Mettez un Treillis dans votre Modèle ! Réflexions sur la Place et les Moyens de la Classification dans l'Ingénierie des Modèles

Michel Dao, Marianne Huchard, Amine Mohamed Rouane Hacene, Cyril Roume, Petko Valtchev

Mettez un treillis dans votre modèle !

Réflexions sur la place et les moyens de la classification dans l'Ingénierie Des Modèles

Dao M.[1], Huchard M.[2], Rouane Hacène M.[3], Roume C.[3], Valtchev P.[3]

1 France Télécom R&D/MAPS/AMS, 38-40 rue du Général Leclerc, 92794 Issy les Mx cedex 9

2 LIRMM, UMR 5506, CNRS et Univ. Montpellier 2, 161 rue Ada, 34392 Montpellier cedex 5

3 DIRO, Université de Montréal, C.P. 6128, Succ. "Centre-Ville", Montréal, Québec, Canada, H3C 3J7

Résumé

Dans cet article nous nous interrogeons sur la place de la classification dans l'Ingénierie Des Modèles. Nous envisageons deux scenarii de transformation de modèles basés sur des opérations de construction automatique de classifications. Nous introduisons l'analyse formelle de concepts (AFC) et sa version relationnelle comme un cadre théorique utile pour définir ces opérations. Nous appliquons à notre contexte un patron associant trois transformations de modèles qui met l'AFC au cœur d'un processus de transformation de classifications, qui considère toute description d'un ensemble d'individus par un ensemble de propriétés pour produire une classification de ces mêmes individus. Une expérience de reconstruction de modèles statiques UML, qui tire parti de l'analyse relationnelle de concepts en concevant les entités des modèles UML comme des individus, est ensuite décrite et nous en tirons les leçons en proposant de nouveaux axes de travail.

1. Introduction

La classification tient une place particulière dans l'intelligence humaine. Réunir objets ou idées selon des propriétés communes et abstraire ces regroupements dans des « classes » structure la connaissance, facilite la mémorisation et contribue à certaines formes de compréhension et de raisonnement sur le monde qui nous entoure. Nous considérerons pour cet article la définition selon laquelle la classification est le processus et le résultat des activités suivantes : division en classes d'un ensemble d'entités (classes décrites en extension) ; description d'un ensemble d'entités par des propriétés caractéristiques (classes décrites en intension) ; organisation des classes entre elles (classification de classes) ; placement d'une entité dans une classe (classification d'instances).

Les approches par objets, qui s'intègrent à présent dans l'ingénierie des modèles comme l'un des espaces technologiques possibles [1], se sont caractérisées par une introduction systématique de la classification dans les représentations informatiques, notamment du fait de la recherche d'une certaine correspondance entre les entités du monde réel et les entités informatiques [2, 3, 4]. Les classifications y sont au cœur des modèles et des programmes.

- Les *classes*, les *interfaces*, les *types* regroupent des données, des valeurs, des objets présentant des régularités, et les décrivent en intension par des caractéristiques communes, telles que des attributs ou des opérations.

- Les *hiérarchies* de classes ou d'interfaces reflètent l'organisation des concepts d'un domaine métier ou classifient les artefacts logiciels utiles à l'expression d'un problème et de sa solution. Elles véhiculent des sémantiques variées, notamment les notions de spécialisation/généralisation, de sous-typage ou d'héritage de propriétés, suivant que l'on se place plutôt du point de vue des modèles ou plutôt du côté des programmes.
- Les *paquetages*, les modules, sont des classes extensionnelles d'entités généralement regroupées de manière thématique.
- Les *patrons* de conception ou de programmation, qui généralisent des régularités de taille réduite dans les architectures, sont des classes de figures architecturales décrites en intension.
- Les *frameworks*, qui définissent des architectures réutilisables dans des familles d'applications, sont fondamentalement construites par abstraction d'applications existantes ou projetées.

Menés à l'origine dans les deux domaines de l'analyse de données [5, 6] et de l'intelligence artificielle (apprentissage symbolique non supervisé) [7, 8, 9], les travaux sur la construction automatique de classifications touchent actuellement de nombreux domaines apparemment aussi différents que la fouille de données [10, 11], la reconstruction phylogénétique [12] ou le génie logiciel. Dans ce dernier contexte sont concernées certaines classifications mentionnées plus haut telles que la construction ou l'analyse de hiérarchies [13, 14], le regroupement en modules [15] ou la découverte de patrons [16, 17]. Nous pensons que cette prégnance de l'abstraction et de la classification ne peut que se prolonger et s'accentuer dans l'ingénierie des modèles, en dépassant largement le cadre des objets. Elle devra se décliner en considérant les aspects *productifs* et non pas seulement *contemplatifs* des modèles [1], c'est-à-dire en mettant en avant les opérations de construction mais aussi de transformation des classifications.

Dans cet article, nous resterons dans le paradigme objet et nous nous concentrerons sur la généralisation d'éléments des modèles statiques tels que les classes, les interfaces, mais aussi les associations et les propriétés en tant qu'entités de premier plan dans les modèles. Nous montrerons quelques cas typiques de transformations de classifications qui nous semblent avoir leur place dans un développement dirigé par les modèles (section 2). Nous exposerons un cadre théorique, l'analyse formelle et l'analyse relationnelle de concepts, pour définir et automatiser ces opérations, ainsi qu'un patron de transformation de classifications (section 3). Nous tirerons quelques conclusions d'une expérience menée dans le contexte d'un projet RNTL. Nous montrerons les directions à suivre pour résoudre les difficultés rencontrées lors de la mise en œuvre pratique, notamment l'écueil de la langue et l'explosion combinatoire des classifications (section 4), puis nous conclurons (section 5).

2. Quelques cas typiques de transformation de classifications

Nous retiendrons ici la définition de [18] selon laquelle une transformation de modèles consiste à générer automatiquement un modèle cible à partir d'un modèle source selon une définition de transformation qui comprend un ensemble de règles décrivant comment un ou plusieurs éléments du modèle source peuvent être transformés en un ou plusieurs éléments du modèle cible.

Toutes les étapes d'un développement dirigé par les modèles peuvent être touchées par des opérations portant sur une transformation de classification. Dans cette partie nous étudierons deux exemples plus en détails : une opération de réingénierie sur un programme Java, contraint par l'héritage simple, qui fait émerger la classification métier sous-jacente (section 2.1) ; une transformation de diagramme de classes UML motivée par le souci d'en augmenter le degré d'abstraction (section 2.2).

2.1. D'un ensemble de classes Java vers un ensemble de classes UML

Dans ce premier exemple de transformation, nous souhaitons retrouver, dans un ensemble de classes Java (spécification de bas niveau), les concepts qui y ont été décrits et comprendre leur organisation (spécification de haut niveau). Dans les classes Java nous conservons seulement les signatures de méthodes publiques et les variables de classes constantes également publiques. Ce choix est dicté par la volonté de

construire une classification selon le point de vue des types abstraits implémentés par les classes. Cette hiérarchie est présentée figure 1 (devant chaque propriété se trouve placé un nom qui la résume et servira dans la suite). Nous pouvons y relever quelques défauts tels que la répétition de certaines signatures dans des classes incomparables (comme `side() :float`) ou l'absence de spécialisation entre `Square` et `Rhombus`, due à la contrainte d'héritage simple.

