Analyse formelle de données relationnelles pour la réingénierie des modèles UML

M. H. Rouane* — M. Dao — M. Huchard*** — P. Valtchev******

** DIRO, U. de Montréal, CP 6128, succ Centre-Ville, Montréal, Canada H3C 3J7 rouanehm@iro.umontreal.ca,*

*** France Télécom R&D/MAPS/AMS/VIP, 38-40 rue du Général Leclerc, 92794 Issy Les Moulineaux- France ; michel.dao@orange-ftgroup.com,*

**** LIRMM, UMR 5506, 161 rue Ada, 34392 Montpellier Cedex 5 - France huchard@lirmm.fr*

***** Dépt. d'informatique, UQÀM, CP 8888, succ. Centre-Ville, Montréal, Canada H3C 3P8 valtchev.petko@uqam.ca*

RÉSUMÉ. *La restructuration des modèles statiques de logiciel, ou diagrammes de classes UML, cherche à remédier à la détérioration de la qualité logicielle suite à une évolution prolongée ou à une maintenance corrective. Elle re-distribue des éléments du modèle sur l'ensemble des classes en abstrayant au besoin de nouvelles classes à partir des initiales. Nous présentons ici une approche de restructuration qui couvre tous les types d'éléments d'un modèle — associations, méthodes, etc., — et donc permet d'en trouver des abstractions. L'approche repose sur l'adaptation de l'analyse de concepts au traitement d'informations relationnelles lesquelles abondent dans les représentations des modèles UML. Les divers aspects de l'analyse des méta-données UML à l'aide des treillis sont abordés, tel la traduction de l'UML vers le format d'analyse, le traitement des ambiguïtés sémantiques, l'assemblage du modèle restructuré, etc.*

ABSTRACT. *Refactoring of static software models, typically UML class diagrams, is aimed at recovering the quality that deteriorates along a protracted evolution or due to a corrective maintenance. Refactoring redistributes class members within the generalization hierarchy while operating rearrangements of various scope on the hierarchy, and possibly abstracting new classes from the initial ones. We present here an approach to refactoring whose scope includes, besides classes, the other elements, of a static model, i.e., associations, methods, attributes, etc. Based on an extension of classical concept analysis capable of processing relational information, our approach puts the emphasis on the proper processing of meta-links that help keep together the elements of a class model. Aspects of the approach, including two-way translation between UML and the analysis formats, semantic ambiguity processing, etc. are discussed as well.*

MOTS-CLÉS : *Analyse formelle de concepts, treillis de concepts, UML, restructuration.*

KEYWORDS: *Formal concept analysis, concept lattices, UML, refactoring.*

1. Introduction

En modélisation conceptuelle, en particulier à l'aide de l'UML [BOO 99], la qualité des modèles est encore une notion très peu formalisée. Néanmoins, deux facteurs très généraux peuvent être identifiés, chacun contribuant à augmenter l'utilisabilité d'un modèle : d'une part, la complétude du modèle, ou la présence de tous les concepts pertinents du domaine représenté, et d'autre part, l'absence de redondances dans celui-ci (qui, comme en bases de données, accroissent l'effort de maintenance). Bien qu'intuitifs, les deux critères sont loin d'admettre une satisfaction triviale et peuvent même s'avérer antinomiques si traités de façon indépendante. En revanche, il est très facile d'enfreindre les deux, surtout dans les modèles de taille ou à évolutions fréquentes. À titre d'illustration, examinons le diagramme de la figure 1 qui reflète une partie d'un modèle statique du domaine bancaire. En effet, les classes représentent des types de comptes, soit chèque (`CheckAccount`) ou hypothèque (`MortgageAccount`), et leurs détenteurs respectifs (`CheckBookHolder` et `Mortgager`).

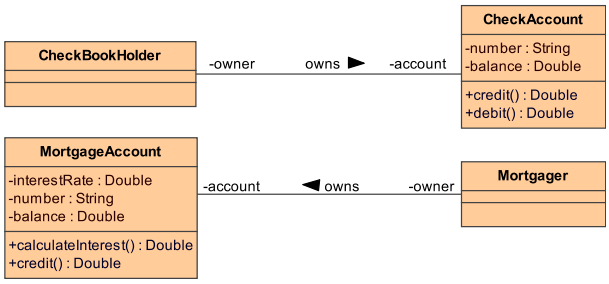


Figure 1. *Portion d'un modèle statique du domaine bancaire.*

Pris tel quel, le diagramme ci-dessus exhibe aussi bien des similarités que de l'incomplétude. En effet, les classes modélisant les deux types de comptes possèdent un bon nombre de membres en commun[1] qui auraient été avantageusement factorisés par une hypothétique super-classe commune, `Account`, absente du diagramme. Le traitement de ce type de situations, fréquentes en construction, intégration ou maintenance de modèles, a été abordé avec des approches semi-automatiques, basées pour la majorité sur l'analyse formelle de concepts (AFC) [GOD 98, DAO 04, SNE 00]. En effet, AFC s'est avéré être un cadré théorique adéquat pour la restructuration des hiérarchies de par la proximité entre la structure d'une hiérarchie de classes et celle d'un treillis de concepts, ainsi que par la factorisation maximale que les treillis et les structures de concepts dérivées assurent aux hiérarchies qui en sont extraites. Cependant, l'AFC classique a ses limites. Pour s'en persuader, il suffit d'observer l'analogie des rapports que les classes des détenteurs entretiennent avec les classes comptes respectifs sur la figure 1. Ainsi, une super-classe `Client` semble être la réponse appropriée

1. Nous faisons l'hypothèse que les éléments de même nom ont une sémantique identique.

qui de plus permettra de factoriser l'association `owns`. Toutefois, `CheckBookHolder` et `Mortgager` n'ont aucun membre en commun, alors qu'aucune technique actuelle d'AFC ne permet de regrouper les individus sur la base d'une analogie dans leurs liens à d'autres individus.

Dans la suite de l'article, un cadre plus expressif pour l'extraction de concepts formels sera présenté, notamment admettant le traitement des liens inter-individus, et les divers problèmes concrets soulevés par son application à la restructuration de modèles UML seront abordés.

Tout d'abord, la section 2 introduit l'AFC classique et situe ses limitations en termes d'expressivité. La section 3 présente l'analyse relationnelles de concepts (ARC), notre extension admettant des données relationnelles. L'utilisation de l'ARC pour une restructuration plus approfondie des modèles UML est discutée dans la section 4. La section 5 décrit quelques expérimentations pratiques avec l'approche de restructuration par ARC.

