Instanciation multiple et classification d'objet

D. Rieu, Toan Gia Nguyen, Annie Culet

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

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° 1480

Programme 3
Intelligence artificielle, Systèmes cognitifs et
Interaction homme-machine

INSTANCIATION MULTIPLE ET CLASSIFICATION D'OBJET

Dominique RIEU
Nguyen GIA TOAN
Annie CULET

GROUPE DE RECHERCHE
GRENOBLE

Juillet 1991

RR-1480

INSTANCIATION MULTIPLE ET CLASSIFICATION D'OBJETS

Dominique Rieu, Nguyen Gia Toan[1], Annie Culet

IMAG & INRIA[1]
Laboratoire Génie Informatique
BP 53 x
38041 Grenoble Cedex
France
Tel: 76.51.45.75
Fax : 76.44.66.75
e-mail : nguyen@imag.fr

Résumé

L'instanciation permet de lier un objet à l'entité conceptuelle qui le décrit: sa classe. Bien qu'étant un des concepts fondamentaux des systèmes à objets elle n'a pas retenue la même attention que la spécialisation ou l'encapsulation. De nombreux travaux ont pourtant montré que:
- l'objet ne peut pas toujours être un moulage parfait de l'entité conceptuelle qui le décrit,
- la distorsion de l'objet par rapport au moule idéal représenté par sa classe peut évoluer,
- une évolution d'un objet peut nécessiter de le re-classer, par exemple dans des sous-classes,
- un lien d'instanciation ne modélise qu'un point de vue de l'objet.

Cet article propose une remise en cause des hypothèses de base concernant l'instanciation. L'instanciation est assouplie: une instance n'est plus forcément un moulage parfait de l'entité conceptuelle qui la décrit. On parle d'instanciation par moulage souple. Elle est aussi étendue: une instance peut être rattachée à plusieurs classes. On parle alors de multi-instanciation.
Nous montrons, ici, en quoi la multi-instanciation par moulage souple peut être un concept unificateur qui permet d'étendre les mécanismes de classification afin de prendre en compte simultanément la notion de point de vue et l'évolution des connaissances. Nous décrivons le mécanisme "MIC" (Multi-Instanciation par Classification) mis en œuvre dans le modèle de représentation de connaissances SHOOD. Il est conçu pour la classification d'objets évolutifs qui peuvent être incomplets et passer par des états incohérents.

MULTIPLE INSTANTIATION AND OBJECT CLASSIFICATION

Abstract

Instantiation mechanisms usually relate object instances to the conceptual entities describing them i.e, their classes. They belong to the fundamental concepts in object-oriented languages and database systems. They have seldom been questioned in the literature before. Recent studies have shown however that object instances cannot always be exact representatives of their classes. They can exhibit distorsions which are subject to change during the application lifetime. Further, object evolution may require migration of instances among classes. Finally, a particular class may represent only one specific point of view on the objects.
This paper relaxes some of the constraints usually associated to instantiation mechanisms. Instances are no longer required to be exact representatives of their classes. This is called here flexible instantiation. Further, instances may be related to several classes simultaneously. This is called multiple instantiation.
It is shown how flexible and multiple instantiation extend classification mechanisms. It allows the latter to take into account multiple object representations and object evolution. A mechanism called MIC - an acronym for Multiple Instantiation and Classification - is described. It is intended to classify evolving objects with changing completeness and consistency. It is designed for an object-based knowledge representation model called SHOOD.

1. INTRODUCTION

Les approches orientées objets des langages de programmation, des systèmes de gestion de bases de données ou de représentation des connaissances reposent sur les concepts bien connus de méta-classes, de classes, d'instances, d'attributs, de méthodes, de messages, de spécialisation, d'héritage [COP84, GOL83, STE86, MEY88, MAS89,KIM90,UNL90]. Ces concepts sont bâtis sur des hypothèses de base telles que:
- *"une classe hérite des attributs et des méthodes définis dans sa (ses) super-classe(s)"*,
- *"une instance appartient à une et une seule classe qui définit sa structure et son comportement"* [GOL83, KEE88, MEY88, UNL90],
- *"une instance peut être incomplète mais ne peut pas être incohérente"* [REC88].

Les hypothèses concernant l'instanciation, parfois appelée par moulage [MAS89], ont des avantages évidents: elle facilite le contrôle de cohérence, la sélection de méthodes et sert de support à la mise en œuvre de concepts de plus haut niveau tels que l'héritage multiple et les objets composites.

Cet article propose une remise en cause de ces hypothèses :
- une instance peut appartenir à plusieurs classes simultanément,
- une instance n'est plus forcément un moulage parfait des entités conceptuelles qui la décrivent.

Le concept d'instanciation par moulage parfait est remplacé par celui de multi-instanciation par moulage souple.

Nous verrons que cette extension est une aide intéressante pour la prise en compte de deux concepts:
• les *points de vue multiples* d'un objet.
La perception que l'on a d'un objet varie dans le temps et d'un individu à l'autre. Je peux voir mon avion comme un objet de collection, un objet de loisir ou un moyen de transport. Dans les systèmes à objets, ces différentes perceptions correspondent à autant de *points de vue* différents d'un même objet. Le concept de points de vue est similaire à celui de *perspectives* en IA [STE86,MAR90], de *vues* en Base de Données [HEI86], de *représentations* en CAO [RIE86, RIE91a, NGU91b]).
• les *objets incomplets et incohérents.*
Dans des applications telles que la CFAO l'objet à concevoir est la plupart du temps incomplet et passe par des états incohérents jusqu'à la validité complète de son processus d'élaboration [RIE90, NGU91a]. La tendance actuelle est d'intégrer dans les modèles de données des règles sémantiques effectuant un contrôle de cohérence à la demande. Certaines règles de cohérence peuvent donc ne pas être respectées [NGU89, FAV90,FAV90b] pendant de longues périodes de temps et les réactions du système en cas de violation ne sont plus standardisées [KOT88].

La connaissance détenue sur un objet incomplet et incohérent est par nature *évolutive*. Dans les systèmes de représentations de connaissances à objets, l'évolution des connaissances est prise en compte par un mécanisme de *classification*. La classification consiste à rechercher à quelles classes une instance particulière peut être rattachée [REC88]. Si, par exemple, la connaissance sur une instance est enrichie, le mécanisme de classification peut tenter de la rattacher à une classe plus spécialisée que celle de sa création.

Nous montrons ici en quoi la multi-instanciation par moulage souple permet d'étendre les mécanismes de classification afin de prendre en compte simultanément les notions d'objet incomplet et incohérent, de points de vue et d'évolution des connaissances.
Le paragraphe 2 décrit les avantages et la complémentarité de la multi-instanciation, de la spécialisation multiple et de l'agrégation pour la prise en compte des points de vues. Le paragraphe 3 montre l'apport de la classification pour le traitement d'objets évolutifs (§ 3.1), puis l'intérêt d'un mécanisme de multi-instanciation par classification pour le traitement simultané des concepts de points de vue et d'évolution d'objets (§ 3.2). Le paragraphe 4 montre comment classifier des objets incomplets et incohérents. Le paragraphe 5 décrit le mécanisme de Multi-Instanciation par Classification (MIC) mis en œuvre dans le modèle de représentation de connaissances SHOOD [ESC90].

2. POINTS DE VUE

La perception que l'on a d'un objet varie dans le temps et d'un individu à l'autre. Dans les systèmes à objets, ces différentes perceptions correspondent à autant de *points de vue* différents d'un même objet. Leur cohabitaion au sein d'une même base pose de multiples difficultés de représentation et de manipulation telles que la gestion de l'évolution simultanée de deux points de vue ou l'expression des liens sémantiques entre les points de vue. En CAO de mécanique, par exemple, les représentations technologiques et géométriques d'un objet sont souvent élaborées simultanément à partir de sa représentation cinématique. Elles utilisent et concrétisent des informations cinématiques et recouvrent donc des informations communes ou dépendantes.

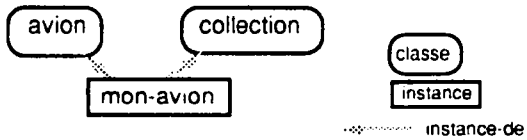
La notion de point de vue peut être modélisée explicitement [MAR90, CAR89] ou implicitement. Dans le cas des modélisations explicites, le concept de point de vue est intrinsèque au modèle de représentation de connaissances. On peut, par exemple le représenter par un nouveau type de lien [CAR89], ou gérer des hiérarchies de classes parallèles [MAR90]. Dans un cas de modélisation implicite on ne rajoute pas de nouveaux concepts. Les points de vue peuvent alors être représentés en utilisant l'agrégation, la spécialisation multiple ou la multi-instanciation.