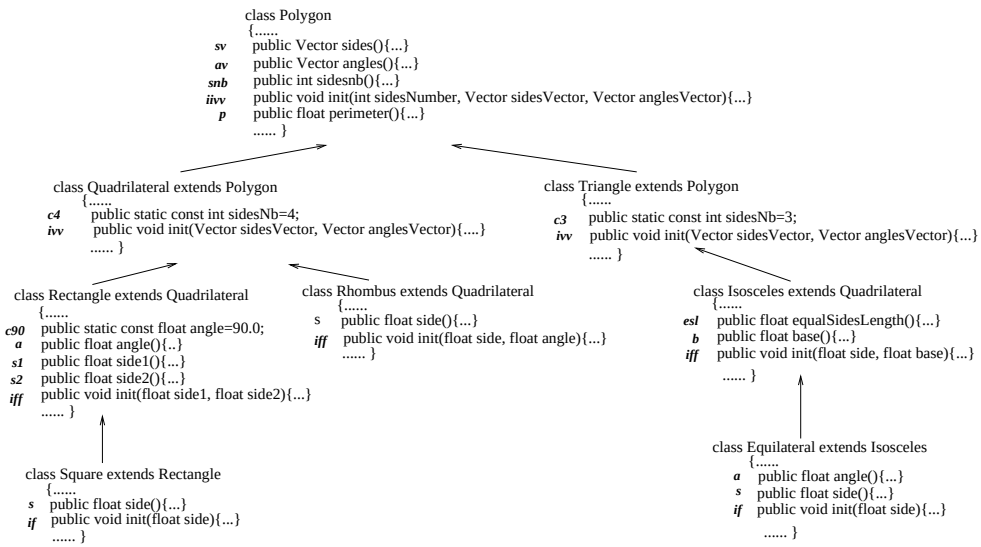

FIG. 1 – Classification des polygones, en Java

Dans une transformation vers UML de la hiérarchie des classes Java (figure 2), divers points peuvent être améliorés en ce qui concerne le niveau d'abstraction et les relations de spécialisation.

- les caractéristiques répétées peuvent être factorisées (sur l'exemple de la figure 2, `side` est factorisée dans la classe `RegularSidePolygon`);
- des liens de spécialisation manquant (comme entre `Square` et `Rhombus`) peuvent apparaître ;
- des généralisations intéressantes peuvent être mises à jour, telles que `RegularPolygon` qui représente les polygones dont les angles (resp. les côtés) sont égaux.

Cette transformation, qui serait qualifiée d'« exogène » selon [19], se définit pour l'essentiel à l'aide d'une règle consistant à transformer toute intersection non vide d'ensembles de propriétés des classes Java (considérées par deux, puis par trois, etc.) en une classe UML. Les relations de spécialisation/généralisation sont ajoutées par respect de l'inclusion des ensembles de propriétés des classes UML.

2.2. Transformation d'un diagramme de classes UML par généralisation systématique

L'élaboration d'un diagramme de classes UML se fait généralement en plusieurs itérations car il est rarement possible d'expliciter et d'organiser dans un premier mouvement les concepts du système ou du domaine que l'on souhaite décrire. Lors de ces itérations, l'une des activités courantes consiste à généraliser : à rechercher des régularités entre certains concepts déjà détaillés pour en tirer de nouvelles abstractions. Dans le cadre des diagrammes de classes UML, ces abstractions ne sont pas uniquement des abstractions de classes ; d'autres entités telles que les associations, les attributs et les opérations peuvent également faire l'objet d'un processus de généralisation. Nous proposons, figure 3, un premier modèle décrivant des situations d'enseignement guidé par un premier souci d'explicitation des informations : un instituteur est affecté à une classe qui se compose d'élèves tandis qu'un professeur enseigne pour une promotion qui se compose d'étudiants.

La figure 4 propose un ensemble de généralisations imaginables pour les entités présentées à la figure précédente. Elle fait apparaître notamment la notion d'enseignant qui enseigne à un groupe composé lui-même d'*enseignés*. Il s'agit là d'une transformation de modèles qui vise à construire des abstractions utiles pour comprendre la structure sous-jacente aux deux parties du modèle d'origine, pour la réutiliser,

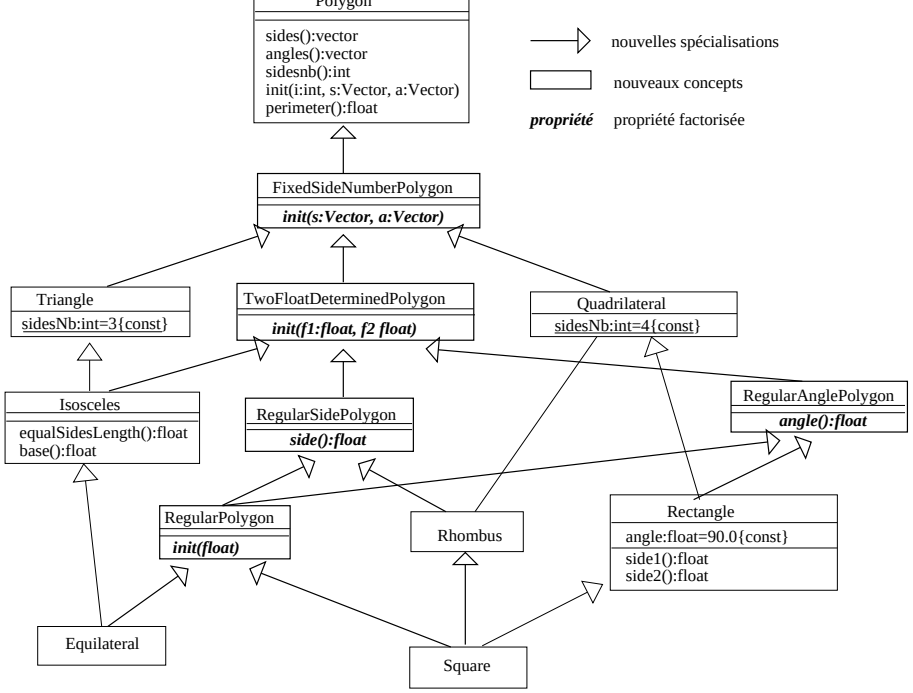

FIG. 2 – Nouvelle classification des polygones, en UML

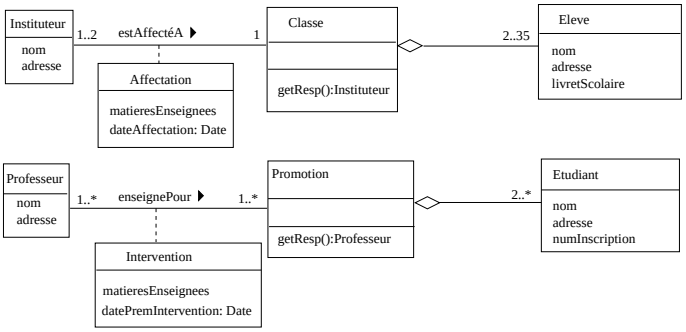

FIG. 3 – Situations d'enseignement

pour la simplifier dans certains cas. Quoique plus complexe, cette transformation peut aussi se définir par des règles apparentées à celle de la section 2.1. Les généralisations créées peuvent être aussi comprises comme le cœur commun de modèles conçus indépendamment (les deux parties connexes du modèle d'origine), ou encore le modèle final peut être vu comme un modèle issu d'une fusion de deux autres modèles [20], ce que [19] appelle transformation avec sources multiples.