2. Analyse Formelle de Concepts

L'AFC [GAN 01] est une approche vers la découverte et la structuration de connaissances. Elle fournit une méthode algébrique de dérivation de hiérarchies d'abstractions à partir d'un ensemble d'entités. L'AFC représente les entités par le biais d'un tableau de données (*contexte formel*), noté $\mathcal{K}$, faisant apparaître un ensemble O d'individus (*objets formels*), un ensemble A de caractéristiques (*attributs formels*) et exprimant la relation binaire I d'incidence objet-attribut. La table illustrée par la figure 2 montre un contexte binaire $\mathcal{K} = (O, A, I)$ où les individus sont les classes du modèle UML de la figure 1 et les attributs formels sont les propriétés de ces classes.

	number (nb)	balance (ba)	interestRate (ir)	credit() (cr)	debit() (de)	calculateInterest() (ci)
CheckBookHolder (CBH)						
CheckAccount (CA)	X	X		X	X	
Mortgager (M)						
MortgageAccount (MA)	X	X	X	X		X

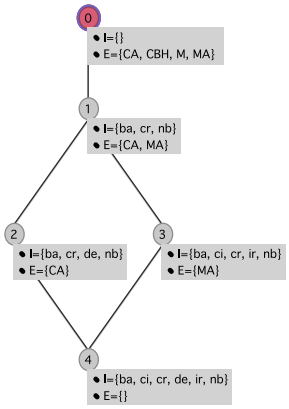


Figure 2. ***Gauche*** *: Codage des classes du modèle UML de la figure 1 par un contexte binaire.* ***Droite*** *: Le treillis de concepts correspondant.*

Deux opérateurs, tous deux notés $'$ lient les sous-ensembles d'individus de $\mathcal{P}(O)$ aux sous-ensembles d'attributs de $\mathcal{P}(A)$. Le premier opérateur se définit comme suit : $' : \mathcal{P}(O) \rightarrow \mathcal{P}(A),\ X' = \{a \in A \mid \forall o \in X, oIa\}$. Le deuxième opérateur $'$ est défini de manière duale sur les attributs. La paire d'opérateurs $'$ induit une correspondance de Galois entre $\mathcal{P}(O)$ et $\mathcal{P}(A)$ [BAR 70]. Les opérateurs de composition $''$ sont des opérateurs de fermeture qui induisent deux familles d'ensembles fermés, à savoir, $\mathcal{C}^o \subseteq \mathcal{P}(O)$ et $\mathcal{C}^a \subseteq \mathcal{P}(A)$. Ces deux ensembles, munis de l'ordre issu de la relation d'inclusion ensembliste forment deux treillis complets (anti-isomorphique par $'$). Une paire (X, Y) où $X \in \mathcal{P}(O)$, $Y \in \mathcal{P}(A)$, $X = Y'$, et $Y = X'$, est un *concept formel*. Les ensembles X et Y sont appelés *extension* et *intension*, respectivement. L'ensemble $\mathcal{C}_\mathcal{K}$ de tous les concepts obtenus à partir de $\mathcal{K}$ et ordonnés par l'inclusion des extensions forme un treillis complet, $\mathcal{L}_\mathcal{K} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$, appelé le *treillis de concepts* du contexte ou encore le *treillis de Galois* de la relation binaire I. La figure 2 illustre un contexte et le treillis de concepts correspondant. Les concepts du treillis représentent toutes les factorisations maximales des propriétés des individus. Le concept formel $c_{\#1}$, par exemple, est composé de l'extension $\{$`CA`,`MA`$\}$ et de l'intension $\{$`ba`, `cr`, `nb`$\}$. Cela signifie que les deux classes `CheckAccount` et `MortgageAccount` partagent les variables `number`, `balance` et l'opération `credit`.

L'AFC de par la définition formelle de la notion de « concept », de « hiérarchie conceptuelle » et de son pouvoir d'abstraction et de factorisation maximale s'adapte très bien au problème d'analyse [SNE 00] et de construction [GOD 98, HUC 00] de hiérarchies de classes ainsi que la découverte de patrons de conception [TON 99]. Le concept en AFC est la représentation formelle d'un concept physique ou abstrait du domaine (classe métier). La hiérarchie conceptuelle précise les relations de spécialisation/généralisation entre concepts. L'abstraction permet de former de nouveaux concepts (superclasses) à partir des traits communs d'un ensemble d'individus (variables/méthodes). Enfin, la factorisation maximale permet d'avoir une structure sans redondances et favorise par conséquent la réutilisation.

Bien que les techniques de l'AFC ont été appliquées avec succès pour la maintenance des logiciels objets, la richesse des données rencontrées en UML, en fort contraste avec la relative simplicité du modèle binaire de données utilisé en AFC, a révélé les limites du paradigme « classique » de l'AFC. Par exemple, le treillis de la figure 2, nous a permis de détecter une seule abstraction possible dans le modèle UML de la figure 1, la classe `Account`, alors que d'autres auraient été possibles. En effet, `CheckBookHolder` et `Mortgager` jouent un rôle analogue par rapport aux classes comptes. Intuitivement, elles pourront être abstraites vers une super-classe commune qu'on appellerait `Client` et qui factoriserait leur rôles de détenteurs de compte. En se penchant sur les éléments du modèle qui pourraient servir de base pour cette dernière abstraction, on constate que `CheckAccount` et `MortgageAccount` sont incidentes à des associations à sémantique proche les reliant à des sous-classes de `Client`. Donc, deux questions surviennent naturellement : Comment reconnaître la proximité des sémantiques « proche » ? et Comment signaler l'existence de la superclasse de `CheckBookHolder` et `Mortgager`, soit `Client`, vers `CheckAccount` et `MortgageAccount`. La première question appelle clairement à un traitement linguis-

tique (par exemple, une analyse des sens des deux termes). La seconde sort du cadre actuel de l'AFC en requérant de nouveaux mécanismes d'abstraction capables de regrouper les individus non seulement en fonction du partage dans leurs propres descriptions, mais aussi dans les descriptions des individus qui leurs sont liés. De plus, le modèle devrait pouvoir être analysé en plus de profondeur, c'est-à-dire, ne pas se limiter aux abstractions de classes mais étendre l'analyse à d'autres types d'éléments tels que les attributs, rôles, etc. Cependant, les liens existant entre ces éléments comme l'incidence classe-attribut, le typage d'un rôle par une classe, etc. constituent des informations *relationnelles* qui ne sont pas à la portée des approches d'AFC actuelles. L'obstacle devant les méthodes automatiques capables de produire des abstractions de cette complexité est de taille car il s'agit de modifier la façon même dont les concepts sont construits en y intégrant le partage de liens à côté de celui des attributs formels.