Nous n'envisageons par la suite que les modélisations implicites. En effet, un des principes de SHOOD est d'offrir un système évolutif et modifiable. Pour cela nous avons choisi une politique consistant à implanter un minimum de concepts, définissables en terme d'un modèle minimal. Chaque concept peut être utilisé pour modéliser d'autres concepts ou pour prendre en compte des connaissances applicatives. La spécialisation, par exemple, se définit comme un attribut, au même titre que l'attribut moteur d'une classe avion. On peut donc la modifier comme un attribut et ainsi obtenir un système complètement différent. De même, la spécialisation est utilisée pour spécialiser les classes d'un domaine d'application, mais aussi pour affiner des attributs ou surcharger des méthodes. En effet, tout attribut et toute méthode sont modélisés dans SHOOD par des classes. Affiner un attribut ou surcharger une méthode revient alors à créer une sous-classe de la classe modélisant l'attribut ou la méthode (§ 3.1).
Une règle d'or du modèle est donc : "peu de concepts et beaucoup de sémantique" .

2.1 DE L'INSTANCIATION A LA MULTI-INSTANCIATION

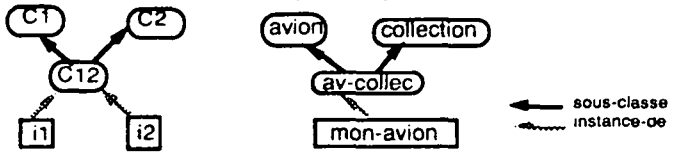
L'instanciation est un des concepts fondamentaux des systèmes à objets [MAS89]. Elle permet de lier un objet à l'entité conceptuelle qui le décrit: sa classe. Dans la plupart des systèmes à objets une instance n'est rattachée qu'à une et une seule classe [GOL83].

La multi-instanciation consiste à permettre le rattachement simultané d'un objet à plusieurs classes [VAN89]. Nous parlons ici de la multi-instanciation explicite qu'il ne faut pas confondre avec la multi-instanciation implicite induite par les liens de spécialisation: mon avion amphibie est aussi un avion. La multi-instanciation explicite permet le rattachement d'une instance à des classes qui ne sont pas directement ou indirectement des spécialisations les unes des autres. Elle autorise donc le rattachement d'une instance à des classes comportant des sémantiques différentes de celles drainées par sa classe de création. Il ne s'agit plus ici d'un affinage ou d'une remise en cause d'un point de vue de l'objet, mais bien d'un enrichissement de connaissances sémantiquement différentes, c'est à dire de la prise en compte d'un nouveau point de vue sur un objet. C'est le cas de mon avion que l'on rattache à la classe des *objets de collection* parce qu'il a un âge respectable. Mon avion peut alors être vu comme un véhicule ou comme un objet de collection.

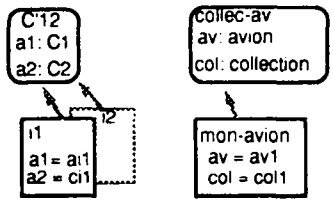


2.2 SPECIALISATION-MULTIPLE, AGREGATION ET MULTI-INSTANCIATION

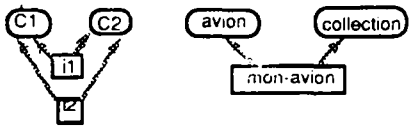
On veut exprimer que deux classes C1(avion) et C2 (collection) modélisent deux points de vue différents du même ensemble {i1,i2..} d'objets.
On peut utiliser la *spécialisation multiple* en créant une sous-classe C12 (av-collec) commune à C1 et C2 dont l'ensemble des instances est {i1,i2..}.



On peut aussi utiliser l'*agrégation* en créant une classe C'12 (collec-av) dont chaque attribut (a1 et a2) correspond à un point de vue.



On peut enfin utiliser la *multi-instanciation* en rattachant directement i1, i2...à C1 et C2.



Par la suite, les classes C12 (av-collec) ou C'12 (collec-av) sont appelées *classes inter-points de vue*. Les classes C1(avion) ou C2 (collection) sont appelées *classes point de vue*.

Ces représentations induisent des différences, notamment sur les points suivants:

• Regroupement de l'information concernant les différents points de vue dans une classe.
Dans les cas de la spécialisation et de l'agrégation les différents points de vue cohabitent au sein d'une même classe inter-points de vue. Ce n'est pas le cas de la multi-instanciation où cette cohabitation n'est pas matérialisée par une classe mais par les différents liens d'instanciation liant chaque instance aux classes points de vue.

• Préservation de l'identité d'objet à travers ses différents points de vue.
Dans les cas de la multi-instanciation et de la spécialisation chaque instance i1, i2 (mon-avion) appartient aux classes points de vue C1 (avion) et C2 (collection): il s'agit du *même* objet perçu sous différents points de vue. Dans le cas de l'agrégation deux points de vue du même objet n'ont aucune raison d'être représentés par le même identifiant. En effet, C'12 n'étant pas une sous-classe de C1 et C2 une de ses instances n'a aucune raison d'appartenir à C1 et C2, de plus C1 et C2 ne sont pas dans la plupart des cas des spécialisations l'une de l'autre. L'agrégation ne permet donc pas d'exprimer qu'il s'agit bien du même objet.

2.3 QUAND UTILISER LA MULTI-INSTANCIATION ?

La multi-instanciation est préférable à la spécialisation multiple ou à l'agrégation dans les cas suivants:
* La classe inter-points de vue (C12 ou C'12) est une classe creuse, c'est à dire que la cardinalité de l'ensemble de ses instances est faible. Si peu de personnes ont de vieux avions, la classe des avions de collection surcharge inutilement le graphe des classes. Par contre, si dans un environnement de CFAO tous les objets mécaniques admettent des représentations fonctionnelle et géométrique il est préférable de créer une classe inter-points de vue. La

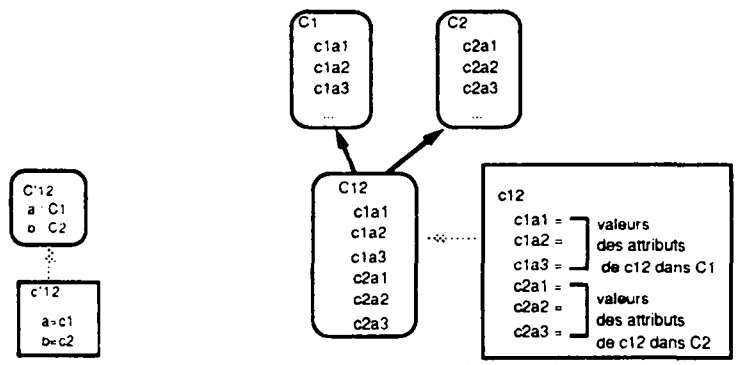
cardinalité de l'ensemble des instances d'une classe est jugée faible ou forte par rapport au nombre d'instances à classer : il s'agit d'un pourcentage.
* les différents points de vue ont des existences et des valeurs indépendantes. Si tel est le cas, aucun lien sémantique inter-points de vue n'est alors à modéliser dans la classe inter-points de vue (C12 ou C'12). Par exemple il se peut qu'aucun lien de dépendance, de calcul ou de cohérence ne soit à exprimer entre mon-avion vu comme un avion et mon-avion vu comme un objet de collection. Par contre il existe des programmes testant la compatibilité entre les représentations cinématique et géométrique d'un objet mécanique. Cette compatibilité peut être exprimée dans C12 ou C'12 par une méthode, une règle de cohérence ou une inférence.

Lorsque ces deux conditions sont réunies la multi-instanciation est un concept suffisant pour modéliser les points de vue. Son utilisation permet:
• de préserver l'identité d'objet à travers ses différents points de vue,
• d'éviter certaines classes creuses, c'est à dire dont la cardinalité de l'ensemble des instances est faible. Si peu de personnes ont de vieux avions, la classe des avions de collection surcharge inutilement le graphe des classes.
• d'éviter certaines modifications du graphe des classes, en particulier l'ajout de classes inter-points de vue non prévues initialement. Si le cas des avions de collection n'a pas été prévu lors de la conception des schémas, il faudra rajouter dynamiquement la classe des avions de collection lors de la création de mon-avion.
• d'éviter dans les classes inter-points de vue, la gestion de valeurs non significatives : certains objets mécaniques n'auront, par exemple jamais de représentation technologique. L'existence de valeurs non significatives complique la gestion des valeurs inconnues. En effet comment savoir pour un objet donné si un point de vue n'est pas encore connu ou s'il est sans signification pour l'objet ("quelle est la couleur de la queue d'un chat sans queue?"),
• de prendre en compte dynamiquement les nouveaux points de vue au fur et à mesure de leur élaboration: il suffit de rajouter un lien d'instanciation entre l'objet et la classe point de vue.