On peut se poser à juste titre la question de savoir si une transformation endogène si complexe entre dans le cadre strict de l'Ingénierie Des Modèles. Nous pensons que c'est le cas, et cette problématique se rapproche à la fois du *refactoring* et de la normalisation des schémas de bases de données : l'objectif est d'améliorer les qualités d'un modèle (par exemple en excluant toute redondance), sans changer son comportement externe.

3. Une usine à classifications : l'Analyse Formelle et Relationnelle de Concepts

Nous étudions à présent les moyens dont nous pouvons disposer pour construire de manière automatique ces classifications.

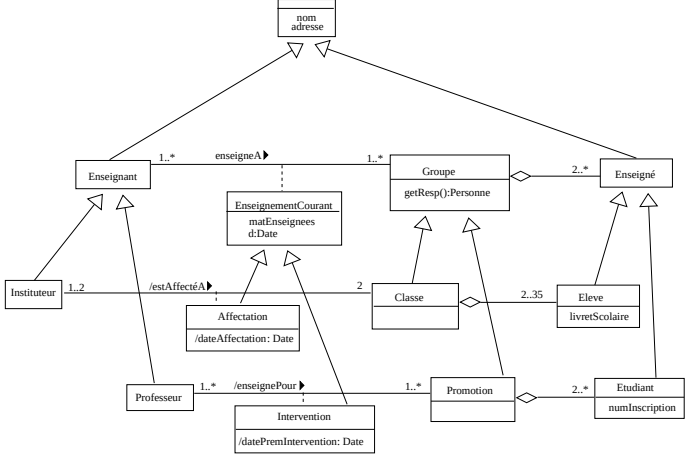

FIG. 4 – Généralisation des situations d'enseignement

3.1. Analyse Formelle de concepts (treillis de Galois)

L'Analyse Formelle de Concepts (AFC) est une théorie d'analyse de données issue de la théorie des treillis [21, 22, 23] qui permet de faire émerger un ensemble de concepts par abstraction d'un ensemble d'individus décrits par des propriétés.

Le modèle de base en AFC est le *Contexte formel*, plus exactement un triplet $\mathcal{K} = (E, P, I)$ où E est l'ensemble des entités ou individus (objets formels), P l'ensemble des propriétés (attributs formels) et I la relation d'incidence qui associe une entité à une propriété : $(e, p) \in I$ lorsque e possède la propriété p. La table 5 présente un contexte formel qui encode une description des classes Java de la figure 1.

La classification construite à partir de la relation d'incidence, appelée *treillis de concepts*, est un ensemble de concepts partiellement ordonnés.

- un concept est un couple $c = (X, Y)$, où $X \subseteq E$ représente l'ensemble des individus couverts par le concept (extension, notée $extent(c)$) et $Y \subseteq P$ correspond à l'ensemble des propriétés vérifiées par tous les individus (intension, notée $intent(c)$). Si on note $X' = \{p \in P \mid \forall e \in X,\ (e, p) \in I\}$ l'ensemble des propriétés partagées par les individus de X et $Y' = \{e \in E \mid \forall p \in Y,\ (e, p) \in I\}$ l'ensemble des individus qui détiennent les propriétés de Y, alors un concept se définit formellement comme un couple (X, Y) avec $X \subseteq E$, $Y \subseteq P$, $X' = Y$ et $Y' = X$. Pour la table 5, $\{\{E, S\}, \{iIVV, SNB, p, sV, aV, iVV, iFF, s, a, iF\}\}$ est un concept, ce qui peut se repérer visuellement car un concept correspond à un pavé maximal de 'x' dans la table (aux permutations de lignes et de colonnes près).
- L'ordre partiel ordonne les concepts par spécialisation. Il se définit d'après l'inclusion des extensions ou l'inclusion inverse des intensions.
  Ainsi le concept $\{\{E, S\}, \{iIVV, SNB, p, sV, aV, iVV, iFF, s, a, iF\}\}$ spécialise le concept $\{\{I, Re, Rh, E, S\}, \{iIVV, SNB, p, sV, aV, iVV, iFF\}\}$.

Dans certaines applications, seules certaines parties du treillis de concepts sont utilisées, dans le cadre de la construction de hiérarchies de classes notamment un sous-ordre caractéristique, la *sous-hiérarchie de Galois* a été fréquemment employé. Ce sous-ordre est induit par les concepts (X, Y) engendrés par un individu ou une propriété, c'est-à-dire tels que $X = \{e\}'$ pour un certain $e \in E$ ou $Y = \{p\}'$ pour un certain $p \in P$. La construction du treillis à partir du contexte peut être perçue comme une transformation de modèles (réversible). Une règle simple définit cette transformation puisque tout pavé maximal de 'x' dans le modèle source (le contexte) est transformé en concept dans le modèle cible (le treillis). L'ordre entre concepts n'est pas issu d'une règle de transformation, mais d'une contrainte portant sur les modèles cibles.

La nouvelle classification des polygones (figure 2) peut être déduite du treillis de concepts ou de la sous-hiérarchie de Galois construite à partir du contexte décrit dans la table 5.

	iIVV	SNB	p	sV	aV	iVV	iFF	C4	s	C3	a	S2	S1	eSL	c90	b	iF
0 -P	X	X	X	X	X												
1 -T	X	X	X	X	X	X				X							
2- Q	X	X	X	X	X	X		X									
3-Rh	X	X	X	X	X	X	X	X	X								
4-I	X	X	X	X	X	X	X			X				X		X	
5-Re	X	X	X	X	X	X	X	X			X	X	X		X		
6-E	X	X	X	X	X	X	X		X	X	X			X		X	X
7-S	X	X	X	X	X	X	X	X	X		X	X	X		X		X

FIG. 5 – *Le contexte formel décrivant les polygones*

Le concept $\{\{E, S\}, \{iIVV, SNB, p, sV, aV, iVV, iFF, s, a, iF\}\}$ donne ainsi naissance à la classe UML `RegularPolygon` (nommée *a posteriori* par le concepteur) qui déclare la propriété `iF`, hérite des autres et se trouve être super-classe des classes `Equilateral` et `Square` qui partagent précisément les propriétés $\{iIVV, SNB, p, sV, aV, iVV, iFF, s, a, iF\}$ et en possèdent chacune d'autres en propre. Le concept $\{\{I, Re, Rh, E, S\}, \{iIVV, SNB, p, sV, aV, iVV, iFF\}\}$, quant à lui, est à l'origine de la classe UML `TwoFloatDeterminedPolygon`. L'ordre partiel entre concepts engendre les relations de spécialisation entre les classes, par exemple l'inclusion des extensions des deux concepts précédents engendre la spécialisation entre `RegularPolygon` et `TwoFloatDeterminedPolygon`.