3. Analyse Relationnelle de Concepts

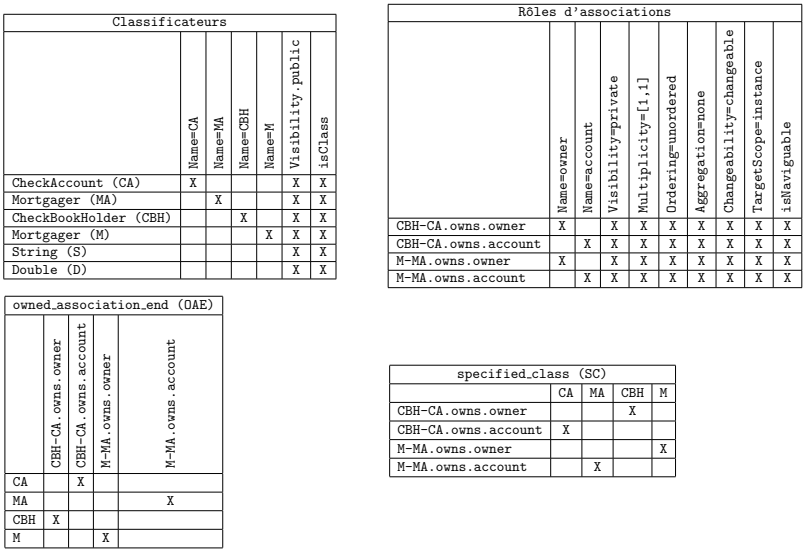
En ARC, les données sont décrites par une collection de contextes et de relations binaires appelée famille de contextes relationnels (FCR). La figure 3 illustre une partie de la FCR extraite du modèle UML de la figure 1 par la façon décrite dans la section 4.

Définition 1 *Une FCR est une paire* $(\mathbf{K}, \mathbf{R})$ *où* $\mathbf{K}$ *est un ensemble de contextes pluri-valués*[2] $\mathcal{K}_i = (O_i, A_i, V_i, I_i)$ *et* $\mathbf{R}$ *est un ensemble de relations (ou fonctions multi-valuées)* $r_k \subseteq O_i \times O_j$ *où* O_i *et* O_j *sont des ensembles d'individus, appelés domaine et co-domaine de* r_k*, respectivement.*

Les domaines de valeurs des relations inter-contextes sont des sous-ensembles d'individus. Ainsi, à l'instar du scaling conceptuel en AFC [GAN 01], le scaling relationnel traduit les structures de ces domaines par des prédicats décrivant les sous-ensembles d'individus. Nous avons choisi de baser ces prédicats sur la structure conceptuelle de l'ensemble des individus référencés. Ainsi, les nouveaux attributs formels issus du scaling —ayant la forme $r : c$ où r et c désignent un nom d'une relation et d'un concept formel, respectivement— reflèteront l'appartenance d'un individu de l'ensemble à analyser à un concept identifié au préalable. Par exemple, si un concept formel représentant les attributs UML de type réel est identifié, un individu du contexte des classes UML pourra se voir affecter un attribut formel signalant la possession d'au moins un tel attribut UML. Formellement :

Définition 2 *Étant donné un contexte* $\mathcal{K}_i = (O_i, A_i, I_i)$*, le treillis de concepts* $\mathcal{L}_j$ *associé au contexte* $\mathcal{K}_j = (O_j, A_j, I_j)$ *et la relation* $r \subseteq O_i \times O_j$*, l'opérateur de scaling* $sc_\times^{(r,\mathcal{L}_j)} : \mathbf{K} \rightarrow \mathbf{K}$ *est défini par :* $sc_\times^{(r,\mathcal{L}_j)}(\mathcal{K}_i) = (O_i^{(r,\mathcal{L}_j)}, A_i^{(r,\mathcal{L}_j)}, I_i^{(r,\mathcal{L}_j)})$*, où :* $O_i^{(r,\mathcal{L}_j)} = O_i$*,* $A_i^{(r,\mathcal{L}_j)} = A_i \cup \{r : c | c \in \mathcal{L}_j\}$*,* $I_i^{(r,\mathcal{L}_j)} = I_i \cup \{(o, r : c) | o \in O_i, c = (X, Y) \in \mathcal{L}_j, r(o) \neq \emptyset, r(o) \subseteq X\}$*).*

2. Un contexte ayant la forme $\mathcal{K} = (O, A, V, I)$ où I est une relation ternaire entre l'ensemble des individus O, celui des attributs A et celui des valeurs V [GAN 01].



Classificateurs

	Name=CA	Name=MA	Name=CBH	Name=M	Visibility.public	isClass
CheckAccount (CA)	X				X	X
Mortgager (MA)		X			X	X
CheckBookHolder (CBH)			X		X	X
Mortgager (M)				X	X	X
String (S)					X	X
Double (D)					X	X

Rôles d'associations

	Name=owner	Name=account	Visibility=private	Multiplicity=[1,1]	Ordering=unordered	Aggregation=none	Changeability=changeable	TargetScope=instance	isNavigable
CBH-CA.owns.owner	X		X	X	X	X	X	X	X
CBH-CA.owns.account		X	X	X	X	X	X	X	X
M-MA.owns.owner	X		X	X	X	X	X	X	X
M-MA.owns.account		X	X	X	X	X	X	X	X

owned_association_end (OAE)

	CBH-CA.owns.owner	CBH-CA.owns.account	M-MA.owns.owner	M-MA.owns.account
CA		X		
MA				X
CBH	X			
M			X	

specified_class (SC)

	CA	MA	CBH	M
CBH-CA.owns.owner			X	
CBH-CA.owns.account	X			
M-MA.owns.owner				X
M-MA.owns.account		X		

Figure 3. *Une famille de contextes relationnels codant les classificateurs et les rôles d'associations du modèle UML de la figure 1 ainsi que leurs liens.*

L'opérateur $sc_{\times}^{(r,\mathcal{L}_j)}$ est dit « *étroit* » car la manière de coder la relation d'incidence entre un individu o de l'ensemble à analyser O_i et les nouveaux attributs formels de la forme $r : c$ repose sur l'opération d'inclusion $\subseteq$ qui exige que tous les individus désignés par les liens de type r en partance de o (l'ensemble $r(o)$) soient inclus dans l'extension du concept c. Par contre, pour l'opérateur de scaling dit « *large* », utilisé pour UML et noté $sc_{+}^{(r,\mathcal{L}_j)}$, il suffit qu'il y ait des individus désignés par les liens de type r dans l'extension de c. Ainsi pour $sc_{+}^{(r,\mathcal{L}_j)}$, $I_i^{(r,\mathcal{L}_j)} = I_i \cup \{(o, r : c) | o \in O_i, c = (X, Y) \in \mathcal{L}_j, r(o) \cap X \neq \emptyset\}$).

