Hewitt *et al.*, 1973] C.E. Hewitt, P. Bishop, and R. Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd IJCAI, Stanford, California*, pages 235-245, 1973.

[Hewitt, 1977] C.E. Hewitt. Viewing Control Structure as Patterns of Passing Messages. *Artificial Intelligence*, 8:323-364, 1977.

[Hewitt, 1985] C.E. Hewitt. The Challenge of Open Systems. *Byte*, 10(4):223-242, 1985.

[Horty *et al.*, 1990] J.F. Horty, R.H. Thomason, and D.S. Touretzky. A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. *Artificial Intelligence*, 42(2-3):311-348, 1990.

[Horty, 1991] J.F. Horty. A Credulous Theory of Mixed Inheritance. In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 13-28. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Hull and King, 1987] R. Hull and R. King. Semantic Database Modeling: Survey, Applications and Research Issues. ACM *Computing Surveys*, 19(3):202-260, 1987.

[Ibrahim and Cummins, 1988] M.H. Ibrahim and F.A. Cummins. KSL: A Reflective Object-Oriented Programming Language. In *Proceedings of the International Conference on Computer Language (ICCL'88), Miami, Florida*, pages 186-193, 1988.

[Ibrahim and Cummins, 1990] M.H. Ibrahim and F.A. Cummins. KSL/Logic: Integration of Logic with Objects. In *Proceedings of the International Conference on Computer Language (ICCL'90), New Orleans, Louisiana*, pages 228-235, 1990.

[Jauffret *et al.*, 1986] P. Jauffret, C. Laurenço, and G. Kaufmann. PSYCHO : Un programme d'aide à la synthèse en chimie organique. *Techniques et Sciences Informatiques*, 5(5):375-390, 1986.

[Johnson-Laird *et al.*, 1984] P.N. Johnson-Laird, D.J. Herrmann, and R. Chaffin. Only Connections: A Critique of Semantic Networks. *Psychological Bulletin*, 96(2):292–315, 1984.

[Johnson-Laird, 1983] P.N. Johnson-Laird. *Mental Models. Towards a Cognitive Science of Language, Inference and Consciousness.* Cambridge University Press, Cambridge, UK, 1983.

[Johnson, 1985] A.P. Johnson. Computer aids to synthesis planning. *Chemistry in Britain*, 21(1):59–67, 1985.

[Kaczmarek *et al.*, 1986] T.S. Kaczmarek, R. Bates, and G. Robins. Recent Developments in NIKL. In *Proceedings of AAAI'86, Philadelphia, Pennsylvania*, pages 978–985, 1986.

[Kayser, 1984] D. Kayser. Examen des diverses méthodes utilisées en représentation des connaissances. In *Actes du 4ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA'84), Paris*, pages 115–144, 1984.

[Kayser, 1985] D. Kayser. Comment représenter la typicalité ? In *Actes du Congrès AFCET Informatique Matériels et Logiciels pour la 5ème Génération, Paris*, pages 177–186, 1985.

[Keene, 1989] S.E. Keene. *Object-Oriented Programming in Common Lisp. A Programmer's Guide to CLOS.* Addison Wesley, Reading, Massachusetts, 1989.

[Kempf and Stelzner, 1987] R. Kempf and M. Stelzner. Teaching Object-Oriented Programming with the KEE System. In *Proceedings of the 2nd OOPSLA, Orlando, Florida, ACM SIGPLAN Notices 22(12)*, pages 11–25, 1987.

[Kim *et al.*, 1987] W. Kim, J. Banerjee, H.T. Chou, J.F. Garza, and D. Woelk. Composite Object Support in an Object-Oriented Database System. In *Proceedings of the 2nd OOPSLA, Orlando, Florida, ACM SIGPLAN Notices 22(12)*, pages 118–125, 1987.

[Kim *et al.*, 1989] W. Kim, E. Bertino, and J.F. Garza. Composite Objects Revisited. In *Proceedings of the ACM/SIGMOD International Conference on the Management of Data, Portland, Oregon, SIGMOD RECORD, 18(2)*, pages 337–347, 1989.

[Kim, 1990] W. Kim. *Introduction to Object-Oriented Databases.* MIT Press, Cambridge, Massachusetts, 1990.