Dans le cas contraire, et surtout lorsque des liens inter-points de vue sont à exprimer, il faudra avoir recours à la spécialisation multiple ou à l'agrégation. Le paragraphe 2.4 montre l'avantage de l'agrégation par rapport à la spécialisation multiple. Le paragraphe 2.5 montre comment combiner l'agrégation à la multi-instanciation de manière à préserver l'identité de l'objet.

2.4 AVANTAGES DE L'AGREGATION

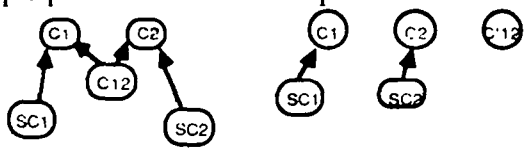
La spécialisation multiple permet, comme la multi-instanciation, de préserver l'identité d'objet à travers ses différentes représentations. Cependant, l'agrégation nous semble plus élégante pour diverses raisons:
* Un point de vue correspond à une instance.
Soient c'12, c1 et c2 des instances appartenant respectivement à C'12, C1 et C2. Utilisant l'agrégation (figure de gauche) les points de vue a et b de l'instance c'12 sont modélisés par les instances c1 et c2 de C1 et C2. Ce n'est pas le cas pour la spécialisation multiple (figure de droite) où toutes les propriétés des différents points de vue sont mises à plat et regroupées dans la classe inter-points de vue C12.



C'12 classe inter-points de vue par agrégation

C12 classe inter-points de vue par spécialisation

* Les sous-graphes des classes points de vue et inter-points de vue peuvent être élaborés indépendamment rendant ainsi le graphe des classes plus simple et plus lisible.
Dans le cas de la spécialisation, une même classe (C1 ou C2) est impliquée dans des liens de spécialisation dont certains sont destinés à *affiner* un point de vue (SC1 ou SC2) et d'autres à *regrouper* des points de vue (C12). Il en découle une certaine confusion et ambiguïté d'utilisation du lien de spécialisation. Par contre, dans le cas de l'agrégation un lien de spécialisation n'est utilisé que pour affiner un concept .



C12 classe inter-points de vue par spécialisation

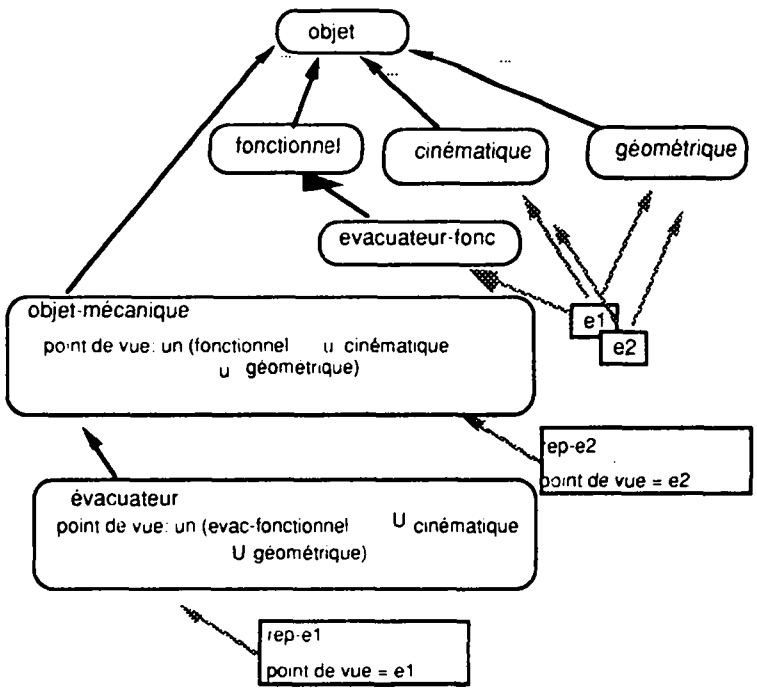
C'12 classe inter-points de vue par agrégation

Plus le graphe de spécialisation est simple (une sémantique unique du lien de spécialisation), plus le mécanisme de classification est performant. Dans l'exemple précédent, si la classe inter-points de vue est créée par agrégation (C'12), le mécanisme de classification peut rechercher s'il est possible de transférer une instance c1 de C1 dans SC1 sans être perturbé par les classes inter-points de vue.

2.5 AGREGATION ET MULTI-INSTANCIATION

La multi-instanciation est un concept suffisant pour modéliser les points de vue lorsque:
- on ne s'intéresse pas à une catégorie d'objet mais à des objets individuels,
- les différents points de vue n'ont pas de liens sémantiques entre eux.

Dans le cas contraire, il est préférable de combiner la multi-instanciation avec l'agrégation.
L'agrégation a pour principaux inconvénients de ne pas préserver l'identité de l'objet à travers ses différents points de vue et de compliquer la gestion des objets incomplets par celle des valeurs non significatives. Ces problèmes peuvent être résolus en prenant en compte la multi-instanciation au niveau des domaines des attributs.



Le domaine d'un attribut correspond à l'ensemble des valeurs possibles de cet attribut [ESC90b]. Les constructeurs U et u, utilisés dans les classes inter-points de vue ont la sémantique suivant:

A u B : est l'ensemble des instances qui appartiennent à A et pas à B.
union l'ensemble des instances qui appartiennent à B et pas à A
union l'ensemble des instances qui appartiennent à A et à B.

Un objet géométrique peut être décrit d'un point de vue fonctionnel et/ou cinématique et/ou géométrique.

A U B : est l'ensemble des instances qui appartiennent à A et à B.
Un évacuateur, pour être complet, doit obligatoirement admettre une représentation fonctionnelle, cinématique et géométrique.

Si une des représentations d'un évacuateur e3 n'est pas encore connue, rep-e3 est géré comme un objet incomplet et non comme un objet incohérent (§ 4.2).
Il est possible, dans la classe inter-points de vue, d'exprimer des liens sémantiques entre les différents points de vue, par exemple des contraintes d'intégrité ou des méthodologies d'élaboration.
Cette approche permet :
- d'élaborer simultanément plusieurs points de vue d'un même objet,
- d'imposer des ordres méthodologiques d'élaboration entre eux,
- de vérifier la cohérence interne d'un point de vue,
- de tester de manière incrémentale les cohérences inter-points de vue.

Malheureusement cette approche n'est pas sans inconvénient. Bien que l'identité d'un évacuateur à travers ses différents points de vue ait été préservée, celle-ci est perdue dans sa classe inter-points de vue. En effet, une instance de la classe inter-points de vue "évacuateur" ne représente pas vraiment un évacuateur, mais une instance ayant un attribut qui pointe vers le "vrai" évacuateur. D'autres solutions sont actuellement étudiées, leur but étant de préserver l'identité de l'objet dans les classes points de vue et inter-points de vue.
Malgré cet inconvénient nous pensons que la solution proposée ici constitue actuellement le meilleur compromis.

L'exemple de l'évacuateur de téléphérique a été entièrement traité dans le cadre du projet CIM-ONE[1] dans le but de modéliser en SHOOD toutes les informations issues d'un processus d'élaboration d'objets mécaniques.

3. EVOLUTION DES CONNAISSANCES

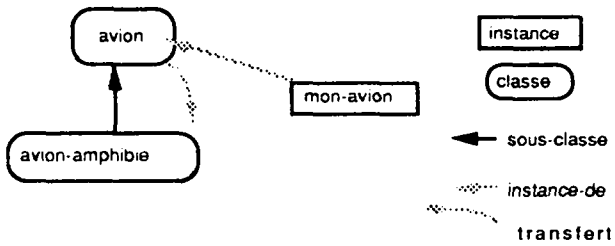
3.1 CLASSIFICATION

Un des mécanismes clés des systèmes de représentation de connaissances à objets concerne la classification : on cherche dans le graphe des classes, à quelles classes une instance particulière peut être rattachée [REC88]. La classification permet :
• d'affiner les connaissances détenues sur un objet,
• de modifier des connaissances préalablement acquises.