Par exemple, le scaling de la relation `Owned_Association_End` (`OAE`) reliant les classes à leurs rôles respectifs (voir la figure 3) étend le contexte des classificateurs par les nouveaux attributs formels `OAE :c0` et `OAE :c3` où `c0` et `c3` sont des concepts du treillis dérivé à partir du contexte des rôles de la figure 3 et correspondent aux rôles `owner` et `account` de la figure 1, respectivement. L'objet formel `CA` (classe `CheckAccount`), par exemple, a l'attribut formel `OAE :c3` car la classe `CheckAccount` possède le rôle `account` comme illustré sur la figure 1.

Le scaling relationnel est au cœur d'un processus qui, à partir d'une FCR construit une famille de treillis relationnels (FTR), un treillis par contexte. Au sein d'une FTR, les concepts renferment dans leurs extensions des individus qui, en plus des caractéristiques de départ, partagent aussi des liens vers d'autres individus. Ce nouveau partage se matérialise à travers des attributs formels appelés « *relationnels* ». Tout attribut relationnel peut être interprété par une relation entre deux concepts, d'une part

un concept représentant un groupe d'individus et d'autre part un concept représentant le groupe d'individus référencé par un même type de lien. À noter que le processus d'extraction des treillis est de nature itérative car le scaling relationnel modifie l'état des contextes ce qui nécessite la mise à jour des treillis associés qui à son tour implique un re-scaling de toutes les relations ayant utilisé les treillis qui ont été modifiés comme source de concepts pour la fabrication de prédicats décrivant des individus référencés. Le processus itératif de construction de la FTR s'arrête dès qu'un point fixe est atteint, c'est-à-dire lorsqu'un nouveau scaling de toutes les relations de la FCR ne conduit à l'enrichissement d'aucun contexte.

L'algorithme 1 de dérivation d'une FTR commence par le scaling des attributs pluri-valués (primitive CONCEPTUAL-SCALING) des différents contextes (ligne 4) dans le but de calculer les treillis initiaux (ligne 6). La structure $\Delta\mathbf{L}^+$, à partir de laquelle l'algorithme récupère les concepts qui serviront dans l'itération suivante au scaling des relations, est initialisée (primitive GET-CONCEPTS) aux concepts actuels des treillis (ligne 7). L'algorithme termine l'étape d'initialisation en positionnant les valeurs du vecteur `Off` à `faux` (ligne 8) car les contextes ne sont pas encore saturés.

1: **Algorithm** MULTI-FCA(**In :** $\mathbf{K}$ array $[1..n]$ of contexts, $\mathbf{R}$ set of inter-context relations, mode : scaling mode, **Out :** $\mathbf{L}$ array of lattices)
2: Local $\Delta\mathbf{K}$: array of the context relational extensions, $\mathbf{L}$: array of lattices, $\mathbf{L}^+$: array of updated lattices, $\mathbf{L}^+[i]$ is obtained from $\mathbf{L}[i]$ and $\Delta\mathcal{K}_i$, $\Delta\mathbf{L}$: array of sets of concepts (for relational scale attributes), $\Delta\mathbf{L}^+$: array of sets of concepts (concepts of the next rel. scaling round), `Off` : array of boolean variables indicating whenever a context is saturated, $\mathcal{A}$: a set of context relations.
3: **for all** $i = 1, n$ **do**
4: **if** MANY-VALUED($\mathbf{K}[i]$) **then**
5: $\mathbf{K}[i] \leftarrow$ CONCEPTUAL-SCALING($\mathbf{K}[i]$)
6: $\mathbf{L}^+[i] \leftarrow$ COMPUTE-INITIAL-LATTICE($\mathbf{K}[i]$)
7: $\Delta\mathbf{L}^+[i] \leftarrow$ GET-CONCEPTS($\mathbf{L}^+[i]$)
8: `Off`[i] $\leftarrow$ **false**
9: `fixedpoint` $\leftarrow$ **false**
10: **while not** `fixedpoint` **do**
11: `fixedpoint` $\leftarrow$ **true**
12: **for** i **from** 1 **to** n **do**
13: $\mathbf{L}[i] \leftarrow \mathbf{L}^+[i]$
14: $\Delta\mathbf{L}[i] \leftarrow \Delta\mathbf{L}^+[i]$
15: **for** i **from** 1 **to** n **do**
16: **if** (**not** `Off`[i]) **then**
17: $\mathcal{A} \leftarrow$ GET-CONTEXT-RELATIONS($\mathbf{K}[i]$,$\mathbf{R}$)
18: **for all** $r \in \mathcal{A}$ **do**
19: $\Delta\mathbf{K}[i] \leftarrow$ GET-RELATIONAL-EXTENSION($\mathbf{K}[i]$,r,$\Delta\mathbf{L}$[RANK($\mathbf{K}$, $cod(r)$)])
20: MERGE($\mathbf{K}[i]$,$\Delta\mathbf{K}[i]$){merges context and its relational extension along r.}
21: UPDATE($\mathbf{L}^+[i]$,$\Delta\mathbf{K}[i]$){updates the corresponding lattice}
22: $\Delta\mathbf{L}^+[i] \leftarrow$ GET-NEW-CONCEPTS($\mathbf{L}^+[i]$,$\mathbf{L}[i]$)
23: **if** $\Delta\mathbf{L}^+[i] == \emptyset$ **then**
24: `Off`[i] $\leftarrow$ **true** {eliminating $\mathbf{K}[i]$ from the working contexts}
25: **else**
26: `fixedpoint` $\leftarrow$ **false**
27: **return** $\mathbf{L}^+$

Algorithm 1: Traitement d'une FCR par la méthode MULTI-FCA.

Par la suite, l'algorithme entre dans une série d'itérations permettant l'enrichissement des différents contextes et des treillis associés et conduisant à un point fixe dans lequel tous les contextes sont saturés. Ainsi, tant que ce point fixe n'est pas encore atteint (ligne10), l'algorithme initialise la variable de saturation globale (ligne

11). Ensuite, comme l'algorithme se prépare à une nouvelle mise à jour des treillis, il réactualise les structures différentielles $\mathbf{L}$ et $\Delta\mathbf{L}$ (lignes $13-14$). Arrivé à ce stade, l'algorithme parcourt tous les contextes de la FCR (ligne 15) et examine leur état de saturation (ligne 16). Pour tout contexte non saturé, l'algorithme détermine l'ensemble de ses relations (ligne 17). Ensuite, pour chaque relation r de cet ensemble, l'algorithme réalise les trois opérations suivantes : calcul d'une nouvelle extension relationnelle (primitive GET-RELATIONAL-EXTENSION) du contexte suivant r (ligne 19) en utilisant le treillis du co-domaine de r obtenu au niveau de l'itération précédente ($\Delta\mathbf{L}$), mise à jour du contexte courant (ligne 20) et mise à jour des treillis du contexte courant suite à l'évolution du contexte associé (ligne 21). Par la suite, l'ensemble des nouveaux concepts qui ont émergé dans le treillis suite à l'enrichissement du contexte correspondant est calculé (ligne 22), d'une part, pour savoir si le contexte est saturé ou pas et, d'autre part, pour être utilisé dans de futurs scaling. Si cet ensemble s'avère vide (ligne 23), le contexte est saturé et doit être ignoré au cours des prochaines itérations (ligne 24). Dans le cas où au moins un treillis a évolué, le point fixe n'est pas atteint (ligne 26) obligeant l'algorithme à effectuer d'autres itérations. En fin de calcul du point fixe, l'algorithme retourne les treillis finaux construits. La figure 4 et la figure 5 illustrent le treillis final du contexte des classificateurs et du contexte des rôles de la figure 3, respectivement.

