Aspects classificatoires des systèmes à objets

R. Ducournau[1], M. Huchard[1], T. Libourel[1], A. Napoli[2]
[1] *LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France*
email: name@lirmm.fr
[2] *LORIA, UMR 7503, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France*
email: name@loria.fr

Résumé

Nous cherchons à mettre en évidence le phénomène de « classification » dans les divers systèmes à objets. Pour cela nous nous appuyons sur les acceptions usuelles de ce terme. Une « classification » est à la fois le processus et le résultat :

– de la division en classes d'un ensemble d'entités (découverte de classes extensionnelles),
– de la description d'un ensemble d'entités par des propriétés (découverte de classes intensionnelles),
– de l'organisation des classes les unes par rapport aux autres (classification de classes),
– du placement d'une entité relativement aux classes existantes (classification d'instances).

Notre propos sera illustré au travers de la pratique des systèmes à objets établie dans divers domaines de l'informatique : génie logiciel, représentation des connaissances, bases de données, etc.

Mots-clés Systèmes à objets, treillis de Galois, classification.

1 Introduction

Nous souhaitons identifier et définir de manière systématique les aspects des systèmes à objets qui relèvent d'une problématique de classification. L'approche objet connaît un succès non démenti dans la construction, l'utilisation et la maintenance de systèmes établis dans divers domaines de l'informatique : génie logiciel, représentation des connaissances, bases de données, etc. En ce qui concerne la construction de ces systèmes, le succès est dû à une étroite correspondance entre la modélisation et le monde réel. En effet, les entités informatiques construites se veulent être des images, les plus fidèles possibles, des objets du domaine d'application. Au cœur de cette représentation se retrouve naturellement, comme dans le monde réel, une classification des objets qui joue un rôle prépondérant [Lakoff, 1987; Smith et Medin, 1981; Mervis et Rosch, 1981]. Durant la phase d'exploitation du système, l'une des activités consiste à utiliser la classification issue de la phase de construction pour classer des entités puis pour calculer ou raisonner sur ces mêmes entités. L'intérêt de l'approche objet dans le cadre de l'exploitation des systèmes est certes principalement dû à la réutilisation aisée des constituants, mais également à la possibilité d'évolution des classifications.

Nous poursuivons deux objectifs. D'une part nous explicitons les questions de classification qui sont clairement traitées comme telles dans le cadre d'un développement objet standard, et nous nous attachons à spécifier les points sur lesquels la classification intervient de manière non formulée. D'autre part, nous montrons que l'approche objet apporte un point de vue particulier et de nouvelles interrogations sur le domaine de la classification [Ducournau *et al.*, 1998; Simon *et al.*, 1998].

Essentiellement, nous montrerons que l'approche objet, dans la partie construction, se fonde traditionnellement sur une classification « a priori », c'est-à-dire propose les classes à partir d'une description du domaine. En génie logiciel et en bases de données, de manière standard, les individus (instances) ne sont créés qu'ensuite, en respectant le schéma imposé par la classe (instanciation). En représentation des connaissances, dans les systèmes à prototypes [Dony *et al.*, 1998], ou plus récemment dans le cadre des recherches sur les données semi-structurées [Abiteboul, 1997; Abiteboul *et al.*, 1997], les individus peuvent être créés indépendamment des classes et éventuellement classés ensuite (classification d'instances). La classification de l'analyse de données [Diday *et al.*, 1982; Lebart *et al.*, 1995; Lerman, 1970] ou la classification conceptuelle de l'apprentissage [Gennari *et al.*, 1989; Langley, 1996], quant à elles, sont des classifications « a posteriori », issues de l'analyse d'un ensemble d'individus (ou exemples). Les classifications obtenues peuvent servir ensuite également à classer des individus (classification d'instances).

