Mise en oeuvre de l'heritage au moyen de relations

Mireille Fornarino

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1417

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

MISE EN OEUVRE DE L'HERITAGE AU MOYEN DE RELATIONS

Mireille FORNARINO

Avril 1991

RR-1417

Mise en œuvre de l'héritage au moyen de relations
Implementation of inheritance by means of relations

Mireille Fornarino

Résumé

L'héritage est un mécanisme de partage ou de factorisation des informations dans les langages à objets. Selon les approches (génie logiciel ou intelligence artificielle), il permet de n'exprimer que la spécialisation, ou également d'autres sortes de relations comme l'instanciation ou la composition. Dans le premier cas, son utilisation est limitée mais sa sémantique précise, dans le second, il est un bon outil de partage, mais il n'a pas de signification très claire.

Notre propos dans ce rapport est une modélisation de ce mécanisme, habituellement figé, en terme de propagation d'informations le long de liens d'héritage. Chaque transfert d'information entre deux objets est réalisé au moyen d'objets liens qui permettent un filtrage des informations émises, une vérification et un maintien de la cohérence en fonction de la relation à modéliser. L'héritage procède alors par envoi de message aux différents protagonistes. Chaque objet présente à la fois une interface fonctionnelle et une interface relationnelle. L'encapsulation des données est ainsi mieux préservée.

Dans le système résultant, l'utilisateur peut définir des familles de relations d'héritage spécifiques à la connaissance qu'il doit modéliser, tout en leur associant une sémantique précise. Ainsi le système résultant est à la fois plus expressif et plus contraint.

Mots clés

Héritage, relations, liens, visibilité du graphe d'héritage, représentation des connaissances, langages à objets.

Abstract

The inheritance mechanism is a way to share and factorize information in object oriented languages. Depending on application domains, it is used to express strict specialization or other kinds of relations as instanciation or composition. Its semantics is then more or less clearly established and its use is more or less strictly.

Our purpose in this report is a modelling of this mechanism usually predefined, in term of information propagation along inheritance links. Each transfer between two given objects is realized by means of link objects that filter the emitted information and check and manage consistency with respect to the relation to model. Inheritance acts then by message sending to the different protagonists; each object has both a functional and a relational interface. So the encapsulation principle is better preserved.

In the resulting system, the user can define several sorts of inheritance relations, specific to the knowledge to be modeled, and in the same time he can specify their semantics. So the resulting system is both more expressive and more constrained.

Keywords

Inheritance, relations, links, visibility of inheritance graph, knowledge representation, object oriented languages.

Table des matières

1 Introduction **2**

2 Différents traits de l'héritage **4**
2.1 Moment de l'héritage 4
2.2 Visibilité du graphe d'héritage 5
2.3 Décomposition et granularité du mécanisme d'héritage 8
2.4 Informations structurelles et de valeurs 11
2.5 Conclusion 12

3 Mise en œuvre de l'héritage en terme de propagation d'informations **13**
3.1 Concept de lien et héritage 13
3.1.1 Détermination des propriétés 14
3.1.2 Etablissement d'une relation d'héritage : vérifications préliminaires . 14
3.1.3 Propagation des informations 15
3.1.4 Parcours des liens d'héritage 17
3.1.5 Maintien de la cohérence d'une relation d'héritage 20
3.2 Implantation au moyen de classes de liens 22
3.2.1 Classes *Métalien* et *Lien* 22
3.2.2 Classe *lien-heritage* 23
3.3 Conclusion 24

4 Différents liens d'héritage **25**
4.1 Lien de spécialisation 25
4.1.1 Généralisation : lien dual de la spécialisation 28
4.2 Liens de composition 28
4.3 Dérivation 30
4.4 Instanciation 30
4.5 Conclusion 31

5 Conclusion **32**

Chapitre 1

Introduction

"Inheritance is the concept in object languages that is used to define objects that are almost like other objects. Mechanisms like this are important because they make it possible to declare that certain specifications are shared by multiple parts of a program... The concepts of inheritance arise in all object languages, whether they are based on specialization or delegation and copying." [SB86]

L'héritage est un mécanisme de partage et/ou de factorisation de l'information dans les langages à objets. Elément essentiel pour la modularité, la réutilisabilité et le développement incrémental d'applications, il suscite plusieurs controverses quant à l'interprétation qui doit lui être attribuée et les restrictions devant assujettir son exploitation. Selon que les langages s'inscrivent dans le cadre du génie logiciel, de la représentation des connaissances ou comme outils de programmation, ils offrent un mécanisme d'héritage de sémantique et de fonctionnalités différentes [HK89].

En effet, souvent considéré en représentation des connaissances comme simple mécanisme de partage, il est alors utilisé pour exprimer aussi bien le sous-typage, la spécialisation-généralisation que la composition ou la dérivation. Dans ce cas, il n'y a pas de règles strictes sur la constitution des objets mis en relation d'où une grande capacité de réutilisation des objets existants, mais une compréhension plus difficile du code résultant.

A l'inverse, dans les langages de programmation plus orientés génie logiciel, il est souvent assimilé à l'expression du sous-typage et dans ce cas les objets liés par héritage doivent être cohérents les uns vis-à-vis des autres (contraintes sur les propriétés). Le mécanisme n'est pas alors assez souple pour permettre la représentation des différentes relations qui impliquent un partage d'information.

Aussi, afin d'accroître ces paradigmes et ainsi d'augmenter la puissance d'abstraction des langages ou leurs domaines d'application, de nombreux travaux tendent à donner une définition la plus stricte ou la plus laxiste possible de ce mécanisme[Pat89]. C'est ainsi que sont discutés aujourd'hui délégation, héritage ou sélection explicite des classes [Ste87,Lie86a, CG90], points de vues [Rec,Dug89], mixins [BC90]... Il sous-tend à tous ces travaux un fait commun, une propagation d'information, à l'envoi de message ou à la création, contrôlée ou non, totale ou partielle, de la classe vers la sous-classe ou inversement, etc.

Notre proposition est donc de modéliser ce transfert au moyen de relations entre les objets. L'exploitation de ces relations par les différents mécanismes d'héritage permet d'offrir à la fois des relations d'héritage strict au sens par exemple du sous-typage, mais aussi d'exprimer

des relations de parties où seules certaines propriétés sont propagées ou de stipuler des exceptions par refus de transmission. L'abstraction et la structuration des connaissances sont ainsi favorisées puisque la sémantique des différentes relations peut être précisée. Liens de généralisation, partie, déduction ne sont plus confondus, d'où une réutilisation plus efficace et une réelle incrémentalité dans le développement.

D'autre part, l'expression des relations telle que nous la proposons est réalisée au moyen d'objets liens. L'accès aux objets mis en relation passe alors nécessairement par envoi de message, d'où une encapsulation des informations, puisque chaque objet est maître de son comportement vis-à-vis de l'héritage. Les objets présentent ainsi deux interfaces. Une interface fonctionnelle qui correspond à l'interface traditionnelle des langages à objets et une interface relationnelle qui spécifie le comportement de l'objet vis-à-vis des relations pouvant le lier.

Ce rapport présente donc dans un premier temps le mécanisme d'héritage comme une propagation d'informations, le long d'objets relations nommés liens. Des relations d'héritage plus précises telles que la spécialisation ou la composition sont ensuite modélisées dans ce formalisme. Un langage, Othelo, plus particulièrement destiné à la représentation des connaissances, est le résultat de ce travail sur la représentation conjointe objet et relations et il sert de base pour présenter l'implantation du mécanisme d'héritage.

Chapitre 2

Différents traits de l'héritage

L'héritage est le procédé de partage ou de factorisation de l'information dans les langages à objets. C'est un mécanisme que l'on trouve aussi bien dans les langages de programmation qu'en intelligence artificielle. Il permet de définir des objets à partir d'autres. En général, il consiste à inférer des propriétés d'objets plus spécifiques à partir de celles d'objets plus généraux. On parle alors d'une relation de spécialisation, même si toutes les règles inhérentes à une telle relation ne sont pas toujours vérifiées. Ainsi plusieurs travaux soulignent le manque de sémantique associée à ce mécanisme tout particulièrement en représentation des connaissances [Bra83,Pat89,DH89].

Dans ce rapport, nous présentons l'héritage comme un mécanisme de propagation des informations le long des liens d'héritage qui unissent les objets entre eux. Aussi nos propos ne se limitent-ils pas à l'expression de la spécialisation, mais englobent d'autres relations d'héritage comme la composition et l'instanciation.

Notre approche de l'héritage a été motivée par un certain nombre de constatations quant aux différents aspects de l'héritage dans la littérature. Afin de clarifier notre travail, présentons brièvement ces faits et nos conclusions.

2.1 Moment de l'héritage

> *"Il y a de nombreux points de vue sur l'héritage... Il existe deux points communs dans cette diversité. Le premier statique, est la relation de généralisation-spécialisation qui lie l'objet qui hérite à ceux dont il hérite. Le second dynamique, est le mécanisme - le processus- par lequel l'héritage se fait, au moment où il se fait." [DH89]*

Il faut distinguer lorsque l'on parle d'héritage, l'établissement d'une relation entre deux objets, et la propagation des informations par cette relation. Cette propagation peut se situer à des moments très différents, comme la création d'une classe ou d'une instance (Art [Cla85], TELOS [DGN89]), l'envoi de message (Flavors [Moo86]), la compilation (C++[Str86]), le premier accès à un objet[Lie86a],...

Nous parlons d'héritage statique, lorsqu'il y a "assimilation" des propriétés héritées. Un futur accès à celles-ci n'implique alors pas de reparcourir le graphe d'héritage.

Il y a héritage dynamique lorsque la propagation d'information n'est pas "assimilée" par l'objet qui hérite. Dans ce cas, l'accès ultérieur à une propriété héritée impliquera de reparcourir le graphe d'héritage.

L'héritage statique favorise les temps d'accès et l'héritage dynamique la place mémoire et la modifiabilité.

La distinction entre classes et instances, entre spécialisation et instanciation n'est pas indispensable à l'étude du mécanisme d'héritage. Aussi, dans un premier temps, ferons-nous abstraction des différences inhérentes à ces deux sortes d'objets et de relations.

> $\rightsquigarrow$ Notre propos dans ce rapport est une étude du mécanisme de propagation de l'information, qu'il soit définif (héritage statique) ou non (héritage dynamique). En effet, quel que soit le moment de l'héritage, il s'agit de propager des informations d'un objet vers un autre, et c'est plus précisément à ce mécanisme que nous nous sommes intéressées dans ce rapport.

† Le langage Othelo qui servira de base à la présentation de l'implantation propose un héritage statique élaboré à la déclaration des relations d'héritage entre objets.

2.2 Visibilité du graphe d'héritage

L'héritage consiste en une propagation des informations. Lorsqu'il est simple, c'est à dire qu'un nœud du graphe peut avoir au plus un père, il ne peut pas y avoir de conflit. En effet, le graphe d'héritage est alors un arbre, et l'ensemble des ascendants de tout nœud (hiérarchie de ce nœud [DH89]) dans ce graphe est totalement ordonné. L'héritage consiste alors à rechercher dans cette hiérarchie du plus spécialisé au plus général, le premier ascendant qui possède la propriété désirée. Ainsi la définition d'une propriété dans un objet peut masquer la définition de cette propriété dans ses ascendants.

Selon la vision que l'on a de l'arbre d'héritage, le parcours peut être interprété différemment.

Soit l'arbre de d'héritage suivant :
$A \rightarrow B \rightarrow C \rightarrow F \rightarrow L$ où $\rightarrow$ signifie *hérite-de*. Nous noterons $H(A, B)$ la relation d'héritage entre A et son ascendant B, qui retourne l'objet A après héritage de B. Nous noterons X_{init} l'objet X restreint à l'ensemble des propriétés qu'il a déclarées.

Un objet a une vision restreinte de son graphe d'héritage lorsqu'il ne "voit" que l'interface de ses ascendants directs. Ainsi, en héritage simple, la relation d'héritage entre A et B peut être interprétée comme suit :
$H(A, B) = H(A_{init}, H(B_{init}, H(C_{init}, H(F_{init}, L_{init}))))$.

L'objet A hérite donc de l'objet B après que celui-ci a hérité de C et ainsi de suite. Cette première approche correspond à une délégation. L'héritage par l'objet A de l'objet B équivaut à réaliser l'héritage par A des propriétés "connues" de B (soit par déclaration soit par héritage).

Avec les algorithmes de linéarisation du graphe, la vision qu'un objet a du graphe est le plus souvent globale, ce que nous interprétons ainsi:
$H(A, B) = H(H(H(H(A_{init}, B_{init}), C_{init}), F_{init}), L_{init})$.
L'héritage par l'objet A de l'objet B équivaut à réaliser l'héritage par A des propriétés déclarées par B, puis des propriétés déclarées par C et ainsi de suite.