[Kleiber, 1990] G. Kleiber. *La sémantique du prototype.* Presses Universitaires de France, Paris, 1990.

[Kobsa, 1991] A. Kobsa. Utilizing Knowledge: The Components of the SB-ONE Knowledge Representation Workbench. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 457–486. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Kodratoff, 1986] Y. Kodratoff. *Leçons d'Apprentissage Symbolique Automatique.* Cepadues-Editions, Toulouse, 1986.

[Kolodner *et al.*, 1985] J.L. Kolodner, R.L. Simpson, and K. Sycara-Cyranski. A Process of Case-Based Reasoning in Problem Solving. In *Proceedings of the 9th IJCAI, Los Angeles, California*, pages 284–290, 1985.

[Kolodner, 1983] J. Kolodner. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science*, 7:243–280, 1983.

[Kolodner, 1991] J.L. Kolodner. Improving Human Decision Making through Case-Based Decision Aiding. *The AI Magazine*, 12(2):52–68, 1991.

[Kuipers, 1975] B.J. Kuipers. A Frame for Frames. In D.G. Bobrow and A. Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 151–185. Academic Press, New York, 1975.

[Lakoff, 1987] G. Lakoff. *Women, Fire, and Dangerous Things (What Categories Reveal about the Mind)*. The University of Chicago Press, Chicago, 1987.

[Laurenço *et al.*, 1990] C. Laurenço, M. Py, A. Napoli, J. Quinqueton, and B. Castro. Représentation de connaissances en synthèse organique à l'aide d'un langage à objets. *New Journal of Chemistry*, 14(12):921–931, 1990.

[Laurenço *et al.*, 1992] C. Laurenço, M. Py, J. Quinqueton, J-C. Regin, and P. Vismara. RESYN. Rapport technique, LIRMM, Montpellier, 1992.

[Laurenço, 1985] C. Laurenço. Synthèse organique assistée par ordinateur. Thèse de Doctorat d'Etat, Université Louis Pasteur, Strasbourg, 1985.

[Laurière, 1982] J.L. Laurière. Représentation et utilisation des connaissances. *Techniques et Sciences Informatiques*, 1(1):25–42 et 1(2):115–144, 1982.

[Lebowitz, 1987] M. Lebowitz. Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, 2:103–138, 1987.

[Lebowitz, 1988] M. Lebowitz. Deferred Commitment in UNIMEM: Waiting to Learn. In *Proceedings of the Fifth Machine Learning Conference (IMAL), Ann Arbor, Michigan*, pages 80–86, 1988.

[Lenat and Guha, 1990] D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems*. Addison Wesley, Reading, Massachusetts, 1990.

[Lerman, 1970] I.C. Lerman. *Les bases de la classification automatique*. Gauthier-Vilars, Paris, 1970.

[Levesque and Brachman, 1987] H.J. Levesque and R.J. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence*, 3(2):78-93, 1987.

[Levesque, 1986a] H.J. Levesque. Knowledge Representation and Reasoning. *Annual Reviews of Computer Science*, 1:255–287, 1986.

[Levesque, 1986b] H.J. Levesque. Making Believers out of Computers. *Artificial Intelligence*, 30:81–108, 1986.

[Levinson, 1984] R.A. Levinson. A Self-Organizing Retrieval System for Graphs. In *Proceedings of AAAI'84, Austin, Texas*, pages 203–206, 1984.

[Levinson, 1985] R.A. Levinson. A Self-Organizing Retrieval System for Graphs. Technical Report AI-85-05, Artificial Intelligence Laboratory, The University of Texas at Austin, 1985.

[Lieberman, 1986] H. Lieberman. Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. In *Proceedings of the 1st OOPSLA, Portland, Oregon, ACM SIGPLAN Notices 21(11)*, pages 214–223, 1986.

[Lipkis, 1982] T. Lipkis. A KL-ONE Classifier. In J.G. Schmolze and R.J. Brachman, editors, *Proceedings of the 1981 KL-ONE Workshop, Jackson, New Hampshire*, pages 126–143, 1982. The proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

[MacGregor and Burstein, 1991] R. MacGregor and M.H. Burstein. Using a Description Classifier to Enhance Knowledge Representation. IEEE *Expert*, 6(3):41–46, 1991.