3.2. Analyse Relationnelle de concepts

L'Analyse Formelle de Concepts rend de nombreux services dans la construction de classifications, mais ne permet pas de prendre en compte des descriptions *relationnelles* dans lesquelles un individu est décrit par ses liens avec d'autres individus. C'est une situation commune, qui se présente par exemple si l'on s'intéresse à décrire les entités UML de la figure 3. Dans notre étude si nous considérons seulement les classes UML comme des individus, le diagramme fait apparaître un individu (la classe `Professeur`) qui se décrit par des propriétés `nom` ou `adresse` (description non relationnelle) mais aussi par l'association `enseignePour` qui l'attache à la classe `Promotion`, un autre individu (description relationnelle).

Plutôt que de considérer un seul contexte formel comme dans l'AFC, l'Analyse Relationnelle de Concepts (ARC ou RCA en anglais) [24, 25] privilégie un modèle de description des entités comprenant une collection de contextes ainsi qu'un ensemble de relations connectant deux contextes en spécifiant les liens entre individus de ces deux contextes.

Plus formellement, une *famille de contextes relationnels (FCR)* est un couple $\mathcal{R} = (K, R)$ où K est un ensemble de contextes $K_i = (E_i, P_i, I_i), 1 \leq i \leq n$ et R un ensemble de relations $r_j \subseteq E_p \times E_q$ avec p, q dans $[1, n]$. E_p est le domaine de départ de r_j, et E_q son domaine d'arrivée.

Dans le cadre d'UML, la famille de contextes relationnels contient les entités que l'on juge de premier plan suivant le type de transformation recherché et la précision attendue. Les individus peuvent être notamment les classes du méta-modèle UML telles que `Class`, `Association`, `Operation`, `Property` (en UML 2.0). Les attributs du méta-modèle UML, comme `name`, `isAbstract` sont les propriétés. Les associations (leurs rôles) du méta-modèle fournissent enfin les relations puisqu'elles relient les individus : par exemple l'association qui détient le rôle `ownedAttribute` relie dans le méta-modèle UML 2.0 des individus `Class` à des individus `Property`.

La figure 6 présente partiellement la famille de contextes relationnels correspondant à l'exemple de la figure 3. Seuls trois contextes sont présentés et nous avons restreint le diagramme aux quatre classes `Classe`, `Promotion`, `Eleve` et `Etudiant`, ainsi qu'aux propriétés et aux associations. Les extrémités d'associations sont nommées d'après les classes qui les typent. Volontairement les contextes ne contiennent pas toutes les informations, c'est ici pour simplifier l'exemple et nous reviendrons sur ce

point dans la section suivante. Par exemple, pour les six premières `Property`, les multiplicités ne sont pas données et aucune valuation de `agregation` n'est inscrite. On peut noter que lorsqu'un individu possède une caractéristique (par exemple `upper`≤ 35), il en possède également les généralisations (pour l'exemple `upper`$\leq *$). Comme relations inter-individus nous avons retenu uniquement `type`, ainsi que les deux paires de rôles opposés (`class`, `ownedAttribute`) et (`association`, `memberEnd`).

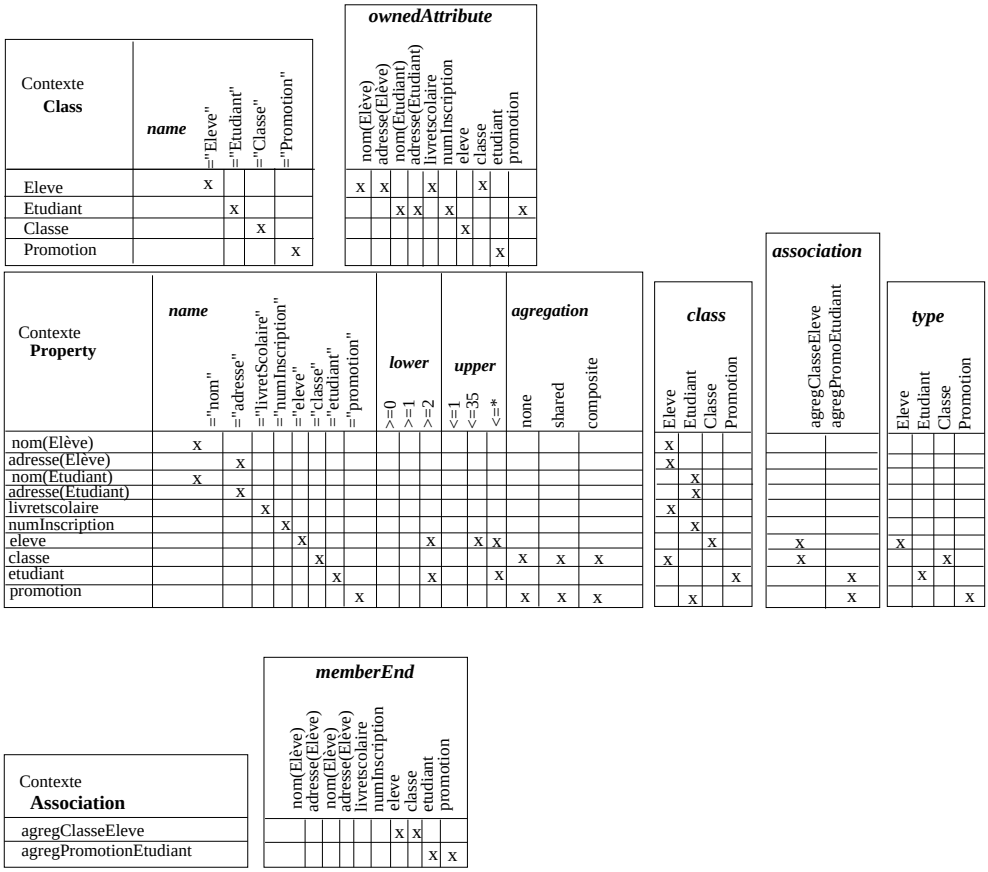

FIG. 6 – Famille de contextes relationnels pour l'enseignement (partielle)

Les relations complètent les contextes, c'est pourquoi elles apparaissent à leur droite : pour chaque contexte $K_i = (E_i, P_i, I_i)$ étendu par les relations qui ont pour départ E_i on peut construire le treillis de Galois ou la sous-hiérarchie de Galois associée (par la suite nous ne parlerons que des treillis).

Par exemple, on peut construire le treillis $\mathcal{L}_{Property}$ associé au contexte `Property` complété par les relations `class`, `type` et `association`.

On y trouvera notamment le concept $c_1 = (\{nom(Eleve), nom(Etudiant)\}, \{name = "nom"\})$ qui présente ce que les attributs `nom` des classes `Eleve` et `Etudiant` partagent ou encore le concept $c_3 = (\{eleve, etudiant\}, \{lower \geq 2, upper \leq *\})$ qui regroupe deux extrémités d'association. c_1 s'interprète en particulier comme un attribut généralisant les deux attributs `nom(Eleve)` et `nom(Etudiant)`.