Dans cette dernière approche, il est donc indispensable de distinguer les propriétés déclarées par un objet des propriétés dont il a pu hériter de ses ancêtres, ce qui n'est pas le cas, lorsque la vision du graphe d'héritage est restreinte.

Cette remarque est d'autant plus importante lorsque l'on introduit l'héritage multiple. Un objet peut alors avoir plusieurs ascendants directs. Il s'agit de manipuler un graphe d'héritage,

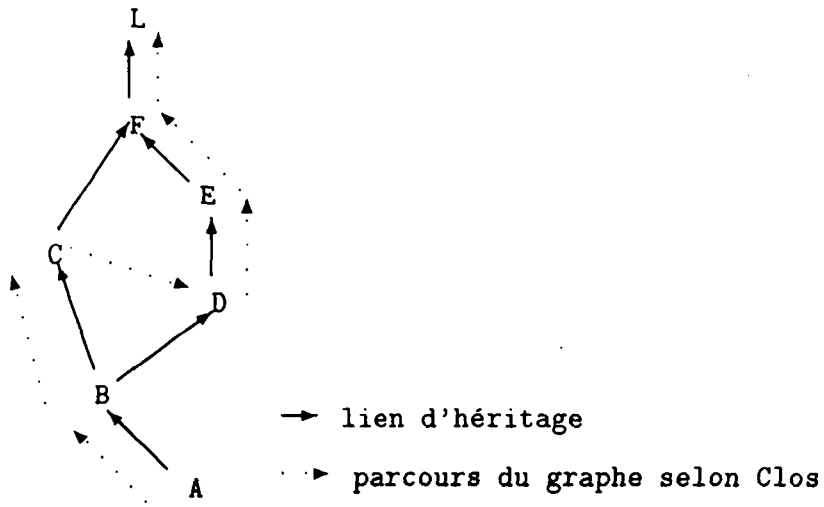
pour lequel il n'existe pas de façon inhérente un ordre total de parcours, d'où l'apparition d'ensemble de conflits. Une propriété de nom P peut alors être héritable pour un objet A de deux ascendants dont aucun ne masque l'autre pour P.

Une résolution du conflit pour une propriété donnée peut consister, en fonction de la sémantique de la propriété, à combiner les valeurs conflictuelles ou à vérifier l'égalité des valeurs (Smalltalk [GR82]). Mais le plus souvent, ne serait-ce que pour des raisons de performances, un parcours du graphe d'héritage est déterminé, et le conflit est levé au profit du "premier" rencontré selon cet ordre. La difficulté réside alors dans la recherche du parcours idéal.

Les différents parcours existants (DFS, DFO, BDS, .. [DH89]) prennent le plus souvent en compte l'ordre dans lequel les ascendants sont associés à un objet. Cette priorité entre ascendants est locale à l'objet et Ducournau parle alors de la *multiplicité locale* à l'objet.

Elle correspond en général à l'ordre de déclaration des parents : Si x est fils de a et de b, alors les propriétés de a sont héritées par x avant celles de b. En cas de conflit, elles l'emportent sur celles de b. Dans un langage tel que Sina/st [AT88], la priorité est précisée selon les méthodes héritées. La multiplicité locale n'est alors plus une relation d'ordre totale entre ascendants, mais entre propriétés des ascendants, d'où une granularité plus fine de l'héritage.

Etendons notre arbre d'héritage à un graphe :
$A \rightarrow B \rightarrow C \rightarrow F \rightarrow L$ et $B \rightarrow D \rightarrow E \rightarrow F$



Si notre objet B a une vision restreinte du graphe, il est possible de distinguer l'héritage de C par B de celui de D. L'objet B "voit" l'ascendant C puis l'ascendant D.
L'héritage de C par B consiste en de l'héritage simple.
$H(B,C) = H(B_{init}, C) = H(B_{init}, H(C_{init}, H(F_{init}, L_{init})))$.

L'héritage peut alors être réalisé entre B et D :
$H(B,D) = H(H(B_{init}, C), H(D_{init}, H(E_{init}, H(F_{init}, L_{init}))))$.

Dans ce cas, nous constatons que l'objet B hérite deux fois de l'objet F.
Toute information héritée d'un ascendant qui n'est pas direct est "filtrée", par le parent via

lequel on en a hérité. Cette approche garantit le principe d'encapsulation. En effet, chaque objet reste ainsi maître de son interface. L'utilisateur d'un objet pré-existant n'a pas besoin d'en connaître le graphe d'héritage pour l'appréhender. De plus, il devient possible de combiner au niveau d'un objet plusieurs valeurs conflictuelles et de transmettre la valeur résultante aux descendants. Le traitement de l'héritage multiple est alors géré par l'objet récepteur qui sait comment combiner les informations qu'il reçoit. C'est donc à lui de se charger des problèmes de redondance d'informations et de cohérence, tels que l'héritage des propriétés de F à la fois par C et D. Ce type de combinaison est plus difficile lorsque les objets ont une vision globale de leur graphe d'héritage.

Nous ne développerons pas ici les divers parcours existants mais davantage la différence existant entre une vision restreinte ou étendue du graphe d'héritage. Aussi, pour l'exprimer, travaillerons-nous sur le parcours de Clos. Une vision globale du graphe n'exclut cependant pas un parcours en profondeur tel que celui induit par une vision restreinte du graphe.

Le parcours du graphe d'héritage d'un objet en Clos est une extension linéaire de son graphe. Dans notre exemple l'extension linéaire associée à B selon cet algorithme est {C , D, E, F, L}. Si la vision que notre objet B a du graphe est globale, nous modéliserons l'héritage de D par B, selon le parcours de Clos, comme suit :
$H(B,D) = H(H(H(H(H(B_{init}, C_{init}), D_{init}), E_{init}), F_{init}), L_{init})$.

Le lien qui unit C et F n'est alors jamais parcouru, tandis que tout se passe comme si un lien unissait C et D. Par exemple, si l'on utilise la primitive *call-next-method* de Clos dans une méthode redéfinie par C, le concepteur de cette classe s'attend à exécuter la méthode définie par F. Or, l'appel à call-next-method du point de vue B impliquera un appel à la méthode redéfinie par D, d'où une erreur éventuelle [Sny86]. Ainsi lorsqu'un objet a une vision globale de son graphe d'héritage, il ne peut pas agir réellement comme un filtre vis-à-vis des informations qu'il reçoit. Il peut juste les masquer. Par contre, les problèmes de redondances dans l'héritage sont ici annulés.

Lorsqu'un objet a une vision globale de son graphe d'héritage, il est nécessaire de distinguer au niveau de son interface les connaissances qu'il a effectivement définies de celles qui sont héritées. Sans cette distinction, la compréhension de l'héritage au niveau des sous-classes est particulièrement difficile.

Soit $\mathcal{H}$ la fonction qui, pour un nœud A avant héritage (noté A_{init}), retourne le nœud A (noté A_{fin}) après héritage.

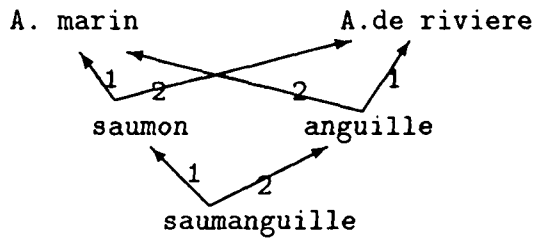
Lorsque la vision d'un objet sur sa hiérarchie est restreinte, cette fonction correspond à :
$\mathcal{H}(A) = A_{fin} = H_n(...H_2(H_1(A_{init}, X1_{fin}), X2_{fin})...Xn_{fin})$,
$\forall Xi$ ascendants directs de A, où $\forall i \leq j, Xi$ a une plus grande priorité que Xj selon la multiplicité locale à A.

Lorsque la vision d'un objet sur sa hiérarchie est globale, si le parcours du graphe est celui de Clos, cette fonction correspond alors:
$\mathcal{H}(A) = A_{fin} = H_n(H_2(H_1(A_{init}, X1_{init})...Xn_{init})$,
$\forall Xi$ élement de l'extension linéaire associée à A, où $\forall i \leq j, Xi$ a une plus grande priorité que Xj selon l'extension linéaire de la hiérarchie pour A.

Selon la vision que l'on a du graphe, la propagation statique des informations ("assimilation des propriétés héritables par l'objet récepteur") est plus ou moins incrémentale.

En effet, dans le cas d'une vision restreinte du graphe, l'ajout d'un nouvel ascendant à un nœud, qui ne pertube pas la multiplicité locale de celui-ci (noeud de plus faible priorité que les autres ancêtres), consiste alors simplement à propager les nouvelles informations et à les "assimiler" éventuellement par combinaison aux informations précédemment héritées. Un ajout équivalent, mais avec une vision étendue du graphe, implique par contre de refaire la totalité de la propagation, la liste de précédence associée au nœud étant nécessairement modifiée par cet ajout (présence d'une racine commune).

D'autre part, il faut noter que lorsque les objets ont une vision restreinte du graphe, la multiplicité locale à un nœud ne peut pas inférer sur la multiplicité locale d'un de ses descendants. Ce n'est plus le cas, lorsque les objets ont une vision globale du graphe, car, le respect de la multiplicité locale à chaque noeud, peut conduire à des contradictions internes de la multiplicité, telles que l'exemple suivant de Ducournau en témoigne : Saumanguille doit-il hériter d'abord d'A.marin ou d'A.de rivière?



Par contre, avec une vision restreinte du graphe, il risque fort d'y avoir redondance d'informations.

> ⇝ L'héritage consiste en un parcours d'un graphe qui procède à chaque traversée d'arc à une propagation d'informations. Cette propagation est un mécanisme qui met en jeu deux objets, un qui transmet de l'information, un autre qui la reçoit.
> Lorsqu'un objet a plusieurs ascendants, la multiplicité locale indique dans quel ordre parcourir les différents liens d'héritage.
> La réception d'informations par héritage dépend de l'objet récepteur, qui en fonction de la vision qu'il a du graphe résoudra les éventuels conflits.
> Nous parlerons des propriétés associées à un objet de façon générale. Selon la visibilité du graphe d'héritage, il s'agira uniquement des propriétés définies par l'objet, ou également des propriétés héritées.
> Dans toute la suite de ce rapport, nous considèrerons que la vision que l'on peut avoir du graphe est soit restreinte, soit globale mais ne peut pas différer d'un noeud à l'autre.

† En Othelo, la vision que tout objet a de sa hiérarchie est restreinte à ses ascendants directs, et la multiplicité locale d'un objet correspond à l'ordre de déclaration de ses parents.

2.3 Décomposition et granularité du mécanisme d'héritage

Dans les langages de programmation à objets, le mécanisme d'héritage est lié à la spécialisation des classes avec une connotation de sous-typage [For91,Car84]. Dans les langages destinés à la représentation des connaissances, il est également utilisé pour exprimer d'autres

relations telles que la composition (un avion hérite de son fuselage), la généralisation(l'ellipse hérite des propriétés du cercle), l'association(une pile est un ensemble d'éléments dont elle hérite), etc.

L'utilisation de l'héritage dans des situations si différentes implique une plus grande flexibilité de ce mécanisme. Il doit permettre diverses combinaisons d'informations et ne doit pas contraindre le programmeur par une mécanique trop figée, telle est du moins l'approche de langages tels que Lap[LAP89], Kool[Lac89], Kee[KEE85], ...

C'est ainsi que l'héritage des propriétés dans les langages centrés objets présente des paradigmes très différents.

- Les langages distinguent souvent plusieurs natures d'attributs [Cla85,DG87,GR82], qui présentent des comportements différents à l'héritage. Ainsi en Kee [KEE85], les own slots ne sont hérités ni par spécialisation, ni par instanciation. De même, des langages orientés génie logiciel comme C++ [Str86], Treillis/Owl [HO87], ou Sina/st [AT88] préconisent l'expression de propriétés privées aux objets qui ne sont alors pas accessibles directement dans leurs descendants (instances ou sous-classes). De telles distinctions permettent d'exprimer des familles d'attributs qui se comportent différemment à l'héritage, éventuellement en fonction de la relation d'héritage [Dug89]. Elles introduisent implicitement le refus de transmission d'informations. Ainsi selon la nature des propriétés, l'héritage propage ou non l'information.