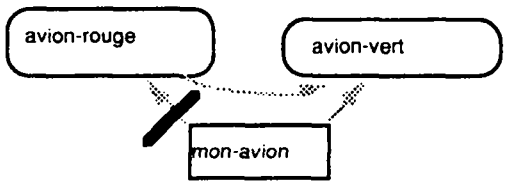
On affine la connaissance en permettant le rattachement d'une instance à des classes plus spécialisées. La prise en compte de connaissances relatives à un objet incomplet peut, par exemple, conduire à le re-classer dans une classe plus spécialisée. C'est le cas par exemple

[1] CIM-ONE est un projet qui regroupe cinq équipes universitaires lyonnaises et grenobloises. Son but est de concevoir un support d'intégration pour un environnement de Conception, Etude, Développement et Production assistée par ordinateur.

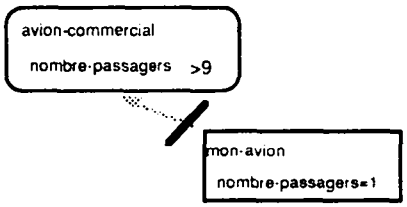
d'un avion que l'on peut rattacher à la classe *avions amphibies* parce que l'on sait à présent qu'il a des flotteurs.



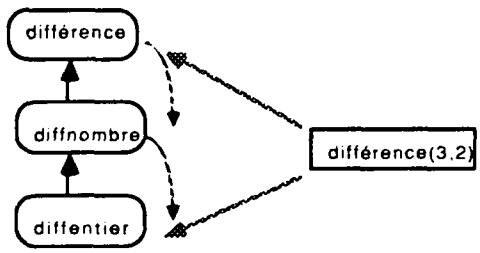
Si les connaissances détenues sur un objet sont modifiées, sa classe d'appartenance peut changer. Si mon avion est de *couleur rouge*, il fait partie de la classe *avion-rouge*. Si je le repeins en vert, il doit être classifié dans la classe *avion-vert*.



De même, toute évolution de schéma concernant l'ajout, la modification ou la suppression d'une classe ou d'un lien de spécialisation, peut entraîner une re-classification des instances impliquées. Si par exemple le concept d'*avion-commercial* évolue : on modifie la classe en rajoutant une contrainte sur l'attribut *nombre-passagers*, les instances ne vérifiant pas cette contrainte ne peuvent plus être rattachées à la classe.



La classification peut donc être une aide non négligeable pour gérer des évolutions de valeurs (dynamique des instances) mais aussi des évolutions de classes ou du graphe des classes (dynamique des schémas). Elle peut aussi être utilisée dans les modèles réflexifs et plus particulièrement dans les modèles méta-circulaires [COI87] pour implanter ou faire évoluer les fonctionnalités du système (dynamique du modèle) [MCL88].
Dans SHOOD par exemple, les méthodes sont modélisées par des classes, instances d'une méta-classe particulière META_METHODE. Les méthodes sont organisées en treillis en fonction du type de leurs paramètres ainsi que de leurs pré et post conditions. Cette approche est similaire à celle du système CLOS [KEE88]. La méthode *diffentier* est, par exemple une sous-classe de *diffnombre* elle-même sous-classe de *différence*. La recherche de la meilleure méthode pour effectuer un calcul donné, par exemple *différence(3,2)*, se fait par classification à partir de la classe correspondant à la méthode la plus générale. C'est ainsi que *différence(3,2)* est classifiée à partir de la classe *Différence* et rattachée à la classe la plus spécialisée *(Diffentier)*.

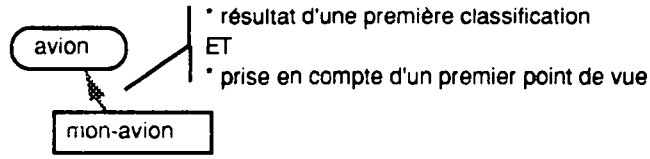


3.2 VERS "MIC" (MULTI-INSTANCIATION PAR CLASSIFICATION)

La classification permet de prendre en compte l'évolution des connaissances détenues sur un objet. La multi-instanciation prend en compte le fait qu'un objet peut être perçu sous différents points de vue. Le rattachement d'un objet à une classe peut être vu comme le résultat d'une classification de cet objet. *On ne peut rattacher l'objet qu'à une classe retenue par le mécanisme de classification.* C'est le cas, par exemple lorsque l'on crée un objet. La création d'un objet dans une classe consiste à rechercher si cette classe est possible pour l'objet avant de faire le rattachement effectif.

Par extension, on peut considérer la création d'un objet comme :
- le résultat d'une première classification de cet objet et
- la prise en compte d'un premier point de vue sur cet objet.

En effet, créer un objet O dans une classe C1 revient à :
- vérifier que l'objet O est *classifiable* dans C1, on dit que C1 est possible pour O.
- si c'est le cas, le rattacher effectivement à C1 par *un lien d'instanciation qui modélise un premier point de vue de l'objet.*

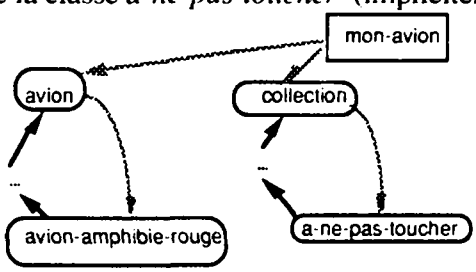


Prendre en compte un nouveau point de vue de l'objet O revient alors à tenter de le classifier dans une classe C2 qui draine une sémantique différente de C1 : C1 et C2 ne sont pas directement ou indirectement des spécialisations l'une de l'autre. Il suffit donc de:
- vérifier que O est classifiable dans la classe C2,
- puis, si c'est le cas, le rattacher effectivement à C2 par *un lien d'instanciation qui modélise un deuxième point de vue de l'objet.*

Affiner les connaissances détenues sur un objet O suivant le point de vue modélisé par C1 revient à présent à classifier O dans le graphe d'origine C1 de manière à :
- rechercher les sous-classes de C1 qui sont possibles pour O,
- puis rattacher O à une ou plusieurs sous-classes de C1 (qui correspondent à des sous-points de vue du point de vue modélisé par C1).

L'enrichissement d'un mécanisme de classification par le concept de multi-instanciation permet donc de gérer, pour une même instance et simultanément plusieurs énoncés du type :
- je veux que *mon-avion* soit instance des classes *avion* et *collection*,
- rechercher et rattacher *mon-avion* à la sous-classe de *collection* qui lui corresponde le mieux,
- rattacher mon-avion à la classe *avion-amphibie-rouge* (sous-classe de avion) dès que la condition *couleur=rouge* est vérifiée.

Si tout c'est bien passé *mon-avion* est instance de la classe *avion-amphibie-rouge* (donc implicitement de avion) et de la classe *a-ne-pas-toucher* (implicitement de collection).



Dans le cas du treillis de méthode (§ 3.1), la recherche et l'exécution de la meilleure méthode à exécuter pour le calcul de *différence(3,2)* correspond à l'énoncé:
- *différence(3,2)* est une instance de différence,

- rechercher la sous-classe de différence qui lui correspond le mieux, si elle est trouvée, rattacher *différence (3,2)* à cette classe. Dans ce dernier cas, l'utilisateur du mécanisme de multi-instanciation par classification est le système lui-même.

4.CLASSIFICATION D'OBJETS INCOMPLETS & INCOHERENTS

La *classification* consiste à chercher dans le treillis des classes, à quelles classes une instance particulière peut être rattachée. Par exemple si une *personne* a un âge inférieur à 18, le mécanisme de classification peut proposer d'en faire un *enfant* (si la classe *Enfant* est une sous-classe de la classe *Personne*). Le but est donc souvent de rattacher une instance à une classe plus spécialisée que sa classe de création.
Dans [REC88] l'exécution du mécanisme de classification délivre trois listes:
- l*es classes admissibles* pour lesquelles l'instance vérifie toutes les contraintes et auxquelles elle peut donc être rattachée. Une personne dont l'âge est inférieur à 18 peut être rattachée à la classe des enfants.
- *les classes possibles* pour lesquelles l'instance n'est pas encore en contradiction, mais qui pourrait le devenir en complétant les informations manquantes. Pour une personne dont l'âge est inconnu, la classe des enfants est possible. L'instance ne viole pas encore la contrainte d'âge (*âge inférieur à 18*).
- *les classes impossibles* c'est à dire pour lesquelles l'instance présente une contradiction, par exemple une violation de contrainte. Pour une personne de quarante ans, la classe des enfants est une classe impossible.