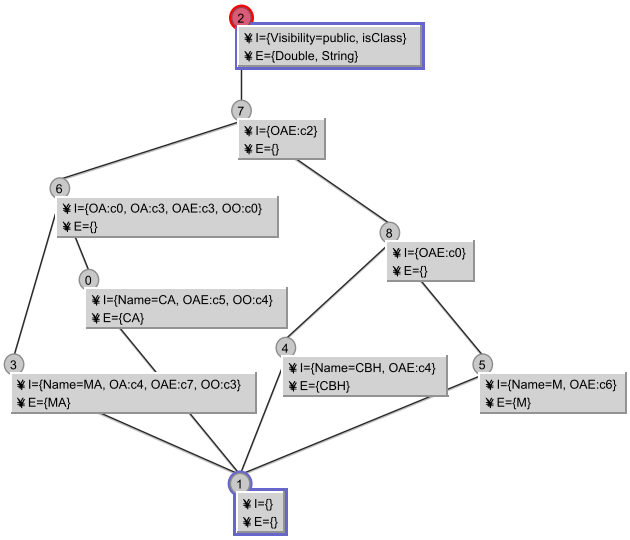


Figure 4. *Le treillis de concepts final du contexte des classificateurs.*

Par exemple, le concept $c_{\#5} = (\{M\}, \{name{=}M, OAE :c6\})$ du treillis des classificateurs représente la classe `Mortgager` de la figure 1. L'attribut relationnel $OAE :c6$ indique que le concept $c_{\#5}$ du treillis des classificateurs (figure 4) a une relation de type OAE avec le concept $c_{\#6}$ du treillis des rôles (figure 5). Ceci signifie que la classe `Mortgager` possède le rôle `owner` (voir figure 1).

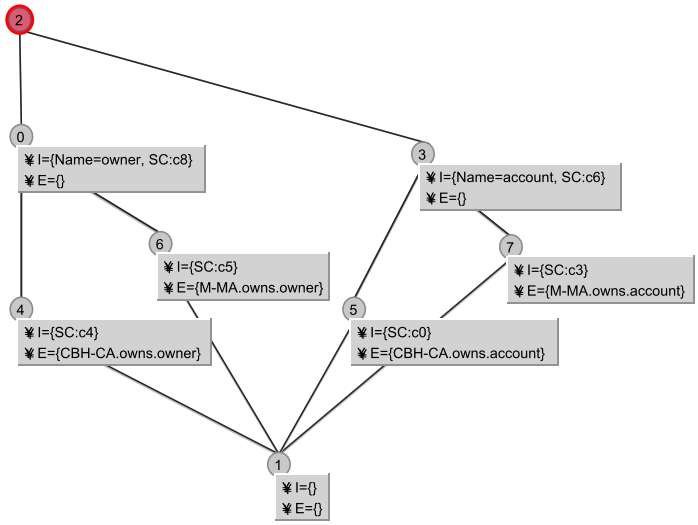


Figure 5. *Le treillis de concepts final du contexte des rôles d'associations.*

4. ARC et restructuration des modèles UML

Transformation d'un modèle UML vers une FCR : le modèle UML est décomposé en plusieurs contextes, un par type d'élément du modèle qui sera sujet à un processus d'abstraction (typiquement, classes et associations) ; puis ces contextes sont reliés par le biais de méta-associations, appelées *techniques*. Ces associations ont une sémantique indépendante du domaine sous-jacent et traduisent les liens qui existent entre éléments au sein du modèle. L'avantage d'un tel codage est la possibilité d'abstraire sur plusieurs types d'éléments et d'opérer des enrichissements mutuels entre les hiérarchies d'abstractions sous-jacentes.

La liste complète des contextes et des relations techniques a été établie suite à une étude du méta-modèle UML 1.5. Ceux-ci sont illustrés sur la figure 6, où certaines relations ont dû être nommées de façon plus explicite. En général, les objets formels de chaque contexte représentent des instances de la même méta-classe. Les attributs formels dans les différents contextes représentent les propriétés locales aux individus, c'est-à-dire, des méta-attributs. Toutefois, tous les méta-attributs ne sont pas nécessairement pertinents pour nos objectifs et certains peuvent même nuire à l'abstraction. Ainsi, il est inutile de chercher à factoriser des propriétés comme `isLeaf` ou `isRoot` car leur valeurs définitives ne peuvent que se déterminer une fois la restructuration terminée. La figure 3 montre une manière de coder les classificateurs et les rôles d'associations du modèle UML de la figure 1.

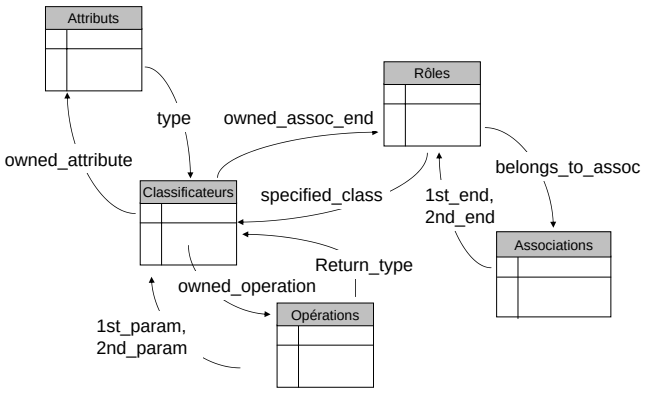


Figure 6. *Le schéma générique de la FCR d'un modèle UML (méta modèle UML 1.5).*

Parmi les difficultés que rencontre l'application systématique de l'ARC, il y a l'apparente antinomie entre la nécessité de préserver toutes les informations incluses dans le modèle UML initial, d'une part, et le besoin de réduire les abstractions fortuites, d'autre part. Ainsi, un modèle de taille raisonnable peut engendrer un grand nombre d'abstractions à faible utilité. En outre, un facteur de bruit important réside dans la polysémie du langage naturel qui est utilisé pour nommer les éléments du modèle. Par exemple, il est possible de trouver dans les grands modèles des éléments dont le nom est différent mais la sémantique est identique et inversement [DAO 03]. Ces problèmes ont été considérés dans le cadre de l'intégration de schémas de base de données et la solution standard habituellement suggérée était de résoudre les conflits de noms par une normalisation des noms en se servant de ressources linguistiques telles que des dictionnaires électroniques ou des ontologies de domaine [RAH 01].