- Certains langages, comme Kool[ALB86], Kee ou les flavors [Moo86] introduisent la possibilité de combiner les informations obtenues par héritage et ainsi de résoudre d'éventuels conflits d'héritage multiples. D'autres langages comme CommonObjects, Sina/st [AT88] permettent de refuser l'héritage de certaines propriétés et facilitent ainsi l'expression des exceptions. D'autres résolvent les problèmes de conflits par l'introduction de points de vue [CD88,CG90].
  Ainsi les cas de conflit pour l'héritage multiple d'attributs ou de méthodes sont gérés différemment selon les langages [DH89]. La réception des informations par héritage dépend toujours de l'objet qui hérite et dans certaines langages, elle peut également être fonction des propriétés.

- Selon le rôle attribué à une propriété (connaissance propre à chaque élément d'une classe, peu utilisée ou commune à un ensemble d'objet) l'héritage de cette propriété peut être réalisé par copie systématique[RL89], par cache [BS83,Lie86b] ou par référence [BDGK88,DGN89].

- La prise en compte de relations de composition implique de prendre en compte des relations d'héritage où seules certaines propriétés sont propagées comme la couleur des portières transmise à la couleur de la voiture, alors que la couleur des sièges n'intervient pas.

⇝ De ces différentes constations, nous avons conclu à un mécanisme d'héritage qui dépend non seulement de deux objets, mais également de la relation qui les lie. Au niveau de chaque objet, la dépendance est fonction des propriétés mises en jeu. Pour une même relation d'héritage, chaque propriété n'a pas le même comportement.

Ainsi il est possible de distinguer trois étapes dans la propagation : l'emission, le transfert et la réception des informations.

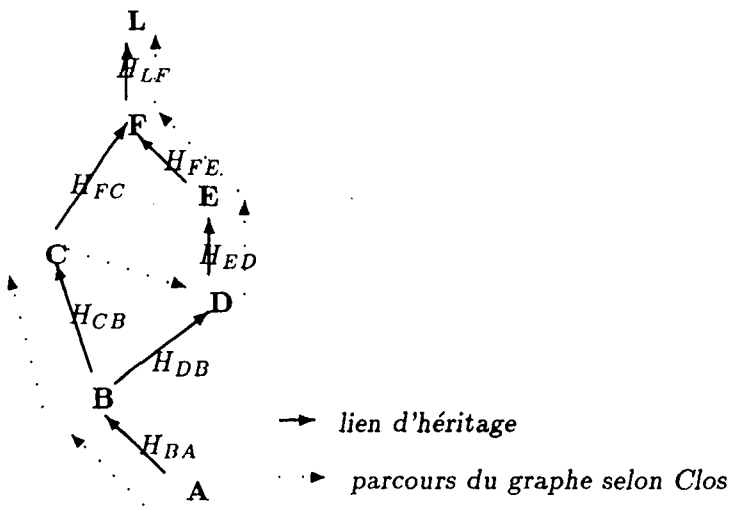
A l'émission, selon la relation d'héritage, toutes les propriétés de l'objet "émetteur" ne sont pas nécessairement concernées ou le sont différemment. Ainsi au niveau de l'objet émetteur se trouve un filtre qui favorise l'encapsulation des informations, en laissant l'objet maître de son interface relationnelle.

Selon les relations d'héritage, le transfert concerne seulement une partie ou la totalité des propriétés héritables de l'émetteur. C'est donc à ce niveau que doit être associé un filtre de transfert qui précise la sémantique de la relation.

A la réception de l'information, il existe différents comportements possibles de l'objet "récepteur" à la fois en fonction des propriétés et de la relation. Ainsi chaque objet doit pouvoir préciser les propriétés qu'ils refusent de réceptionner (exceptions) et la façon de combiner les informations reçues.

La prise en compte de différentes relations d'héritage modifie la fonction $\mathcal{H}$.

Reprenons le graphe précédent, mais en nommant les différents liens d'héritage selon le schéma suivant :



En vision restreinte du graphe par le nœud B, nous aurons :
$\mathcal{H}(B) = H_{DD}(H_{CD}(B_{init}, C), H_{ED}(D_{init}, H_{FE}(E_{init}, H_{LF}(F_{init}, L_{init}))))$.
Le lien H_{LF} est donc parcouru deux fois, $\mathcal{H}(C)$ incluant le parcours de ce lien.

Avec une vision globale du graphe par le nœud B, nous aurons :
$\mathcal{H}(B) = H_{LF}(H_{FE}(H_{ED}(H_{DD}(H_{CB}(B_{init}, C_{init}), D_{init}), E_{init}), F_{init}), L_{init})$

Le parcours du lien H_{ED} est alors réalisé entre l'objet E et l'objet B , après héritage de C_{init} et D_{init}!

Si nous émettons l'hypothèse qu'il existe une unique relation d'héritage entre deux objets données, nous pouvons généraliser la fonction presentée précédemment comme suit :

Soit H_{YX} l'unique relation d'héritage entre deux nœuds Y et X, où X hérite de Y.

Lorsque la vision d'un objet sur sa hiérarchie est restreinte, la fonction $\mathcal{H}$ correspond à :
$\mathcal{H}(A) = A_{fin} = H_{XnA}(...H_{X2A}(H_{X1A}(A_{init}, X1_{fin}), X2_{fin})...Xn_{fin})$,
$\forall Xi$ ascendants directs de A, où $\forall i \leq j, Xi$ a une plus grande priorité que Xj selon la multiplicité locale à A.

Lorsque la vision d'un objet sur sa hiérarchie est globale, il est nécessaire non seulement de déterminer l'extension linéaire associée à un nœud, mais les liens d'héritage qui lient les différents nœuds. Ainsi, dans notre exemple, pour le nœud A, la liste associée, si le parcours du graphe est celui de Clos, sera :
$(B, H_{BA}), (C, H_{CB}), (D, H_{DB}), (E, H_{ED}), (F, H_{FE}), (L, H_{LF})$.

De façon générale, lorsqu'un nœud a une vision globale du graphe d'héritage,
soit $(X1, H_1), (X2, H_2), ...(Xn, H_n)$ la liste de précédence correspondant à l'extension linéaire du graphe pour le nœud A :
$\mathcal{H}(A) = A_{fin} = H_n(H_2(H_1(A_{init}, X1_{init}), X2_{init})...Xn_{init})$.

> $\rightsquigarrow$ Dans ce travail, nous avons limité notre approche à un seul lien d'héritage entre deux objets donnés et supposé qu'il existe toujours une multiplicité locale à un objet A, même si des liens différents lient cet objet à d'autres. Il est probable qu'une priorité entre familles de relations d'héritage existe (l'instanciation est prioritaire sur la composition) mais nous n'en débattrons pas ici.

† En Othelo, il est possible de définir différents liens d'héritage (cf. 4). De plus chaque propriété, par le biais du mode d'héritage, peut particulariser son comportement à l'émission et à la réception d'information, en fonction de la relation et de l'objet récepteur.

2.4 Informations structurelles et de valeurs

La nature de l'information transmise par héritage est structurelle lorsqu'elle contribue à la définition de la structure d'un objet. On parle alors d'*héritage de structure*. Lorsqu'elle concerne uniquement la valeur des propriétés, il s'agit d'*héritage de valeur*.

Un lien de spécialisation consiste en majeure partie en de l'héritage de structure. Que ce soit lors de la définition de la sous-classe [Cla85], ou lors de la création d'une instance, l'information propagée le long de ce lien concerne la structure des propriétés héritables. Les relations structurelles entre instances dans [Dug87] ont la même définition : elles représentent une agrégation d'objets pour former un objet de complexité supérieure.

Le lien *partie-de*, proposé par Blake [BC87], propage également des informations structurelles des parties vers l'objet composite. De même, la partie de la création qui consiste à transmettre des informations structurelles d'un objet classe à son instance est assimilable à de l'héritage de structure.

L'héritage de structure est rarement paramétrable dans les langages classiques; il a toujours le même comportement : recopie ou partage des structures. En le considérant simplement comme une propagation de structures, il perd son aspect figé et devient paramétrable.

En général, un héritage de structure s'accompagne d'un héritage de valeur, qu'il s'agisse de la valeur d'un attribut ou du corps d'une méthode éventuellement par référence.

Dans ce rapport, nous présenterons plus précisément le mécanisme de propagation que nous préconisons pour représenter l'héritage.

† L'application de cette décomposition au langage Othelo nous a permis de paramétrer l'héritage dans ce langage au moyen de modes d'héritage tant de structure (copie, partage, nécessité) que de valeur(défaut, union, produit, intersection, ..). L'extensibilité du langage permet également de définir de nouveaux comportements à l'héritage [FP90b].

2.5 Conclusion

Face à l'ensemble des problèmes résumés ici et largement développés dans la littérature, notre travail est original sur plusieurs points. Dans notre modèle, le lien d'héritage ne se résume pas au lien de spécialisation. Il représente un lien d'héritage genéral qui modélise un transfert d'informations d'un objet vers un autre. Le transfert est différent selon la relation modélisée (spécialisation, composition, déduction, ..) et les objets mis en dépendance. Ainsi il est possible de préciser la sémantique des différents liens d'héritage à leur niveau même.

Un algorithme de parcours indépendant du mécanisme des liens orchestre alors la propagation de l'information sur l'ensemble du graphe d'héritage en déclenchant au moment et dans un ordre opportun les liens qui constituent le réseau d'héritage.

D'autre part, dans un souci de préservation de l'encapsulation inhérente aux objets, chaque objet offre une interface relationnelle par laquelle il précise les informations qu'il accepte d'émettre ou de recevoir et le long de quels liens. On aboutit ainsi pour un objet, à deux interfaces : une interface de messages (qui traite les messages acceptés) et une interface d'héritage ou relationnelle (qui précise le code qui peut être hérité d'autres objets).

Après cette vision générale du mécanisme d'héritage, nous allons présenter plus particulièrement de son implantation en terme d'objets, au travers du concept de lien.

Chapitre 3

Mise en œuvre de l'héritage en terme de propagation d'informations

L'étude de diverses relations de dépendance nous a conduites à dégager un mécanisme général de gestion de la cohérence de ces relations [FP90a]. A l'origine de ce travail, nous cherchions à uniformiser le mécanisme d'héritage à partir des constatations présentées au chapitre précédent : diversité des relations à représenter, comportements particuliers de certaines familles d'attributs, différentes implantations du mécanisme d'héritage et néanmoins décomposition binaire de l'héritage multiple en des transferts entre deux objets dont l'un émet et l'autre reçoit l'information, avec un algorithme de parcours de ces dépendances particulier aux différents langages.

Bien que le concept de lien soit général et ait été appliqué à de nombreuses autres formes de relations de dépendances, dans ce rapport, nous le détaillerons en l'appliquant plus précisément à l'héritage. Ce mécanisme habituellement figé est ici décomposé en transfert d'informations entre objets le long de liens binaires. Ceux-ci jouent alors à la fois un rôle de transmission, de contrôle et de maintien de la cohérence. Un objet peut recevoir de l'information par plusieurs liens d'héritage; cette présentation de l'héritage n'est donc pas limitée à l'héritage simple. L'ordre de parcours des liens est un mécanisme extérieur aux liens, qui a pour rôle d'exploiter l'expression de ces relations en les déclenchant à bon escient.

Les liens d'héritage sont eux-même des objets organisés selon une hiérarchie de spécialisation. Nous en donnerons brièvement l'implantation.

Le parcours des liens d'héritage permet de retrouver des informations dans les classes "mères" et donc de réaliser un héritage dynamique. Nous présenterons donc cette approche de l'héritage.

3.1 Concept de lien et héritage

Nous parlerons de façon générale de relations ou liens pour les relations d'héritage telles que la spécialisation ou la composition et de relations individuelles ou liens individuels pour les relations effectivement établies entre deux objets donnés (Sc spécialisation de C).

L'algorithme général de parcours des liens individuels est un mécanisme indépendant de ceux-ci. Le même lien d'héritage peut être parcouru avec une vision restreinte ou étendue du graphe, avec un parcours en profondeur ou à la Clos. Seule différencie ces approches la

connaissance propagée.

Un objet "émetteur" pour un lien d'héritage émet via ce lien des informations qui sont réceptionnées par l'objet "récepteur" du lien. Le lien agit alors comme un filtre. Nous distinguons la déclaration de la relation d'héritage, la propagation des informations et le maintien de la cohérence de la relation.

3.1.1 Détermination des propriétés

Pour certains liens, il est possible de préciser statiquement les propriétés qu'il concerne. Ainsi si nous reprenons l'exemple de Dugerdil [Dug87], la relation *a-pour-embrasure* qui peut unir un mur ou une paroi à une embrasure de porte ou de fenêtre ne propage que l'attribut *épaisseur*.