En ce qui concerne les objectifs de la classification, dans l'approche objet, nous en distinguerons trois en nous référant à [I. Bournaud, 1996]. La classification produite vise tout d'abord à donner un modèle des connaissances du domaine. Le deuxième aspect, qui est très spécifique de la programmation par objets est que la classification est la base du développement des programmes : les classes produites sont une partie du programme qui va s'exécuter. Dans le cas précis des systèmes à objets, le modèle de connaissances, tout comme le programme, servent de base à un processus de réutilisation : les abstractions produites lors d'une classification dans un contexte donné sont en effet supposées être réutilisables dans d'autres contextes. Enfin, la classification d'une instance permet une forme de déduction. Ce dernier objectif est primordial pour la représentation des connaissances à objets.

Nous présentons tout d'abord un modèle objet abstrait simplifié, illustré avec un exemple. Nous décrivons ensuite brièvement quelques usages de la classification dans le développement d'un système à objets (les phases d'analyse et de conception). Enfin nous évoquons, au travers d'une discussion, d'autres questions de classification que posent ces systèmes.

2 Un modèle de base pour les objets

Nous donnons ici une définition abstraite d'un système à objets issue des travaux de [Ducournau, 1996] que nous utilisons ici sous une forme extrêmement simplifiée. Les systèmes à objets ainsi décrits relèvent majoritairement des systèmes dits « à classes »[1].

Le modèle. Les constituants d'un système à objets S sont les suivants.

- Un ensemble de classes $\mathcal{C}_S$, un ensemble de propriétés $\mathcal{P}_S$, un ensemble d'instances $\mathcal{I}_S$.
- Une hiérarchie (une classification de l'ensemble des classes) H_S qui est un ordre partiel[2] $(\mathcal{C}_S, \leq_S)$.
- Chaque classe est décrite par une intension (l'ensemble de ses propriétés) qui lui est associée par l'application décroissante $Int_S \ : \ (\mathcal{C}_S, \leq_S) \longrightarrow (2^{\mathcal{P}_S}, \subseteq)$, où 2^X dénote l'ensemble des parties de X.
- Chaque classe est décrite en extension par un ensemble d'instances qui lui est associé par l'application croissante $Ext_S \ : \ (\mathcal{C}_S, \leq_S) \longrightarrow (2^{\mathcal{I}_S}, \subseteq)$.

1. Les systèmes « à prototypes » ne font donc pas partie de ce résumé.
2. Dans certains cas, cet ordre partiel se restreint à une arborescence.

– Enfin on dispose d'une description de l'état des instances par un ensemble d'expressions. Les expressions de la forme $(i\ p\ v)$ associent, à une instance $i \in \mathcal{I}_S$ et une propriété $p \in \mathcal{P}_S$, une valeur v qui peut être une instance.

On associe à tout système à objets S une sémantique dénotationnelle qui permet d'interpréter les constituants dans un domaine d'interprétation. Cette sémantique, inhérente à tout système à objets, répond principalement à la préoccupation de fonder logiquement les opérations de raisonnement, rejoignant ici la démarche des logiques de descriptions [Donini *et al.*, 1996; Nebel, 1990; Napoli, 1997].

Un exemple. L'exemple présenté ci-dessous nous sert à illustrer de manière concrète la syntaxe et la sémantique du système abstrait présenté. La figure 1 représente schématiquement l'ensemble des classes et des instances décrivant partiellement la notion de *suite* mathématique en programmation par objets.

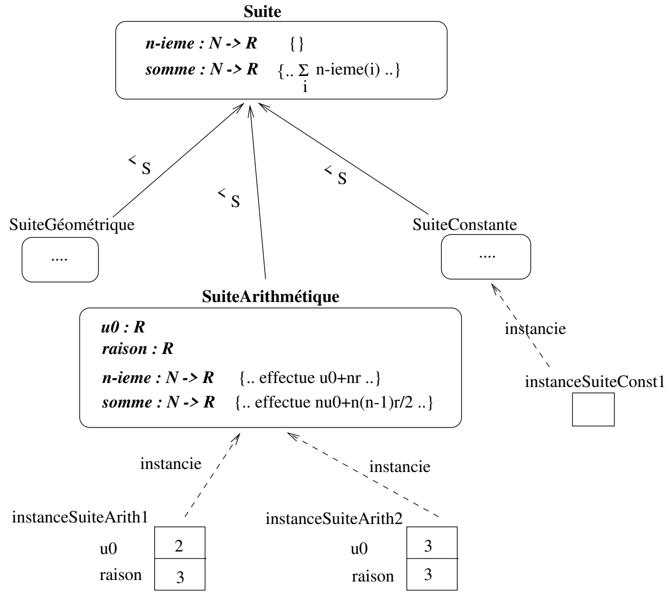

FIG. 1 – *Un ensemble de classes et d'instances*

Cet exemple s'interprète ainsi dans notre modèle.