Si l'on s'intéresse à présent au contexte `Class`, on peut noter qu'aucun concept ne pourra en être déduit car on n'y trouve pas deux classes partageant des propriétés. Pourtant on aura pu remarquer qu'élèves et étudiants possèdent tous un nom et une adresse, et appartiennent à un groupe. Cette information peut être déduite de la construction des concepts du contexte étendu `Property`. La relation `ownedAttribute` va subir une transformation pour intégrer les connaissances acquises lors de la construction des concepts de $\mathcal{L}_{Property}$: son domaine d'arrivée, à l'origine constitué des individus de $E_{Property}$ est remplacé par l'ensemble des concepts de $\mathcal{L}_{Property}$ construit à l'étape précédente. Pour un individu $o_i \in E_{Class}$ et un concept $c_j \in \mathcal{L}_{Property}$, o_i possède l'attribut <`ownedAttribute` $: c_j >$, si `ownedAttribute`$(o_i) \cap extent(c_j) \neq \emptyset$, ce que l'on peut interpréter comme le fait que o_i possède un attribut spécialisant la généralisation d'attribut c_j. Le résultat est présenté figure 7 et on peut y remar-

quer que cette fois les classes possèdent des caractéristiques communes et que l'on pourra construire un concept généralisant `Eleve` et `Etudiant` ainsi qu'un concept généralisant `Classe` et `Promotion`.

Par un même procédé, les généralisations d'extrémités d'associations telles que les concepts c_3 et c_4, mais utilisées cette fois pour `memberEnd` permettront de faire apparaître une généralisation des deux associations.

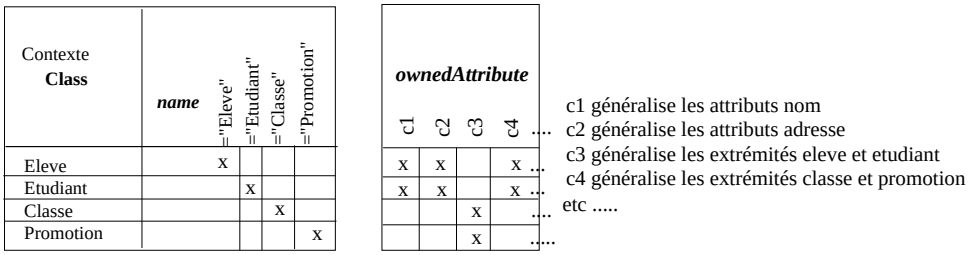

FIG. 7 – Evolution du contexte `Class`

L'opération que nous venons de décrire porte le nom d'*échantillonnage relationnel*. Étant donné un contexte $K_j = (E_j, P_j, I_j)$, le treillis associé $\mathcal{L}_j$ sert de base pour l'échantillonnage des relations $r \subseteq E_i \times E_j$. Une relation r_e, destinée à remplacer r pour progresser dans le calcul est construite de la manière suivante. Pour un individu $o_i \in E_i$ et un concept $c_j \in \mathcal{L}_j$, $(o_i, < r : c_j >) \in r_e$ si $r(o_i) \cap extent(c_j) \neq \emptyset$.

La procédure de construction de classification, MULTI-FCA [24], est une procédure qui consiste d'abord à construire les treillis initiaux de tous les contextes de la FCR (partie non relationnelle), ensuite à mettre à jour à chaque étape ces treillis par ajout des attributs relationnels issus de l'échantillonnage des relations correspondantes (partie relationnelle). L'algorithme s'interrompt lorsqu'on atteint une étape où, pour chaque contexte étendu par les relations, le treillis obtenu est isomorphe à celui obtenu à l'étape précédente. Accompagnée d'une étape de réinterprétation des treillis obtenus à l'étape finale, MULTI-FCA permet de transformer le diagramme de la figure 3 en celui de la figure 4.

3.3. Un patron de transformation de classification autour de l'AFC

Sur la figure 8 nous retraçons les différentes opérations de transformation effectuées par une triple application du patron de transformation de modèle proposé dans [26] : transformation de UML vers FCR (en bas à gauche), puis de FCR vers des collections de treillis (`Multi-FCA`), puis des collections de treillis vers UML (en bas à droite). Nous employons les notions et les notations de [1] que nous comprenons ainsi dans notre contexte :

- la relation *ReprésentationDe* (notée μ) associe le *modèle* et le *système étudié* : par exemple le formalisme FCR est une représentation du langage FCR (ensemble des familles de contextes relationnels) ; le diagramme UML 1 est une représentation du monde de l'enseignement ; le modèle de transformation UML vers FCR est une représentation d'une fonction de transformation (par fonction on entend ici un ensemble de couples).
- la relation *EstConformeA* (notée $\mathcal{X}$) associe un méta-modèle à un modèle valide : par exemple les deux modèles UML enseignement sont des instances du méta-modèle UML ; la famille de contextes relationnels enseignement ($rcfe$) respecte le formalisme FCR.
- la relation $\in$ relie un modèle (à comprendre comme une phrase conforme au méta-modèle qui serait vu comme une grammaire) à un langage (au sens de l'ensemble des phrases).

Nous avons trouvé intéressant d'utiliser ce patron de transformation car il nous permet de nous affranchir des implémentations d'outils que nous avons réalisées pour présenter de manière générique le processus. Il est utile aussi pour en expliciter les points problématiques. La triple application du patron de transformation peut être généralisée de manière à mettre toute opération de calcul automatique de classifications (transformation Contexte formel vers treillis de l'AFC, transformation FCR vers collections de treillis de l'ARC) au cœur d'un patron général de calcul de classification. Nous faisons une telle proposition avec la figure 9 qui présente notre point de réflexion actuel sur le sujet. La partie haute du