Pour les liens d'héritage classiques, c'est l'objet émetteur du lien qui sait quelles propriétés sont concernées. Ainsi, en KEE, l'héritage le long d'un lien *sort-of* ne concerne que les *member-slots* tandis que le long d'un lien *member-of* toutes les propriétés sont concernées. De même en C++, selon la relation d'héritage (publique ou non), seules les variables publiques sont visibles par la sous-classe et restent ou non publiques. Par contre des données privées à une classe peuvent être visibles via la relation d'ami. En Eiffel [Mey88], une classe précise la connaissance qu'elle accepte d'exporter.

Dans certains langages permettant d'exprimer des exceptions [Pat89,NF87] ou de refuser la réception de certaines informations d'un objet donné [AT88], l'ensemble des propriétés concernées par un lien individuel d'héritage peut être restreint par l'objet récepteur.

Selon les liens d'héritage, la liste des propriétés concernées est soit définie statiquement au niveau du lien, par exemple au moyen d'une liste de noms, soit déterminée dynamiquement par envoi de message aux objets émetteurs et/ou récepteurs.

A tout lien d'héritage est donc associée une méthode *propriétés-concernées*. Selon la nature des liens, cette méthode peut interroger par envoi de message les objets mis en relation pour constituer cette liste. La méthode, appelée au niveau de ces objets, appartient à leur interface relationnelle. Nous retrouvons donc les trois niveaux présentés au paragraphe 2.3 : émission d'informations, filtrage par la relation et réception des informations.

Lorsque la vision qu'un objet a du graphe est restreinte, l'ensemble des propriétés concernées peut inclure des propriétés héritées d'ancêtres de l'objet émetteur. C'est nécessairement le cas pour un lien de spécialisation (cf. 4).

Par contre, la notion de propriétés concernées n'est pas suffisante lorsque la vision que les objets ont du graphe est globale. En effet, pour qu'un objet puisse interdire l'héritage par ses descendants d'une propriété provenant d'un de ses ascendants, il est nécessaire de distinguer l'ensemble des propriétés concernées, qui, dans le cas d'une vision globale du graphe, ne peut représenter que des propriétés déclarées dans l'objet émetteur, et l'ensemble des propriétés non propagées qui peut englober des propriétés héritées (cf. 3.1.4).

3.1.2 Etablissement d'une relation d'héritage : vérifications préliminaires

L'établissement d'une relation d'héritage entre deux objets donnés peut en premier lieu conduire à vérifier que les objets à lier sont bien cohérents l'un vis-à-vis de l'autre au sens de la relation qui doit les unir. Dans le cas contraire, le système doit refuser d'établir la relation.

> Dans un lien de spécialisation, au sens du sous-typage, toutes les propriétés de l'objet émetteur (déclarées et héritées) doivent être héritables. Si certaines propriétés sont redéfinies par l'objet récepteur, la redéfinition doit être correcte sémantiquement [Car84.Ame87.RM90] (cf. 4.1).

Lorsque l'émetteur d'une relation d'héritage connaît statiquement l'ensemble des propriétés qu'il a déclarées et celles dont il a héritées et que l'objet récepteur de la relation d'héritage a déjà hérité statiquement d'informations d'ancêtres ayant une plus forte priorité, les vérifications sont facilitées. Par contre, la chose est plus difficile en héritage dynamique, où il faut d'une part déclencher l'héritage sur l'émetteur pour obtenir des informations sur l'ensemble des propriétés qu'il transmet et d'autre part, en cas d'héritage dynamique multiple du récepteur, rechercher l'ensemble des propriétés héritées de ses ancêtres ayant une plus forte priorité.

Lorsque l'héritage est considéré comme un simple mécanisme d'implantation pour permettre le partage d'informations, aucune vérification n'est opérée. C'est le cas par exemple pour des liens de composition qui permettent un simple partage d'information.

Il n'est pas toujours facile de séparer les étapes de vérification et de propagation, en effet la complexité des calculs à réaliser amène souvent à confondre ces deux étapes. Par exemple, certains modes d'héritage ont des implications sur le type des propriétés héritables (union de listes, intersection d'intervalles, borne inférieure d'entiers, ...). La vérification préliminaire consiste alors à s'assurer que les propriétés héritables sont bien typées en fonction du mode des propriétés à la reception. Lorsque l'utilisateur peut définir ses propres modes d'héritage[FP90b], la vérification est encore plus complexe.

Lorsqu'il y a héritage statique, l'établissement d'une relation d'héritage peut s'accompagner immédiatement d'une propagation des informations, qui pourra modifier l'objet récepteur pour le rendre cohérent vis-à-vis de la relation nouvellement établie.

L'établissement d'une relation induit également un maintien de la cohérence en cas de modification d'un des objets.

3.1.3 Propagation des informations

Lorsque la relation a été établie, les propriétés concernées par celle-ci peuvent alors être propagées.

A un moment donné, la propagation des informations, le long d'un lien donné concerne une propriété précise.

Othelo est une couche objet au dessus de Prolog aussi les méthodes sont-elles rédigées dans une syntaxe à la Prolog.

Soit la méthode *heritage-propriété* associée à un lien, et ainsi définie par défaut :

```
heritage-propriété(Lien-individuel,Propriété,Information) :-
    send(Lien-individuel,propriétés-concernées(PC)),
    element(Propriété,PC),
    send(Lien-individuel,valeur(objet-influant,OE)),
    send(Lien-individuel,valeur(objet-dependant,OR)),
    send(OE,heritable(Lien-individuel,OR,Propriété,Info),
    send(OR,heritee(Lien-individuel,OE,Propriété,Info,Information)).
```

Lorsque l'héritage est statique ou définitif, la méthode *héritée* associée à l'objet récepteur de la relation "assimile" l'information reçue. Un accès ultérieur à la propriété n'impliquera pas un reparcours du graphe. La propagation est donc censée être déclenchée à un instant "logique" vis-à-vis de l'algorithme de parcours, et, en cas d'héritage statique, sur un objet récepteur n'ayant connaissance que des propriétés qu'il a déclarées ou dont il a hérité d'ancêtres ayant une plus forte priorité que celui qui est émetteur de la relation traitée.

Lorsque l'héritage est statique, l'ensemble des propriétés concernées par une relation donnée doit être propagé. Cette propagation, qui est déclenchée en Othelo lors de l'établissement de la relation, est réalisée comme suit.

heritage(Lien,OE,OR) :-
send(Lien,propriétés-concernées(PC)),
Pour toute propriété P de PC
si send(OE, heritable(Lien,OR,P,Info))
alors send(OR, heritee(Lien,OE,P,Info,Information)).

Cette décomposition permet de filtrer à l'émission l'information contenue dans un objet, en fonction de l'objet dépendant et de la relation qui les unit. Le concepteur d'un objet peut ainsi cacher ou transmettre certaines informations à certains objets selon certains liens. En général, seule la relation intervient dans la détermination des propriétés à transmettre, l'objet émetteur n'ayant pas a priori connaissance de ses descendants. Ce filtrage à l'émission permet la définition de propriétés locales ou privées aux objets et est bien adaptée à l'encapsulation et à l'abstraction de données (types privés en Ada[Ref83], propriétés private et classes amies en C++).

Un filtre peut également être posé à la réception; il agit alors comme une protection lorsque l'objet récepteur refuse des informations d'une catégorie d'émetteurs le long de liens non prévus pour se protéger d'un flux d'informations qu'il considère inadaptées (exceptions, multiplicité locale différente selon les propriétés) [AT88]. Le filtre à la réception peut également servir à combiner les informations déjà connues par le récepteur (soit par déclaration, soit par héritage).

Les méthodes *héritable*, *héritée* appartiennent à l'interface relationnelle de l'objet.

En Othelo, elles font à leur tour appel, après détermination de la propriété effectivement concernée, aux méthodes de même nom définies au niveau des propriétés. Ainsi la granularité de l'héritage est plus fine.

En Othelo, la détermination de la propriété concernée en réception est réalisée à partir de son nom.

Lorsque la propriété existe, l'héritage est alors directement réalisé par elle, qui combine les informations transmises à celles qu'elle a déjà (soit par déclaration, soit par héritage).

Si aucune propriété déjà connue de l'objet récepteur n'est concernée, il y a ajout de la propriété en fonction des informations transmises par héritage. Cet ajout peut consister à créer une nouvelle propriété ou simplement à mémoriser une référence. Mais après l'héritage, la propriété sera connue de l'objet récepteur.

C'est au niveau de cette réception que doit être envisagée la prise en compte des points de vues tels que les propose Dugerdil [Dug89]. En fonction de la provenance et du contenu de l'information, l'équivalence sémantique entre la propriété à l'émission et celle à la réception peut être établie et une nouvelle propriété selon un nouveau point de vue peut être créée.

Format Othelo :
(Nom, Référence, Valeur, Liste-Informations-descripteurs).
avec :
- Référence : la référence de la propriété dans l'objet émetteur. Il s'agit dans ce cas d'un objet propriété. Cette information est exploitée dans les cas de mode d'héritage à l'émission tels que partage ou nécessité [FP90b].
- Valeur : la valeur héritable de la propriété. Un transfert d'informations structurelles concernant une propriété implique naturellement la propagation de sa valeur. Comme pour la structure, la propriété est maître de la valeur émise.
- Liste-information-descripteurs : les informations structurelles des propriétés (descripteurs et démons) de la propriété considérée. Cette liste est constituée d'éléments correspondant à des propriétés de la propriété, définis selon le même format.

Figure 3.1: Format des informations propagées en Othelo par héritage de structure

Quelles informations sont transmises ? *Info* peut porter aussi bien des informations structurelles que de valeurs. Quel que soit le schéma choisi pour représenter les informations transmises par une propriété, celui-ci doit être compréhensible par l'objet récepteur. Lorsqu'il s'agit d'un héritage de valeur, l'information transmise peut ne consister qu'en la valeur héritable de la propriété, qui peut être différente de la valeur (ou valeur par défaut) de la propriété dans l'objet émetteur (élimination de la précision pour un réel, masquage de valeur aux sous-classes,...)[Dug86].

> En Kool, lorsque le mode d'héritage associé à un attribut est *NO*, la valeur par défaut de l'attribut ne sera "héritée " que par les instances directes de la classe et non par les instances indirectes.

Lorsqu'il s'agit d'informations structurelles, les schémas peuvent être plus complexes, surtout si l'on veut, comme en Othelo, proposer le partage de structure, la copie systématique, ou la copie lors du premier accès. On est alors obligé d'établir un formalisme compréhensible par l'ensemble du système.

La figure 3.1 présente succintement le format choisi pour l'héritage de structure en Othelo.

3.1.4 Parcours des liens d'héritage

Lorsque l'héritage est dynamique, le parcours des liens est déclenché soit à la lecture d'un attribut, soit à l'envoi de message et l'on peut considérer les propriétés comme partagées. Il s'agit alors de parcourir les liens d'héritage afin de propager les informations nécessaires le long de ceux-ci. Par exemple, lors d'envoi de message, si nous ne trouvons pas localement la méthode recherchée, nous déclenchons un parcours inverse des liens d'héritage qui concernent dynamiquement cette méthode. L'information recherchée dans le graphe d'héritage dépend des différentes implantations. En général dans le cas d'un attribut, il s'agit de sa valeur et dans celui d'une méthode de la fonction ou du prédicat à appeler. L'héritage dynamique implique donc un parcours du graphe pour une propriété donnée. Nous modélisons, en Othelo, cette recherche par le prédicat *rechercher* qui parcourt le graphe d'héritage à la recherche

d'informations sur une propriété de nom donnée. Dans le cas d'une méthode, le nom est son sélecteur.

L'ordre de parcours des liens d'héritage est imposé par un mécanisme annexe à celui des liens. Selon que les objets ont une vision globale ou restreinte du graphe d'héritage, nous n'aurons pas les mêmes algorithmes de parcours.

- Si l'ensemble de nos objets a une vision restreinte du graphe, l'implantation de ce prédicat peut être :

```
rechercher(Objet,Propriété,Info) :-
        recepteur(Objet, Liste-liens),
        % determination de l'ensemble des liens dont Objet est récepteur
        parcours-liens(Propriété,Liste-liens,Info).

parcours-liens(Propriété,[L1|Liste-liens],Info) :-
        (send(L1,heritage-propriété(Propriété,Info);
        parcours-liens(Propriété,Liste-liens,Info)).
```

  1. La liste des liens dont *Objet* est récepteur est ordonnée selon la multiplicité locale à cet objet.
  2. Une seule solution est retournée.
  3. La méthode *héritable* associée aux objets émetteurs, peut, en cas d'héritage dynamique, faire à son tour appel au prédicat *rechercher*. Mais comme la vision que l'objet a du graphe est restreinte, cet appel n'est réalisé que si l'ensemble des propriétés concernées englobe la propriété dont on cherche à hériter.