[MacGregor, 1988] R. MacGregor. A Deductive Pattern Matcher. In *Proceedings of AAAI'88, St Paul, Minnesota*, pages 403–408, 1988.

[MacGregor, 1991] R. MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 385–400. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Maes and Nardi, 1987] P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection.* North-Holland, Amsterdam, 1987.

[Maes, 1987] P. Maes. Computational Reflection. Technical Report 87_2, Artificial Intelligence Laboratory, Vrije Universiteit, Brussel, 1987.

[Marie *et al.*, 1989] S. Marie, A. Adam-Nicolle, and D. Villemin. Utilisation des représentation par objets en chimie organique. In *Actes du 7ème Congrès AFCET Reconnaissances des Formes et Intelligence Artificielle (RFIA'89), Paris*, pages 1709–1723, 1989.

[Marie, 1991] S. Marie. Contribution à l'apprentissage et à la représentation par objets de mécanismes réactionnels en Chimie Organique (Le système MAROCO). Thèse de l'Université de Caen, 1991.

[Mariño *et al.*, 1990] O. Mariño, F. Rechenmann, and P. Uvietta. Multiple Perspectives and Classification Mechanism in Object-Oriented Representation. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90), Stockholm, Sweden*, pages 425–430, 1990.

[Mariño, 1989] O. Mariño. Classification d'objets dans un modèle multi-points-de-vue. Rapport de dea, Laboratoire ARTEMIS/IMAG, Institut National Polytechnique de Grenoble, 1989.

[Marquis, 1991] P. Marquis. *Contribution à l'étude des méthodes de construction d'hypothèses en intelligence artificielle.* Thèse de l'Université de Nancy 1, 1991.

[Martin, 1977] W.A. Martin. OWL. In *Proceedings of the 5th IJCAI, Cambridge, Massachusetts*, pages 985–987, 1977.

[Martin, 1979] W.A. Martin. Descriptions and the Specialization of Concepts. In P.H. Winston and R.H. Brown, editors, *Artificial Intelligence, A MIT Perspective*, pages 378–419. MIT Press, Cambridge, Massachusetts, 1979.

[Masini *et al.*, 1989] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Les langages à objets.* InterEditions, Paris, 1989.

[McAllester and Zabih, 1986] D. McAllester and R. Zabih. Boolean Classes. In *Proceedings of the 1st OOPSLA, Portland, Oregon, ACM SIGPLAN Notices 21(11)*, pages 417–423, 1986.

[Mervis and Rosch, 1981] C.B. Mervis and E. Rosch. Categorization of Natural Objects. *Annual Review of Psychology*, 32:89–115, 1981.

[Métivier, 1987] P. Métivier. Synthèse organique assistée par ordinateur : L'approche synthétique. Thèse de l'Université Louis Pasteur, Strasbourg, 1987.

[Mettrey, 1987] W. Mettrey. An Assessment of Tools for Building Large Knowledge-Based Systems. *The AI Magazine*, 8(4):81–89, 1987.

[Meyer, 1990] B. Meyer. *Conception et programmation par objets, pour du logiciel de qualité.* InterEditions, Paris, 1990.

[Michalski and Stepp, 1983] R.S. Michalski and R.E. Stepp. Automated Construction of Classifications: Conceptual Clustering versus Numerical Taxonomy. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396–410, 1983.

[Michalski and Stepp, 1984] R.S. Michalski and R.E. Stepp. Learning From Observation: Conceptual Clustering. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach, Volume I*, pages 331–363. Springer-Verlag, Berlin, 1984.

[Michalski, 1984] R.S. Michalski. A Theory and Methodology of Inductive Learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach, Volume I*, pages 83–134. Springer-Verlag, Berlin, 1984.

[Miezitis, 1988] M.A. Miezitis. Generating Lexical Options by Matching in a Knowledge Base. Technical Report CSRI-217, Computer Systems Research Institute, University of Toronto, Toronto, 1988.

[Minsky, 1975] M. Minsky. A Framework for Representing Knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–281. MacGraw Hill, New York, 1975.

[Minsky, 1988] M. Minsky. *La société de l'esprit.* InterEditions, Paris, 1988.