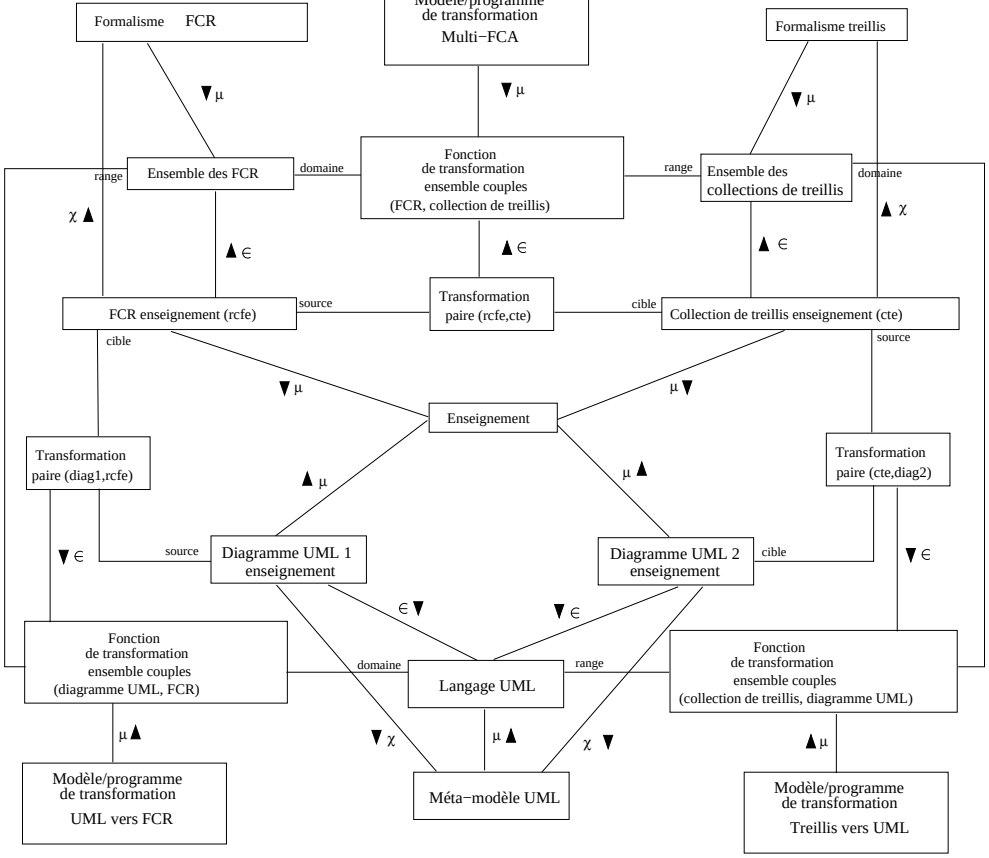

FIG. 8 – Transformations de classifications UML

schéma, qui est concernée par ce qui touche à l'AFC ou ses extensions, nous pose peu de problèmes de définition. Essentiellement ce sont les aspects d'efficacité algorithmique qui sont en jeu, au moment de l'implémentation des modèles de transformation des contextes ou famille de contextes vers les treillis ou les collections de treillis. Les points sensibles et sujets à de multiples variations de spécification dans ce patron de transformation sont :

- les modèles de transformation LCS (Langage de Classification Source) vers AFC : lorsque le LCS est le langage UML, le choix des informations placées dans les contextes est crucial pour d'une part obtenir les généralisations réutilisables, qui vont inspirer le concepteur pour simplifier son diagramme, d'autre part éviter la création de généralisations sans intérêt ; dans notre premier exemple où le LCS était Java, un choix a dû être fait également pour extraire des informations de niveau conceptuel à l'intérieur du code Java, écrit naturellement avec des contraintes techniques.
- les modèles de transformation des treillis vers un LCC (Langage de Classification Cible) : lorsque le LCC est UML, des spécialisations qui n'ont plus d'intérêt peuvent par exemple disparaître ; ainsi les attributs `nom` des classes `Eleve` et `Etudiant` n'ont pas véritablement de raison de rester dans un diagramme final après les avoir généralisés par l'attribut `nom` placé dans la classe `Personne` (figure 4).

La qualité et la facilité d'utilisation des outils mettant en œuvre ce patron de transformation de classification tiennent beaucoup à ces étapes sensibles : elles doivent être intégrées sous forme de composants paramétrables et surtout remplaçables afin de permettre à un concepteur d'expérimenter diverses hypothèses.

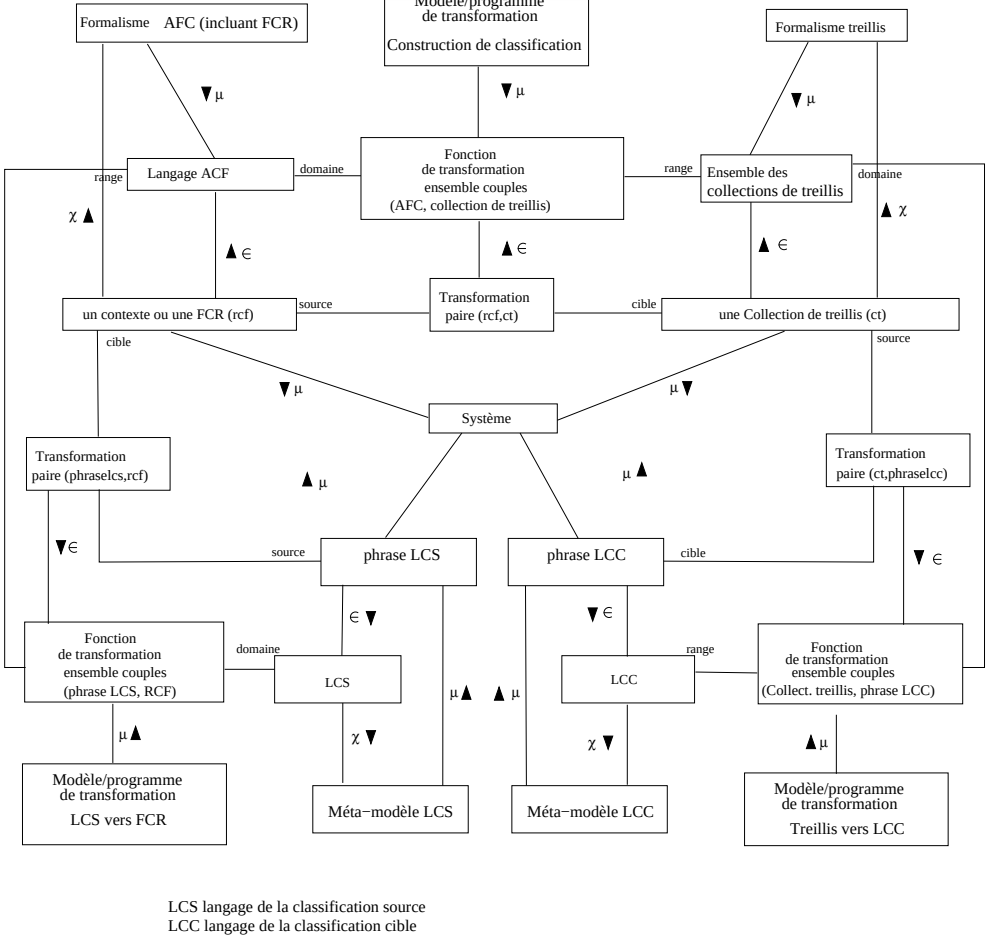

FIG. 9 – Patron de transformation de classification d'un langage source (LCS) vers un langage cible (LCC)

4. Bilan d'une expérience et nouvelles directions

La plate-forme GALICIA et Le projet MACAO Nos premières expériences concrètes de transformation de classifications sur la base de l'AFC ont été menées :

- dans le cadre d'un projet associant France Télécom R&D et le LIRMM dont l'un des résultats notables était une transformation de classes Java vers des interfaces Java [27] ;
- dans le cadre du projet RNTL MACAO[1] qui associait France Télécom R&D, le LIRMM et SOFTEAM[2], et dont certains travaux ont été menés en collaboration avec l'Université de Montréal [28, 29, 24, 25].

Pour le projet MACAO et dans nos développements en cours toutes les opérations de transformation de classification (contexte formel vers treillis ou sous-hiérarchie de Galois, famille de contextes relationnels vers collection de treillis ou collection de sous-hiérarchie de Galois) sont implémentées dans la plate-forme GALICIA [30], créée à l'initiative de l'Université de Montréal pour la manipulation de treillis[3].

L'application de l'Analyse Relationnelle de Concepts sur les diagrammes UML a été testée dans le cadre de MACAO sur trois projets de taille moyenne de France Télécom R&D [31] : gestion d'un système d'information, logiciel intranet et modèle de données utilisateurs pour des applications de services de télécommunication. Les diagrammes de classes de ces projets comprenaient plusieurs douzaines de classes et le nombre de nouveaux éléments (classes, attributs, opérations, associations, liens) créés par

[1] http ://www.lirmm.fr/~macao/
[2] http ://www.softeam.fr/
[3] http ://sourceforge.net/projects/galicia/ ou http ://www.iro.umontreal.ca/~galicia/

le processus de généralisation a atteint quelques centaines avec le paramétrage utilisé à l'époque. On peut considérer que l'expérience a été (positivement) concluante car les concepteurs des diagrammes initiaux ont trouvé de nombreuses généralisations construites pertinentes ou révélatrices d'un problème de conception initial, ainsi que des structures récurrentes telles que le patron *singleton*. Cependant l'explosion du nombre d'éléments construits, qui s'explique théoriquement, est le verrou à lever pour rendre réellement utilisable la méthode en contexte industriel et nous nous y appliquons actuellement en suivant deux directions, d'une part en étudiant une nouvelle spécification des modèles de transformation UML vers FCR et collection de treillis vers UML, d'autre part en facilitant l'appréhension du résultat par le concepteur.