- Si par contre l'ensemble des objets a une vision globale du graphe d'héritage, il n'est plus possible de déléguer comme précédemment la recherche d'une propriété aux ancêtres. De plus l'ensemble des propriétés concernées ne représente alors que des propriétés déclarées. L'implantation du prédicat *rechercher* pourrait alors être la suivante.
A un objet est associée son extension linéaire, qui consiste en une liste de liens individuels. L'information obtenue par héritage est celle correspondant au premier ascendant PA selon l'ordre de l'extension linéaire qui permet d'hériter de cette propriété. La suite des liens unissant cet ascendant à l'objet qui nous intéresse est parcourue afin d'éventuellement stopper la recherche en cas de propriété non propagée par un des descendants de cet ascendant!

  > La relation qui unit les classes *oiscau* et *autruche* ne propage pas la propriété de *voler*. Aussi une sous-classe de *autruche* ne devra-t-elle pas hériter par cette relation de la propriété de *voler*, mais pourra hériter de la classe *animal*, super-classe de *oiseau*, la propriété de se *mouvoir*.

  L'information recherchée est alors "héritée" par parcours du premier lien de l'extension linéaire dont l'ascendant PA est émetteur. PA émet l'information qui est éventuellement filtrée par ce lien, mais la réception de l'information est réalisée par l'objet pour lequel la recherche a été entreprise! Tout se passe donc comme si ce lien unissait PA à cet objet. C'est là un paradigme de la visibilité globale du graphe : il n'y a pas alors de filtrage effectif par l'ensemble du graphe d'héritage menant à la propriété, mais par un seul des liens.

rechercher(Objet,Propriété,Info) :-
extension-linéaire(Objet,Liste-EL),
rechercher(Objet,Propriété,Liste-EL ,Info).

rechercher(Objet,Propriété,[LH|Liste-EL],Info) :-
send(LH,propriétes-concernées(PC)),
element(P,PC),
(her-prop(Objet,LH,P,Info);
send(LH,propriétés-non-propagées(PNP),
not(element(P,PC)),
rechercher(Objet,P,Liste-EL,Info)).

où

her-prop(Objet,LH,P,Info) :-
send(LH,valeur(objet-influant,OE)),
send(OE,heritable(Lien-individuel,OR,Propriété,I),
send(Objet,heritee(Lien-individuel,OE,Propriété,Info,Information)).

La méthode *heritable* associée aux différentes classes du graphe ne doit pas faire appel au prédicat *rechercher* : une propriété n'est héritable que si elle est définie par la classe elle-même (cf. 2.2).

Si nous reprenons notre exemple du paragraphe 2.3, si la propriété P est définie par l'objet F, l'héritage de P par A n'est possible que si tous les liens entre F et A n'en interdisent pas la propagation. Par contre aucun filtrage de cette propriété par un autre lien que H_{FE} ne sera pris en compte au niveau de A.

La compilation du graphe d'héritage, qui conduit à un héritage statique consiste à propager toutes les propriétés concernées. Lorsque les objets ont une vision restreinte du graphe, la fonction $\mathcal{H}$ peut être implantée comme suit :

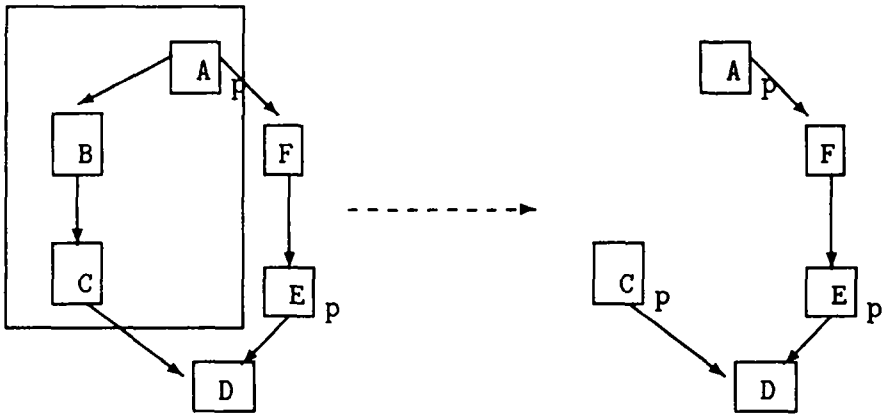
heritage-total(Objet) :-
recepteur(Objet, Liste-liens),
heritage-parcours-liens(Objet,Liste-liens).

heritage-parcours-liens(Objet,[L1|Liste-liens]) :-
(send(L1,valeur(objet-influant,OE)),
send(L1,heritage(OE,Objet)),
parcours-liens(Propriété,Liste-liens,Info)).

Si la vision du graphe est globale il suffit de remplacer dans cette fonction l'appel à *récepteur* par celui à *extension-linéaire* pour obtenir la fonction $\mathcal{H}$.

L'héritage statique correspond à une compilation du graphe d'héritage. Il entraîne un encombrement mémoire plus important pour un accès aux informations plus rapide tandis que l'héritage dynamique conduit à un gain en place mémoire pour des accès plus lents. Un avantage incontestable à ce dernier choix est la modifiabilité des systèmes obtenus. Trait primordial en Intelligence artificielle où l'acquisition de la connaissance est incrémentale et procède par succès-échec.

Il est intéressant de combiner héritage statique et héritage dynamique lorsqu'une part de la connaissance est bien établie et l'autre, correspondant à une sous-arborescence de la première, est en cours d'élaboration. L'héritage est alors statique pour la partie bien établie et dynamique pour l'autre. De la sorte, les modifications du sous-graphe sont facilitées. Lorsque la vision que les objets ont du graphe est restreinte, cette approche semble tout à fait réalisable. Les mécanismes de parcours des liens n'étant pas déclenchés par les objets en héritage statique. Par contre, une vision globale du graphe dans un tel cadre peut faire perdre l'intérêt d'une telle approche puisque soit il n'est pas tenu compte de l'héritage statique (prise en compte des seules propriétés déclarées), soit on peut aboutir à des erreurs.



Les classes B et C héritent statiquement de la propriété *p* définie dans A tandis que les classes F, E et D en héritent dynamiquement. La visibilité que la classe D a du graphe pour la propriété *p* est alors celle représentée par le graphe de droite. Dans le cas d'une linéarisation du graphe par tri topologique telle que celle proposée en Clos, la propriété définie par E eût été visible avant celle définie par A, qui apparaît par héritage statique directement au niveau de C.

Cette dernière remarque n'exclut pas l'orthogonalité entre vision du graphe et moment de l'héritage. Elle souligne simplement que mêler héritage statique et dynamique avec une vision globale du graphe est plus ardu qu'avec une vision restreinte.

3.1.5 Maintien de la cohérence d'une relation d'héritage

Le maintien de la cohérence d'une relation d'héritage implique lors de certaines modifications de l'objet émetteur de vérifier qu'elles ne compromettent pas la cohérence de la relation et éventuellement de mettre à jour l'objet récepteur pour la rétablir.

- L'ajout d'une propriété dans un objet émetteur d'un lien de sous-typage implique de vérifier que cette nouvelle propriété est bien héritable par le lien de sous-typage, et, si elle est redéfinie par une sous-classe, éventuellement par héritage multiple, il faut vérifier qu'il y a bien compatibilité sémantique (cf. 4.1). En cas d'héritage statique, l'"héritage" de la propriété doit alors être réalisé.
- La modification d'une propriété ne concernant pas le lien d'héritage n'implique, elle, aucune vérification.
- La modification d'une propriété héritable de façon dynamique n'implique aucune mise à jour de l'objet récepteur alors que si elle est héritable de façon statique, la modification survenue dans l'objet émetteur doit être propagée.

Lorsqu'il s'agit d'héritage dynamique, le maintien de la cohérence consiste simplement à vérifier qu'une modification de l'objet émetteur n'altère pas la cohérence de la relation, par exemple, qu'il n'y a pas violation de contraintes imposées par une relation de spécialisation. Cette étape est importante, car elle permet de détecter des concepts incohérents avant l'instanciation.

Lors d'héritage multiple statique, l'ajout ou la modification d'une propriété au niveau de l'objet émetteur, peut impliquer de recommencer l'héritage pour cette propriété dans l'objet récepteur. Il est donc très important de distinguer connaissances déclarées et connaissances héritées. En cas de modification de la propriété dans l'objet émetteur, après avoir rétabli au niveau de l'objet récepteur la propriété telle qu'elle a été déclarée, il faut faire appel à la fonction *rechercher(Objet,Propriété,Info)* (cf. 3.1.4).

Un ajout d'une propriété non encore héritée par l'objet récepteur implique simplement une propagation de la nouvelle propriété, soit l'appel pour le lien individuel déclenché à *heritage-propriété.*

Nous avons modélisé cette mise à jour incrémentale au moyen des *points d'activations.* Ils servent à préciser au niveau même des relations lors de quelles opérations sur l'émetteur, la cohérence d'une relation individuelle doit être rétablie et quelles actions il faut réaliser pour cela. Un point d'activation est constitué d'une liste de sélecteurs de méthodes[1](quelles opérations impliquent de rétablir la cohérence d'une relation?) et d'une procédure de mise à jour (comment faut-il la rétablir?). Celle-ci est appelée *action* et prend pour arguments un lien individuel et un message.

> Par exemple, un des points d'activation associés à un lien de sous-typage peut être défini comme suit:
> *sélecteurs : [affecter, affecter-descripteur].*
> *action : verification-propagation-propriété.*
>
> Un autre point d'activation concerne l'ajout d'une propriété. Il s'agit alors de vérifier que cette nouvelle propriété est bien héritable le long du lien de sous-typage puis de vérifier la compatibilité sémantique de la propriété vis-à-vis de la sous-classe, avant d'éventuellement réaliser effectivement la propagation.
> *sélecteurs : [ajouter].*
> *action : heritable-verification-propagation-propriété.*
>
> Pour un lien modélisant simplement un partage d'informations, ces différents points d'activations consisteront simplement à propager les informations héritables statiquement.

Une modification dans un objet peut entraîner la vérification et éventuellement le rétablissement de la cohérence de plusieurs liens d'héritage. En effet, tous les objets dépendant par héritage de l'objet modifié devront éventuellement être mis à jour et des vérifications à nouveau réalisées. Ces objets peuvent à leur tour être objet influant d'autres liens d'héritage et dans ce cas propager la modification. De la sorte, il peut être nécessaire d'imposer un ordre de parcours des différents liens d'héritage pour rétablir la cohérence de l'ensemble du

[1]Un sélecteur de méthode est constitué du nom de la méthode, de son arité et éventuellement du type de ses arguments.

graphe. Un mécanisme extérieur aux liens doit alors être déclenché par ceux-ci pour rétablir la cohérence de l'ensemble du graphe.

Lorsqu'une vérification est associée à l'établissement d'une relation d'héritage, il peut être nécessaire de vérifier la cohérence de celle-ci, lorsque l'objet récepteur est modifié. Pour obtenir un tel comportement, il est alors nécessaire de définir une relation inverse, afin que sa vérification soit automatique. En général, l'implantation de l'héritage statique prend en compte cette caractéristique et fixe les bornes de modifiabilité des objets (interdiction de modifier le type des objets, limitation dans la modification d'intervalles, ...).

De l'étude des mécanismes des relations, nous avons dégagé la classe de lien que voici.

3.2 Implantation au moyen de classes de liens

Une relation donnée est représentée par une classe dont les instances sont des relations individuelles qui se comportent toutes de la même façon. La classe *lien* définit les mécanismes nécessaires à l'établissement de la cohérence des relations individuelles, tandis que la méta-classe *métalien* définit les méthodes de création et de destruction des relations individuelles. Toute classe représentant une relation est une spécialisation directe ou indirecte de *lien* et une instance de *métalien*. Nous appellerons ces classes *des liens* et les instances de ces classes *des liens individuels*.

La classe *lien-heritage* est une spécialisation de *lien* et toute relation d'héritage en est une spécialisation.

3.2.1 Classes *Métalien* et *Lien*

L'implantation actuelle des liens a été réalisée en Othelo, langage à objets au dessus de Prolog. Aussi les méthodes sont-elles spécifiées en Prolog.