$\mathcal{C}_S = \{Suite, SuiteArithmetique, SuiteGeometrique, SuiteConstante\}$

L'ordre $\leq_S$ est représenté sur la figure 1.

$\mathcal{I}_S = \{instanceSuiteArith1, instanceSuiteArith2, instanceSuiteConst1\}$

$Ext(Suite) = \{instanceSuiteArith1, instanceSuiteArith2, instanceSuiteConst1\}$

$Ext(SuiteArithmetique) = \{instanceSuiteArith1, instanceSuiteArith2\}$

$Ext(SuiteConstante) = \{instanceSuiteConst1\}$

Les propriétés sont, dans le cadre de la programmation par objets, des attributs ou des méthodes. Les attributs (comme $u0$ qui représente le premier terme d'une suite arithmétique) sont habituellement décrits par un type (pour $u0$, le type réel, noté R). Les méthodes, comme $n\text{-}ieme$ qui calcule le $n$ième terme de la suite, et $somme$ qui calcule la somme des n premiers termes, sont décrites par une « signature » (ensembles de départ et d'arrivée), et un « code » (une expression dans le langage décrivant le calcul). Les descriptions des propriétés sont simplifiées ci-dessous pour la lisibilité, en particulier une expression comme $n\text{-}ieme_{suite}$ représente la méthode $n\text{-}ieme$ telle qu'elle est décrite dans la classe $Suite$.

$Int(Suite) = \{n\text{-}ieme_{suite}, somme_{suite}\}$

$Int(SuiteArithmetique) = \{u0, raison, n\text{-}ieme_{suite}, somme_{suite}, n\text{-}ieme_{suiteArithmetique}, somme_{suiteArithmetique}\}$

Dans le cas de l'intension de la classe $SuiteArithmetique$, les propriétés $n\text{-}ieme_{suite}$ et $somme_{suite}$ sont héritées de la classe $Suite$, et elles sont de plus *masquées*[3] par les propriétés $n\text{-}ieme_{suiteArithmetique}$ et $somme_{suiteArithmetique}$ qui sont plus spécifiques et qui seront utilisées pour les calculs concernant les instances de $SuiteArithmetique$.

Une instance comme $instanceSuiteArith1$, notée ci-dessous $iA1$, est décrite par les expressions suivantes : $(iA1, u0, 2)$ $(iA1, raison, 3)$.

En programmation par objets standard, si la description de la méthode est donnée par sa signature accompagnée du code, il n'y a pas de valeur dans les instances (ce que l'on peut figurer par un domaine de valeurs vide). Cependant les méthodes sont invocables sur les instances, si l'on veut calculer par exemple la somme des 4 premiers termes de $iA1$, on utilise une expression comme $iA1.somme(4)$.

3 Quelques usages de la classification dans le développement de systèmes à objets

Le développement commence par des étapes d'analyse et de conception qui ne sont pas toujours clairement différenciées. L'analyse permet, au cours d'un dialogue avec les experts du domaine d'application, d'ébaucher une modélisation du monde réel. Pendant la phase de conception, on précise l'organisation du système, on affine les modèles issus de l'analyse, on prévoit comment on implémentera (architecture de composants[4], langage, système de gestion de bases de données, etc.).