4.1 CLASSES POSSIBLES ET IMPOSSIBLES

Le partitionnement des classes en listes admissibles, possibles et impossibles est bâti sur le postulat que toutes les contraintes sont fortes, c'est à dire non violables. Dans des applications comme la CFAO, une telle approche n'est pas possible. L'objet à concevoir évolue en structure et valeur jusqu'à validité complète de son processus d'élaboration. Il est la plupart du temps incomplet et passe par des états incohérents. La tendance actuelle est d'intégrer dans les modèles de données des règles sémantiques effectuant un contrôle à la demande. Certaines règles de cohérence peuvent donc être violées [NGU89, FAV90] pendant de longues périodes de temps et les réactions du système en cas de violation ne sont plus standardisées [KOT88].

Dans SHOOD, une contrainte peut être définie comme forte ou faible. Une *contrainte forte* est non violable. C'est le cas, par exemple, des contraintes de types. Une *contrainte faible* est violable, c'est à dire qu'une instance peut être rattachée à une classe dont elle viole une ou plusieurs contraintes faibles. De la même manière, un attribut peut être *obligatoire* ou non. Un attribut obligatoire doit être forcément valué à la création de l'instance. Un attribut obligatoire est vu dans SHOOD comme une contrainte forte. Un attribut obligatoire non instancié est donc assimilé à une contrainte forte violée. Pour qu'une contrainte soit *décidable* il faut et il suffit que tous ses paramètres soient instanciés. Les paramètres d'une contrainte sont souvent des attributs de la classe. Par exemple, on exprime par une contrainte la cohérence entre les valeurs des attributs *âge* et *date de naissance* des instances de la classe Personne. Si l'âge ou la date de naissance ne sont pas instanciés, la contrainte n'est pas décidable.

Une contrainte non décidable ne dénote pas un état d'incohérence mais plutôt un état d'incomplétude de l'instance. Dans un environnement d'instances incomplètes, incohérentes et fortement évolutives, l'utilisateur doit avoir une idée précise de l'état de complétude et de cohérence de son instance et non pas uniquement une information de "tout ou rien". Nous préférons donc, ne partitionner les classes qu'en deux catégories les classes possibles et les classes impossibles. Puis donner pour chaque classe possible l'état exact de complétude et de cohérence de l'instance dans cette classe (§ 4.2). Nous n'avons alors plus que deux types de classes:

- les *classes impossibles* pour lesquelles l'instance viole au moins une contrainte forte.
- les *classes possibles* pour lesquelles aucune contrainte forte n'est violée par l'instance.

Une instance ne peut être rattachée qu'à des classes possibles. Elle peut rester rattachée à ces classes tant qu'une modification ou une valuation de valeur ne provoque pas de violation de contrainte forte.

4.2 CRITERE DE CLASSIFICATION

Le mécanisme de classification doit pouvoir répondre à des questions telles que:
- «est-il possible de rattacher i à C ?», mais aussi
- «quelle est la sous-classe de C qui correspond le mieux à i ?»

Il faut donc pouvoir juger parmi les classes possibles, celles qui sont *les meilleures*.
Pour chaque classe possible, l'utilisateur peut juger de l'opportunité de rattacher une instance à cette classe grâce à la donnée de la *structure significative, du pourcentage d'incohérence et du pourcentage d'incomplétude* de l'instance dans cette classe [RIE86, NGU89].

Ces différents concepts sont gérés dans SHOOD par un module de gestion de l'évolution des instances [FAV90]. Ce mécanisme gère l'évolution d'une instance à l'intérieur de sa classe grâce à des structures dites "significatives". Ces structures sont construites à partir des classes et reflètent tous les états possibles d'une instance en tenant compte des instances incomplètes et incohérentes. Chaque instance est alors rattachée dans sa classe à sa structure significative maximale: c'est à dire celle pour laquelle elle value tous les attributs et vérifie toutes les contraintes. Toutes les instances d'une classe valuant les mêmes attributs et respectant les mêmes contraintes sont donc rattachées à la même structure. A chaque structure sont associés un pourcentage d'incohérence et un pourcentage d'incomplétude communs à toutes ses instances.

Soient Nc(C) le nombre d'attributs facultatifs de la classe C et Ns(C,i) le nombre d'attributs facultatifs de la classe C instanciés par i. Le *pourcentage d'incomplétude* de i dans C, noté N(C,i), est égal au pourcentage d'attributs facultatifs qu'il reste à instancier (les attributs obligatoires sont forcément instanciés).

$$N(C,i) = \frac{Nc(C)-Ns(C,i)}{Nc(C)}$$

Soient Lc(C) le nombre total de contraintes faibles de la classe C et Ls(C,i) le nombre de contraintes faibles de la classe C indécidables ou satisfaites par i. Le *pourcentage d'incohérence* de i dans C, noté L(C,i), est égal au pourcentage de contraintes faibles actuellement violées par l'instance (les contraintes fortes ne pouvant pas être violées).

$$L(C,i) = \frac{Lc(C)-Ls(C,i)}{Lc(C)}$$

Plus les pourcentages d'incomplétude et d'incohérence sont faibles plus les instances sont complètes et cohérentes dans leurs classes.
Si le contexte ne pose pas d'ambiguïté, nous utiliserons par la suite les notations N pour le pourcentage d'incomplétude et L pour le pourcentage d'incohérence.

Pour une instance i à classifier, le mécanisme de classification donne pour chaque classe possible CP:
- la structure significative Scp de rattachement de i dans CP,
- le pourcentage d'incomplétude Ncp de i dans CP,
- le pourcentage d'incohérence Lcp de i dans CP.

Le triplet (Scp,Ncp,Lcp) pour chaque classe possible CP de i est appelé: *critère de classification* de i dans CP. Il permet de juger de l'opportunité de classifier i dans CP.

Si i appartient à d'autres classes que CP, l'utilisateur peut utiliser les critères de classification comme moyen de comparaison. En effet, une énorme croissance des pourcentages

d'incomplétude ou d'incohérence doit le faire réfléchir sur l'opportunité de rattacher l'instance i à Cp. Nous verrons au paragraphe 5.3, comment modéliser et automatiser cette comparaison.

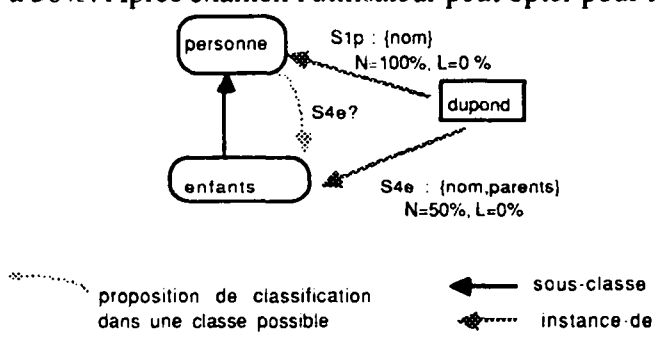
Exemple:
La classe Personne a deux attributs *nom* et *âge*. L'attribut nom est obligatoirement instancié. Personne admet deux sous-classes:
-la classe des enfants, qui admet un attribut supplémentaire : *parents* et une contrainte faible :*âge inférieur à 18*.
- la classe des adultes qui admet une contrainte faible: *âge supérieur à 18* (non traitée ici).

Les structures significatives de la classe Personne sont:
• S1p : { nom } à laquelle sont rattachées les personnes dont on ne connait que le nom.
N= 1/1=100% (un seul attribut facultatif qu'il reste à instancier),
L = 0 % (aucune contrainte faible violée)
• S2p : { nom , âge} à laquelle sont rattachées les personnes dont on connait le nom et l'âge,
N=0% (le seul attribut facultatif est instancié), L = 0%.

Les structures significatives de la classe Enfants sont:
• S1e = S1p à laquelle sont rattachés les enfants dont on ne connait que le nom,
N= 2/2 =100% (deux attributs facultatifs âge et parents non instanciés),
L = 0 % (la seule contrainte faible est indécidable).
• S2e : {nom, âge, inférieur(âge,18)}
C-à-d les enfants dont on connait le nom et l'âge et vérifiant la contrainte d'âge,
N=1/2=50% (l'attribut *parents* n'est pas instancié), L=0%.
• S3e : {nom, âge,not-inférieur(âge,18)}
C-à-d les enfants dont on connait le nom et l'âge et ne vérifiant pas la contrainte d'âge,
N=1/2=50%, L=1/1=100% (violation de la seule contrainte faible).
• S4e : {nom,parents}
C-à-d les enfants dont on connait le nom et les parents,
N=1/2=50% (l'attribut *âge* n'est pas instancié), L=0 %.
• S5e : {nom,âge,inférieur(âge,18),parents}
C-à-d même cas que S2e mais on connait en plus les parents,
N=0% (tous les attributs facultatifs sont instanciés), L=0%.
• S6e : {nom,âge,not-inférieur(âge,18),parents}
Même cas que S3e mais on connait en plus les parents , N=0% , L=1/1=100%.