La métaclasse *métalien* définit les propriétés suivantes :
Attributs-d-instances : points-d-activation
Méthodes-d-instances : créer, détruire

Un lien individuel est caractérisé par sa classe, un objet influant et un objet dépendant. La classe *lien* définit les propriétés :
Attributs-d-instances: objet-influant, objet-dépendant
Méthodes-d-instances : cohérence, activer

Il est possible de préciser pour les attributs *objet-influant* et *objet-dépendant* le type de leur valeur. Ainsi, il sera possible de définir des liens d'héritage ne pouvant être établis qu'entre des familles d'objets bien précises.

Pour un lien donné, l'utilisateur doit définir les propriétés suivantes :

coherence(lien-individuel) Cette méthode qui précise la sémantique de la relation est exécutée à la création d'un lien individuel. Le corps de la méthode *cohérence*, associée à un lien, définit l'action qui établit la cohérence de chaque instance.

points d'activation Ce champ spécifie lors de quelles actions sur l'objet influant, la cohérence d'une relation individuelle doit être rétablie et quelles actions il faut réaliser pour

cela (cf. 3.1.5).

Soit la classe *point-d-activation* définie par :
Attributs-d-instances: selecteurs.
Méthodes-d-instances : action, activer.

Les méthodes indispensables à la mise en œuvre du mécanisme de maintien de la cohérence ne sont pas figées, et un utilisateur avancé peut les redéfinir pour en spécialiser le comportement.

Lorsqu'un lien individuel est déclenché, il y a exécution de la méthode *activer* définie dans sa classe. Cette méthode prend pour argument le message qui a causé l'activation du lien individuel. Par défaut, si le message correspond à un sélecteur reconnu par un point d'activation, l'action associée à ce dernier est alors exécutée.
Un utilisateur avancé pourra redéfinir la méthode *activer* pour, par exemple, retarder une activation ou vérifier des conditions avant l'activation.

Une instance d'un lien est créée par une méthode *créer* qui établit un lien individuel entre *objet-influant* et *objet-dépendant* passés en argument et déclenche la méthode *cohérence* définie dans le lien.

Le nombre important des relations individuelles à modéliser nous a conduites à minimiser la structure de ces objets, d'où la nécessité de la métaclasse *métalien*. En particulier dans cette mise en œuvre, nous avons exploité au maximum les possibilités relationnelles de Prolog.

En Othelo, la représentation interne d'un lien individuel de la classe l entre les objets oi et od est le fait l(oi,od).

3.2.2 Classe *lien-heritage*

La classe *lien-heritage* modélise l'ensemble des relations d'héritage. Toute relation d'héritage en est une spécialisation.

Cette classe, instance de *métalien* et sous-classe de *lien*, définit les méthodes suivantes :
Méthodes-d-instances : propriétés-concernées, verification, cohérence, héritage, héritage-propriété.

propriétés-concernées(Lien,PC) .
Par défaut, cette méthode s'adresse à l'objet émetteur du lien individuel pour récupérer l'ensemble des propriétés concernés.
Une sous-classe de la classe *lien-heritage*, la classe *héritage-selectif* redéfinit cette méthode de telle sorte que la liste des propriétés concernées corresponde à une liste stockée au niveau du lien individuel.

vérification(Lien,Objet-influant,Objet-dependant) .
Cette méthode est appelée avant l'établissement d'une relation d'héritage et a en charge de vérifier que la relation peut effectivement être établie. Si cette méthode échoue, la relation précedemment créée est détruite.
Par défaut, aucune vérification n'est réalisée.

héritage-propriété(Lien,Propriete,Information) .
Cette méthode détermine la propagation d'une propriété le long d'un lien et retourne l'information résultante. Lorsque l'héritage est statique, cette information est assimilée par l'objet récepteur du lien. L'implantation de cette méthode est donnée au paragraphe 3.1.3.

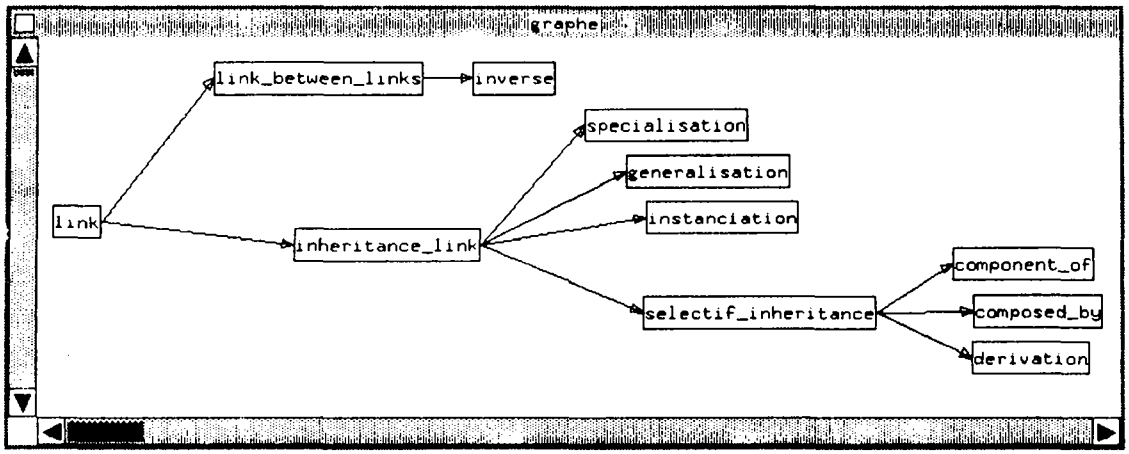


Figure 3.2: hiérarchie des relations d'héritage en Othelo

héritage(Lien-individuel,Objet-influant,Objet-dependant) .

Cette méthode exprime la propagation de l'ensemble des propriétés concernées le long d'un lien, elle est donc particulièrement utilisée en cas d'héritage statique. Elle est appelée par défaut par la méthode *cohérence* du lien d'héritage.

Plusieurs sous-classes de *lien-héritage* au sens de la spécialisation ont été définies en Othelo (cf. 4).

3.3 Conclusion

La décomposition de l'héritage en transferts d'informations le long de liens, permet une correspondance entre la représentation intuitive de l'héritage en terme de graphe de dépendances et l'implantation. De plus parcours et transfert sont ici séparés, ce qui permet une plus grande souplesse d'approche de l'un et l'autre, sans que le principe d'encapsulation des données soit violé, puisque tout lien d'héritage s'adresse nécessairement aux objets mis en dépendance pour réaliser la propagation des informations.

D'autre part, la représentation objet des relations d'héritage entre objets nous a permis d'émettre des contraintes sur l'établissement de certaines relations d'héritage et ainsi de leur associer une sémantique claire. Le chapitre suivant présente donc quelques liens d'héritage spécifiques. Enfin notons que cette modélisation objet des relations nous a également permis d'inférer automatiquement l'existence de relations comme les inverses des relations implantées, les arcs de transitivité lorsqu'ils sont utiles (les amis de nos amis sont nos amis) en tenant compte pour certaines relations de la transitivité faible, et d'interdire la mise en place de certaines relations lorsqu'elles se révélaient inconsistentes vis-à-vis du réseau pré-établi (contradiction pure avec l'existence de liens d'exceptions [DH89], métaclasses différentes [Gra89]).

Chapitre 4

Différents liens d'héritage

Dans les langages à objets de façon générale, on voit apparaître de plus en plus de liens d'héritage différents : lien de spécialisation, lien de partie, d'instanciation, de dérivation, etc.

Parmi les mécanismes d'héritage offerts, tous n'ont pas une sémantique très claire [Pat89, Bra83].

Ainsi la sémantique du lien *sorte-de* est largement débattue dans la littérature. Doit-il unir un ensemble à un sous-ensemble, signifier une spécialisation stricte ou simplement être considéré comme un outil de structuration des informations? A-t-on le droit d'exprimer des exceptions? Comment faire remonter des informations par généralisation?

La nécessité de représenter la composition entre objets a conduites soit à détourner les liens initialement prévus pour exprimer la spécialisation [Ame87], soit à l'introduction de nouveaux mécanismes [BC87,BS83].

Le besoin d'exprimer simultanément dans une même application des relations différentes, tout en offrant une sémantique claire pour chacune de ces relations nous a conduit à modéliser différents liens d'héritage. L'expression de la connaissance s'en trouve facilitée ainsi que sa compréhension ultérieure.

Dans ce chapitre, nous nous proposons de montrer comment la modélisation de l'héritage en terme de propagation d'informations permet d'implanter différentes sortes de liens d'héritage dont la sémantique est clairement établie.

4.1 Lien de spécialisation

La relation de spécialisation est souvent comparée à une relation de sous-typage. En effet, si l'on se base sur la définition donnée par America *"A type is a collection of objects that share some properties, notably the operations that can be performed on them"*, une classe correspond bien à un type [Car84]. Il est alors possible d'étendre cette approche en concluant que la spécialisation correspond à du sous-typage. Moultes controverses se sont opposées à cette conclusion [Ame87]. Cependant, puisque notre décomposition de l'héritage nous permet d'associer à un lien particulier une sémantique précise, nous nous proposons de présenter ici, un lien de "sous-typage" entre classes.

Soit la relation de sous-typage notée $\leq$ entre types de bases que nous nous proposons d'étendre aux classes.

Pour que deux classes Sc et C puissent être liées par cette relation, certaines propriétés doivent être vérifiées.

Une classe ou type consiste en une spécification du comportement de ses instances ou

élements, ce qui inclut les messages que ses instances peuvent recevoir (les méthodes), les contraintes sur les valeurs des paramètres de méthodes et les attributs.

Avant d'énoncer la règle sur laquelle est basée l'implantation du lien de sous-typage présenté dans ce paragraphe, précisons quelque peu notre vocabulaire :
- Une classe définit des propriétés lorsque celles-ci ont été déclarées à son niveau.
- Une classe connaît les propriétés qu'elle déclare et les propriétés dont elle hérite.

Dans ce qui suit, nous présentons un lien de spécialisation particulier qui exclut l'existence de propriétés privés et impose des restrictions identiques sur le type des arguments de méthodes qu'ils soient en entrée ou en sortie. Telle n'est pas toujours l'approche de cette relation [Car84,Ame87]. Notre objectif n'est pas de débattre ici de la justesse de ces choix, mais plus modestement de présenter une implantation parmi d'autres de la relation de sous-typage.

Regle 1 :
Soient deux classes Sc et C , pour que Sc $\leq$ C , il faudra que

- pour toute méthode *mi* connue par C de nom *Nmi* et de signature *t1,...,tn*$\rightarrow$ *t*, il existe une méthode *m'i* connue de Sc de nom *Nmi* et de signature *t'1,...,t'n* $\rightarrow$ *t'*, telle que pour tout j dans 1..n *t'j*$\leq$ *ti* et *t'* $\leq$ *t*.
- pour tout attribut *ai* connu par C de nom *Nai* et typé par *ti*, il existe un attribut *a'i* connu de Sc de nom *Nai* et typé par *t'i* tel que *t'i*$\leq$ *ti*.

Dans cette définition, nous ne prenons pas en compte les problèmes de surcharge tels que ceux soulevés dans [For91].

Afin de simplifier les règles présentées ici, nous nous plaçons dans le cas d'un héritage simple. L'implantation d'un lien de sous-typage consiste alors à :

1. vérifier avant établissement de la relation entre deux classes C et SC que

   (a) sur la classe C prétendante à être une "super-classe" :

   - toutes les propriétés pour instances (variables et méthodes) connues de C sont bien héritables par le lien de sous-typage pour la classe SC.

   (b) sur la classe prétendante SC à être une spécialisation de la classe C :

   - toute propriété redéfinie par la sous-classe respecte bien la règle 1 énoncée précédemment vis-à-vis de la super-classe.
   - toute propriété héritable de C qui n'est par redéfinie par SC est bien héritée par SC.

   En cas d'héritage multiple, les combinaisons sur le type des attributs (intersection) ou des arguments de méthodes permettent d'accepter la relation, même lorsque le type de la propriété précédemment héritée est un sur-type de la propriété connue de C. Cette approche implique de prendre en compte au niveau de SC des propriétés déjà connues par héritage d'autres classes de plus forte précédence, et de les "assimiler" puisqu'elles ne correspondent pas exactement aux propriétés héritables.

2. propager l'ensemble des informations statiques,

3. maintenir la consistence de la relation. C'est à dire, hormis la propagation de l'information en cas d'héritage statique, vérifier la consistence des éventuelles modifications. Par exemple,

   - l'ajout d'une propriété pour instance dans la super-classe implique de vérifier que cette nouvelle propriété est bien héritable par la sous-classe et que cet héritage préservera les règles de sous-typage.
   - une modification de contraintes implique également de vérifier que les contraintes dans les sous-classes respectent bien les règles de sous-typage.