Principes généraux de l'analyse et de la conception. Le point de départ est une spécification textuelle dont on tire une description plus formelle où sont mises en avant les classes et les relations structurelles qu'elles entretiennent, parfois aussi les échanges dynamiques entre classes qui traduisent les calculs qu'est supposé effectuer le logiciel, enfin l'architecture générale du système. La construction de cette description formelle ne se fait pas en une fois. Elle est plus généralement le fruit d'abstractions successives qui constituent une activité de classification, selon les propres termes de G. Booch [Booch, 1990]. Par exemple un texte ayant amené au développement des classes de la figure 1 pourrait être le suivant :

> « *Une* suite réelle *est une application d'une partie de* $\mathbb{N}$ *dans* $\mathbb{R}$*. On considère ici les suites définies sur* $\mathbb{N}$*.* »
>
> « *Une suite est* arithmétique *si la différence de deux termes consécutifs quelconques de la suite est une constante* r *que l'on appelle la raison de la suite. Une suite arithmétique vérifie les deux propriétés suivantes :* $\forall n, u_n = u_0 + n.r$ *et la somme des* n *premiers termes vaut* $n.u_0 + n.(n-1).r/2$*.* »
>
> « *Une suite est* géométrique *si (...) Une suite est* constante *si (...)* »
>
> « *Soit une suite* u*, on doit pouvoir lui appliquer les opérations suivantes : calcul du* $n^{\text{ième}}$ *élément de la suite, calcul de la somme des n premiers éléments.* »

Une première étape consiste à extraire de ce texte ce qui relève de la notion d'instance, et ce qui relève de la notion de classe ou de relation entre classes. Dans un deuxième temps, les mécanismes d'abstraction portent sur les classes elles-mêmes et peuvent être, suivant la spécification

3. Ceci invite à étendre le modèle de système à objets qui est présenté au début du §2.

4. Architecture du système, centralisé ou distribué, client-serveur, etc.

textuelle disponible, descendants (par spécialisation) ou ascendants (par généralisation) ou mixtes (en itérant des étapes ascendantes et descendantes).

Dans le cas du génie logiciel et des bases de données, un formalisme graphique est utilisé pour exprimer les descriptions issues de la phase d'analyse, par exemple celui d'OMT [Rumbaugh *et al.*, 1991] ou UML [Rat, 1997]. Il existe des outils d'aide pour établir ces représentations graphiques, mais le travail de classification est effectué par l'esprit humain.

Dans le cas d'une méthode comme OMT, les instances sont utilisées, en particulier au travers de « diagrammes d'instances », essentiellement à titre d'exemples, pour illustrer les classes ou déterminer la multiplicité des relations.

Une part automatisable. Dès qu'une première représentation relativement formalisée est disponible, on peut lui appliquer des processus d'affinement, processus qui méritent d'être définis avec précision si on veut les automatiser. La sémantique joue ici un rôle important puisqu'elle fonde les modifications apportées au système. Ainsi on peut reconnaître des relations de généralisation/spécialisation entre classes à partir de l'inclusion de leurs intensions, ou chercher à factoriser des propriétés communes, ce qui nécessite fréquemment l'ajout de nouvelles classes.

Ces deux problèmes peuvent être résolus de diverses manières, mais en particulier le recours à la notion de treillis de Galois [Barbut et Monjardet, 1970; Godin et Mili, 1993; Dicky *et al.*, 1996] permet de travailler dans un cadre mathématique précis et rigoureux, et parallèlement d'avoir un guide lors de la classification (des classes).

> **Construction à l'aide d'un treillis de Galois**
> Etant donné un système S_1, trouver un système S_2 tel que $(\mathcal{C}_{S_2}, \leq_{S_2})$ soit isomorphe au treillis de Galois associé à la relation binaire $\mathcal{R} \subseteq \mathcal{C}_{S_1} \times \mathcal{P}_{S_1}$ définie par $(C,p) \in \mathcal{R}$ ssi $p \in Int_{S_1}(C)$.

Avec une telle construction, on s'assure de trois propriétés très fortes :

- la relation de spécialisation $\leq_S$ coïncide avec la relation d'inclusion entre l'intension des classes ;
- la factorisation est maximale, en effet, pour chaque propriété, il existe une unique classe maximale pour l'ordre qui la possède dans son intension (en termes objets, qui la « déclare ») ;
- c'est la structure qui assure une factorisation maximale en utilisant le moins de classes possibles pour factoriser.