Mesure de similarité : afin de détecter les conflits de noms entre les éléments UML, nous évaluons une mesure de similarité entre ceux-ci et détectons les cas suspects qui sont ensuite rapportés à l'expert humain. Une première méthode pour le calcul de similarité entre noms, suit la distance entre termes atomiques au sein de WORDNET [FEL 98]. L'autre mesure est basée sur des distances inter-documents bien connues en recherche d'information et utilise pour leur calcul le moteur de recherche LUCENE[3]. Rappelons que pour chaque terme, WORDNET enmagasine sa définition, ses synonymes (synset), ses homonymes, ses formes dérivées etc. De son côté, LUCENE permet de traduire les données en une collection de documents qui est par la suite soumise à une analyse et une indexation.

Avant de se lancer dans une recherche de similarité entre noms, les données du modèle UML (classes, attributs, méthodes, etc.) sont utilisées pour créer des index, un par type d'élément, de la manière suivante : d'abord l'analyseur procède à la *tokenisation*

3. http ://lucene.apache.org/java/docs/index.html

du nom de l'élément, par exemple le nom de classe `CheckBook` est séparé en `Check` et `Book` ; ensuite, les sens possibles des différents termes sont extraits de WORDNET. Par exemple `check` donne {`check`, `assay`, `arrest`} et `book` donne {`book`, `record`, `script`}. Le couple composé, d'une part, du nom de l'élément UML et d'autre part, de ses termes avec leurs synsets est placé dans l'index approprié. Le calcul de la similarité peut être réalisé de deux manières différentes :

– Distance WORDNET : en parcourant les noms des éléments de même type et en calculant les distances deux à deux. La mesure de similarité entre deux noms repose sur un appariement optimal, initialement implémenté dans l'outil d'alignement d'ontologies OLA[4] (OWL-Lite Alignment) [EUZ 04] et dont le principe consiste à chercher un appariement maximal entre les pools de termes des deux noms. Par exemple, étant donnés deux noms du modèle `invoiceAmount` et `customerOrderSum`, l'appariement retenu en se basant sur les distances WORDNET entre les termes composant ces deux noms est représenté par les deux paires (`amount`,`sum`) et (`invoice`,`order`).

– LUCENE : Le nom de l'élément UML avec ses termes et leurs synsets est considéré comme un document. Les documents des différents noms sont indexés par LUCENE. Ensuite, chaque nom d'élément est considéré comme une requête dans le moteur de recherche LUCENE. Par exemple, la requête qui correspond au nom `CheckBook` est composée de ce nom ainsi que ses termes {`check` et `book`} et finalement les synsets des termes {`check`, `assay`, `arrest`, `book`, `record`, `script`}. LUCENE fait alors un calcul de distance du cosinus entre chaque terme de l'élément dans la requête et tous les autres termes des noms éléments de la collection pris un par un. Il retourne une liste de documents (noms) triés selon le degré de similarité.

Génération du modèle UML à partir d'une FTR : la génération du modèle UML à partir d'une FTR est une opération complexe car, d'une part, les treillis peuvent être de taille exponentielle par rapport au nombre d'éléments du modèle UML initial et d'autre part, les relations inter-treillis peuvent être de nature circulaire et/ou transitive. La méthode est basée sur un processus exploratoire et semi-automatique faisant intervenir des outils de navigation de treillis et le concepteur pour préciser des choix de modélisation selon ses objectifs. La méthode se déroule en trois principales étapes :

– Sélection des concepts du treillis des classificateurs (concepts-classificateurs) de départ : cette étape consiste à déterminer les concepts-classificateurs qui correspondent aux classes du modèle initial. Ces concepts, une fois traduits en UML, représentent forcément des classes métier. Par exemple, les concepts du treillis final des classificateurs illustrés par la figure 4 représentant les classes du modèle initial sont : $c_{\#0}$, $c_{\#3}$, $c_{\#4}$ et $c_{\#5}$ et correspondent aux classes `CheckAccount`, `MortageAccount`, `CheckBookHolder` et `Mortager`, respectivement.

– Détection des abstractions significatives en parcourant les concepts-classificateurs retenus à l'étape précédente et en cherchant parmi leurs successeurs immédiats (Cov^u) les autres pertinents. Les concepts pertinents sont des concepts dont l'intension contient des informations métiers, soit directement, c'est-à-dire, à

4. http ://www.iro.umontreal.ca/∼owlola/align_files.html

travers les attributs formels comme *nom*=..., *type*=..., etc. exprimant la factorisation d'une information du domaine, soit indirectement, par la référence relationnelle vers un autre concept, éventuellement sur un autre type d'élément du modèle, qui a été reconnu comme pertinent auparavant. Par exemple, les concepts-classificateurs successeurs des concepts retenus lors de la première étape sont : $c_{\#6}$, $c_{\#7}$, $c_{\#8}$ et $c_{\#2}$ dont seuls $c_{\#6}$ et $c_{\#8}$ sont pertinents.

– Génération du modèle UML par la traduction en éléments UML des concepts-classificateurs retenus et des concepts des autres treillis qu'ils désignent. Ainsi, les concepts-classificateurs retenus sont traduits en classes UML, les liens d'héritage sont déduits de la relation d'ordre entre les concepts-classificateurs et l'intension de chaque concept-classificateur retenu est traduite en éléments de modélisation tels que attributs, rôles, opérations, etc. en exploitant les informations fournies par les concepts désignés dans les différents attributs relationnels.