Une nouvelle spécification L'une des difficultés avec les transformations depuis et vers UML est l'hétérogénéité et la variabilité des méta-modèles et des formats d'échange. Dans le projet MACAO, la transformation UML vers FCR et collection de sous-hiérarchies de Galois vers UML s'est faite naturellement en utilisant le méta-modèle UML interne à l'atelier OBJECTEERING. Dans le sens UML vers FCR nous avions même simplifié notre transformation en ne considérant pas les extrémités d'association comme des entités de premier plan et en imaginant une orientation naturelle à chaque association.

Nous pensons revenir sur cette hypothèse et à terme nous rapprocher plutôt du méta-modèle UML 2.0 (comme montré partiellement sur l'exemple enseignement). Cela assure plus de lisibilité au résultat pour toute personne connaissant le méta-modèle UML 2.0, évite de placer une orientation parfois artificielle (certaines associations sont tout à fait symétriques) qui peut cacher des généralisations et donne une relative pérennité à la transformation. Le choix dans le méta-modèle UML 2.0 de fusionner les notions d'attribut et d'extrémité d'association sous certaines conditions de navigabilité résout également l'un des problèmes que nous avions rencontrés dans l'hétérogénéité de la représentation : certains concepteurs choisissent systématiquement d'utiliser un attribut lorsque le type est primitif, mais tous ne le font pas et nous pensions devoir uniformiser les modèles avant de reconstruire afin de repérer dans un attribut `propriétaire` ou un rôle `propriétaire` une même notion à généraliser.

Pour réduire la combinatoire du résultat et surtout la génération de généralisations aussi peu intéressantes que *property de type entier*, *property de multiplicité upper=**, il est fondamental de ne pas utiliser toute l'information contenue dans le diagramme. Par contre, il faut compléter *a posteriori* une généralisation intéressante : il est utile de généraliser les deux agrégations sur notre schéma, mais cela doit se faire parce que les classes extrémités se généralisent et non pas parce que les multiplicités se généralisent. Ensuite, ayant décidé de généraliser les deux agrégations, on généralisera les multiplicités pour avoir une description plus précise. Ce paramétrage dépend beaucoup du point de vue du concepteur et nous devons privilégier un paramétrage interactif et très flexible.

Enfin l'un des objectifs que nous poursuivons est de construire, en liaison avec des chercheurs du domaine du traitement automatique des langues, des dictionnaires locaux, associés à un ou plusieurs modèles, vraisemblablement dans le même domaine métier pour plus de précision [32, 33]. Ces dictionnaires devraient nous permettre de résoudre les problèmes linguistiques rencontrés, notamment homonymie, synonymie, hyponymie et hyperonymie qui sont des indications importantes pour autoriser ou interdire des généralisations. Le nommage des généralisations découvertes devrait pouvoir profiter de ces dictionnaires, car nous devrions théoriquement être en mesure, dans de nombreux cas, de proposer un terme hyperonyme des noms des spécialisations concernées, par exemple avec un dictionnaire détaillé, nous pouvons espérer trouver le terme `Personne` pour généraliser des noms de classes tels que `Etudiant`, `Professeur`, `Eleve`, `Instituteur`, ou encore le nom de la classe pourra être proposé d'après les propriétés incluses telles que `nom`, `age`, `adresse`.

Évolution de l'outillage Les programmes de transformation UML vers FCR et collection de treillis vers UML ont été, dans le cadre du projet MACAO, développés en J, langage interne de l'atelier OBJECTEERING, puis complétés par des outils d'interprétation du résultat qui permettent de comprendre les correspondances entre éléments des diagrammes initial et final. D'autres outils pourraient nous être utiles

tels que le déroulement pas à pas et interactif de la transformation ou encore comme il a été suggéré récemment [4], de présenter le résultat final comme une succession d'opérations élémentaires de *refactoring* [34]. La libre mise à disposition de la chaîne complète de transformation demande de s'affranchir de l'atelier OBJECTEERING et nous étudions actuellement un développement autour d'ECLIPSE qui soit le plus générique possible, une approche qui nous intéresse particulièrement étant le projet *Eclipse MDDi* [35].

5. Conclusion

Dans cet article nous avons présenté l'AFC et son extension relationnelle comme un support théorique pour la construction et la transformation de classifications. Nous avons également exposé brièvement les résultats d'une expérience menée avec l'Analyse Relationnelle de Concepts et tracé les grandes lignes des développements à venir.

Nous avons placé les modèles de transformations issus de l'AFC au centre d'un patron de transformation de classification dont nous pensons qu'il a une portée générale dans l'ingénierie des modèles : au-delà des travaux présentés ici, de multiples activités d'ingénierie logicielle se structurent en effet autour du support théorique de l'AFC [36, 17], incluant notamment l'identification de classes à partir de cas d'utilisation [37], la réingénierie vers les approches à objets [38], la construction de schémas de bases de données [39], la construction de hiérarchies de classes d'après les propriétés des classes [40, 41, 42] ou d'après l'utilisation qui est faite de ces propriétés [43], l'extraction de patrons de conception [16, 17]. Nous sommes bien loin d'avoir fait le tour du sujet, mais nous espérons avoir au moins ouvert une réflexion générale sur la place de la classification dans l'ingénierie des modèles en montrant une expérience concrète de mise en œuvre.

Remerciements Nous remercions les relecteurs anonymes pour leurs remarques constructives qui ont nous permis d'améliorer l'article.

Références

[1] J. Bézivin, M. Blay, M. Bouzeghoub, J. Estublier, and J.-M. Favre. Rapport de synthèse. Action spécifique CNRS sur l'Ingénierie Dirigée par les Modèles, janvier 2005. http ://www-adele.imag.fr/mda/as/rapport/AS-MDA-IDM-Synthese-1.1.pdf.

[2] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Modélisation et conception orientées objet*. Masson, 1995.

[3] Jean François Perrot. Objets, classes, héritage : définitions. In Roland Ducournau, Jérôme Euzenat, Gérald Masini, and Amedeo Napoli, editors, *Langages et Modèles d'Objets*, chapter 1. INRIA, collection didactique, 1998.

[4] P.-A. Muller and N. Gaertner. *Modélisation objet avec UML*. Eyrolles, snd edition, 2002.

[5] E. Diday. La méthode des nuées dynamiques. *Revue de statistique appliquée*, XIX(1), 1971.

[6] J-P. Benzecri. *L'analyse des données (Tomes 1 et 2)*. Dunod, 1973.

[7] R.S. Michalski and R.E. Stepp. Learning from observation : Conceptual clustering. In *Machine Learning, An Artificial Intelligence Approach*, volume I, pages 331–363. Morgan Kaufmann, 1983.