On peut aussi être amené à regrouper ou dissocier des classes, afin de mieux représenter le domaine d'étude. De telles stratégies se rapprochent des opérateurs de fusion ou d'éclatement utilisés dans les méthodes de classification conceptuelle [Fisher, 1987; Gennari *et al.*, 1989].

4 Discussion

Après ce premier aperçu, nous discutons brièvement d'autres aspects de la classification dans le cadre des systèmes à objets qui méritent une définition et une étude plus précise.

Une relation de spécialisation plus fine. Dans le modèle simplifié ci-dessus, nous identifions la spécialisation entre classes à l'inclusion entre les intensions de ces classes. En pratique, la spécialisation peut être plus riche ; en particulier elle peut s'appuyer sur une notion de spécialisation des propriétés. Par exemple, la méthode $n\text{-}ieme_{suite}$ se spécialise par la méthode $n\text{-}ieme_{suiteArithmetique}$, la méthode $somme_{suite}$ se spécialise par la méthode $somme_{suiteArithmetique}$.

Cela nous invite à préciser dans le modèle précédent que l'intension d'une classe C_2 ne devrait étendre l'intension d'une super-classe C_1 que par ajout de propriétés qui sont soit nouvelles, soit des spécialisations des propriétés de C_1 [Snyder, 1987; Wegner et Zdonik, 1988]. Toutes les relations de spécialisation ne sont pas calculables (on ne peut pas toujours comparer les codes de deux méthodes), mais celles qui le sont peuvent être utilisées pour organiser les classes.

Le cas des relations. Un concept supplémentaire est généralement mis en évidence dans les activités de modélisation, celui de *relation*[5] entre objets, ou entre classes. Pour le modèle que nous avons présenté à la section 2, ces relations se traduisent soit par des classes, soit par des attributs, c'est justement un travail du concepteur que d'effectuer ce choix parce que dans leur grande majorité les systèmes à objets ne proposent pas la notion de relation comme constituant de base ou en proposent une version trop pauvre [Rumbaugh, 1987; Rumbaugh *et al.*, 1991]. La définition des processus de classification doit, en tout état de cause, inclure cette notion.

Les instances. Les instances jouent bien évidemment un rôle fondamental. En particulier, il nous semble important de mettre en avant les points suivants :

- ce sont toutes les instances « potentielles » qui déterminent la sémantique des classes ;
- les instances peuvent être la « matière première » des mécanismes de classification (au sens de l'analyse de données [Diday *et al.*, 1982; Lebart *et al.*, 1995; Lerman, 1970]) et d'abstraction (au sens de la classification conceptuelle [Gennari *et al.*, 1989; Fisher, 1987]) ;
- dans le cadre de la représentation des connaissances, la classification d'instances est la base du raisonnement, elle permet par exemple de déduire des valeurs manquantes pour une instance donnée ;
- les classes sont généralement amenées à évoluer, et dans ce cas, reclasser les instances peut devenir nécessaire, mais peut poser des problèmes complexes [Huchard, 1999].

Pour bien exprimer ces problèmes, il faut prendre en compte les contraintes existant sur la relation classe/instance (mono-instanciation, multi-instanciation) et qui varient suivant les systèmes.

Les métaclasses. Certains systèmes à objets admettent la notion de « classe de classes », encore appelée « métaclasse » [Cointe, 1987; Briot et Cointe, 1989]. Une classe entretient avec sa métaclasse le même rapport qu'une instance avec sa classe. Toutes les questions concernant la classification de métaclasses sont ouvertes.

Les aspects dynamiques. Ils font l'objet de diverses représentations (diagrammes d'états, diagrammes de collaboration, diagrammes d'événements, etc.) pour lesquelles il est important de définir une notion de spécialisation. Quelques travaux ont déjà été effectués sur la classification des opérations [Borgida, 1981; Prieto-Diaz, 1991; Kristensen *et al.*, 1987; Harito Shteto, 1997].

Les patrons. Dans le domaine du génie logiciel s'est récemment développée la notion de « patron », qui représente une solution d'analyse, de conception ou de programmation répondant à un problème donné. Ces patrons sont actuellement classés par objectif dans des sortes de catalogues [Gamma *et al.*, 1995], mais il serait intéressant de définir une notion de spécialisation entre patrons.