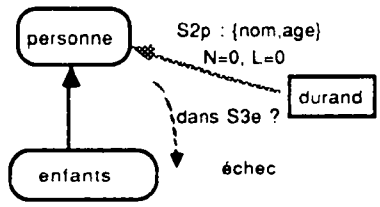
Soit l'instance p1 de la classe personne :
p1 : {nom = dupond}
p1 est rattachée à la structure S1p de la classe personne où ses pourcentages d'incomplétude et d'incohérence sont égaux à N= 100%, L=0%.

Supposons que l'on connaisse les parents de p1 et que l'on désire classifier p1 dans la classe des enfants. La classe enfant est possible: p1 value le seul attribut obligatoire (nom) et ne viole aucune contrainte forte. Le critère de classification de p1 dans la classe enfant est (S4e,50%,0%). Le pourcentage d'incohérence est inchangé, et celui d'incomplétude décroît: il passe de 100% à 50%. Après examen l'utilisateur peut opter pour le changement.



Soit l'instance p2 de la classe personne:

p2 : {nom=durand,âge=20}
p2 appartient à S2p où elle est complète et cohérente (N=0%,L=0%).
La classe *enfants* est possible puisque la contrainte d'âge est une contrainte faible. *durand* peut être rattaché à la structure S3e. Les pourcentages d'incomplétude et d'incohérence croissent. Après examen l'utilisateur renonce car il sait que l'âge de *durand* est une donnée sûre qu'il ne modifiera pas et donc que *durand* ne vérifiera jamais la contrainte d'âge : il n'y a pas d'espoir de regagner un pourcentage d'incohérence correct.



4.3 CONDITION DE CLASSIFICATION

Une classe CP est possible pour une instance i ssi i ne viole aucune contrainte forte de la classe. Nous avons vu que le critère de classification (Scp,Lcp,Ncp) permet à l'utilisateur de juger de l'opportunité de rattacher i à la structure Scp de CP. Il reflète l'état de complétude et de cohérence de l'instance.

Ce jugement peut-être spécifié et automatisé par une *condition de classification*. L'utilisateur n'a plus alors à examiner classe par classe celles qui seront retenues. Une condition de classification porte sur les critères de classification. Elle peut être absolue ou relative:

• Une condition *absolue* est exprimée en chiffrant les valeurs de N et L maximales qui sont tolérées. C'est ainsi qu'une condition de classification égale à "L=O%,N=O%" indique que seules sont retenues les classes possibles où l'instance est complète et cohérente.
Reprenons les exemples précédents (§ 4.2). Soit p1 une personne de nom = dupond. Dans la classe *Personne*, p1 est rattachée à la structure S1p (N=100%,L=0%). On connait ses parents et on accepte son rattachement à une sous-classe de Personne si le pourcentage d'incomplétude N de p1 dans cette sous-classe est au moins de 50%. La condition de classification de p1 dans une sous-classe de *Personne* est donc : N≤50%. C'est le cas de la classe *Enfants* où p1, enrichie de la valuation de l'attribut parents, peut être rattachée à la structure S4e (N=50%,L=0%) de *Enfants*.
• Une condition *relative* exprime que l'on ne veut pas de détérioration de l'instance (en complétude et/ou cohérence). Elle doit toujours être exprimée par rapport à l'état de l'instance dans une de ses classes d'appartenance. Dans l'exemple précédent, on aurait pu exprimer que p1 peut être classifiée dans une sous-classe possible de *Personne* si et seulement si son pourcentage d'incomplétude N dans cette sous-classe est inférieur ou égal au pourcentage d'incomplétude de l'instance dans Personne (noté N.Personne).

On dit qu'une classe C est *retenue* pour une instance i lors d'une demande de classification ssi:
- C est possible pour i,
- i satisfait, dans C, la condition de classification.

L'ensemble des *classes retenues* {CR} lors d'une demande de classification de i est un sous-ensemble de l'ensemble des classes possibles {CP} de i : {CR}⊆{CP}.
Tant que i n'est pas modifiée, {CP} reste inchangé. Par contre, deux demandes de classification de i, dont les conditions de classification sont différentes, ne rendent pas en général le même ensemble de classes retenues : {CR1} ≠ {CR2}.

5. LE MECANISME MIC DE SHOOD

Le mécanisme "MIC" (Multi-Instanciation par Classification) permet de classifier puis de rattacher une instance à une ou plusieurs classes. Il s'applique aussi bien sur des instances terminales que sur des classes. Le mécanisme de classification peut être exécuté sur n'importe quel sous-graphe, réduit éventuellement à une classe et incluant ou non une classe d'appartenance de l'instance à classifier. La classification permet donc :

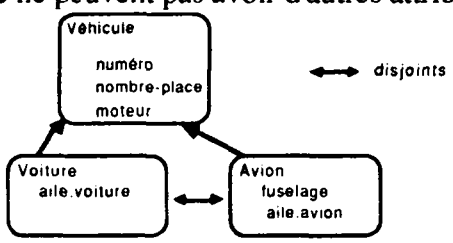
- non seulement de rattacher une instance à des classes plus spécialisées que ses classes d'appartenance. On parle alors d'affinage des connaissances.
- mais aussi à n'importe quelles classes du treillis pourvue qu'elles soient possibles et retenues par la condition de classification. On parle alors de prise en compte de nouveaux points de vue.

Une telle démarche est possible dans SHOOD car la relation de spécialisation respecte l'inclusion des ensembles (§5.1). En effet, quelles que soient ses classes d'appartenance (C1,C2..Ci) une instance appartient à toutes les super-classes de C1, C2..Ci et donc à la classe Objet. Nous pouvons donc toujours nous ramener à une recherche de classe plus spécialisée.
Nous abordons le mécanisme par un exemple d'utilisation de la primitive MIC (§5.2) qui permet de classifier un objet. Une commande MIC délivre un ou plusieurs graphes retenus, qui correspondent à des alternatives de rattachement. Le rattachement effectif de l'objet à une ou plusieurs classes d'un graphe retenu est mis en œuvre par la commande INST qui ne sera pas décrite ici.

5.1 SEMANTIQUE DU TREILLIS

Un mécanisme simple est un mécanisme qui s'appuit sur des règles générales que l'utilisateur maîtrise. Dans le cas de MIC, ces règles doivent exprimer la *sémantique du treillis* des classes et en particulier celle des relations inter-classes. Les relations inter-classes modélisent des liaisons structurelles mais aussi ensemblistes. Nous rappelons qu'une classe est un *ensemble d'instances* de même structure et de même comportement. Une liaison ensembliste permet l'expression de règles générales exploitables par MIC. Si cette liaison change, les règles aussi doivent changer. Le mécanisme de classification est alors différent. Dans SHOOD une relation inter-classe est modélisée comme un attribut : on peut donc la modifier comme un autre attribut [ESC90a, ESC90b].

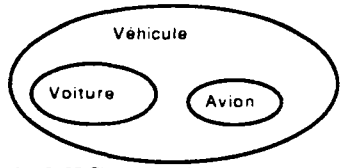
Dans SHOOD nous avons deux liens inter-classes : la *spécialisation* et la *disjonction*.
Leur expression structurelle est:
- Si C' est une sous-classe de C, alors C' hérite de la structure et du comportement définis dans C. Les Voitures et les Avions héritent, par exemple, des attributs numéro, nombre-place et moteur définis dans leur plus petite super-classe commune : la classe Véhicule.
- Si C' et C" sont disjointes alors la structure commune à C' et C", ainsi qu'à deux quelconques de leurs sous-classes, est réduite à celle héritée de leur(s) super-classe(s) commune(s) la (les) plus spécialisée(s). La structure commune à Voiture et Avion est l'ensemble des attributs hérités de Véhicule. Avion et Voiture ne peuvent pas avoir d'autres attributs communs.



Dans SHOOD les conflits de nom sont résolus par les concepts de nom complet d'attributs et d'attributs pré-déclarés [ESC90a]. Si un attribut est défini ou pré-déclaré dans une classe, son nom complet est égal au nom de l'attribut post-fixé de la classe de définition ou de pré-déclaration. Cela signifie que deux attributs de même nom mais n'ayant pas la même origine (classe de définition ou de pré-déclaration) ont des noms complets différents et sont considérés

comme sémantiquement différents. C'est le cas ici pour les attributs *aile* définis dans Voiture et Avion.