[Mittal *et al.*, 1986] S. Mittal, D.G. Bobrow, and K.M. Kahn. Virtual Copies: At the Boundary Between Classes and Instances. In *Proceedings of the 1st OOPSLA, Portland, Oregon, ACM SIGPLAN Notices 21(11)*, pages 159–166, 1986.

[Moon, 1986] D. Moon. Object-Oriented Programming with Flavors. In *Proceedings of the 1st OOPSLA, Portland, Oregon, ACM SIGPLAN Notices 21(11)*, pages 1–8, 1986.

[Napoli and Masini, 1989] A. Napoli and G. Masini. Objets et représentation des connaissances. Rapport de Recherche 89-R-192, Centre de Recherche en Informatique de Nancy, 1989.

[Napoli *et al.*, 1986] A. Napoli, C. Laurenço, and M.P. Heitz. Synthèse automatique optimale en chimie organique. In *Actes des 6èmes Journeés Internationales sur les Systèmes Experts et leurs Applications, Avignon*, pages 1219–1234, 1986.

[Napoli *et al.*, 1991] A. Napoli, C. Laurenço, and R. Ducournau. Techniques de classification pour modéliser la synthèse de molécules organiques. In D. Hérin-Aime, R. Dieng, J.P. Regourd, and J.P. Angoujard, editors, *Proceedings of the First International Conference on Knowledge Modeling and Expertise Transfer (KMET'91), Sophia-Antipolis*, pages 241–252. IOS Press, Amsterdam, 1991.

[Napoli, 1990] A. Napoli. Using Frame-Based Representation Languages to Describe Chemical Objects. *New Journal of Chemistry*, 14(12):913–919, 1990.

[Nebel, 1988] B. Nebel. Computational Complexity of Terminological Reasoning in BACK. *Artificial Intelligence*, 34:371–383, 1988.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Computer Science 422. Springer-Verlag, Berlin, 1990.

[Nguyen and Rieu, 1989] G.T. Nguyen and D. Rieu. Schema Evolution in Object-Oriented Database Systems. *Data & Knowledge Engineering*, 4:43–67, 1989.

[Niwa *et al.*, 1984] K. Niwa, K. Sasaki, and H. Ihara. An Experimental Comparison of Knowledge Representation Schemes. *The AI Magazine*, 5(2):29–36, 1984.

[Nugues, 1989] P. Nugues. *Interprétation de gels d'électrophorèses bidimensionnelles.* Thèse de l'Université de Nancy 1, 1989.

[Ogden and Richards, 1923] C.K. Ogden and I.A. Richards. *The Meaning of Meaning.* Harcourt, Brace and World, New York, 1923. (8th edition, 1946).

[Østerbye, 1988] K. Østerbye. Active Objects: An Access Oriented Framework for Object-Oriented Languages. *Journal of Object-Oriented Programming*, 1(2):6–10, 1988.

[Owsnicki-Klewe, 1988] B. Owsnicki-Klewe. Configuration as a Consistency Maintenance Task. In *Proceedings of the 12th German Workshop on Artificial Intelligence (GWAI'88)*, pages 77–87, Berlin, 1988. Springer-Verlag.

[Owsnicki-Klewe. 1989] B. Owsnicki-Klewe. A General Characterization of Term Description Languages. In K.H. Bläsius, U. Hedstück, and C.R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, Lecture Notes in Computer Science 418, pages 183–189. Springer-Verlag, Berlin, 1989.

[Patel-Schneider *et al.*, 1990] P.F. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. MacGregor, W.S. Mark, D.L. McGuinness, B. Nebel, A. Schmiedel, and J. Yen. Report on the Workshop on Term Subsumption Languages in Knowledge Representation. *The AI Magazine*, 11(2):16–22, 1990.

[Patel-Schneider, 1984] P.F. Patel-Schneider. Small can be Beautiful in Knowledge Representation. Technical Report 660, Fairchild Laboratory for Artificial Intelligence Research, Palo Alto, California, 1984.

[Patel-Schneider, 1989] P.F. Patel-Schneider. A Four-Valued Semantics for Terminological Logics. *Artificial Intelligence*, pages 319–351, 1989.

[Patel-Schneider, 1991] P.F. Patel-Schneider. What's Inheritance got to do with Knowledge Representation. In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 1–11. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Peltason *et al.*, 1989] C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK System Revisited. KIT-REPORT 75, Technische Universitæt, Berlin, 1989.