5. Sous ce terme « relation », nous évoquons ce que les méthodes d'analyse et de conception appellent lien entre objets et association entre classes, autre que généralisation/spécialisation.

5 Conclusion

Tout au long du cycle de vie d'un système à objets (développement, exploitation, rétroconception), nous retrouvons ce même souci d'abstraction où la classification joue un grand rôle : abstraction d'instances pour trouver des classes, abstraction de classes pour trouver des métaclasses, abstraction de classes pour trouver de nouvelles classes, pour n'évoquer que la partie structurelle des objets. L'un des intérêts de cette approche est que toutes ces abstractions sont généralement réutilisables.

Nous soutenons la thèse selon laquelle tout système à objet devrait, quel que soit le contexte (programmation, base de données, etc.), toujours disposer d'un véritable « classifieur », comportant en particulier une aide à la construction et à la maintenance des classes et des hiérarchies de classes, ainsi qu'une gestion de l'évolution des instances et de la propagation sur les instances de l'évolution des classes.

Références

[Abiteboul *et al.*, 1997] S. Abiteboul, D. Quass, J. McHugh, J. Widom et J.L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1:68–88, 1997.

[Abiteboul, 1997] S. Abiteboul. Querying Semi-Structured Data. Dans *Database Theory – ICDT'97, 6th International Conference, Delphi, Greece*, rédacteurs F. Afrati et P. Kolaitis, Lecture Notes in Artificial Intelligence 1186, pages 1–18. Springer, Berlin, 1997.

[Barbut et Monjardet, 1970] M. Barbut et B. Monjardet. *Ordre et classification — Algèbre et combinatoire (2 tomes)*. Hachette, Paris, 1970.

[Booch, 1990] G. Booch. *Object Oriented Design with Applications*. Benjamin/Cummings Publishing, Reading (MA), USA, 1990.

[Borgida, 1981] A. Borgida. On the definition of specialization hierarchies for procedures. Dans *Proceedings of IJCAI'81, Vancouver, Canada*, pages 254–256, 1981.

[Briot et Cointe, 1989] J.-P. Briot et P. Cointe. Programming with explicit metaclasses in Smalltalk-80. Dans *Proceedings of OOPSLA'89, New Orleans (LA), USA*, Special issue of ACM SIGPLAN Notices, 24(10), pages 419–431, 1989.

[Cointe, 1987] P. Cointe. Metaclasses are first class: The ObjVlisp model. Dans *Proceedings of OOPSLA'87, USA*, Special issue of ACM SIGPLAN Notices, 22(12), pages 156–167, 1987.

[Dicky *et al.*, 1996] H. Dicky, C. Dony, M. Huchard et T. Libourel. On automatic class insertion with overloading. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96*, 31(10):251–267, 1996.

[Diday *et al.*, 1982] E. Diday, J. Lemaire, J. Pouget et F. Testu. *Eléments d'analyse des données*. Dunod, Paris, 1982.

[Donini *et al.*, 1996] F.-M. Donini, M. Lenzerini, D. Nardi et A. Schaerf. Reasoning in description logics. Dans *Principles of Knowledge Representation*, rédacteur G. Brewka, pages 191–236. CSLI Publications, Stanford (CA), USA, 1996.

[Dony *et al.*, 1998] C. Dony, J. Malenfant et D. Bardou. *Les langages à prototypes*, pages 227–254. In Ducournau et al. [1998], 1998.

[Ducournau *et al.*, 1998] rédacteurs R. Ducournau, J. Euzenat, G. Masini et A. Napoli. *Langages et modèles à objets — État des recherches et perspectives*. Collection Didactique D-019. INRIA, Le Chesnay, 1998.

[Ducournau, 1996] R. Ducournau. Les systèmes classificatoires. Rapport technique, LIRMM, 1996.

[Fisher, 1987] D. Fisher. Knowledge Acquisition via Incremental Concept Clustering. *Machine learning*, 2:139–172, 1987.