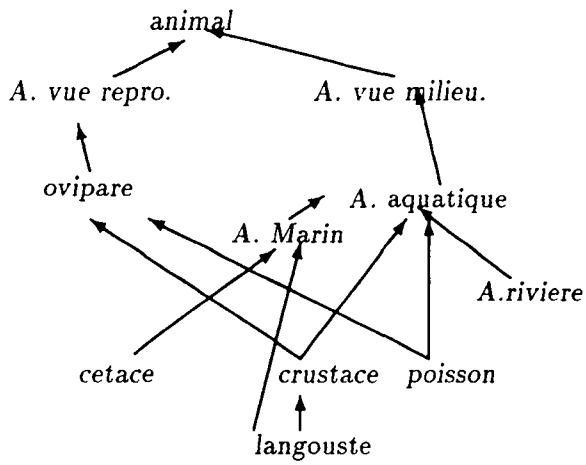
Dans un lien de *sous-typage* les propriétés d'instances *objet-dépendant* et *objet-influant* auront nécessairement pour valeur une classe.

La méthode de vérification a en charge de vérifier que les protagonistes d'un lien de spécialisation que l'on désire établir vérifient bien le point 1 précédent.

La méthode *cohérence* réalise le point 2 par appel à la méthode *heritage*, tandis que l'ensemble des points d'activation a en charge d'implanter le point 3.

Remarques ◊ En Othelo, l'héritage est statique, aussi est-il relativement facile d'implanter l'héritage multiple, tout en vérifiant les règles de cohérence énoncées par la règle 1. L'héritage multiple implique d'introduire le concept d'intersection de classes.
Deux approches sont alors possibles selon que l'on adopte une résolution statique ou non de l'intersection.

- Une résolution statique induit l'hypothèse que le graphe des classes est figé et connu au moment du calcul. L'intersection entre deux classes A et B est non vide, lorsqu'il existe un descendant commun. Il peut en exister plusieurs et dans ce cas, le résultat de l'intersection est la disjonction des classes X telles que $X \leq A$ et $X \leq B$ et il n'existe pas de classe Y telle que $Y \leq X$ et $Y \leq A$ et $Y \leq B$.



L'intersection entre *ovipare* et *A.aquatique* est la disjonction : *crustace* ou *poisson.*

- Si la résolution de l'intersection est dynamique, alors vérifier qu'un objet appartient bien à l'intersection de deux classes A et B consiste à s'assurer qu'il est bien instance d'une classe X sous-classe de A et sous-classe de B.

Lorsqu'il y a héritage multiple, il s'agit en plus des lois énoncées précédemment de vérifier lorsqu'une propriété héritable de C a été héritée d'une autre classe par SC que :

- $\forall$ *m'i* une méthode héritée dans SC de nom *Nmi* et de signature *t'1,...,t'n* $\rightarrow$ *t'*, alors, soit la méthode *mi* connue par C de nom *Nmi* et de signature *t1,...,tn*$\rightarrow$ *t*, pour tout i dans 1..n, $t'i \cap ti = t"i \neq \{\}$ et $t' \cap t = t" \neq \{\}$, et la méthode *mi* connue par SC de nom *Nmi* aura maintenant pour signature *t"1,...,t"n*$\rightarrow$ *t"*.
- $\forall$ *a'i* un attribut hérité par SC de nom *Nai* et typé par *t'i*, alors soit l'attribut *ai* connu par C de nom *Nai* et typé par *ti*, il faut vérifier que $t'i \cap ti = t" \neq \{\}$, et l'attribut *a'i* connu de SC sera après héritage typé par *t"*.

$\Diamond$ En Othelo, les méthodes correspondent à des prédicats Prolog. Aussi chaque paramètre d'une méthode peut-il être, par défaut, aussi bien en entrée qu'en sortie [Djo89]. Comme les lois imposées pour les paramètres en entrée et en sortie sont les mêmes, ce qui n'est pas toujours le cas [Car84,Ame87], les différents algorithmes présentés ici s'appliquent à notre langage.

4.1.1 Généralisation : lien dual de la spécialisation

Contrairement à l'approche des langages de classes, les langages à base de prototypes proposent eux le plus souvent de procéder par généralisation des concepts [Lie86b]. Cette méthodologie est particulièrement bien adaptée à la phase d'acquisition des connaissances en Intelligence Artificielle. En effet, l'expert a souvent plus de facilité à parler de cas particuliers dont il extraira une généralisation, que de prime abord à parler en terme de concept qu'il spécialisera. A moins d'une grande capacité d'abstraction et d'un domaine parfaitement cerné, nous agissons tous de même. Aussi nous semble-t-il très utile de permettre de s'exprimer en terme de généralisation. Les propriétés de tels liens sont alors exactement la fonction inverse de celle du lien de spécialisation. Le programmeur qui réalise que plusieurs types ou classes partagent des caractéristiques communes, pourra alors les factoriser pour produire une classe plus générale [HO87].

Aujourd'hui, nous avons limité la représentation de ce lien à l'expression de la relation inverse de la spécialisation, et nous ne lui avons donc pas associé un comportement particulier. Dès qu'une relation de spécialisation est établie, la relation inverse de généralisation, très utile pour le parcours du réseau de dépendance, est créée.

Une perspective de ce travail est d'extrapoler à partir d'un réseau de relations de généralisation, les propriétés qui peuvent être "remontées" par héritage au niveau des super-classes. Nous trouvons un tel comportement en Smeci [SME88], où le type des attributs dans les super-classes peut être déduits par l'union du type de ces attributs dans les sous-classes.

De façon générale, un des intérêts de préciser la sémantique de telles relations, est la possibilité escomptée d'un jour pouvoir déduire leur existence à la seule vue des propriétés des objets [RM90].

4.2 Liens de composition

Lorsqu'un objet est le résultat de l'assemblage de plusieurs autres, nous parlerons de lien de composition. L'héritage le long de ces liens est sélectif: seules certaines informations sont partagées entre les objets composites et composants.

Ainsi dans [Dug89], nous trouvons la définition du lien *Part-of* entre les classes *Face* et *Wall* qui permet de partager entre deux instances de ces classes la hauteur du mur par la face. La relation n'est effectivement établie que lorsque l'on déclare par exemple que *Face1* est une partie de *Wall1*. Il y a alors partage du champs *hauteur* entre ces instances.

Les liens de compositions unissent des instances, mais selon qu'il s'agit d'aggrégation ou de décomposition, des comportements différents sont à envisager.

Nous distinguons ainsi le lien d'*aggrégation* ou *partie-de* des parties vers l'objet composite. Dans ce cas, l'objet récepteur est constitué de plusieurs caractéristiques de l'objet émetteur, dont il a hérité. Il ne s'agit le plus souvent pas de recopier les propriétés définies dans les parties mais de les combiner ou de les partager.

> Un guéridon est constitué d'un plateau et d'un pied. La hauteur du guéridon est la somme de la hauteur du plateau et du pied. Le "diamètre" du guéridon et le "diamètre" du plateau sont la même propriété.

Même si le partage d'information n'est effectif qu'entre des instances, il est fréquent de vouloir définir une relation d'aggrégation directement au niveau de la classe de l'objet composite.

> Ainsi nous définirons la classe *guéridon* comme composée d'une instance de la classe *pied* dont elle hérite de la *hauteur*, et d'une instance de la classe *plateau* dont elle hérite de la *hauteur* et avec laquelle elle partage le *diamètre*.
> La création d'une instance de la classe *guéridon* implique alors immédiatement la création d'une instance de la classe *pied* et la création d'une instance de la classe *plateau*. Des relations de d'aggrégation unissent alors ces instances au guéridon.

Dans ce cas, l'existence des parties peut devenir invisible à l'utilisateur de l'objet composite. Une modification du tout peut alors, lorsque l'héritage ne consiste pas en un partage de structure, être incohérente vis-à-vis des informations héritées des parties.

> Si la hauteur d'un des pieds d'une table est modifiée, la hauteur de la table doit être modifiée ainsi que, par héritage, la hauteur des autres pieds de la table. (Le cycle est évité soit par un test quant à la nécessité de modifier la valeur d'un champs, soit par un mécansime dynamique de détection de boucles, basé sur le principe des interruptions [FP90b].)

Dans ce cas, il apparaît indispensable de définir le lien inverse de l'aggrégation, qui est le lien de *décomposition* ou *a-pour-partie*. Il permet de maintenir la consistence des parties vis-à-vis de l'objet composite.

> Dans une pièce, la position du guéridon détermine la position de son pied et de son plateau.

Les liens de décomposition sont effectifs entre instances mais peuvent être spécifiés au niveau des classes. Dans ce cas, il s'agit le plus souvent de préciser les propriétés à propager le long de ces liens si des instances de ceux-ci sont effectivement créées.

> Reprenons l'exemple de Dugerdil dans [Dug87].
> La classe *Embrasure-de-fenetre* déclare qu'elle peut être partie d'un mur dont elle hériterait de l'épaisseur, ou d'une paroi dont elle hériterait également de l'épaisseur ou d'un précadre dont elle hériterait de la hauteur et de la largeur. Ainsi une instance *Embrasure-de-fenetre1* déclarée explicitement comme partie d'un précadre *precadre-1* héritera automatiquement de ce dernier les champs *épaisseur*, *hauteur et largeur*.

D'autre part, certaines relations de composition permettent de mettre en relation automatiquement toute instance d'une classe à une autre instance.

> Soit la classe *Navette-spatiale* dont une instance est *Hermes*. Soit la classe *composant-de-Hermes* dont toutes les instances directes ou indirectes sont des parties de *Hermes* dont elles héritent un numéro d'immatriculation.

De façon générale, les relations de composition sont des relations d'héritage sélectif entre instances. Pour ces relations, l'ensemble des propriétés concernées est explicité statiquement au niveau de chaque relation individuelle.

La spécification de ces relations au niveau des classes d'objets devant être liés est réalisée au moyen de liens génériques [FP90a]. Ces relations impliquent la création de relations individuelles lorsqu'une instance de l'objet émetteur est créée (création du pied et du plateau qui composent un guéridon).

Identification des liens de parties Lorsque l'utilisateur veut permettre un accès nominatif aux parties d'un objet, il apparaît particulièrement utile d'associer un nom à certains liens individuels liant l'objet composite à une de ses parties.

Par exemple, il est très utile de parler de la jambe droite d'un homme, du fuselage d'un avion, etc. L'objectif est donc de rendre direct l'accès à une partie. L'accès est alors équivalent pour l'utilisateur à l'accès à un attribut.

Cette approche n'est pas opposée à l'approche relationnelle que nous préconisons. Mais dans ce cas, des accesseurs [DGN89,Kee89] doivent être définis qui autorise alors l'accès direct aux parties. Dans un tel cas, l'existence des liens inverses d'aggrégation entre ces parties et l'objet composite paraissent indispensables.

4.3 Dérivation

Il s'agit de représenter la transformation d'un objet en un autre[RM90]. Seules certaines caractéristiques sont alors héritées, et doivent être préservées. Il s'agit donc non seulement d'avoir une déduction automatique des propriétés héritées par l'objet récepteur mais de vérifier à tout moment la cohérence de la dérivation.

Par exemple, une représentation graphique ou textuelle d'un programme ou d'un objet est dérivée d'un arbre de syntaxe abstraite.

Il s'agit en l'occurrence de liens d'héritage sélectifs.

Le maintien de la cohérence consiste alors à vérifier que les propriétés de l'objet dérivé vérifient bien les contraintes imposées par l'objet maître.

4.4 Instanciation

Si nous reprenons la définition que nous avons donnée de l'héritage de structure : "propagation d'information structurelle d'un objet vers un autre", la création d'un objet est constituée en majeure partie d'héritage. La représentation de l'instanciation en terme de relation est fréquente dans les langages centrés schémas dédiés à la représentation des connaissances [RL89,RG77]. Dans ce cas, l'objet récepteur de l'information n'existe pas encore (sauf en cas d'instanciation multiple). Pour exister vis-à-vis de l'héritage, il suffit qu'il sache répondre au message *heritee* pour le lien *instance*.

L'établissement d'un lien d'instanciation consiste donc en premier lieu en une pré-création qui crée un objet minimal, objet sachant répondre au message :
heritee(instance,Classe,Propriété,Info).

La relation d'instanciation est la part de la création qui consiste à propager les informations d'une classe vers une instance.

4.5 Conclusion

Outre la puissance d'expression des liens, leur utilisation pour implanter l'héritage, présente plusieurs avantages :

- la mise en place de l'héritage sélectif.

  L'aspect sélectif de l'héritage se situe au niveau du lien où il est possible de stipuler les propriétés mises en relations (lien de composition, de dérivation). Cet aspect peut éventuellement être affiné au niveau des objets liés qui peuvent refuser d'émettre ou de réceptionner l'information. Ainsi le principe d'encapsulation des objets est bien respecté.