[Pfertzel-Ramphft, 1989] S. Pfertzel-Ramphft. Synthèse organique assistée par ordinateur : l'approche système expert. Thèse de l'Université Louis Pasteur, Strasbourg, 1989.

[Pivot and Prokop, 1987] B. Pivot and M. Prokop. Définition et réalisation d'une fonctionnalité de classification dans Shirka. Rapport de projet, Institut National Polytechnique de Grenoble/ENSIMAG, Grenoble, 1987.

[Porter *et al.*, 1990] B.W. Porter, E.R. Bareiss, and R.C. Holte. Concept Learning and Heuristic Classification in Weak-Theory Domains. *Artificial Intelligence*, 45(1–2):229–263, 1990.

[Qrps, 1984] Qualitative Reasoning about Physical Systems. Special Issue of *Artificial Intelligence*, 24(1–3), 1984.

[Quillian, 1968] M.R. Quillian. Semantic Memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, Cambridge, Massachusetts, 1968.

[Rao, 1971] M.R. Rao. Cluster Analysis and Mathematical Programming. *Journal of the American Statistical Association*, 66:622–626, 1971.

[Read and Cesa, 1991] S.J. Read and I.L. Cesa. This Reminds Me of the Time When...: Expectation Failures in Reminding and Explanation. *Journal of Experimental and Social Psychology*, 27(1):1–25, 1991.

[Rechenmann, 1988] F. Rechenmann. *SHIRKA : système de gestion de bases de connaissances centrées-objet. Manuel de référence.* INRIA/ARTEMIS, Grenoble, 1988.

[Reed, 1972] S.K. Reed. Pattern Recognition and Categorization. *Cognitive Psychology*, 3:392–407, 1972.

[Regoczei and Hirst, 1990] S. Regoczei and G. Hirst. The meaning triangle as a tool for the acquisition of abstract, conceptual knowledge. *International Journal of Man-Machine Studies*, 33(5):505–520, 1990.

[Reimer and Hahn, 1983] U. Reimer and U. Hahn. A Formal Approach to the Semantics of a Frame Data Model. In *Proceedings of the 8th IJCAI, Karlsruhe, West Germany*, pages 337–339, 1983.

[Reimer and Hahn, 1985] U. Reimer and U. Hahn. On Formal Semantic Properties of a Frame Data Model. *Computers and Artificial Intelligence*, 4(4):335–351, 1985.

[Reiter, 1987] R. Reiter. Non-Monotonic Reasoning. *Annual Reviews of Computer Science*, 2:147–186, 1987.

[Resnick *et al.*, 1990] L.A. Resnick, A. Borgida, R.J. Brachman, D.L. McGuinness, and P.F. Patel-Schneider. CLASSIC Description and Reference Manual For the Common Lisp Implementation (Version 1.0). Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1990.

[Rich, 1979] E. Rich. User Modeling via Stereotypes. *Cognitive Science*, 3:329–354, 1979.

[Riesbeck and Schank, 1989] C.K. Riesbeck and R.C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[Rips, 1989] L.J. Rips. Similarity, typicality, and categorization. In S. Vosniadou and A. Ortony, editors, *Similarity and analogical reasoning*, pages 21–59. Cambridge University Press, Cambridge, UK, 1989.

[Roberts and Goldstein, 1977] R.B. Roberts and I.P. Goldstein. The FRL Manual. AI Memo 409, AI Lab, MIT, Cambridge, Massachusetts, 1977.

[Roche and Laurent, 1989] C. Roche and J.P. Laurent. Les approches objets et le langage LRO2 (KEOPS). *Techniques et Sciences Informatiques*, 8(1):21–39, 1989.

[Rosch and Mervis, 1975] E. Rosch and C. Mervis. Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology*, 7:573–605, 1975.

[Rosch *et al.*, 1976] E. Rosch, C. Mervis, W. Gray, D. Johnson, and P. Boyes-Bream. Basic Objects in Natural Categories. *Cognitive Psychology*, 8:382–439, 1976.

[Rosch, 1977] E. Rosch. Human Categorization. In N. Warren, editor, *Studies in Cross-Cultural Psychology.* Academic Press, London, 1977.