L'expression ensembliste des liens de spécialisation et de disjonction est:
- si C' (Voiture) est une sous-classe de C (Véhicule) alors l'ensemble des instances de C' est inclus dans celui de C,
- si C' (Voiture) et C" (Avion) sont disjointes alors les ensembles de leurs instances sont disjoints (ainsi que les ensembles de deux quelconques de leurs sous-classes).



Les règles générales sur lesquelles le MIC peut s'appuyer sont:
- Si i peut être rattachée C', alors elle peut aussi l'être à toutes les super-classes de C'.
- Si i ne peut pas être rattachée à C', alors elle ne peut pas être rattachée à une sous-classe de C'.
- Si i devient instance de C', alors elle devient instance de toutes les super-classes de C'.
- Si i devient instance de C' ou d'une de ses sous-classes, alors elle ne peut plus être rattachée à C" (disjointe de C') et à ses sous-classes....

La sémantique du lien de spécialisation, c'est à dire l'inclusion ensembliste, n'admet pas les exceptions, ce qui à pour conséquence d'interdire la surcharge ou le masquage de propriétés. La surcharge étant surtout utilisée pour les méthodes, nous avons résolu ce problème en organisant les méthodes en graphe de spécialisation indépendant de celui des classes de l'application (§ 3.1). Une méthode peut donc être spécialisée mais pas surchargée. De même, nous ne pouvons pas traiter le cas de "Gertrude" qui est une autruche et donc un oiseau mais qui ne vole pas. Nous pensons que dans la plupart des cas l'existence d'individus exceptionnels traduit une erreur de conception de schéma ou des schémas inadaptés qu'il faut modifier [BRA85, NGU89a].

5.2 LA COMMANDE : MIC

La demande de classification d'une instance existante d'identificateur *i* ou d'une nouvelle instance *(new)* est exprimée par la commande suivante :

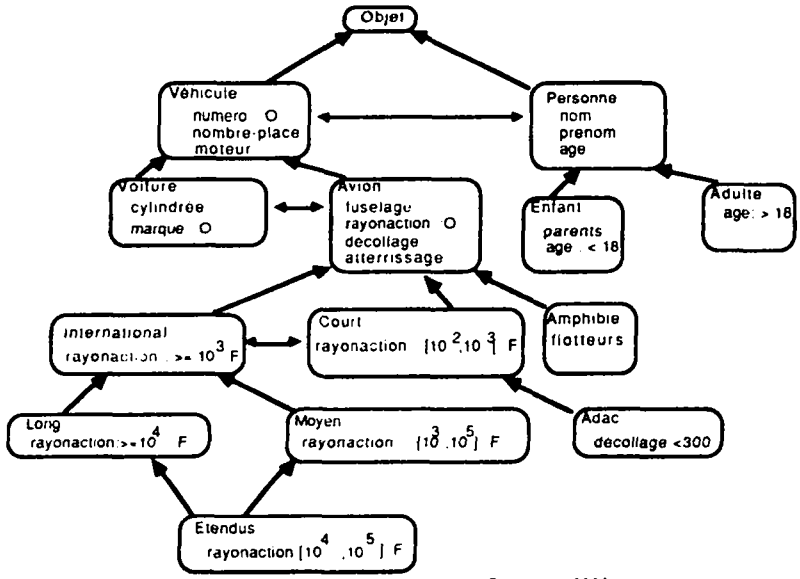
MIC (i / new) [dans Cd1) / (depuis Cd2 [sauf LCE])] [si cond] [valsattributs]

Dans SHOOD tout objet est instance d'un autre objet. Autrement dit une condition nécessaire et suffisante pour qu'un objet existe est qu'il soit rattaché par un lien d'instanciation à au moins une classe du graphe des classes. Un nouvel objet *(new)* sera effectivement créé dans la base lorsque l'objet sera explicitement rattaché *(commande INST)* à au moins une classe retenue par le mécanisme de classification. La création d'un objet peut donc être vue comme le résultat d'une première classification de cet objet.

La classification sera faite:
- dans une classe *(dans Cd1)*,
- ou dans le graphe de racine Cd2 *(depuis Cd2)* à l'exclusion des sous-graphes dont les racines sont des classes citées dans la liste des classes exclues *(sauf LCE)*,
- ou dans tout le treillis *()*.

[cond]: Une classe est retenue si elle est possible et si l'instance à classifier vérifie la condition de classification *cond* portant sur les critères de classification.
[valattributs] permet de donner des valeurs à des attributs non encore instanciés. En effet, le besoin de classification découle souvent de l'acquisition de nouvelles informations sur un objet. Une extension de la commande MIC permettra par la suite la modification des connaissances.

Le graphe retenu est constitué :
- des classes retenues : ce sera Cd1 si Cd1 est retenue, ou les classes retenues du sous-graphe de racine Cd2, à l'exclusion des classes exclues, ou toutes les classes retenues du treillis. Pour chaque classe retenue Cd, on donne son critère de classification (Scd,Ncd,Lcd),
- des classes d'appartenance de l'instance, s'il s'agit de la classification d'une instance existante,
- de toutes les super-classes des classes retenues et des classes d'appartenance.



ADAC signifie Avion à Décollage et Atterrissage Court. Le treillis est composé de nœuds qui représentent les classes (Objet,Véhicule,Avion...), de liens de spécialisation (Véhicule sous-classe de Objet...) et de liens de disjonction (Véhicule disjointe de Personne...). Chaque classe a un ensemble d'attributs qui sont hérités par ses sous-classes et éventuellement affinés (comme rayonaction). Les attributs obligatoires sont indiqués par *O* (numéro, rayonaction, marque) et les contraintes fortes par *F*.

MIC new depuis Véhicule sauf Voiture [numéro = F-GFXT, rayonaction= 120]

Un identificateur *il* est généré. Il sera définitivement conservé si la commande MIC est suivie d'une commande INST dont l'exécution provoque le rattachement de l'instance *il* à une ou plusieurs classes retenues.
Le graphe retenu est ici un arbre: (Objet Véhicule Avion ((Court Adac)Amphibie))

Si le graphe retenu comporte des classes disjointes le système délivre plusieurs graphes retenus ne comportant aucun lien de disjonction et correspondant à des *alternatives de rattachement.*
Dans l'exemple précédent, supposons que Adac et Amphibie appartiennent à deux branches disjointes (par exemple Adac disjointe de Amphibie). Si la commande MIC n'exclue pas une des deux branches (*sauf Amphibie* par exemple) le système fournit deux graphes retenus:
-(Objet Véhicule Avion Amphibie)
-(Objet Véhicule Avion Court Adac)

Certaines branches disjointes sont automatiquement exclues par:
-*[sauf LCE]*. La liste des classes exclues peut comporter un certain nombre de classes qui auraient été retenues et en conflit (car disjointes) avec d'autres classes retenues.
- *[valattributs]*. Dans la liste d'attributs cités, l'absence d'un attribut obligatoire exprime d'une manière indirecte une exclusion de classes. Dans l'exemple précédent, l'attribut *marque* est un attribut obligatoire de la classe *voiture*, il suffit de ne pas l'instancier pour être sûr que notre avion ne sera pas transformé en voiture : la clause *sauf voiture* est donc inutile.

A la suite d'une commande MIC, l'instance classifiée peut être rattachée à toutes les classes d'un sous-graphe retenu auxquelles elle n'appartient pas déjà.