- l'association explicite d'une sémantique aux liens d'héritage.

  Le concepteur du langage a les moyens d'implanter de nouvelles relations d'héritage auxquelles il associera une sémantique particulière. L'acquisition et la compréhension de la connaissance s'en trouvent facilitées.

- l'expression de relations entre relations.

  L'inverse, la contradiction, la transitivité sont des relations. Exprimer leur existence complète la description des relations d'héritage et améliore l'aspect déductif des langages qui est très souvent limité à une exploitation primaire des relations d'héritage.

- la mise à jour automatique des relations.

  Il suffit d'exprimer au niveau du lien quels sont les points d'activation de la dépendance pour le mettre dynamiquement à jour. Il peut être nécessaire d'ordonner les mises à jour mais leur déclenchement reste indépendant des autres mécanismes du langage.

En fonction des applications, un nombre limité de sortes de relations d'héritage est nécessaire pour modéliser de façon adéquate la connaissance que l'on a du monde réel. Par les liens, le concepteur dispose d'outils pour construire les briques relationnelles spécifiques au domaine traité. Ainsi il ne s'agit pas de définir beaucoup de relations d'héritage, mais davantage de définir les relations adéquates. De la sorte, l'acquisition et l'exploitation de la connaissance sont facilitées sans que la complexité de la maintenance soit augmentée.

Chapitre 5

Conclusion

Ce rapport présente une décomposition de l'héritage en propagation d'informations le long de liens d'héritage. Chaque transfert peut être décomposé en trois étapes essentielles : l'émission, la propagation et la réception de l'information. L'ordre de déclenchement de ces transferts est un mécanisme annexe.

Les liens étant des objets et les différents accès aux objets liés passant nécessairement par envoi de message, l'héritage n'est plus un mécanisme figé. Cette modélisation en fait un mécanisme facilement redéfinissable par le programmeur. Ce dernier peut intervenir à la fois sur le parcours et dans la définition même des différents liens d'héritage. Le système résultant gagne ainsi en réflexivité.

De plus l'encapsulation des objets est d'autant mieux préservée que l'héritage ne se situe plus au niveau même de la structure des objets mais "passe" par leur interface relationnelle. Ainsi chaque objet est maître des informations qu'il partage ou propage.

Cette décomposition est d'autant plus puissante qu'elle peut être exploitée pour paramétrer le comportement des propriétés de propriétés à l'héritage. Ainsi il est possible d'exprimer différentes combinaisons sur le type des attributs, l'union de contraintes, la combinaison de démons,....

La distinction entre le moment de l'héritage et la propagation des informations favorise les comparaisons entre les différentes implantations de ce mécanisme, tandis que la notion de visibilité du graphe d'héritage éclaircit, à notre sens, la nature des informations propagées par héritage et l'approche qui doit donc être faite d'un langage selon qu'il choisit l'une ou l'autre implantation. Avec une visibilité globale du graphe d'héritage, il est indispensable de connaître le graphe d'héritage d'une classe pour l'utiliser comme émetteur d'un nouveau lien d'héritage. Tel n'est pas le cas en visibilité restreinte.

La puissance expressive du langage est accrue avec l'utilisation des relations d'héritage.

Toute relation d'héritage est représentée de façon uniforme. Relations de spécialisation, de composition, de dérivation ou d'ami (C++) sont toutes exprimées au moyen de liens.

Cette représentation permet de leur associer, via des méthodes, une sémantique claire. Des vérifications quant à la validité de l'établissement de certaines relations peuvent alors être réalisées automatiquement. Ainsi les méthodes virtuelles ou retardées doivent être redéfinies dans les sous-classes pour qu'une relation de spécialisation puisse effectivement être établie, tandis que certaines contraintes de type doivent être vérifiées pour la mise en place de relations de sous-typage. Ce travail va donc dans le sens d'une formalisation de l'héritage adaptée à chaque sorte de relation.

Pour déterminer le parcours du graphe, interviennent alors la multiplicité locale à tout

objet et éventuellement un ordre sur les sortes de relation. En effet, bien que nous n'ayons pas résolu ce problème, il semble que certaines sortes de relations d'héritage soient prioritaires par rapport à d'autres. Ainsi il est indispensable d'effectuer l'héritage le long de lien d'instanciation, avant de le réaliser pour la spécialisation.

Cette modélisation de l'héritage permet d'améliorer conjointement la compréhension et la puissance expressive du système.

Bibliographie

[ALB86] P. Albert, V. Lacroix, and S. Le Bars. *KOOL Manuel de référence.* Bull, 1986.

[Ame87] P. America. Inheritance and subtyping in a parallel object oriented language. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 281–289, Paris, 1987.

[AT88] M. Aksit and A. Tripathi. Data abstraction mechanisms in sina/st. In *OOPSLA '88 Proceedings*, pages 267–275, ACM, September 1988.

[BC87] E. Blake and S. Cook. On including part hierarchies in object-oriented languages, with an implementation in Smalltalk. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 45–54, Juin 1987.

[BC90] G. Bracha and W. Cook. Mixin-based inheritance. In ACM/Sigplan, editor, *Proceedings of European Conference on Object-Oriented Programming*, pages 303–311, ACM, Ottawa, Canada, October 1990. Published by Sigplan Notices Volume 25, Number 10.

[BDGK88] D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, and S.E. Keene. *Common Lisp Object System Specification.* 1988. Draft submitted to X3J13 March.

[Bra83] R.J. Brachman. What is-a is and isn't : an analysis of taxonomic links in semantic networks. In *Computer Knowledge Representation, IEEE*, pages 37–41, Octobre 1983.

[BS83] D.G. Bobrow and M. Stefik. *The LOOPS Manual.* 1983.

[Car84] L. Cardelli. A semantic of multiple inheritance. In *Lectures Notes in Computers Science, Semantics of data types*, Springer-Verlag, New York, 1984.

[CD88] E. Chouraqui and Ph. Dugerdil. Conflits solving in a frame-like multiple inheritance system. In *European Conference on Artificial Intelligence ECAI 88*, Munchen, August 1988.

[CG90] B. Carre and JM. Geib. The point of view notion for multiple inheritance. In ACM/Sigplan, editor, *Proceedings of European Conference on Object-Oriented Programming*, pages 312–321, ACM, Ottawa, Canada, October 1990. Published by Sigplan Notices Volume 25, Number 10.

[Cla85] B.D. Clayton. *Art, Programming tutorial.* Mars 1985.

[DG87] L.G. Demichiel and R.P. Gabriel. The common Lisp object system : an overview. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 201–222, Paris, June 1987.

[DGN89] H. Davis, N. Graube, and G. Nuyens. Telos - the EuLisp object system. 1989. Draft of August 3, 1989.

[DH89] R. Ducournau and M. Habib. La multiplicité de l'héritage dans les langages à objets. *TSI*, 8(1):41–62, Janvier 1989. Numéro Spécial Langages orientés objet.

[Djo89] C. Djossou. *Etude sur l'envoi de message et la modélisation des méthodes par des objets.* Caen, 1989.

[Dug86] P. Dugerdil. A propos des mécanismes d'héritage dans un langage orienté objet. In *CIIAM*, pages 67–77, Marseille, 1986.

[Dug87] P. Dugerdil. Les mécanismes d'héritage d'OBJLOG : vertical et sélectif multiple avec point-de-vue. In *Reconnaissance des formes et intelligence artificielle*, Antibes, France, November 1987.

[Dug89] P. Dugerdil. Inheritance mechanisms in the OBJLOG language: multiple selective and multiple vertical with points of view. In *Proceedings of the workshop on Inheritance hierarchies in knowledge representation and programming languages*, pages 233–242, Viareggio, Italy, February 1989.

[For91] C. Fornarino. *ObjectivAda : extension objet du langage Ada. Application à un environnement pour la conception de systèmes experts.* PhD thesis, Université de Nice, Fevrier 1991.

[FP90a] M. Fornarino and AM. Pinna. *Expression des relations et maintien de la cohérence : le concept de lien.* Research report 1346, INRIA, Novembre 1990.

[FP90b] M. Fornarino and A.M. Pinna. *Un modèle objet logique et relationnel. Le langage Othelo.* PhD thesis, Université de Nice, Avril 1990.

[GR82] A. Goldberg and D. Robson. *SMALTALK-80 : The language and its Implementation.* Assison-Wesley, Xerox Palo Alto Research Center, 1982.

[Gra89] N. Graube. Metaclass compability. In *Object-Oriented Programming: Systems, Languages and Application*, pages 305–315, ACM, New Orleans, Louisiana, October 1989. Published in Sigplan Notices Volume 24, Number 10.

[HK89] Manfred Hein and Gerhard K. Kraetzschmar. Inheritance in programming languages revisited: some problems and steps towards their solution. In *Proceedings of the workshop on Inheritance hierarchies in knowledge representation and programming languages*, pages 161–187, Viareggio, Italy, February 1989.

[HO87] D.C. Halbert and P.D. O'Brien. Using types and inheritance in object-oriented languages. In Bezivin, Hullot, Cointe, and Lieberman, editors, *ECOOP'87, European Conference on Object-Oriented Programming*, pages 20–31, Paris, France, June 1987. Lecture Notes in Computer Science, no 276, Springer-Verlag.

[KEE85] *KEE v.2. Software Development System, User's Manual.* Intellicorp., 1985.

[Kee89] S.E. Keene. *Object-Oriented Programming in Common Lisp: A Programmer'Guide to CLOS. Symbolics Press*, Addison-Wesley Publishing Company, 1989.

[Lac89] V. Lacroix. Kool: a reflexive representation language. In *Technology of Object-Oriented Languages and Systems*, pages 309–321, CNIT Paris - La defense - France, November 1989.

[LAP89] *LAP, Reference Manual.* Elsa-Software, Paris, 1989. Version 3.0.

[Lie86a] H. Lieberman. Delegation and inheritance: two mechanisms for sharing knowledge in object-oriented systems. *Bigre + Globule*, 48:79–89, 1986. Journées Langages Orientés objets.

[Lie86b] H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In Norman Meyrowitz, editor, *OOPSLA'86 Proceedings*, pages 214–223, Portland, Oregon, September 1986.

[Mey88] B. Meyer. *Object-oriented Software Construction. in Computer Science*, Prentice Hall, 1988.

[Moo86] D.A. Moon. Object-oriented programming flavors. In Norman Meyrowitz, editor, *OOPSLA'86 Proceedings*, pages 1–8, Portland, Oregon, September 1986.

[NF87] R. Nado and R. Fikes. Semantically sound inheritance for a formally defined frame language with defaults. In *AAAI87 Proceedings of the Sixth National Conference on AI*, pages 443–448, Seattle Washington, 13–17 July 1987. Tome 2.

[Pat89] P. F. Patel-Schneider. What's inheritance got to do with knowledge representation. In *Proceedings of the workshop on Inheritance hierarchies in knowledge representation and programming languages*, pages 5–18, Viareggio, Italy, February 1989.

[Rec] Rechenman. Modèle dde connaissances à objets. Compte-rendu d'avancement du projet G.S.E.-B.D. Rapport Final. Annexe, Decembre 1990.

[Ref83] *Reference manual for the Ada programming language.* Alsys, 29, Avenue de Versailles 78170 La Celle-Saint-Cloud, France, 1983. ANSI/MIL-STD 1815A.

[RG77] R. Bruce Roberts and Ira P. Goldstein. *The FRL MANUAL.* MIT, 1977. Report 409, Artificial Intelligence Laboratory.

[RL89] C. Roche and JP. Laurent. Les approches objets et le langage LRO2 (KEOPS)*. *TSI*, 8(1):21–39, Janvier 1989. Numéro Spécial Langages orientés objet.

[RM90] M. Rueher and C. Michel. Using objects evolution for software processes representation. In IEEE Computer Society Press, editor, *Proc. Software Track*, pages 121–130, Hawai, 1990.

[SB86] M. Stefik and D.G. Bobrow. Object oriented programming : themes and variations. *AI Magazine*, 6(4):40–62, 1986.

[SME88] *SMECI Manuel de référence.* Ilog, 1.4 edition, 1988.

[Sny86] A. Snyder. Encapsulation and inheritance in object-oriented programming languages. In *OOPSLA'86 Proceedings*, pages 38–45, ACM, September 1986.

[Ste87] L.A. Stein. Delegation is inheritance. In *OOPSLA'87 Proceedings*, pages 138–146, October 1987.

[Str86] B. Stroustrup. *The C++ Programming Language. Computer Science*, Addison-Wesley Publishing Company, 1986.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399