[Rumbaugh *et al.*, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Sathi *et al.*, 1985] A. Sathi, M.S. Fox, and M. Greenberg. Representation of Activity Knowledge for Project Management. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 7(5):531–552, 1985.

[Schank, 1982] R.C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, Cambridge, UK, 1982.

[Schild, 1989] K. Schild. Towards a Theory of Frames and Rules. KIT-REPORT 76, Technische Universitæt, Berlin, Germany, 1989.

[Schmidt-Schauss, 1989] M. Schmidt-Schauss. Subsumption in KL-ONE is Undecidable. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, Toronto*, pages 421–431, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.

[Schmolze and Israel, 1983] J.G. Schmolze and D.J. Israel. KL-ONE: Semantics and Classification. In C.L. Sidner, editor, *Research in Knowledge Representation and Natural Language Understanding (BBN Technical Report 5421)*, pages 27–39. Bolt, Beranek and Newman, Cambridge, Massachusetts, 1983.

[Schmolze and Lipkis, 1983] J.G. Schmolze and T.A. Lipkis. Classification in the KL-ONE Knowledge Representation System. In *Proceedings of the 8th IJCAI, Karlsruhe, West Germany*, pages 330–332, 1983.

[Segev and Shoshani, 1987] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In *Proceedings of the ACM/SIGMOD International Conference on the Management of Data, San Francisco, California, SIGMOD RECORD, 16(3)*, pages 454–466, 1987.

[Shastri, 1989] L. Shastri. Default Reasoning in Semantic Networks: A Formalization of Recognition and Inheritance. *Artificial Intelligence*, 39(3):283–355, 1989.

[Shoham, 1987] Y. Shoham. Temporal Logics in AI: Semantical and Ontological Considerations. *Artificial Intelligence*, 33:89–104, 1987.

[Slade, 1991] S. Slade. Case-Based Reasoning: A Research Paradigm. *The AI Magazine*, 12(1):42–55, 1991.

[Smith and Medin, 1981] E.E. Smith and D. Medin. *Categories and Concepts.* Harvard University Press, Cambridge, Massachusetts, 1981.

[Smith and Smith, 1977a] J.M. Smith and D.C.P. Smith. Database Abstractions: Aggregation. *Communications of the* ACM, 2(6):405–413, 1977.

[Smith and Smith, 1977b] J.M. Smith and D.C.P. Smith. Database Abstractions: Aggregation and Generalisation. ACM *Transactions on Database Systems*, 2(2):105–133, 1977.

[Smith *et al.*, 1988] E.E. Smith, D.N. Osherson, L.J. Rips, and M. Keane. Combining Prototypes: A Selective Model. *Cognitive Science*, 12(4):485–527, 1988.

[Snyder, 1987] A. Snyder. Inheritance and the Development of Encapsulated Software Components. In B. Shriver and P. Wegner, editors, *Research Directions in Object Oriented Programming*, pages 165–188. MIT Press, Cambridge, Massachusetts, 1987.

[Snyder, 1991] A. Snyder. Inheritance in Object-Oriented Programming Languages. In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 153–171. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Sowa, 1984] J.F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison Wesley, Reading, Massachusetts, 1984.

[Stefik and Bobrow, 1986] M. Stefik and D.G. Bobrow. Object-Oriented Programming: Themes and Variations. *The AI Magazine*, 6(4):40–62, 1986.

[Stefik *et al.*, 1983] M. Stefik, D.G. Bobrow, S. Mittal, and L. Conway. Knowledge Programming in LOOPS: Report on an Experimental Course. *The AI Magazine*, 4(3):3–13, 1983.

[Stefik *et al.*, 1986] M.J. Stefik, D.G. Bobrow, and K.M. Kahn. Integrating Access-Oriented Programming into a Multi-Paradigm Environment. IEEE *Software*, pages 10–18, January 1986.

[Stefik, 1979] M.J. Stefik. An Examination of a Frame-Structured Representation System. In *Proceedings of the 6th IJCAI, Tokyo*, pages 845–852, 1979.

[Stefik, 1981a] M. Stefik. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–139, 1981.

[Stefik, 1981b] M. Stefik. Planning with Constraints (MOLGEN: Part 2). *Artificial Intelligence*, 16:141–170, 1981.