6. CONCLUSION

Cet article propose une remise en cause des hypothèses de base concernant l'instanciation. Le concept d'instanciation par moulage parfait est remplacé par celui de multi-instanciation par moulage souple.
La multi-instanciation permet de préserver l'identité de l'objet à travers ses différents points de vue. Le moulage souple permet de prendre en compte le fait que certaines connaissances peuvent être incomplètes et momentanément incohérentes : l'objet n'est pas toujours un moulage parfait des entités conceptuelles qui le décrivent.
La multi-instanciation souple permet d'étendre les mécanismes de classification pour une prise en compte plus homogène et dynamique des évolutions d'objets. Des manipulations d'objets aussi différentes que la création, la prise en compte d'un nouveau point de vue, l'affinage des connaissances peuvent alors être traitées de manière similaire.
Le mécanisme MIC de SHOOD introduit les concepts de:
- classes impossibles et de classes possibles. Une classe est possible si l'objet à classifier ne viole aucune contrainte forte de la classe : l'objet vérifie la partie du moule obligatoire et inviolable,
- critère de classification qui permet de juger de l'opportunité de rattacher un objet à une classe possible. Un critère de classification est un triplet constitué de la structure significative, du pourcentage d'incomplétude et du pourcentage d'incohérence de l'objet dans la classe. Il modélise la distorsion de l'objet par rapport au moule idéal représenté par sa classe,
-condition de classification qui permet d'automatiser la classification d'un objet. Elle modélise la tolérance de distorsions.
Les travaux en cours portent sur l'utilisation de MIC pour gérer les reclassifications automatiques des instances lors de leurs mises à jour. La (re)classification d'une instance pourra être conditionnée par l'apparition d'un évènement, par exemple la modification d'une valeur d'attribut : lorsque mon-avion rouge est repeint en vert, il faut le reclassifier dans la classe des avions verts. Les mises à jour peuvent être plus ou moins importantes et donc avoir des conséquences plus ou moins grandes sur les instances concernées:
- L'instance reste rattachée aux mêmes structures significatives dans ses différentes classes de rattachement.
C'est le cas d'une modification mineure, par exemple, une modification de valeurs d'attributs qui n'interviennent pas dans les contraintes.
- L'instance peut changer de structures significatives dans ses classes de rattachement.
C'est le cas, par exemple, si l'on value un attribut jusqu'alors non instancié ou inversement si l'on supprime une valeur d'attribut non obligatoire. C'est aussi le cas d'une modification de valeurs d'attribut si cette modification inverse le résultat de l'évaluation d'une contrainte faible.
Pour gérer l'évolution des instances dans une classe, une relation d'ordre est définie entre les structures significatives de cette classe [NGU89a,FAV90]. Si, lors d'une mise à jour, l'évolution de l'instance satisfait la relation d'ordre, nous considérons que l'instance s'améliore: la distorsion est moins importante, l'objet est plus proche du moulage parfait. Le but est donc ici de guider vers une conception de plus en plus complète et cohérente des objets.
- L'instance peut changer de classes d'appartenance.
C'est le cas si la modification d'une valeur d'attribut provoque une violation de contrainte forte: la classe où cette contrainte est définie devient une classe impossible pour l'instance.

BIBLIOGRAPHIE

[BAK87] BANERJEE J. & al.
Semantics and implementations of schema evolution in object-oriented databases.
Proc. ACM SIGMOD Conference. San Fransisco, Mai 1987.

[BOB83] BOBROW D.G, STEFIK M.
The LOOPS manual : a data and object-oriented programming system for InterLisp.
Xerox PARC. Palo Alto. 1983.

[BRA85] BRACHMAN R.J.
"I Lied about the Trees"" Or, Defaults and Definitions in Knowledge Representation.
The AI Magazine, Fall 1985.

[CAR89] CARRE B. *Méthodologie orientée objet pour la représentation des connaissances.*
Thèse de Doctorat. Université des Sciences et Techniques de Lille Flandres Artois. 1989.

[COI87] COINTE P.
Metaclasses are First Class : the ObjVlisp Model. Proceedings OOPSLA'87, ACM, Octobre 1987.

[COP84] COPELAND G, MAIER D. *Making Smalltalk a database system..*
Proc. ACM SIGMOD Conf. Boston (Mass). 1984.

[ESC90] ESCAMILLA & al. *Représentation de connaissances dynamiques dans SHERPA.*
Actes Congrès INFORSID 90. Biarritz. Mai 1990.
Egalement Rapport de Recherche INRIA n° 1208. Avril 1990.

[ESC90a] ESCAMILLA J., JEAN P.
Relations verticales et horizontales dans un modèle de représentation de connaissances. Actes 4e Journées INRIA Bases de Données Avancées, Montpellier, Septembre 1990.

[ESC90b] ESCAMILLA J.,JEAN P.
Relationships in an Object Knowledge Representation Model. Proc. 2nd Intl .Conf. Tools for Artificial Intelligence, Washington D.C, Novembre 1990.

[FAV90] FAVIER V., RIEU D.
Manipulation d'objets dynamiques dans les bases de connaissances. Actes MICAD90, Paris, février 1990.

[FAV90b] FAVIER V., RIEU D.
Dynamique dans les Bases de Connaissances - Projet SHERPA. Revue MBD - Février 1990.

[FER84] FERBER J.
Quelques aspects du caractère self réflexif du langage MERING. Proc. 2° JLOO. Brest. 1984.

[FOX86] FOX M. & al.
Experiences with SRL : An analysis of a frame-based knowledge representation.
Proc. 1st Intl. Conf. Expert Database Systems, Kiawah Island (SC). 1984.

[GOL83] GOLDBERG A. ROBSON D. *Smalltalk 80 : the language and its implementation.*
Addison-Wesley Publ. Co. 1983

[HEI88] HEILER S., ZDONIK S.
Views,Data Abstraction, and Inheritance in the FUGUE Data Model.
Advances in Object-Oriented Databases, Bad Münster, FRG, Septembre 1988.

[KEE88] KEENS S.E
Object-oriented programming in Common Lisp. A programmer's guide to CLOS.
Addison-Wesley. 1988.

[KIM90] KIM W. *Object-oriented databases : definition and research directions.*
IEEE Trans. on Knowledge and Data Engineering., Vol. 2, n°3. Septembre 1990.

[KOT88] KOTZ M.H & al.
Supporting Semantic Rules by a Generalized Event/Trigger Mechanism.
Advances in DB technology ,EDBT 88, Venise , Mars 1988.

[MAI86] MAIER D. & al.
Development of an object-oriented DBMS.
Proc. OOPSLA '86 Conf. Portland (Oregon). September 1986.

[MAR90] MARINO O. & al.
Multiple perspectives and classification mechanism in object oriented representation.
Proc. ECAI Conf. Stockolm (S.), Juillet 1990.

[MAS89] MASINI G. et al.
Les langages à objets. InterEditions, Paris 1989.

[MCL88] McLEOD D.
A learning-Based Approach to Meta-Data Evolution in an Object-Oriented Database.
Advances in Object-Oriented Databases Systems, Bad-Munster , septembre 1988.

[MEY88] MEYER B. *Object-oriented software construction.* Prentice Hall. 1988.

[NGU89a] NGUYEN G.T., RIEU D.
Schema evolution in object-oriented database systems.
Data & Knowledge Engineering, North Holland, Vol. 4, n°1, Juillet 1989.
Egalement Rapport de Recherche INRIA n° 947. Décembre 1988.

[NGU89b] NGUYEN G.T., RIEU D.
Schema change propagation in object-oriented databases.
Proc XIth World Computer Congress, IFIP Congress '89. San Francisco (Ca.), Août 1989. Egalement Rapport de Recherche INRIA n° 1045. Juin 1989.

[NGU91a] NGUYEN G.T, RIEU D.
Database issues in Object Oriented Design. Tools '91, Paris, Mars 1991.
Egalement Rapport de Recherche INRIA n° 1373. Février 1991.

[NGU91b] NGUYEN G.T, RIEU D.
Representing design objects. Proc. 1st Intl. Conf. Artificial Intelligence in Design, Edinburgh (U.K), Juin 1991.

[REC88] RECHENMANN F.
Shirka : un système de gestion de bases de connaissances.
Manuel de référence. IMAG, laboratoire Artémis. juin 1988.

[RIE86] RIEU D., NGUYEN G.T.
Semantics of CAD Objects for Generalized Databases.
Proc. 23rd Design Atomation Conference, Las Vegas (Nevada), juin 1986.

[RIE87] RIEU D., NGUYEN G.T.
Contrôle automatique de cohérence sur des objets dynamiques.
3e Journées INRIA Base de Données Avancées, Port-Camargue, Mai 1987.

[RIE90] RIEU D. & al.
SHERPA: un support d'intégration pour le processus CEDF. Colloque CIM 90, Bordeaux, juin 1990.

[RIE91a] RIEU D., G.T NGUYEN
De l'Objet à l'Objet CFAO. Actes MICAD 91, Paris, Février 1991.

[RIE91b] RIEU D., G.T NGUYEN
Multiple instantiation and object representations. Soumis à publication.

[RIE91c] RIEU D., CULET A.
Classification et représentations d'objets. Actes Congrès INFORSID 91, Paris , Juin 1991.

[STE86] STEFIK M., BOBROW D.G
Object oriented programming: themes and variations. The AI magazine, janvier 86.

[UNL90] UNLAND R., SCHLAGETER G.
Object-Oriented Database Systems: Concepts and Perspectives.
Database Systems of 90s, Int. Symposium, Berlin , novembre 1990.

[VAN89] VAN DE RIET R.P. *Mokum: An Object-oriented active Knowledge base system.*
Data & Knowledge Engineering, North Holland, Vol. 4, n°1, Juillet 1989.

ISSN 0249-6399