[8] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, (2) :139–172, 1987.

[9] D. Fisher and M. Pazzani. Computational Models of Concept Learning. In *Concept formation : Knowledge and Experience in Unsupervised Learning*. Morgan Kaufmann, 1991.

[4] P. Seuring, communication orale.

[10] A. Simon and A. Napoli. Building Viewpoints in an Object-Based Representation System for Knowledge Discovery in Databases. In *1st International Conference on Information Reuse and Integration, IRI-99*, 1999.

[11] N. Pasquier, J. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 19(4) :33–54, 1999.

[12] V. Berry. *Méthodes et algorithmes pour reconstruire les arbres de l'évolution*. PhD thesis, Université Montpellier 2, 1997.

[13] R. Godin, H. Mili, G. Mineau, R. Missaoui, A. Arfi, and T.T. Chau. Design of Class Hierarchies Based on Concept (Galois) Lattices. *Theory and Practice of Object Systems*, 4(2), 1998.

[14] Gabriela Arévalo, S. Ducasse, and O. Nierstrasz. X-ray views : Understanding internals of classes. In *Proceedings of the 18th IEEE International Conference on Automate Software Engineering conference (ASE 2003)*, pages 267–270, Montreal, Canada, October 2003. IEEE Computer Society.

[15] C. Lindig and G. Snelting. Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. In *Proc. International Conference on Software Engineering (ICSE 97)*, pages 349–359, Boston, USA, May 1997.

[16] Paolo Tonella and Giuliano Antoniol. Object-Oriented Design Pattern Inference. In *ICSM*, pages 230–238, 1999.

[17] Gabriela Arévalo. *High-Level Views in Object-Oriented Systems using Formal Concept Analysis*. PhD thesis, Software Composition Group, University of Bern, 2004.

[18] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture : Practice and Promise*. Addison-Wesley, 2003.

[19] Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. A taxonomy of model transformations. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany. http ://drops.dagstuhl.de/opus/volltexte/2005/11.

[20] Mohamed H. Rouane, Petko Valtchev, Houari Sahraoui, and Marianne Huchard. Concept Merging conceptual hierarchies using concept lattices. In *Proceedings of The 3rd International Workshop on MechAnisms for SPEcialization, Generalization and inHerItance, MASPEGHI 2004 (Workshop Ecoop 2004)*, pages 51–58, 2004. http ://www.i3s.unice.fr/maspeghi2004/.

[21] G. Birkhoff. *Lattice theory*. AMS Colloquium Publication, vol. XXV, 1940.

[22] M. Barbut and M. Monjardet. *Ordre et Classification. Algèbre et Combinatoire*, volume 2. Hachette, 1970.

[23] B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer, Berlin, 1999.

[24] Michel Dao, Marianne Huchard, Mohamed Rouane Hacene, Cyril Roume, and Petko Valtchev. Improving Generalization Level in UML Models Iterative Cross Generalization in Practice. In *ICCS'04*, volume 3127 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2004.

[25] Marianne Huchard, Mohamed Rouane Hacene, Cyril Roume, and Petko Valtchev. Relational concept discovery in structured datasets. submitted. Research report version http ://www.iro.umontreal.ca/˜ valtchev/Papiers/valtchev-et-al-DAM-JIM03.pdf.gz.

[26] J.-M. Favre. Towards a Basic theory to Model Model Driven Engineering. 3rd Workshop in Software Model Engineering, WiSME 2004, october 2004. http ://www-adele.imag.fr/˜ jmfavre/papers/TowardsABasicTheoryToModelModelDrivenEngineering.pdf.

[27] M. Huchard and H. Leblanc. Computing Interfaces in Java. In *Proc. IEEE International conference on Automated Software Engineering (ASE'2000)*, pages 317–320, 11-15 September, Grenoble, France., 2000.

[28] M. Huchard, C. Roume, and P. Valtchev. When concepts point at other concepts : the case of UML diagram reconstruction. In *Proceedings of the 2nd Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD)*, pages 32–43, 2002.

[29] P. Valtchev, M. Hacene Rouane, M. Huchard, and C. Roume. Extracting Formal Concepts out of Relational Data. In E. SanJuan, A. Berry, A. Sigayret, and A. Napoli, editors, *Proceedings of the 4th Intl. Conference Journées de l'Informatique Messine (JIM'03) : Knowledge Discovery and Discrete Mathematics, Metz (FR), 3-6 September*, pages 37–49. INRIA, 2003.

[30] P. Valtchev, D. Grosser, C. Roume, and M. Rouane Hacene. GALICIA : an open platform for lattices. In A. de Moor B. Ganter, editor, *Using Conceptual Structures : Contributions to 11th Intl. Conference on Conceptual Structures (ICCS'03)*, pages 241–254, Aachen (DE), 2003. Shaker Verlag.

[31] M. Dao. Validation sur de grands projets, Projet MACAO (RNTL). Technical Report sous-projet MACAO 5.1, France Télécom R&D, december 2003.

[32] M. Lafourcade and V. Prince. Synonymies et vecteurs conceptuels. In *Proceedings of TALN'2001*, pages 233–242, 2001.

[33] D. Schwab, M. Lafourcade, and V. Prince. Amélioration de liens entre acceptions par fonctions lexicales vectorielles symétriques. In *Proceedings of TALN'2003*, pages 225–253, 2003.

[34] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley, 1999. Object Technologies Series.

[35] Eclipse MDDi project. http ://www.eclipse.org/proposals/eclipse-mddi/index.html.

[36] T. Tilley, R. Cole, P. Becker, and P. Eklund. A survey of formal concept analysis support for software engineering activities. In Gerd Stumme, editor, *Proceedings of the First International Conference on Formal Concept Analysis - ICFCA'03*. Springer-Verlag, February 2003.

[37] D. Richards, K. Boettger, and O. Aguilera. A controlled language to assist conversion of use case descriptions into concept lattices. In *Proceedings of AI 2002 (15th Australian Joint Conference on Artificial Intelligence*, number 2557 in LNAI, pages 579–590, Canberra, Australia, 2002. Springer Verlag.

[38] Houari A. Sahraoui, Walcélio L. Melo, Hakim Lounis, and F. Dumont. Applying concept formation methods to object identification in procedural code. In *ASE*, pages 210–218, 1997.

[39] A. Yahia, L. Lakhal, R. Cicchetti, and J.P. Bordat. iO2 - An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In *Proceedings of the 15th International Conference on Conceptual Modeling ER'96*, volume 1157, pages 422–437, 1996.

[40] R. Godin and H. Mili. Building and maintaining analysis-level class hierarchies using Galois lattices. In *Proceedings of OOPSLA'93, Washington (DC), USA*, pages 394–410, 1993.

[41] H. Dicky, C. Dony, M. Huchard, and T. Libourel. On Automatic Class Insertion with Overloading. In *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96*, pages 251–267, 1996.

[42] Samira Si-Said Cherfi and Nadira Lammari. Towards and Assisted Reorganization of Is-A Hierarchies. In Z. Bellahsène, D. Patel, and C. Rolland, editors, *Proceedings of Object-Oriented Information Systems*, volume 2425 of *LNCS*, pages 536–548. Springer-Verlag, 2002.

[43] G. Snelting and F. Tip. Understanding class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems*, 22(3) :540–582, May 2000.