[Stein *et al.*, 1989] L.A. Stein, H. Lieberman, and D. Ungar. A Shared View of Sharing: The Treaty of Orlando. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 31–48. Addison Wesley, Reading, Massachusetts, 1989.

[Stepp and Michalski, 1986] R.E. Stepp and R.S. Michalski. Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach, Volume II*, pages 471–498. Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.

[Stickel, 1986] M.E. Stickel. An Introduction to Automated Deduction. In W. Bibel and Ph. Jorrand, editors, *Fundamentals of Artificial Intelligence*, Lecture Notes in Computer Science (232), pages 75–132. Springer-Verlag, Berlin, 1986.

[Stroustrup, 1989] B. Stroustrup. *Le langage C++*. InterEditions, Paris, 1989.

[Sun, 1989] Y. Sun. Résolution de problème dans l'espace d'états avec abstraction de concepts – le système expert OASIS. Thèse de l'Université de Nancy 1, 1989.

[Tichy, 1987] W.F. Tichy. What Can Software Engineers Learn from Artificial Intelligence? COMPUTER, 20(11):43–54, 1987.

[Trombini, 1987a] C. Trombini. Planning Organic Syntheses: I- Synthetic plans and processes. *La chimica e l'industria*, 69(5):82–86, 1987.

[Trombini, 1987b] C. Trombini. Planning Organic Syntheses: II- Analysis of target functionality. *La chimica e l'industria*, 69(6):82–86, 1987.

[Trombini, 1987c] C. Trombini. Planning Organic Syntheses: III- Analysis of target skeletons. *La chimica e l'industria*, 69(9):99–104, 1987.

[Ungar and Smith, 1991] D. Ungar and R.B. Smith. SELF: The Power of Simplicity. *LISP and Symbolic Computation*, 4(3):187–205, 1991.

[van Hentenryck, 1989] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, 1989.

[Van Marcke, 1986] K. Van Marcke. A Parallel Algorithm for Consistency Maintenance in Knowledge Representation. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI'86), Brighton, England*, pages 278–290, 1986.

[Van Marcke, 1988] K. Van Marcke. The Use and Implementation of the Representation Language KRS. Technical Report 88-2, Vrije Universiteit, Brussel, 1988.

[Veitch, 1991] J. Veitch. Frames in CLOS. *AI Expert*, pages 41–47, June 1991.

[Vignard, 1985] P. Vignard. *Un mécanisme d'exploitation à base de filtrage flou pour une représentation des connaissances centrée objets.* Thèse de 3ème cycle, Institut National Polytechnique de Grenoble, 1985.

[Vilain, 1985] M. Vilain. The Restricted Language Architecture of a Hybrid Representation System. In *Proceedings of the 9th IJCAI, Los Angeles, California*, pages 547–551, 1985.

[von Luck *et al.*, 1987] K. von Luck, B. Nebel, C. Peltason, and A. Schmiedel. The Anatomy of the BACK System. KIT-REPORT 41, Technische Universitæt, Berlin, 1987.

[Wand and Friedman, 1987] M. Wand and D.P. Friedman. The Mystery of the Tower Revealed: A Non-Reflective Description of the Reflective Tower. In P. Maes and D. Nardi, editors, *Meta-Level Architectures and Reflection*, pages 111–134. North-Holland, Amsterdam, 1987.

[Wilcox and Levinson, 1986] C.S. Wilcox and R.A. Levinson. A Self-Organized Knowledge Base for Recall, Design, and Discovery in Organic Chemistry. In T.H. Pierce and B.A. Hohne, editors, *Artificial Intelligence Applications in Chemistry*, pages 209–230. ACS Symposium Series 306, New-York, 1986.

[Winograd, 1975] T. Winograd. Frame Representation and the Declarative/Procedural Controversy. In D.G. Bobrow and A. Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 185–210. Academic Press, New York, 1975.

[Winograd, 1978] T. Winograd. On primitives, prototypes, and other semantic anomalies. In D.L. Waltz, editor, *Theoretical Issues in Natural Language Processing-2, University of Illinois at Urbana-Champaign*, pages 25–32, 1978.