1: **Algorithm** FTR2UML(**In :** RLF a family of lattices, **Out :** $\mathbf{M}$ an UML model)
2: Local : `Candidates` - a set of classifier-concepts to explore
3: Local : `Relevant` - a set of classifier-concepts to interpret as UML classes
4: Local : `Dummy` - a set of irrelevant classifier-concepts
5: Local : `Unknown` - a set of concepts whose relevancy can't be determinated
6:
7: `Candidates` ← COMPUTE-STARTING-CONCEPTS($\mathcal{L}^{\infty}_{classifiers}$)
8: `Relevant` ← $\emptyset$, `Dummy` ← $\emptyset$, `Unknown` ← $\emptyset$
9: **for all** $c \in$ `Candidates` **do**
10: **switch**(RELEVANT(RLF,c,$\emptyset$))
11: **case 1 :** {relevant classifier-concept}
12: `Relevant` ← `Relevant` $\cup \{c\}$
13: `Candidates` ← `Candidates` $\cup\, Cov^{u}(c) \setminus$ `Dummy`
14: **case 2 :** {irrelevant classifier-concept}
15: `Dummy` ← `Dummy` $\cup \uparrow c$
16: **case 3 :** {relevancy can't be determinated}
17: `Unknown` ← `Unknown` $\cup \{c\}$
18: USER-ASSESSMENT(`Unknown`,`Relevant`)
19: **for all** $c \in$ `Relevant` **do**
20: UPDATE($\mathbf{M}$,INTERPRET(FTR,c))
21: **return** $\mathbf{M}$

Algorithm 2: Génération d'un modèle structuré à partir d'une FTR.

L'algorithme 2 exprime les diverses étapes de la génération du modèle UML restructuré. Un stockage séparé est utilisé pour les divers types de concepts-classificateurs, à savoir, les concepts à examiner (`Candidates`), ceux déjà examinés et libellés pertinents (`Relevant`), les non pertinents (`Dummy`) et finalement ceux dont la pertinence n'a pu être établie (`Unknown`). L'algorithme s'appuie sur les routines RELEVANT, INTERPRET et UPDATE qui permettent de calculer la pertinence d'un concept, traduire en élément UML un concept donné et de mettre à jour le modèle final, respectivement. À noter que $\uparrow c$ désigne l'ensemble des successeurs du concept c dans le treillis. La routine USER-ASSESSMENT renvoie aux concepteurs du modèle les concepts de l'ensemble `Unknown` afin qu'ils se prononcent sur leur pertinence.

Nommage des nouvelles abstractions : il est nécessaire de donner de noms aux éléments du modèle restructuré. Alors que les éléments ayant un homologue dans le modèle de départ pourront se voir affecter le nom correspondant (ou combinaison de noms s'il s'agit d'une fusion d'éléments nommés différemment), les abstractions nou-

vellement découvertes posent un réel problème qui est encore ouvert. Les diverses pistes exploitées incluent la composition des noms des entités nommées ayant mené, par le partage d'éléments dans leurs description, à la création de l'entité abstraite. La composition pourrait prendre la forme d'un prefix commun à toutes les entités nommées ou, de façon moins rigide, à une concaténation de tous les noms. L'alternative est la recherche d'un terme, éventuellement composite, généralisant les noms des éléments en question. Celui-ci pourrait être recherché au sein des structures hiérarchiques du WORDNET, EUROWORDNET, ou tout autre thesaurus électronique.

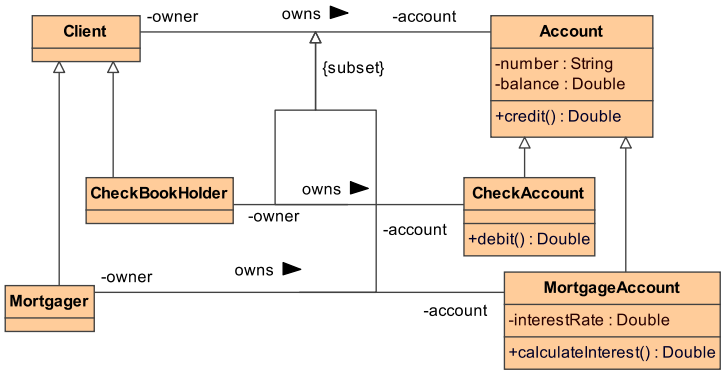


Figure 7. *Le modèle UML final obtenu après la restructuration du modèle de la figure 1.*

La figure 7 montre la traduction en éléments UML de tous les concepts-classificateurs pertinents ainsi que tous les autres types de concepts qui leur sont affiliés. Comparativement au modèle initial de la figure 1, le nouveau modèle présente un meilleur niveau d'abstraction et de factorisation. On constate l'émergence de deux nouvelles classes clés dans le domaine bancaire, à savoir, `Client` et `Account` avec une nouvelle association les reliant qui généralise les associations entre les classes dérivées.

5. Expérimentations

L'ARC et ses mécanismes ont été implémentés dans la plate-forme de manipulation de treillis GALICIA [VAL 03]. L'application de l'ARC à la restructuration des modèles statiques UML a été tout d'abord étudiée et validée dans le cadre du projet MACAO[5]. Pour cela, GALICIA a été interfacée avec l'atelier OBJECTEERING[6]. Dans le cadre du projet MACAO, divers systèmes propriétaires de France Télécom ont été analysés [DAO 03]. Ces systèmes, composés de plusieurs douzaines de classes, appartiennent à différents domaines d'application tels que les systèmes d'information, systèmes intranet et systèmes de services de télécommunication.

5. Un projet entre France Télécom, SOFTEAM et LIRMM. http ://www.lirmm.fr/~macao
6. http ://www.objecteering.com

Sans reprendre dans le détail tous les résultats des expériences, voici un aperçu de ceux qui concernent le premier projet (le système d'information général). Nous y trouvons à l'origine 57 classes réparties dans 8 sous-packages. La hiérarchie construite comporte 110 nouvelles classes, 9 nouvelles associations et 59 nouvelles propriétés. Une analyse par les concepteurs des modèles étudiés a conclu à l'utilité de 62 classes parmi les 110 nouvelles classes créées. Cependant, de nombreuses petites classes, comportant seulement 1 ou 2 propriétés propres, sont apparues ce qui n'est pas un facteur de lisibilité du modèle et constitue un point à améliorer.

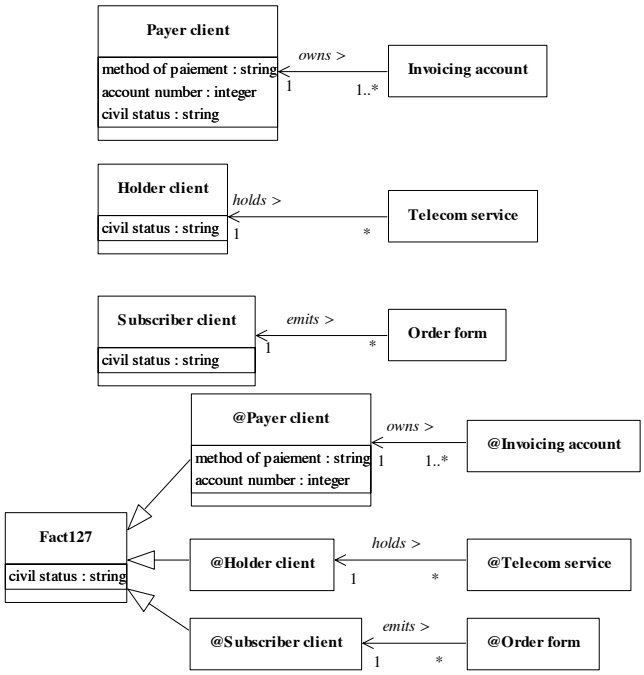


Figure 8. ***Haut :*** *le diagramme de classes original.* ***Bas :*** *le diagramme restructuré.*

La figure 8 présente la restructuration d'un petit sous-ensemble d'un diagramme de classes qui met en évidence la factorisation de l'attribut `civil status` présent dans les trois classes de gauche du diagramme original. Il est important de noter que les classes `Holder client` et `Subscriber client` sont conservées dans le diagramme restructuré grâce aux associations dont elles sont extrémités. De plus, la nouvelle classe `Fact127` pourrait certainement être automatiquement renommée (en `Client` par exemple) grâce à des techniques de traitement de langue que nous avons décrits.