[Gamma *et al.*, 1995] E. Gamma, R. Helm, R. Johnson et J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Gennari *et al.*, 1989] J.H. Gennari, P. Langley et D. Fisher. Models of Incremental Concept Formation. *Artificial Intelligence*, 40:11–61, 1989.

[Godin et Mili, 1993] R. Godin et H. Mili. Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'93*, 28(10):394–410, 1993.

[Harito Shteto, 1997] M. Harito Shteto. *Technique à objets pour les algorithmes de configuration du réseau téléphonique commuté*. Thèse d'Informatique, Université Pierre et Marie Curie (Paris VI), 1997.

[Huchard, 1999] M. Huchard. Classification de classes contre classification d'instances. Evolution incrémentale dans les systèmes à objets basés sur des treillis de Galois. *Actes de la conférence Langages et Modèles à objets*, pages 179–196, 1999.

[I. Bournaud, 1996] I. Bournaud. *Regroupement conceptuel pour l'organisation de connaissances*. Thèse d'informatique, Université Pierre et Marie Curie (Paris 6), 1996.

[Kristensen *et al.*, 1987] B.B. Kristensen, O.L. Madsen, B. Møller-Pedersen et K. Nygaard. Classification of Actions or Inheritance also for Methods. Dans *Proceedings of* ECOOP*'87, Paris, Special issue of Bigre 54, Lecture Notes in Computer Science 276*, pages 109–118, 1987.

[Lakoff, 1987] G. Lakoff. *Women, Fire, and Dangerous Things (What Categories Reveal about the Mind)*. The University of Chicago Press, Chicago, 1987.

[Langley, 1996] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann Publishers, San Francisco, California, 1996.

[Lebart *et al.*, 1995] L. Lebart, A. Morineau et M. Pinon. *Statistique exploratoire multidimensionnelle*. Dunod, Paris, 1995.

[Lerman, 1970] I.C. Lerman. *Les bases de la classification automatique*. Gauthier-Villars Éditeurs, Paris, 1970.

[Mervis et Rosch, 1981] C.B. Mervis et E. Rosch. Categorization of Natural Objects. *Annual Review of Psychology*, 32:89–115, 1981.

[Napoli, 1997] A. Napoli. Une introduction aux logiques de descriptions. Rapport de Recherche RR-3314, INRIA, 1997.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence 422. Springer Verlag, Berlin, West Germany, 1990.

[Prieto-Diaz, 1991] R. Prieto-Diaz. Implementing Faceted Classification for Software Reuse. *Communications of the ACM*, 34(5):88–97, 1991.

[Rat, 1997] Rational Software Corporation. *UML v 1.1, Semantics*, version 1.1 édition, septembre 1997. ad/97-08-04.

[Rumbaugh *et al.*, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy et W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall Inc. Englewood Cliffs, 1991.

[Rumbaugh, 1987] J. Rumbaugh. Relations as Semantic Constructs in an Object-Oriented Language. Dans *Proceedings of OOPSLA'87, Orlando (FL), USA*, pages 466–481, 1987.

[Simon *et al.*, 1998] A. Simon, A. Napoli, J. Lieber et A. Ketterlin. Aspects de la classification dans un système de représentation des connaissances par objets. Dans *Actes des sixièmes rencontres de la Société Francophone de Classification (SFC'98), Montpellier*, rédacteurs O. Gascuel et G. Caraux, pages 205–209. Publication de l'École d'Agronomie de Montpellier, 1998.

[Smith et Medin, 1981] E.E. Smith et D. Medin. *Categories and Concepts*. Harvard University Press, Cambridge, MA, 1981.

[Snyder, 1987] A. Snyder. Inheritance and the Development of Encapsulated Software Components. Dans *Research Directions in Object Oriented Programming*, rédacteurs B. Shriver et P. Wegner, pages 165–188. The MIT Press, Cambridge, Massachusetts, 1987.

[Wegner et Zdonik, 1988] P. Wegner et S.B. Zdonik. Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like. Dans *Proceedings of ECOOP'88, Oslo, Norway*, Lecture Notes in Computer Science 322, pages 55–77. Springer-Verlag, Berlin, 1988.