[Winston *et al.*, 1987] M.E. Winston, R. Chaffin, and D. Herrmann. A Taxonomy of Part-Whole Relations. *Cognitive Science*, 11:417–444, 1987.

[Winter, 1984] J.H. Winter. Classification of Synthetic Pathways in Organic Chemistry. *Journal of Chemical Information and Computer Sciences*, 24:263–265, 1984.

[Winter, 1985] J.H. Winter. Planning of Synthetic Pathways on the Basis of Synthesis Strategies. *Journal of Chemical Information and Computer Sciences*, 25:389–391, 1985.

[Wipke and Dolata, 1986] W.T. Wipke and D.P. Dolata. A Multivalued Logic Predicate Calculus Approach to Synthesis Planning. In T.H. Pierce and B.A. Hohne, editors, *Artificial Intelligence Applications in Chemistry*, pages 188–208. ACS Symposium Series 306, New York, 1986.

[Wipke and Rogers, 1984] W.T. Wipke and D. Rogers. Artificial Intelligence in Organic Synthesis. SST: Starting Material Selection Strategies. An Application of Superstructure Search. *Journal of Chemical Information and Computer Sciences*, 24:71–81, 1984.

[Woods, 1983] W.A. Woods. What's Important About Knowledge Representation? COMPUTER, 16(10):22–27, 1983.

[Woods, 1991] W.A. Woods. Understanding Subsumption and Taxonomy: A Framework for Progress. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 45–94. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Yen *et al.*, 1991a] J. Yen, H.-L. Juang, and R. MacGregor. Using Polymorphism to Improve Expert System Maintainability. IEEE *Expert*, 6(2):48–55, April 1991.

[Yen *et al.*, 1991b] J. Yen, R. Neches, and R. MacGregor. CLASP: Integrating Term Subsumption Systems and Production Systems. IEEE *Transactions on Knowledge and Data Engineering*, 3(1):25–32, 1991.

UNIVERSITE DE NANCY I

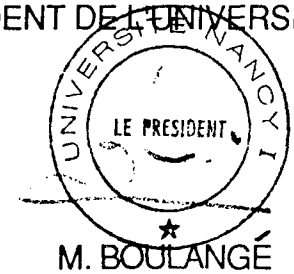
NOM DE L'ETUDIANT : Monsieur NAPOLI Amadeo

NATURE DE LA THESE : DOCTORAT D'ETAT ES SCIENCES MATHEMATIQUES
Spécialité Informatique

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 21 JAN. 1992 n°27

LE PRESIDENT DE L'UNIVERSITE DE NANCY I



M. BOULANGÉ

Résumé

Cette thèse aborde les notions de catégorisation et de raisonnement par classification dans le contexte de la représentation de connaissances. La catégorisation et le raisonnement par classification y jouent des rôles complémentaires : la tâche de la première consiste à organiser des connaissances en hiérarchies sur lesquelles opère le second. Le raisonnement par classification consiste à reconnaître un objet en appariant ses caractéristiques avec celles de catégories connues d'objets, afin de découvrir la catégorie à laquelle l'objet pourrait se rattacher. Il joue un rôle primordial dans la recherche, la gestion et la maintenance d'informations organisées en catégories hiérarchisées, ainsi que comme méthode de résolution de problèmes. Les représentations à objets sont fondées sur ces principes. Elles font partie, avec les systèmes à subsomption, des formalismes de représentation à base d'objets dans lesquels le raisonnement par classification tient une place fondamentale. La partie expérimentale de cette thèse décrit la construction d'un système de conception de plans de synthèse de molécules organiques. Dans le modèle présenté, une synthèse est considérée comme un système temporel qui évolue d'un état initial, la molécule à fabriquer, vers un état final, un ensemble de molécules facilement accessibles, en passant par une suite d'états intermédiaires. Le système est implanté sous la forme d'une représentation à objets, où chaque état est décrit par un objet temporel. Le raisonnement employé pour résoudre les problèmes de synthèse est le raisonnement par classification, qui opère simultanément sur des hiérarchies croisées qui reflètent des ordres partiels différents, donc une organisation en catégories orthogonales.

Mots clés : représentation de connaissances, raisonnement, classification, catégorisation, représentation à objets, subsomption, résolution de problèmes, synthèse organique, planification de synthèse.