Un ensemble d'outils ont été développés pour assister le concepteur du diagramme de classes dans l'exploitation des résultats de l'ARC par l'analyse de la factorisation des attributs, méthodes, associations, etc. [DAO 06]. Dans le cadre du projet GALICIA complété par des modules spécifiques pour UML, l'ARC a été également

utilisée pour la réorganisation du programme JETSMILES de la compagnie aérienne JETSGO qui comprend l'enregistrement des passagers, des itérinaires de voyage ainsi que les points de récompense cumulés. Là aussi les concepteurs ont fait la découverte de classes pertinentes parmi celles qui ont été suggérées par l'ARC. De plus, certaines abstractions jugées moins utiles ont permis de soulever des problèmes de modélisation négligés de la part des analystes.

6. Conclusion

L'AFC a été utilisée dans la réingénierie des modèles de classes [SNE 00, GOD 93, GOD 98] assurant ainsi une factorisation maximale des propriétés des classes et la détection d'abstractions de classes potentiellement utiles. L'ARC, de par sa faculté à représenter et à traiter l'information relationnelle, offre une meilleure factorisation des éléments d'un modèle car, d'une part, tous les constituants du modèle sont soumis à un processus de généralisation, notamment les rôles et les associations, et d'autre part, des abstractions utiles sont détectées en regroupant les individus non seulement à partir de leurs descriptions propres mais aussi à partir des liens qu'ils partagent. Une mise en perspective de l'ARC dans un cadre MDE est documentée dans [ARE 06].

Le travail rapporté ici se situe dans la continuité de notre étude [DAO 04] sur l'utilisation des techniques de l'AFC dans l'extraction de parties réutilisables de modèles de domaine décrits en UML. Nous apportons des améliorations dans plusieurs directions. Nous proposons un codage du modèle UML en se basant, d'une part sur le méta modèle UML dans la formation de la FCR et, d'autre part, sur des ressources linguistiques dans la résolution de conflits de noms durant ce codage. Le processus de dérivation des hiérarchies d'abstraction a été formalisé. Une méthode semi-automatique de génération d'un modèle UML à partir d'une FTR a été mise en place. Enfin des expérimentations ont été effectuées et rapportées. Afin d'améliorer la qualité des modèles produits par les outils de l'ARC, nous continuons à travailler sur l'optimisation de l'ensemble des techniques utilisées avec les différentes transformations du modèle ainsi que la recherche de nouveaux algorithmes permettant notamment de réduire les abstractions fortuites et l'exploration des treillis de grande taille.

7. Bibliographie

[ARE 06] AREVALO G., FALLERI J.-R., HUCHARD M., NEBUT C., « Building Abstractions in Class Models : Model Transformations coupled with Formal Concept Analysis », *Proceedings of MoDELS/UML 2006, Genova, Italy*, 2006.

[BAR 70] BARBUT M., MONJARDET B., *Ordre et Classification : Algèbre et combinatoire*, vol. 2, Hachette, 1970.

[BOO 99] BOOCH G., RUMBAUGH J., JACOBSON I., *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[DAO 03] DAO M., « Validation sur de grands projets, Projet MACAO (RNTL) », rapport n° sous-projet MACAO 5.1, december 2003, France Télécom R&D.

[DAO 04] DAO M., HUCHARD M., HACÈNE M. R., ROUME C., VALTCHEV P., « Improving Generalization Level in UML Models : Iterative Cross Generalization in Practice », *ICCS'04*, 2004, p. 346-360.

[DAO 06] DAO M., HUCHARD M., HACÈNE M. R., ROUME C., VALTCHEV P., « Towards Practical Tools for Mining Abstractions in UML Models », *Proceedings of ICEIS 2006*, INSTICC publisher, 2006, p. 276–283.

[EUZ 04] EUZENAT J., VALTCHEV P., « Similarity-based ontology alignment in OWL-lite », DE MANTARAS R. L., L. S., Eds., *Proc. of ECAI '04*, IOS Press, Amsterdam, 2004, p. 333–337.

[FEL 98] FELLBAUM C., *Wordnet : An Electronic Lexical Database*, MIT Press, 1998.

[GAN 01] GANTER B., WILLE R., *Formal Concept Analysis. Mathematical Foundations*, Springer, Berlin, 2001.

[GOD 93] GODIN R., MILI H., « Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices », *Proc. of OOPSLA'93, Washington (DC), USA*, 1993, p. 394–410.

[GOD 98] GODIN R., MILI H., MINEAU G., MISSAOUI R., ARFI A., CHAU T., « Design of Class Hierarchies Based on Concept (Galois) Lattices », *Theory and Practice of Object Systems*, vol. 4, n° 2, 1998.

[HUC 00] HUCHARD M., DICKY H., LEBLANC H., « Galois lattice as a framework to specify algorithms building class hierarchies », *Theoretical Informatics and Applications*, vol. 34, 2000, p. 521–548.

[RAH 01] RAHM E., BERNSTEIN P. A., « A survey of approaches to automatic schema matching », *VLDB Journal : Very Large Data Bases*, vol. 10, n° 4, 2001, p. 334–350.

[SNE 00] SNELTING G., « Software reengineering based on concept lattices », SOCIETY I. C., Ed., *Proc. of CSMR'00*, 2000.

[TON 99] TONELLA P., ANTONIOL G., « Object Oriented Design Pattern Inference », *Proceedings of ICSM '99*, Washington, DC, USA, 1999, IEEE Computer Society, page 230.

[VAL 03] VALTCHEV P., GROSSER D., ROUME C., HACENE M. R., « GALICIA : an open platform for lattices », B. GANTER A. D. M., Ed., *Using Conceptual Structures : Contributions to ICCS'03*, Aachen (DE), 2003, Shaker Verlag, p. 241-254.